LogiBLOX Guide

Introduction

Getting Started

Understanding Attributes

Module Descriptions

LogiBLOX Versus X-BLOX/ Memgen

any errors contained herein or to advise any user of this text of any correction if such be made. Xilinx, Inc. will not assume any liability for the accuracy or correctness of any engineering or software support or assistance provided to a user.

Xilinx products are not intended for use in life support appliances, devices, or systems. Use of a Xilinx product in such applications without the written consent of the appropriate Xilinx officer is prohibited.

About This Manual

This manual describes the Xilinx® LogiBLOX™ program, a tool used to create high-level modules for insertion into a schematic or an HDL-based design.

Before using this manual, you should be familiar with the operations that are common to all Xilinx software tools. These operations are covered in the *Quick Start Guide*. Other publications you can consult for related information are the *Development System Reference Guide*, *Libraries Guide*, and your third-party user guide.

Additional Resources

For additional information, go to http://support.xilinx.com. The following table lists some of the resources you can access from this page. You can also directly access some of these resources using the provided URLs.

Resource	Description/URL
Tutorial	Tutorials covering Xilinx design flows, from design entry to verification and debugging http://support.xilinx.com/support/techsup/tutorials/index.htm
Answers Database	Current listing of solution records for the Xilinx software tools Search this database using the search function at http://support.xilinx.com/support/searchtd.htm
Application Notes	Descriptions of device-specific design techniques and approaches http://www.support.xilinx.com/apps/appsweb.htm
Data Book	Pages from *The Programmable Logic Data Book*, which describe device-specific information on Xilinx device characteristics, including read-back, boundary scan, configuration, length count, and debugging http://www.support.xilinx.com/partinfo/databook.htm

Resource	Description/URL
Xcell Journals	Quarterly journals for Xilinx programmable logic users http://www.support.xilinx.com/xcell/xcell.htm
Tech Tips	Latest news, design tips, and patch information on the Xilinx design environment http://www.support.xilinx.com/support/techsup/journals/index.htm

Manual Contents

This manual covers the following topics.

- Chapter 1, “Introduction,” covers the features of LogiBLOX, the program components, the two possible design flows you can use to generate designs with LogiBLOX, and the different outputs generated by the program.
- Chapter 2, “Getting Started,” provides the basic procedures from setting up a LogiBLOX project and creating a module to placing that module in a schematic or in an HDL file.
- Chapter 3, “Understanding Attributes,” explains the major attributes that you can use to customize LogiBLOX modules.
- Chapter 4, “Module Descriptions,” describes each library module, including the input and output pins of the module and the attributes you can specify to change the functionality of the module.
- Appendix A, “LogiBLOX Versus X-BLOX/Memgen,” describes the differences between LogiBLOX and its predecessors, X-BLOX and Memgen.

Conventions

This manual uses the following typographical and online document conventions. An example illustrates each typographical convention.

Typographical

The following conventions are used for all documents.

- `Courier font` indicates messages, prompts, and program files that the system displays.

  ```
  speed grade: -100
  ```

- **`Courier bold`** indicates literal commands that you enter in a syntactical statement. However, braces "{ }" in Courier bold are not literal and square brackets "[ ]" in Courier bold are literal only in the case of bus specifications, such as bus [7:0].

  `rpt_del_net=`

  `Courier bold` also indicates commands that you select from a menu.

  `File` → **`Open`**

- *Italic font* denotes the following items.

  - Variables in a syntax statement for which you must supply values

    `edif2ngd` *design_name*

  - References to other manuals

    See the *Development System Reference Guide* for more information.

- Emphasis in text

  If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.

- Square brackets "[ ]" indicate an optional entry or parameter. However, in bus specifications, such as bus [7:0], they are required.

  edif2ngd [*option_name*] *design_name*

- Braces "{ }" enclose a list of items from which you must choose one or more.

  lowpwr ={**on**|**off**}

- A vertical bar " | " separates items in a list of choices.

  lowpwr ={**on**|**off**}

- A vertical ellipsis indicates repetitive material that has been omitted.

```
IOB #1: Name = QOUT'
IOB #2: Name = CLKIN'
.
.
.
```

- A horizontal ellipsis ". . ." indicates that an item can be repeated one or more times.

  allow block *block_name loc1 loc2* . . . *locn;*

Online Document

The following conventions are used for online documents.

- Red-underlined text indicates an interbook link, which is a cross-reference to another book. Click the red-underlined text to open the specified cross-reference.
- Blue-underlined text indicates an intrabook link, which is a cross-reference within a book. Click the blue-underlined text to open the specified cross-reference.

Chapter 1

Introduction

This chapter contains the following sections.

- "What is LogiBLOX?" is a general introduction to LogiBLOX.
- "Why Use LogiBLOX?" lists specific features by which LogiBLOX enhances design entry and module processing.
- "Program Inputs and Outputs" describes how LogiBLOX receives input in the form of pins and attributes that you specify in the module, and the different kinds of output files that LogiBLOX can create.
- "Schematic Design Flow" describes how to use LogiBLOX in a schematic-based environment.
- "HDL Design Flow" describes how to use LogiBLOX in synthesis-based designs.
- "Program Configuration" describes how LogiBLOX is configured when used in stand-alone mode.

What is LogiBLOX?

LogiBLOX is a graphical interactive tool for creating high-level modules, such as counters, shift registers, and multiplexers. LogiBLOX includes both a library of generic modules and a set of tools for customizing them.

The modules you create with LogiBLOX can be used in designs generated with schematic editors from Aldec®, Viewlogic®, Mentor Graphics®, and Cadence®, as well as third-party synthesis tools such as Synopsys® FPGA Compiler, Xilinx Foundation™, and Exemplar® Logic.

Use LogiBLOX modules whenever you need a customized version of a standard function. With normal design entry libraries (for example, the Xilinx Unified Library), you are constrained to whatever variations of a given function are provided in that library. Each variation of that type of function (for example, a counter) corresponds to a specific library component having a predefined set of inputs, outputs, and bus widths, and a predefined set of capabilities (loadable, synchronous, and so forth). With LogiBLOX, instead of having a separate component for each variation on a given function, you start with a generic template and tailor its I/O size and functionality according to your specific needs.

This manual describes how to create and process LogiBLOX modules for insertion into a schematic or an HDL-based design. The LogiBLOX graphical user interface (GUI) is available from your schematic editor package. With HDL-based designs, you can use the LogiBLOX Module Selector in its standalone mode to explicitly specify and generate LogiBLOX modules.

You can use LogiBLOX in two ways to design your modules.

- The Module Selector is a graphical user interface available in both schematic and synthesis-based environments. Use it to tailor modules to your requirements. This is the most common way to design modules in LogiBLOX.
- If you are a library user, you do not have access to the graphical user interface. Instead, you must describe the modules and their dependencies directly on the schematic.

LogiBLOX supports the following device families.

Table 1-1 Supported Device Families

Device Family	Sub-families
Spartan™	Spartan, SpartanXL™
XC3000™	XC3000A™, XC3000L™, XC3100A™, XC3100L™
XC4000™	XC4000E™, XC4000EX™, XC4000L™, XC4000XL™, XC4000XV™
XC5200™	XC5200, XC5200L™
XC9500™	XC9500, XC9500XL™

Why Use LogiBLOX?

LogiBLOX includes the following features that enhance design entry and module processing.

LogiBLOX Design Entry Features

- LogiBLOX facilitates design entry by allowing you to tailor complex logic blocks to precisely match your design's needs.
- The LogiBLOX graphical user interface allows you to quickly and easily specify complex modules with the assistance of interactive Design Rule Checker (DRC) checks and prompts.
- An image of the module with the specified pins and attributes is updated each time you activate a module attribute or connection in the Module Selector.
- The graphical user interface automatically disables selections that are incompatible with your current design selections.

LogiBLOX Processing Features

- In a synthesis-based environment, the modules you create with LogiBLOX are implemented as you incorporate them into the rest of your design.
- In LogiBLOX, a simulation model (VHDL, EDIF, or Verilog) is generated for each LogiBLOX module during design entry. This enables immediate simulation of LogiBLOX design without having to go through a logic implementation step.
- Many synthesis tools automatically infer LogiBLOX modules. You can also incorporate LogiBLOX modules in HDL designs through instantiation without compromising behavioral simulation support.
- Modules are synthesized quickly "on-the-fly" by the LogiBLOX module compiler.

Program Inputs and Outputs

The Module Selector is the LogiBLOX graphical user interface that you use to create a LogiBLOX module. Specifying a LogiBLOX module consists of a) selecting or deselecting optional pins on the

symbol, and b) specifying various module attributes. The result is a module customized for a specific function.

After you complete the module specification, LogiBLOX uses its symbol generator, model generator, and netlist generator to create one or more of the following outputs and store them in the current project directory.

- A schematic symbol for inclusion on the schematic

  The symbol generator can create a symbol definition file that your third-party interface converts into a schematic symbol.

- A behavioral VHDL simulation model

  The model generator creates a behavioral VHDL simulation model for the LogiBLOX module.

  The behavioral model permits the design to be simulated immediately in those environments that support mixed schematic and behavioral simulation.

- Verilog gate-level simulation netlist
- EDIF gate-level netlist, produced as an alternative simulation medium

  The netlist generator creates an EDIF gate-level netlist for the LogiBLOX module that can be converted to a third-party simulation format. These netlists are for simulation only and are not intended for design implementation.

- An NGO file for implementation of the module

Schematic Design Flow

To use the program in a schematic-based environment, follow these steps.

1. Invoke the Module Selector from within your design entry tool.

Note: For Cadence, the Module Selector must be invoked outside the schematic environment.

2. Specify your project directory using the LogiBLOX Setup window.
3. Select a base module type (for example, Counter, Memory, or Shift-register)

4. Customize the module by selecting pins and specifying attributes.
5. After completely specifying a module, click **OK**. Clicking **OK** initiates the generation of a schematic symbol and a simulation model for the selected module.
6. Place the module on your schematic.
7. Connect the LogiBLOX module to the other components on your schematic using ordinary nets, buses, or both.
8. Functionally simulate your design at any time.
9. Implement your design with the Xilinx implementation tools.
10. To simulate your design post-layout, convert your design to a timing annotated netlist and use the back-annotation flow appropriate to your CAE tools to generate a timing simulation netlist.

HDL Design Flow

The product flow for synthesis-based designs is as follows.

Module-Instantiation Tools

You can instantiate the LogiBLOX components in your HDL code to take advantage of their high-level functionality.

Express each LogiBLOX module in HDL code with a component declaration, which describes the module type, and a component instantiation, which describes how the module is connected to the other design elements.

Follow these steps to use the LogiBLOX program.

1. Invoke the Module Selector from an icon or from the command line.
2. Specify your project directory using the LogiBLOX Setup window. The default directory is your current directory.
3. Select a base module type (for example, Counter, Memory, or Shift-register, and so forth)
4. Customize the module by selecting pins and specifying attributes.

5. After completely specifying a module, click **OK**. Clicking **OK** initiates the generation of a component instantiation declaration, a behavioral model, and an implementation netlist.
6. Deposit the HDL module declaration / instantiation into your HDL design. The declaration is available as a .vei file for Verilog and a .vhi file for VHDL.
7. Complete the signal connections of the instantiated LogiBLOX module to the rest of your HDL design.
8. Behaviorally simulate your design. The HDL simulator sees the component declaration and looks for a behavioral model.

   Note: You may need to analyze the LogiBLOX HDL models with the appropriate CAE HDL tools for your particular simulator.
9. Implement your design by invoking the Xilinx implementation tools.
10. To simulate your design post-layout, convert your design to a timing netlist and use the back-annotation flow appropriate to your CAE tools.

Program Configuration

When you invoke LogiBLOX in stand-alone mode, the program looks for the logiblox.ini configuration file in your current directory. If the configuration file does not exist, the program displays a Setup window that enables you to set the vendor, project directory, device family, and outputs for LogiBLOX. The logiblox.ini file is created and stored in your project directory after you create the first LogiBLOX module. Thereafter, logiblox.ini is updated, if necessary, each time a module is processed.

Chapter 2

Getting Started

The LogiBLOX Module Selector is the LogiBLOX graphical user interface (GUI). This chapter explains how to start the Module Selector, configure your design directory, and create LogiBLOX modules.

This chapter contains the following sections.

- "Starting LogiBLOX" explains how to start LogiBLOX and use the Module Selector in both schematic and synthesis environments.
- "Getting Help" explains how to get online help for LogiBLOX.
- "Configuring Your Program" explains how to use the Setup window to initialize your project directory and specify the types of output files you want produced.
- "Adding a Module to Your Design" explains how to create and edit a LogiBLOX module, and include it in your design.

Starting LogiBLOX

LogiBLOX can be started either in stand-alone mode or from a third-party schematic entry tool.

In stand-alone mode, LogiBLOX is started by entering the following command on the command line.

```
lbgui
```

LogiBLOX is integrated into most third-party schematic entry tools. The Module Selector is available in both schematic and synthesis environments. Depending on which design entry tool you use, you can typically access the Module Selector as follows.

- Invoke the program from within your schematic CAE tool, usually from a pull-down menu or a toolbar.

- If you are in a synthesis-based environment and want to instantiate a LogiBLOX module in your HDL design, you can invoke the program as a stand-alone product by clicking on the LogiBLOX icon on a PC or by executing the program from the command line on a workstation.

After you are in LogiBLOX, you can customize standard modules and process them for insertion into your design. When you invoke the Module Selector from your schematic capture tool, the last-used module and its settings are displayed. If you select a LogiBLOX module in your schematic and start the Module Selector, the selected module appears, ready for editing.

Getting Help

Use the following methods to get help for LogiBLOX.

- To get context-sensitive help about the currently active control, press the F1 key. A dialog box appears.
- To get task-oriented help, use the **Help** button in each dialog box.

Configuring Your Program

When you run LogiBLOX for the first time, you must configure your project directory for LogiBLOX.

Note: You should not manually modify the configuration files used by LogiBLOX. LogiBLOX changes these files when you create or edit LogiBLOX modules.

Configuration Files

When you invoke LogiBLOX, the program looks for the *logiblox.ini* configuration file in your current directory. If logiblox.ini does not exist, the program displays a Setup window that enables you to set the vendor, project directory, device family, and outputs for LogiBLOX. The logiblox.ini file is created and stored in your project directory after you create the first LogiBLOX module. Thereafter, logiblox.ini is updated, when necessary, each time a module is processed.

The logiblox.ini File

The logiblox.ini file contains the information you recorded in the initial setup window. When LogiBLOX is started, it reads this initialization information and uses it to configure the Module Selector.

Following is an example of a logiblox.ini file.

```
GenerateVHDLModel=True
GenerateEdifModel=False
GenerateVerilogModel=False
GenerateVHDLInstantiation=True
GenerateVerilogInstantiation=False
GenerateNGDNetlist=False
IgnoreWarning=True
UserCancelled=True
TargetFamily=xc4000e
CAEVendor=synopsys
BusNotation=B<I>
PreviousModule=framecnt
```

Module Information

Also included in the project directory are the .mod files. These files record information about the modules you add to your design. Each .mod file represents a module that was generated for the current project. The .mod files are created and changed automatically by LogiBLOX.

Each module is recorded in a format similar to the following example.

```
module ACCUM
symbol accum4
family xc4000e
symboltemplate accum02
attributes
    BUS_WIDTH = 4
    OPTYPE = ADD_SUB
    REGISTERED = Q
    STYLE = MAX_SPEED
    ENCODING = SIGNED
    ASYNC_VAL = 2#1100#
    SYNC_VAL = 2#1001#
pins
    ADD_SUB
    C_IN
```

```
B(3:0)
LOAD
CLOCK
ASYNC_CTRL
SYNC_CTRL
Q_OUT(3:0)
OVFL
C_OUT
```

Setup Window

The setup window is displayed if the Module Selector does not find a logiblox.ini file in the current directory. This is the case when you start a new design.

Define the settings in this window each time you start a project. You can also edit these settings after you have created a module by clicking **Setup** in the Module Selector.

If you create a netlist for a symbol in your design and then change an option on the Setup window, you must recreate the netlist to reflect the new option settings.

Vendor Panel

Use the Vendor panel to select the 3rd-party vendor and associated bus notation.

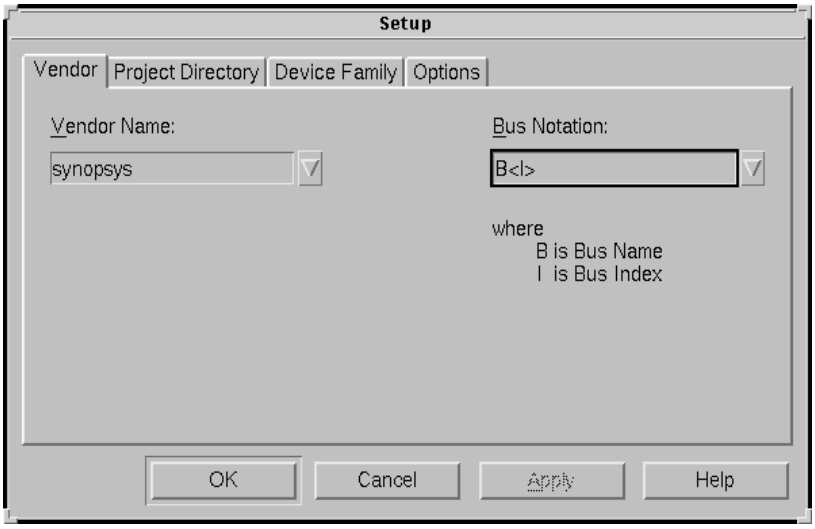

Figure 2-1 Vendor Panel (Setup Window)

Vendor Name

Select a vendor from the pull-down list box. You must choose one of the following in order to return to the Module Selector.

- Cadence
- Foundation
- Mentor
- Synopsys
- Viewlogic
- Other

Bus Notation

Different vendors use different notations to reference a bus index. When you select one of the vendors in the Vendor Name list, Logi-BLOX automatically assigns the default notation for that vendor. If

you select **Other**, you must also select a bus notation from the following choices. You must choose one in order to return to the Module Selector.

- B<I>
- BI
- B(I)
- B[I]

B is the name of the bus. I is the index of the bus. For example, if the bus notation is B<I> and the bus name is Q_OUT[2:0], the expanded bus will be Q_OUT<2>, Q_OUT<1>, Q_OUT<0>.

Project Directory Panel

The project directory is the directory in which LogiBLOX stores the logiblox.ini configuration file and all output files. Use the Project Directory choice in the Setup window to set the directory path to your project directory.

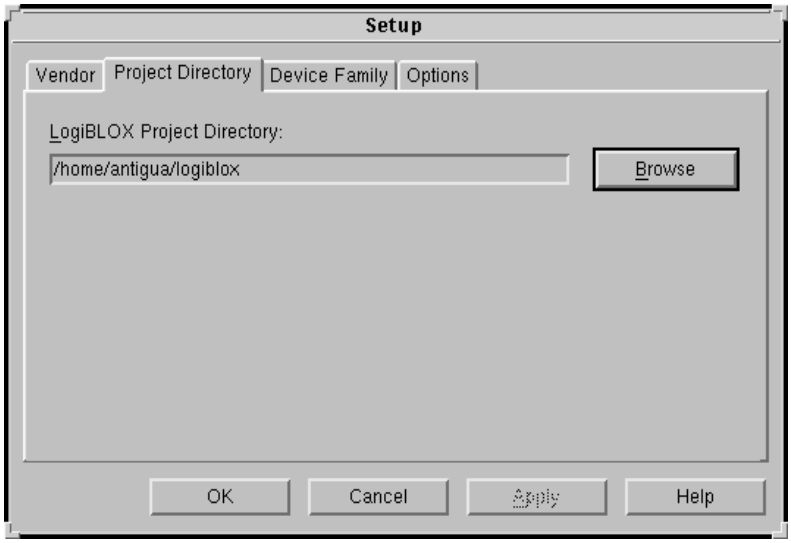

Figure 2-2 Project Directory Panel (Setup Window)

LogiBLOX Project Directory

You can define a project directory by doing one of the following.

- Open the Browse window and select a directory from the list of available directories.
- Enter the directory name directly in the Project Directory field.

Note: In schematic environments such as Viewlogic and Mentor, this option is disabled.

Note: In Windows 95®, if you start LogiBLOX by selecting 'Run' in the Start menu and entering 'lbgui' in the Run window, the default project directory may show as C:\WIN95\DeskTop. This will send all output generated by LogiBLOX to the desktop. You should change the project directory to something else, such as C:\LOGIBLOX.

Device Family Panel

Use the Device Family choice in the Setup window to select one of the supported Xilinx device families.

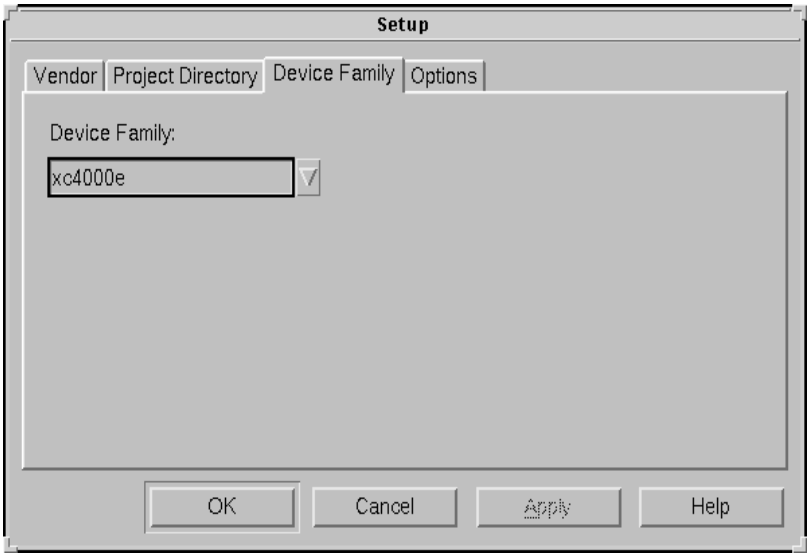

Figure 2-3 Device Family Panel (Setup Window)

Device Family

Select a device family from the pull-down list box. The default is the XC4000E family. The following device families are supported by LogiBLOX.

Spartan
SpartanXL
XC3000A
XC3000L
XC3100A
XC3100L
XC4000E
XC4000EX
XC4000L
XC4000XL
XC4000XV
XC5200

XC9500
XC9500XL

LogiBLOX checks whether the appropriate libraries are installed for each of these families. The families whose libraries are not installed will not appear in the list of choices in the pull-down list box.

This information is needed by the Design Rule Checker (DRC). It is also used for NGO file generation.

Options Panel

Use the Options choice in the Setup window to select the outputs you require for your design. Each of your selections (except the LogiBLOX DRC box) generates a file that is placed in the project directory.

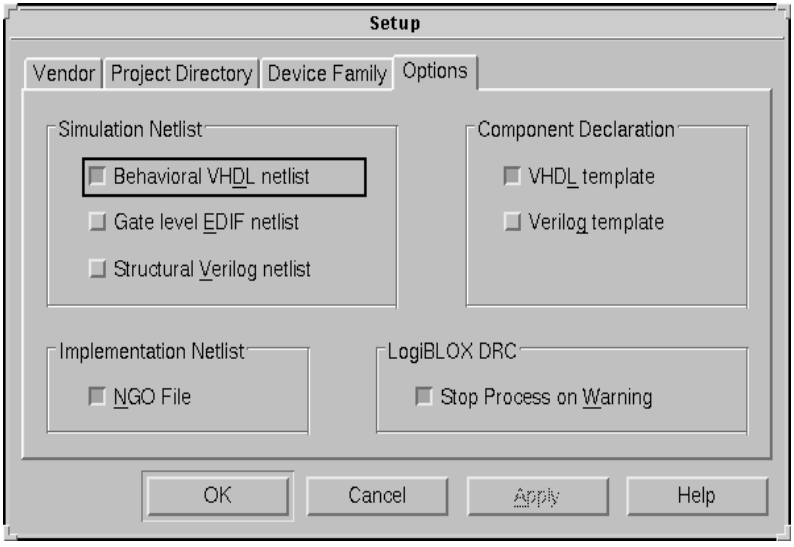

Figure 2-4 Options Panel (Setup WIndow)

There are four selection groups in the Options panel.

Simulation Netlist

The choices in this box create simulation netlists of the selected LogiBLOX module in different formats. You can choose one or more of the following outputs.

- **Behavioral VHDL netlist** — generates a simulation netlist in behavioral VHDL. The output file has a .vhd extension.
- **Gate level EDIF netlist** — generates a simulation netlist in EDIF format. The output file has an .edn extension.
- **Structural Verilog netlist** — generates a simulation netlist in structural Verilog. The output file has a .v extension.

Component Declaration

The choices in this box create instantiation templates in different formats that can be copied into your design. You can choose either, both, or neither of the following outputs.

- **VHDL template** — generates a LogiBLOX VHDL component declaration and instantiation template that can be inserted into your VHDL design when a LogiBLOX module is to be instantiated. The output file has a .vhi extension.
- **Verilog template** — generates a LogiBLOX Verilog module definition and instantiation template that can be inserted into your Verilog design when a LogiBLOX module is to be instantiated. The output file has a .vei extension.

VHDL component instantiations require a matching component declaration, while Verilog instantiations require a matching module port definition.

Implementation Netlist

Select **`NGO File`** to generate an implementation netlist in Xilinx-NGD binary format. You must select this option when instantiating LogiBLOX symbols in an HDL. The output file has an .ngc extension. NGDBuild will read these files when processing the top level module.

LogiBLOX DRC

Select the **`Stop Process on Warning`** check box to halt module processing if any warning messages are encountered during the design entry process.

Control Buttons

OK

Click **`OK`** to save your changes and close the Setup window. LogiBLOX saves the settings in the logiblox.ini file in the project directory. This information is used to configure subsequent sessions with the Module Selector.

Cancel

Click **Cancel** to close the Setup window without saving any of your changes.

Apply

This button is disabled. It is present to comply with the Microsoft Windows standard.

Help

Click **Help** to display help information for the various fields in the window.

User Preferences Window

Use the User Preferences window to select the editor you want to use for editing memory files. The default editor on UNIX™ platforms is vi. The default editor on Windows 95 and Windows NT® is Wordpad.

Select the **User Prefs** button in the Module Selector to bring up the User Preferences window

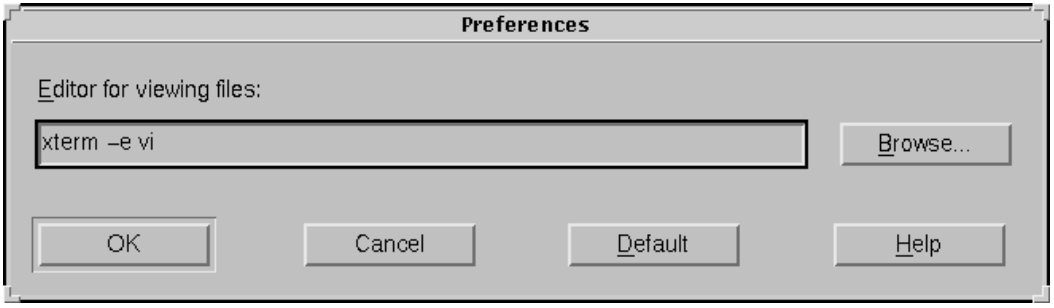

Figure 2-5 User Preferences Window

Adding a Module to Your Design

To design LogiBLOX modules, follow these steps.

1. Start LogiBLOX from your third-party design tool.
2. Select a base module type.
3. Customize the module by changing the pin settings and attributes if the defaults are not what you need.

4. Place the modules in the schematic or in the HDL code and connect them to the rest of your design.

A LogiBLOX module is displayed in the Module Selector GUI as an illustration that is dynamically updated as you modify the module's fields. The LogiBLOX symbols are bundled into a library provided with your third-party design editor. The LogiBLOX library must be in your editor's library search path for you to access it. For more information, please refer to your CAE Interface User Guide.

Choosing a Module

Each LogiBLOX module can be considered a template with a base function. Refer to the "Module Descriptions" chapter for a list of these modules by function. After you have decided which module to use, start the LogiBLOX program. The Module Selector window is displayed.

When you invoke the Module Selector, the program displays the details of the last module you generated. If you are invoking the LogiBLOX program for the first time, the Module Selector displays the Accumulator's default settings.

When you modify an existing module, that module and all its characteristics are displayed for editing.

Creating a Module

The graphical illustration of the module has check-boxes next to the optional pins. If you select a box, the symbol pin is displayed and is connected to the module. If you deselect the box, the pin disappears.

Use the attribute boxes to specify the module attributes. These attributes may affect the appearance of the module. Refer to the "Module Descriptions" chapter for a description of the module pins and attributes.

To specify your module, click on the pins you want to activate and the attributes you want to add to your module. The Module Selector dynamically reveals or hides the attributes and pins that are affected by your choice.

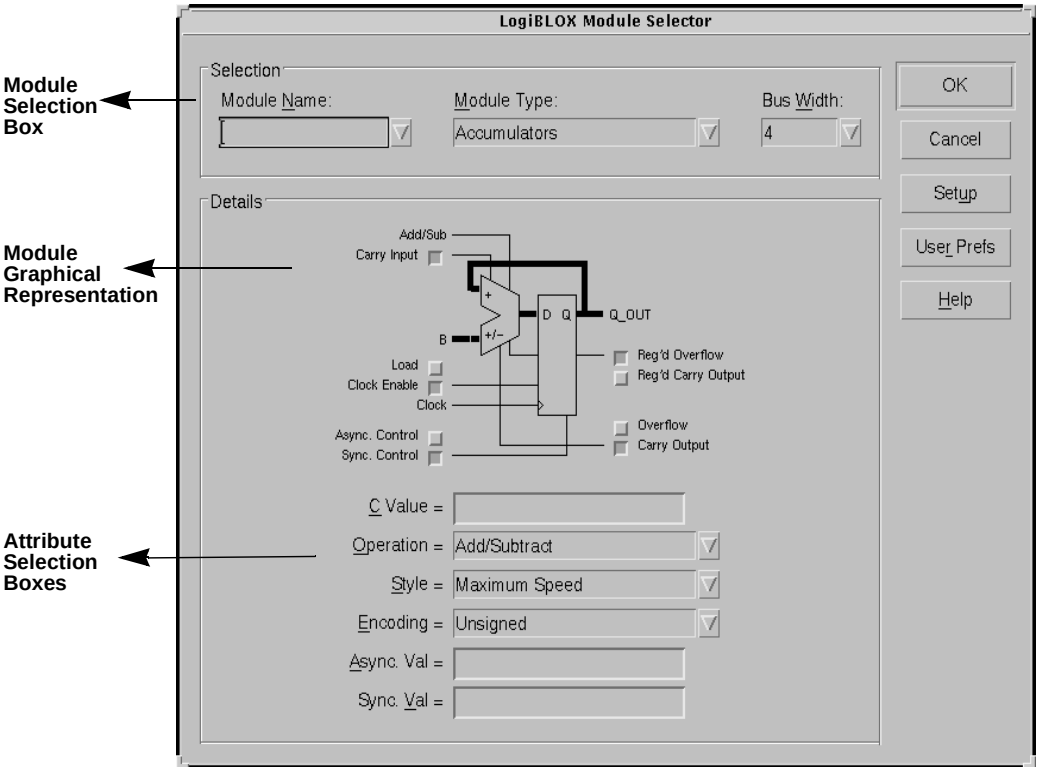

Figure 2-6 Module Selector Window

Module Selector Window Options

Module Name

Either type a name in the Module Name box to identify the module you want to create, or click on the pull-down arrow to see the list of modules that have already been created for your project. LogiBLOX uses this name to create files in which it saves the different outputs it generates for the module. If you select an existing module from the pull-down list and alter its characteristics, you must change the module name or you will overwrite the existing module. Overwriting an existing module implies that you want to update the selected module.

Note: Do not give a LogiBLOX module the same name as an existing component in the Unified Libraries.

Note: If you edit a LogiBLOX module that you have used multiple times in a design, either by repeatedly adding the same module or by copying it within the schematic tool, all copies of the module are also modified.

Module Type

The Module Type box displays the type of the currently selected module. Click on the pull-down arrow to select another module type from the list of available modules.

Bus Width

Specify the data bus width for the module. You can use one of the predefined settings (2, 4, 8, 16, or 32) from the pull-down menu, or type in a custom bus width in the text box. Generally, the minimum value allowed is 2 and the maximum value is 64. Decimal is the only valid base. All LogiBLOX bus pins are big endian, that is, the bits in a bus width of N are numbered N-1 through 0.

Details

The top part of the display shows a graphical illustration of the current module.

Use the check-marks to select or deselect optional pins on the symbol. For example, if you check the Overflow pin check box on the Accumulators module, the pin is automatically drawn on the symbol. Required pins, such as the Clock input pin, do not have check boxes.

Beneath the symbol image, LogiBLOX displays a list of the attributes that pertain to the module. You can assign values to particular attributes to further customize your module. To get online help for the attributes and their possible values, move the mouse pointer over the attribute, click the mouse button to move the focus, and press the F1 key.

OK button

Click **OK** when you have completely specified the module and are ready to insert it into your design. The program implements the module and generates the requested simulation models, component declarations, or both.

A log window is also displayed and records any errors and warnings reported by the DRC.

If the DRC completes without any errors, the Module Selector performs the following operations (depending on what you have requested in the Setup window).

- In schematic-based environments, it generates a symbol definition file (the .mod file) that describes the schematic symbol that is created for the module. It also launches the symbol generator utility with a pointer to this file.
- Instructs the schematic editor to place the symbol on the schematic or to replace the selected symbol with the new one.
- In HDL-based environments, it generates a component instantiation / declaration for inclusion in the HDL code.
- Calls the model generator to create a behavioral or gate-level simulation model for the current module.
- Calls the LogiBLOX module synthesizer to create a gate-level implementation netlist.

Cancel button

Click **Cancel** to quit the Module Selector. This operation discards any details you may have entered and leaves the project unchanged.

Setup button

If you wish to change device families or output types, click **Setup** and make your new choices in the Setup window.

Note: You cannot change the device family after you have generated a module in the current session. To target a different device family, you must exit lbgui first and restart it.

User Prefs button

Click **User Prefs** to set the editor you want to use for editing memory files. The default editor on UNIX platforms is vi. The default editor on Windows 95 and Windows NT is Wordpad.

Help button

Click **Help** to access the online help tool.

Editing a LogiBLOX Module

Part of the power of LogiBLOX is its ability to customize each module to represent many functions. For example, you can customize the ADD_SUB module to work as an adder only, a subtracter only, or as both an adder and a subtracter.

To customize a module, select only the LogiBLOX module control pins that are needed, specify a bus size for the module, and specify module attributes when the default modes are not the desired ones.

To customize the module using the Module Selector, use the pin connections and attribute setup fields in this dialog box.

The "Module Descriptions" chapter lists all LogiBLOX modules and the attributes that are appropriate for each module. The same attributes are described in full detail in the "Understanding Attributes" chapter.

Including the Module in the Design

After designing the module, click **OK** to generate the module. Next, place the module on the schematic or paste the instantiation into your HDL file.

Log Window

When you generate a module, a log window is displayed. The log window is a secondary window with scroll bars. LogiBLOX places any output messages generated by the DRC, symbol generator, model generator, and netlist generator processes in this window.

Each time you complete a module and press **OK**, the Module Selector calls the Design Rule Checker. If the DRC reports no errors, LogiBLOX generates the required output. If the DRC finds errors, LogiBLOX displays the error messages in the log window and the requested modules are not generated.

Connecting the Modules

This section provides information on connecting your LogiBLOX modules to the rest of your design.

Schematic Design

Insert LogiBLOX modules in your schematic and connect them with buses, nets, or both.

Note: You should specify bus widths using your CAE tool's normal mechanism. These values should match those on the module's bus pins.

HDL File

Fill out the instantiation section of the HDL template file by indicating the module's connections to signals in your design.

Changing a Module

This section provides information on changing or copying a LogiBLOX module.

Schematic Module

To change a module that you have already placed on your schematic, select the module and invoke the Module Selector. The Module Selector displays the settings of the module that you want to edit.

Copying Modules

If you copy a module within your schematic or add repeated instances to your schematic, the original module and all of its copies share the same module configuration and simulation model.

Subsequent modifications to any one of these modules change all copies of that module.

Warning: When editing a module, do not use the schematic editor's Editing commands. You must use LogiBLOX exclusively to ensure that the appropriate information in the .mod and behavioral files is up-to-date.

Copying Modules from Another Design

If you choose to copy a module from another design, such as by copying an entire hierarchical module, you must invoke the GUI, regenerate the module, and then dismiss it to recreate the module and create the simulation model for that module. Alternatively, if

your design includes several copied modules, you can copy the HDL files into the new project directory and reanalyze them in the new environment.

HDL Module

To complete your LogiBLOX module description, you must edit an HDL template and specify the module pin connections. Refer to the interface guide for your EDA tool for specific instructions.

After a LogiBLOX HDL module is instantiated, it must be modified with the Module Selector. By invoking the LogiBLOX GUI in standalone mode, you can load the previous module's details and modify them with new edits.

Chapter 3

Understanding Attributes

This chapter covers data values syntax and the most common LogiBLOX attributes. Data values are a combination of radices and numeric values that you assign to some LogiBLOX attributes. Attributes are used to customize LogiBLOX modules.

This chapter contains the following sections.

- "Data Values" explains the use of numeric values for some module attributes.
- "Implementation Styles" explains the different ways that some LogiBLOX modules can be implemented, depending on the device family.
- "Inverting and Decoding Masks for Gated Modules" explains the use of inversion masks and decode masks on Simple Gates modules.
- "Synchronous and Asynchronous Control" explains the use of synchronous and asynchronous pins and attributes in many of the LogiBLOX modules.
- "Location Attributes" explains the use of the Location attribute to place I/O modules in a specific IOB location.
- "OE Phase" explains the use of the OE Phase attribute to control the function of the Output Enable pin.
- "Constraining LogiBLOX Modules" explains how to constrain LogiBLOX modules by using the Floorplanner, or by attaching RLOC_ORIGIN and RLOC_RANGE constraints to them.

Data Values

Some module attributes are assigned numeric values. Data values consist of the arithmetic base, or radix, followed by the numeric data value in the specified base. LogiBLOX only allows bases 2, 4, 8, 10, and 16. The default base is decimal and does not need to be specified. The following formats are allowed.

base#value#
decimal_value

For example, the decimal value 17 can be expressed as follows.

```
Binary       2#10001#
Base 4       4#101#
Octal        8#21#
Decimal      17 or 10#17#
Hexadecimal  16#11#
```

The following attributes require a numeric data value.

- Async. Count and Sync. Count
- Async. Val and Sync. Val
- C Value
- Clock Divisor
- Count Limit
- Decode Mask
- Input Buses (decimal value only)
- Inversion Mask
- Memory Depth (decimal value only)
- Output Duty Cycle

Note: The Bus Width attribute also takes a numeric value, but it is restricted to decimal values between 2 and 64, inclusive.

When a constant value (C Value) is specified, the precision of the value might be less than the precision of the bus; if so, it will be sign-extended on the left (toward the MSB). If the value is greater than the

precision of the bus, the higher order bits are ignored. In this case, a warning is also printed.

The following table shows the valid characters for several radices.

Table 3-1 Valid Characters Using Various Base Values

Base Type	Base	Valid Data Value Characters
Binary	2	0 1 – _
Base 4	4	0 1 2 3 – _
Octal	8	0 1 2 3 4 5 6 7 – _
Decimal	10	0 1 2 3 4 5 6 7 8 9 – _
Hexadecimal	16	0 1 2 3 4 5 6 7 8 9 – _ A B C D E F a b c d e f

- Numeric data values must contain only characters valid for the specific radix.
- Negative data values are handled as twos-complement and are represented by a minus sign in front of the data value (for example, **`–2#0011# = 2#1101# = –3)`**.
- You can use the underscore character to increase the readability of numbers. The underscore characters have no value and are ignored by the software. For example, the value

  `2#0001001001110010011001110110100 1#`

  is more legible when it is formatted as

  `2#0001_0010_0111_0010_0110_0111_0110_1001#`

- If the LogiBLOX module contains two or more registered elements, the data values assigned to the Asynchronous Value and Synchronous Value attributes for each element are combined in a single assignment. The intended recipient of each data value is indicated by a token that precedes the data value. The token and data value are separated by a colon, and two token:data value pairs are separated by a period. For example, an Accumulator module may have the following assignment.

  `OVFL:1.REG:2#1111#`

  This line assigns a value of 1 to OVFL and a binary 1111 to REG.

Asserting a module's Asynchronous Control or Synchronous Control pin affects all the registers contained within that module.

Implementation Styles

Some LogiBLOX modules can be implemented in more than one way within the Xilinx architectures. The implementation methods are called "styles." Some styles use fewer Configurable Logic Blocks (CLBs) at the expense of speed, while other styles use more CLBs to achieve faster performance.

Types of Modules

This section covers modules that require the specification of an implementation style. A STYLE attribute value can be assigned to the LogiBLOX modules listed in the following table.

Table 3-2 Style Specification for Modules

Module	Possible Style
Accumulators Adders/Subtracters Counters[a]	ALIGNED ALIGNED RPM FAST 3KA MAXIMUM SPEED MINIMUM AREA RIPPLE CARRY UNALIGNED UNALIGNED RPM
Comparators	ALIGNED ALIGNED RPM EDGE DECODE MAXIMUM SPEED MINIMUM AREA RIPPLE CARRY TREE UNALIGNED UNALIGNED RPM WIRED AND
Data Registers	D-TYPE LATCHES

Table 3-2 Style Specification for Modules

Module	Possible Style
Decoders[b]	CASCADE MAXIMUM SPEED NORMAL GATES
Multiplexers	CASCADE F5_MUX MAXIMUM SPEED MINIMUM AREA NORMAL GATES WIRED AND
Simple Gates[c]	CASCADE EDGE DECODE MAXIMUM SPEED MINIMUM AREA NORMAL GATES WIRED AND

a.The STYLE attribute applies to binary counters only; it is ignored for counters with a non-binary encoding.

b.The STYLE attribute values listed here apply to the xc5200 family only.

c.The style attribute applies to Type_1 Gates only.

To have LogiBLOX automatically select the best style based on a speed or area preference, specify the MAXIMUM SPEED or MINIMUM AREA values. These attribute values indicate that LogiBLOX should choose the implementation style that best meets your needs in the target architecture.

Note: A particular module's choice of styles will vary depending on the device family selected or the type of logic used.

The following list summarizes the optimal style definitions by module and device family.

Accumulators, Adders/Subtracters, Counters

XC3000

MAXIMUM SPEED: FAST 3KA
MINIMUM AREA: RIPPLE CARRY

XC4000, XC5200

MAXIMUM SPEED: ALIGNED RPM
MINIMUM AREA: ALIGNED RPM

Comparators

XC3000

MAXIMUM SPEED: TREE
MINIMUM AREA: RIPPLE CARRY

XC4000 (Equality Comparisons)

MAXIMUM SPEED: ALIGNED RPM
MINIMUM AREA: TREE

XC4000 (Equality and Magnitude Comparisons)

MAXIMUM SPEED: ALIGNED RPM
MINIMUM AREA: ALIGNED RPM

XC5200

MAXIMUM SPEED: ALIGNED RPM
MINIMUM AREA: ALIGNED RPM

Multiplexers

XC3000, XC4000 (< 4-input Muxes)

MAXIMUM SPEED: NORMAL GATES
MINIMUM AREA: NORMAL GATES

XC3000, XC4000 (> 4-input Muxes)

MAXIMUM SPEED: WIRED AND
MINIMUM AREA: WIRED AND

XC5200 (< 16-input Muxes)

MAXIMUM SPEED: CASCADE
MINIMUM AREA: F5_MUX

XC5200 (> 16-input Muxes)

MAXIMUM SPEED: CASCADE
MINIMUM AREA: CASCADE

Simple Gates

XC3000, XC4000 (<= 4-input AND gates or > 4-input non-AND gates)

MAXIMUM SPEED: NORMAL GATES
MINIMUM AREA: NORMAL GATES

XC3000, XC4000 (> 4-input AND gates)

MAXIMUM SPEED: WIRED AND
MINIMUM AREA: WIRED AND

XC5200 (<= 7-input gates)

MAXIMUM SPEED: NORMAL GATES
MINIMUM AREA: NORMAL GATES

XC5200 (> 7-input gates)

MAXIMUM SPEED: CASCADE
MINIMUM AREA: CASCADE

Types of Styles

This section is an alphabetical reference of available styles.

ALIGNED, ALIGNED RPM, UNALIGNED, and UNALIGNED RPM

These styles apply to Accumulators, Adders/Subtracters, Comparators, and Counters. Aligned and Unaligned apply to all Spartan and XC4000 devices. Aligned RPM and Unaligned RPM apply to all Spartan, XC4000, and XC5200 devices.

Both styles use the fast carry logic and, therefore, constitute the fastest and smallest implementation styles for Spartan, XC4000, and XC5200 devices. On Spartan and XC4000 devices, the logic is aligned into a vertical column of CLBs when the Aligned option is selected. On XC5200 devices, the logic is aligned into two vertical columns of CLBs.

In RPM modules, the CLBs comprising the module maintain a constant relative position to each other.

In aligned modules, the initialization bit, i, occupies a CLB by itself and the first bit of the module, bit 0, starts a new CLB. As a result, the even bits of the module are aligned with CLB boundaries. In Spartan and XC4000 devices, the bits are grouped in twos and in XC5200 devices, bits are grouped in fours. Each group of bits occupies a single CLB.

- Bits (0, 1), (2, 3), (4, 5), … share the same CLBs in Spartan and XC4000 families
- Bits (0, 1, 2, 3), (4, 5, 6, 7), … share the same CLBs in XC5200 families

In unaligned modules, the initialization bit, i, shares the same CLB as bit 0. The rest of the bits are grouped and occupy CLBs as follows.

- Bits (i, 0), (1, 2), (3, 4), (5, 6), … share the same CLBs in XC4000 families
- Bits (i, 0, 1, 2), (3, 4, 5, 6), … share the same CLBs in XC5200 families

CASCADE

This style applies to Decoders, Simple Gates, and Multiplexers in XC5200 devices.

- When used in Decoders, this style implements functions using the dedicated carry logic multiplexer and function generators to serialize the decoding logic chain.
- When used in Simple Gates, this style implements functions using carry multiplexers to combine 4-input sub-functions. Wide functions implemented with this style are significantly faster and smaller.
- When applied to Multiplexers, this style uses carry multiplexer logic that is unique to this family. It is expandable with little area and timing impact.

Note that specifying the cascade style has placement implications. Specifically, the logic becomes aligned into a vertical column of CLBs.

D-TYPE

This style applies to Data Registers. When this style is used, regular flip-flops are constructed.

EDGE DECODE

This style applies to Simple Gates and Comparator symbols in XC4000 devices.

- When applied to Type 1 Simple Gate symbols, this style uses wide edge-decoders to implement the AND and NAND functions.
- When applied to equality operations of Comparators, this style uses a wired-AND implementation built using wide edge-decoders to detect patterns being applied through I/Os. You may only use it to compare a value against a constant.

Wide I/O decode functions using this style can be significantly faster than CLB-based implementations. The I/Os that are used in a decode or compare function will be placed on one edge of the chip.

FAST 3KA

This style applies to Accumulators, Adders/Subtracters, and Counters in XC3000A and XC3100A devices. It uses a gate implementation carry look-ahead adder. It is the fastest implementation style for carry-based modules in XC3000A and XC3100A devices. Modules implemented with this style are 50 percent larger but 30 percent faster than those implemented with the RIPPLE CARRY style.

F5_MUX

This style applies to Multiplexers in XC5200 devices and uses the fast carry logic technique that is unique to this family. It is most efficient for multiplexers with 16 inputs or less.

LATCHES

This style applies to Data Registers in XC4000EX, XC4000XL, XC4000V, and XC5200 devices. When used, the normal CLB registers are configured as transparent level-sensitive latches.

MAXIMUM SPEED

This style applies to Accumulators, Adders/Subtracters, Counters, Comparators, Decoders (XC3000 and XC5200 devices only), Multiplexers, and Simple Gates. It ensures that the fastest implementation style for the target architecture is used.

MINIMUM AREA

This style applies to Accumulators, Adders/Subtracters, Counters, Comparators, Multiplexers, and Simple Gates. It ensures that the smallest implementation style for the target architecture is used.

NORMAL GATES

This style applies to Decoders (XC3000 and XC5200 devices only), Multiplexers, and Simple Gates. It uses a gate implementation, that is, CLB look-up tables.

RIPPLE CARRY

This style applies to Accumulators, Adders/Subtracters, Counters, and Comparators in the XC3000A, XC4000, and XC5200 families. This style is not as efficient as the ALIGNED RPM style for XC4000 and XC5200 devices.

- When applied to Accumulators, Adders/Subtracters, and Counters, the RIPPLE CARRY style uses a gate implementation style.

  The RIPPLE CARRY style is smaller but slower for XC3000A devices than the FAST 3KA style.

- When applied to Comparators, this style uses a gate implementation ripple propagation compare and applies only to equality comparisons. The results are rippled from the MSB to the LSB. The RIPPLE CARRY style is the style that uses the fewest CLBs for equality comparisons in XC3000A devices.

TREE

This style applies to Comparators in XC3000A, XC4000, and XC5200 devices. It uses a gate implementation tree magnitude comparison and applies to all comparison operations in supported devices. It is the only way of implementing magnitude comparisons in XC3000A devices. For XC4000 and XC5200 devices, this style is not as efficient as the ALIGNED RPM or UNALIGNED RPM style.

WIRED AND

This style applies to Simple Gates, Multiplexers, and Comparators in XC3000A and XC4000 devices. Wide input functions with this style can be significantly faster.

This style uses tristate buffers (TBUFs) and horizontal long-lines. Logic is aligned into horizontal rows of CLBs next to the horizontal long lines.

Modules implemented in this style can only be as wide as the number of tristate buffers per horizontal long-line in the target device.

- When applied to Simple Gates, this style implements AND and NAND functions using a wired-AND implementation
- When applied to Multiplexers, this style uses wired-MUX implementation and increases the speed of wide MUX functions significantly, particularly in XC3000A and XC4000 devices.
- When applied to Comparators, this style uses a wired-AND implementation and applies to equality comparisons in XC3000A and XC4000 devices. It uses the same number of CLBs as the RIPPLE CARRY style.

Note: The number of TBUFs allowed in a row for a particular device is limited. PAR may fail if the Bus Width selected for the module is greater than the number of TBUFs allowed in a row for the specific device. Refer to the *The Programmable Logic Data Book* for the number of TBUFs allowed in the target device.

Inverting and Decoding Masks for Gated Modules

The INVERSION MASK and DECODE MASK attributes specify individually inverted or masked inputs and are available on Type 1, Type 2, and INVERT Simple Gates modules.

Table 3-3 Inversion and Decode Masks Summary

Attribute	Decode Input Value (default)	Invert Input Value
INVERSION MASK	0	1
DECODE MASK	1	0

The INVERSION MASK and DECODE MASK attributes use different polarities to achieve the same effect.

You can specify the inversion mask or the decode mask using any of the radices specified in the “Data Values” section in this chapter.

The default value is chosen to be intuitive and, therefore, depends on the nature of the module.

- On the bused AND, NAND, NOR, OR, XNOR, and XOR modules, no inversion is performed on the inputs by default.
- On the INVERT module, the INVERSION MASK indicates which bits in the bus will not be inverted. The default is to invert all the bits in the bus, which is what one would expect from a bus-wide inverter.

Type 1 Modules: One Input Bus

As an example, on a 5-bit Type 1 AND Gate with inputs 01011 and INVERSION MASK = 2#10010#, Bits 1 and 4 are inverted (bit 4 is the most significant bit). The resulting values are 11001. The DECODE MASK achieves the same result with opposite polarities. If DECODE MASK=2#01101#, Bits 1 and 4 are inverted and the resulting values are 11001.

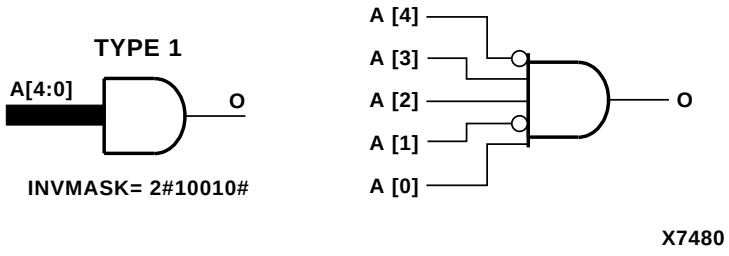

Figure 3-1 5-Input Type 1 AND Gate Using INVERSION MASK

Type 2 Modules: One Input Bus and One Input Signal

On a Type 2 AND Gate, only the input bus is affected by the INVERSION MASK, as shown in the following figure.

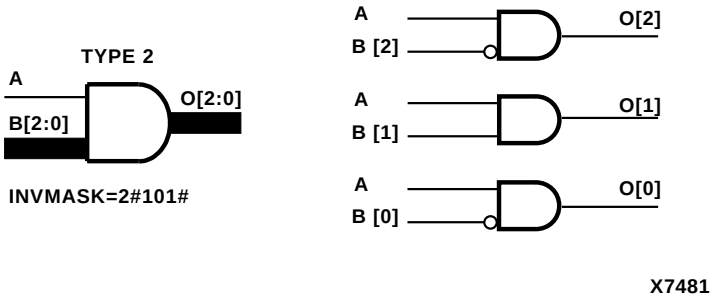

Figure 3-2 Type 2 AND Gate Using INVERSION MASK

INVERT Module

If you do not define INVERSION MASK, all signals connected to the INVERT module are inverted. If you specify INVERSION MASK, the outputs of the INVERT module are determined by the bit pattern of INVERSION MASK. For each bit in the INVERSION MASK that is 1, the corresponding bit in the bus is not inverted (in other words, the bit retains its original value).

You can specify the value of the INVERSION MASK in any base. Refer to the “Data Values” section for more information.

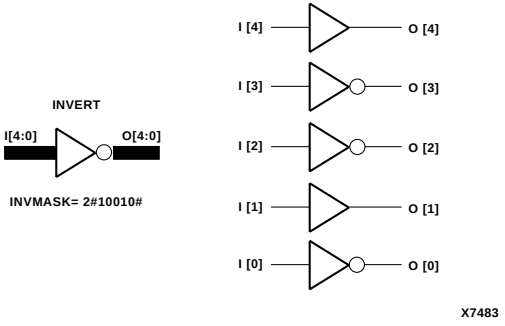

Figure 3-3 5-Input INVERT Using INVERSION MASK

Synchronous and Asynchronous Control

The synchronous/asynchronous control pins and attributes determine how modules containing flip-flops should be initialized after power-up or set or reset during operation.

- When the synchronous control pin is asserted, flip-flops go to their indicated values on the next rising edge of the clock.
- When the asynchronous control pin is asserted, flip-flops go to their indicated values immediately, independent of the clock.

LogiBLOX modules allow both types of control to be specified on the same module with different values for each type of control. This means that LogiBLOX allows you to set an entire register asynchronously to one value or synchronously to a different value. These values are constants specified by the ASYNC_VAL and SYNC_VAL attributes on the LogiBLOX modules and are independent of each other.

The modules that have synchronous (SYNC_CTRL) and asynchronous (ASYNC_CTRL) control pins include the following.

- Accumulators
- Adder/Subtracters
- Clock Dividers
- Counters
- Data Registers
- Shift Registers

You can specify the SYNC_VAL or ASYNC_VAL attribute values on all the above modules except the Clock Dividers module and LFSR Counters, which use the SYNC_COUNT/ASYNC_COUNT attributes. Refer to the Clock Dividers module and LFSR Counter sections for a description of the SYNC_COUNT and ASYNC_COUNT attributes.

ASYNC_VAL may also be applied to some registered I/O modules.

The values are loaded into the registers and counters under the control of the ASYNC_CTRL and SYNC_CTRL inputs of the module.

For Accumulator and Adder/Subtracter modules, which contain C_OUT and OVFL outputs, the values of the C_OUT and OVFL registers can also be specified for the case when ASYNC_CTRL or SYNC_CTRL is asserted. The values are specified in the ASYNC_VAL and SYNC_VAL fields in conjunction with the value of the accumulator register. Use the keyword REG to specify the Accumulator or Sum register value (this is optional), C_OUT to specify the asynchro-

nous or synchronous control value of the C_OUT register, and OVFL to specify the value of the Overflow register. Each data register type value is preceded by a colon, and separated from other data types with a "." (period) symbol.

For example, given an accumulator with the ASYNC_VAL or SYNC_VAL field set to either 1101.C_OUT:0.OVFL:1 or REG:1101.C_OUT:0.OVFL:1, the registers are set as follows.

- The value "1101" or "REG:1101" sets the accumulator or sum register to 1101 (the REG keyword is optional)
- The value "C_OUT:0" sets the carry out register to 0
- The value "OVFL:1" set the Overflow register to 1

The ASYNC_VAL constant is loaded immediately upon asserting the module's ASYNC_CTRL pin.

This load has priority over any clock-activated load.

The SYNC_VAL constant is loaded into the module if the module's SYNC_CTRL pin is High during the rising edge of the clock and the clock is enabled. SYNC_CTRL normally has priority over other synchronous functions on the same module. If the ASYNC_CTRL and SYNC_CTRL inputs are not connected, these functions are not synthesized.

Power-up Reset and Initialization

You can also use the ASYNC_VAL attribute to define a constant that is loaded at power-up or on assertion of the device's global reset net. If you do not specify the ASYNC_VAL attribute, all registers and counters except Linear-Feedback-Shift-Register counters (LFSR) and one-hot counters are set to zero at power-up. LFSR counters are set to their initial count state at power-up. One-hot counters start up with a value of one.

I/O modules can optionally take input and output registers. I/O modules with registers can be given an ASYNC_VAL attribute to define their power-up states.

When the Global Set Reset (GSR) is activated, the register-based modules, including Accumulators, Counters, Data Registers, and Shift Registers, are loaded with the setting given by the ASYNC_VAL

attribute. The same is true for the Global Reset (GR) in the XC3000A and XC5200 devices.

Location Attributes

Use the Location attribute in the Pads module (Pad Loc) to place I/O modules in a specific IOB location.

To assign a location to a specific bit, precede the location with a bit identifier. You can assign multiple bits by using a period as a separator. For example, with a bus width of 8 bits, you could specify the following assignment in the Pad Loc field.

```
0:P44.2:P45.7:P46
```

This specification assigns bit 0 to pad 44, bit 2 to pad 45, and bit 7 to pad 46. Note that all of the bit positions do not need to be specified.

Note: Commas will not work as separators between bit assignments.

OE Phase

The attribute OE_PHASE is used to control the function of Output Enable pin in the Tristate buffer and I/O modules. Its possible values are ACTIVE_LOW and ACTIVE_HIGH.

If OE_PHASE is ACTIVE LOW, then when OE_ENABLE is LOW, the Output = Input, and when OE_ENABLE is HIGH, the Output= High Impedance.

If OE_PHASE is ACTIVE HIGH, then when OE_ENABLE is HIGH, the Output = Input, and when OE_ENABLE = LOW, the Output= High Impedance

For the XC3000, XC4000, XC5200, and Spartan families, the only allowed value of OE_PHASE is ACTIVE_LOW. For XC9500, the only allowed value is ACTIVE_HIGH. For XC9500XL, you can set the value to either ACTIVE_LOW or ACTIVE_HIGH.

Constraining LogiBLOX Modules

LogiBLOX modules can be constrained by using the Floorplanner, or by attaching RLOC_ORIGIN and RLOC_RANGE constraints to them, if they are generated as RPMs (Relationally Placed Macros). Modules that can be generated as RPMs include those which contain

carry logic (Accumulators, Adders, Subtracters, Counters, and Comparators), registers (Data Registers), or RAM or ROM (Memories).

You can generate Accumulators, Adders, Subtracters, and Comparators as RPMs by setting the Module Style for these modules to either ALIGNED RPM or UNALIGNED RPM. Similarly, LogiBLOX Data Registers and RAM can be generated as RPMs by setting the USE_RPM attribute to TRUE in the LogiBLOX GUI.

The modules that can be generated as RPMs are listed in the table below.

Table 3-4 Modules that Can Be Generated as RPMs

STYLE	MODULE Type	Architecture
ALIGNED_RPM, UNALIGNED_RPM	Accumulators Adders/Subtracters Counters Comparators	XC4000, XC5200, Spartan XC4000, XC5200, Spartan XC4000, XC5200, Spartan XC4000, XC5200, Spartan
Attribute	**MODULE Type**	**Architecture**
USE_RPM	Data Registers RAMs	XC3000, XC4000, XC5200, Spartan XC4000, Spartan

LogiBLOX modules are most conveniently constrained to specific locations on an FPGA using the Xilinx Floorplanner. See the *Floorplanner Reference/User Guide* for more information on using this tool.

LogiBLOX modules can also be constrained using RLOC_ORIGIN and RLOC_RANGE constraints on a design schematic or in a UCF file.

Constraining LogiBLOX Modules in Schematics

To constrain a LogiBLOX RPM module to a specific location in a schematic, attach an RLOC_ORIGIN property to the LogiBLOX module symbol that specifies the target location for the upper left hand corner of the RPM.

To constrain a LogiBLOX RPM module to a specific range of CLBs in a schematic, attach an RLOC_RANGE=Rr1Cc1:Rr2Cc2 property to the LogiBLOX module symbol. This property specifies the range of CLBs between rows r1 and r2 and columns c1 and c2 to which the LogiBLOX module is directed.

See the Attributes, Constraints, and Carry Logic section of the *Libraries Guide* for more information on specifying RLOC_ORIGIN and RLOC_RANGE constraints.

Constraining LogiBLOX Modules in a UCF File

To constrain a LogiBLOX data register module into a range of CLBs in a UCF file, you must LOC every individual flip-flop in the UCF.

The flip-flops inside the LogiBLOX modules are named FLOP0, FLOP1, FLOP2, and so forth. A 20-bit data register contains FLOP0 through FLOP19.

For example, if the instance name (as opposed to the module name) of a LogiBLOX 20-bit data register module is L1, you can set the locations as follows in a UCF file.

```
INST L1/FLOP0 LOC=CLB_R1C1;
INST L1/FLOP1 LOC=CLB_R1C1;
INST L1/FLOP2 LOC=CLB_R2C1;
.
.
.
INST L1/FLOP19 LOC=CLB_R10C1;
```

Chapter 4

Module Descriptions

This chapter describes in alphabetical order the different types of modules, their connections, and their attributes. Each module represents a common logic function and is described in detail.

Note: The pin and attribute names in the following module descriptions are given in the following format: **Carry Input (C_IN)**. The first name is how the pin or attribute is listed in the GUI module. The second name, in parentheses, is how the pin or attribute is listed in the .mod file. If only one name is listed, then the name is the same in both places.

The following list divides the different LogiBLOX modules into functional categories and briefly summarizes each module.

Arithmetic

- ACCUMULATOR — Adds data to or subtracts it from the current value stored in the accumulator register.
- ADDER/SUBTRACTER — Adds or subtracts two data inputs and a Carry input.
- COMPARATOR — Compares the magnitude or equality of two values.
- COUNTER — Generates a sequence of count values.

Logic

- CONSTANT — Forces a constant value onto a bus.
- DECODER — Activates 1-of-n lines on the output port, based on the input address.
- MULTIPLEXER: Type 1, Type 2 — Routes 1-of-n input data lines to the output port.

- SIMPLE GATES: Type 1, Type 2, Type 3 — Implements the AND, INVERT, NAND, NOR, OR, XNOR, and XOR logic functions.
- TRISTATE BUFFER — Creates a tristated internal data bus.

I/O

- INPUT/OUTPUT — Connects internal and external pin signals.
- PAD — Represents an input/output pad.

Sequential

- CLOCK DIVIDER — Generates a clock pulse whose period is a multiple of the clock input period.
- COUNTER — Generates a sequence of count values.
- SHIFT REGISTER — Shifts input data bits to the left or right.

Storage

- DATA REGISTER — Captures the input data on active Clock transitions.
- MEMORY: ROM, RAM, SYNC_RAM, DP_RAM — Stores information and makes it readable.
- SHIFT REGISTER — Shifts input data bits to the left or right.

ACCUMULATOR

The Accumulator takes the data on its B input port and its Carry Input port and adds this data to or subtracts it from the current value stored in the accumulator register. It then loads the result back into the register, making it available at the Q_OUT port.

There are two types of accumulators: those that accumulate the value applied to the symbol's B input port and those that accumulate a constant value. The following figure illustrates the accumulation of values applied to the symbol's B pin.

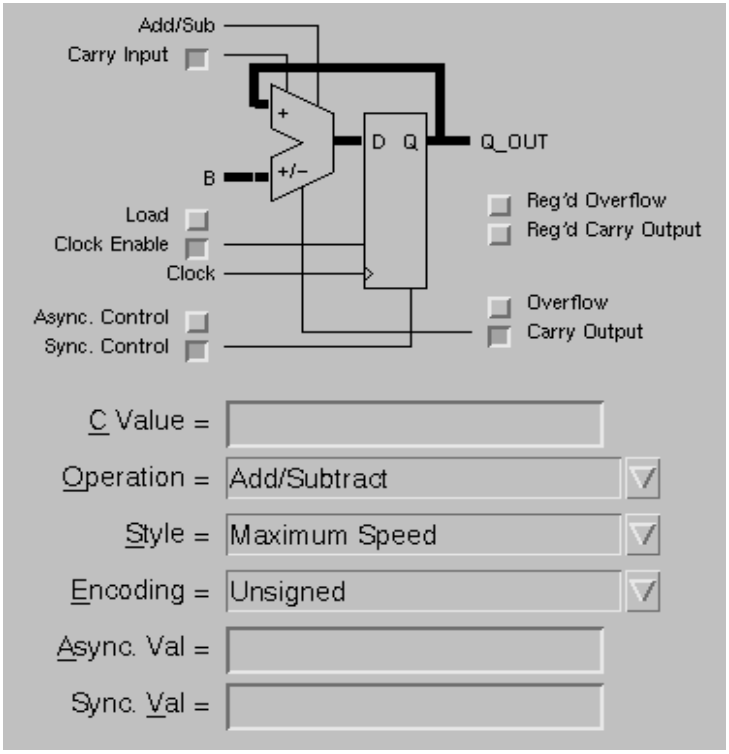

Figure 4-1 The Accumulator Module

The Carry Output and the Overflow outputs are generated by the Adder/Subtracter to indicate the status of the present arithmetic operation. You can latch these outputs by specifying the appropriate

attribute. You can also load separate predefined values synchronously or asynchronously into any or all of the module's registers.

Table 4-1 Accumulator Register Truth Table

Load	Sync. Control	Clock Enable	Clock	Async. Control	Q_OUT
X	X	X	X	H	ASYNC_VAL
X	X	L	X	L	Q_OUT_{prev}[a]
X	H	H	↑	L	SYNC_VAL
L	L	H	↑	L	Q_OUT_{prev} +/- B
H	L	H	↑	L	B

a. Q_OUT_{prev} denotes the previous contents of the register.

Input Pins

Add/Sub (ADD_SUB)

The input on this pin determines whether the module should add the value on the B input port to or subtract it from the value in the Accumulator register. If the input is High, the module performs an addition. If the input is Low, the module performs a subtraction.

Connections: This pin is available only if you set the Operation attribute to Add/Subtract. If you select either of the individual Add or Subtract operation modes, the Add/Sub input pin is removed from the module.

Carry Input (C_IN)

The Carry Input port, together with the data on the B input port, is added to or subtracted from the current accumulator register value.

Connections: Carry Input is optional. Its value depends on whether the Operation mode is Add or Subtract. If Carry Input is not specified, the default values are 0 for Add and 1 for Subtract.

B

The data on the B input port, along with the Carry Input port, is added to or subtracted from the current accumulator register value.

Connections: The B input is present if a Constant value is not specified. If you set the Constant Value attribute**,** you generate a schematic symbol with a Constant input instead of the B input.

Load (LOAD)

When the Asynchronous Control and the Synchronous Control pins are Low and Load is High, the Accumulator register is loaded directly with the value on the B input port on the next active Low-to-High Clock transition.

Connections: Load is optional. If you do not specify this input, the accumulator always adds or subtracts; it never loads.

Note: The Carry Out and Overflow outputs generated by the module's adder / subtracter are unknown during a load operation.

Clock Enable (CLK_EN)

Whenever the Clock Enable input is High, either the data on the B input port, the output of the adder / subtracter, or the value assigned to the Synchronous Value attribute is loaded into the accumulator register on the next active Low-to-High Clock transition.

If the Clock Enable input is Low, the register contents are unaffected by the Clock and the register holds its current value. This input does not affect asynchronous load operations, which occur when the Asynchronous Control pin is asserted.

Connections: Clock Enable is optional. Use this input when you need to disable the clock temporarily. If you do not use the Clock Enable input, the Clock is always enabled.

Clock (CLOCK)

If the Clock Enable input is High, the rising clock edge loads the selected data into the accumulator register. The falling (negative) clock edge can be used by connecting an inverter to the Clock input.

Connections: The Clock pin is always specified.

Async. Control (ASYNC_CTRL)

The Asynchronous Control input is a level-sensitive input. When this input is High, it loads the value assigned to the Asynchronous Value

attribute into the accumulator register independently of the Clock and Clock Enable.

Connections: If you specify the Asynchronous Control pin, you can assign a value to the Asynchronous Value attribute. By default, the Asynchronous Value is assigned a value of zero if it is not specified. The Asynchronous Value attribute may also be specified to define the accumulator register's power-on value.

Sync. Control (SYNC_CTRL)

Whenever the Synchronous Control and Clock Enable inputs are High, the value assigned to the Synchronous Value attribute is loaded into the accumulator register on the next active clock transition. This input has priority over the Load input if both pins are High at the same time.

Connections: If you specify the Synchronous Control pin and do not assign a value to the Synchronous Value attribute, a default value of 0 is used and a warning is issued.

Note: Asserting either Asynchronous Control or Synchronous Control affects all registered elements — the Accumulator register and potentially the Registered Carry Output and Registered Overflow registers. The Asynchronous Value and Synchronous Value attributes may contain data for all three registered elements. See the "Synchronous and Asynchronous Control" section of the "Understanding Attributes" chapter for more information.

Output Pins

Q_OUT

Q_OUT always reflects the current contents of the Accumulator register.

Connections: Q_OUT is always specified.

Overflow, Reg'd Overflow (OVFL)

Overflow is the overflow output from the adder/subtracter. Registered Overflow is the overflow from the Accumulator register. The overflow output is High when the result of the operation exceeds the maximum allowed value of the adder/subtracter.

Connections: Overflow and Registered Overflow are optional. If you specify one, you cannot specify the other.

Carry Output, Reg'd Carry Output (C_OUT)

Carry Output is the carry out from the most significant bit of the adder / subtracter. Registered Carry Output is the carry out from the most significant bit of the Accumulator register.

Connections: Carry Output and Registered Carry Output are optional. If you specify one, you cannot specify the other.

Attributes

C Value (C_VALUE)

Use the Constant Value attribute to replace the B input port with a constant value. If you do not define this attribute, the B pin is present.

Operation (OPTYPE)

Use the Operation attribute to specify one of the three possible arithmetic operation modes: Add, Subtract, or Add / Subtract. If you select Add / Subtract, the Add / Sub input pin is automatically added to the module.

Style (STYLE)

Style defines the implementation style (area or speed preference).

Usage: For more information, see the "Implementation Styles" section of the "Understanding Attributes" chapter.

Encoding (ENCODING)

Encoding defines the encoding scheme assumed in the generation of the overflow output logic. Valid values are Signed and Unsigned.

Async. Val (ASYNC_VAL)

The value of the Asynchronous Value attribute defines the power-on contents of the register. It also determines the data values assigned to the three registered elements: the Accumulator register, the Registered Carry Out register, and the Registered Overflow register. On assertion of the Asynchronous Control pin, the data values for the

specified pins are extracted from the value assigned to the Asynchronous Value attribute.

See the "Synchronous and Asynchronous Control" section of the "Understanding Attributes" chapter for more information about specifying the Asynchronous Value of these registers.

Usage: Asynchronous Value is always available. You can define it whether or not you use the Asynchronous Control pin. If you do not specify a value, the default value is zero.

Sync. Val (SYNC_VAL)

The Synchronous Value attribute defines the value to which the register returns on assertion of the Synchronous Control pin. It also determines the data values assigned to the three registered elements: the Accumulator register, the Registered Carry Out register, and the Registered Overflow register. On assertion of the Synchronous Control pin, the data values for the specified pins are extracted from the value assigned to the Synchronous Value attribute.

See the "Synchronous and Asynchronous Control" section of the "Understanding Attributes" chapter for more information about specifying the Synchronous Value of these registers.

Usage: Synchronous Value is available only if you specify the Synchronous Control pin.

ADDER/SUBTRACTER

The Adder / Subtracter module adds or subtracts two data inputs and a Carry Input. You can use this module as an adder, as a subtracter, or both. The Adder / Subtracter provides a Carry Output and an Overflow output to indicate the status of the current arithmetic operation.

There are two types of Adders / Subtracters: those that add or subtract the values applied via two bus-pins, and those that add a constant to or subtract a constant from a value applied to one bus-pin.

The Adder / Subtracter module permits the user to register any or all of its outputs.

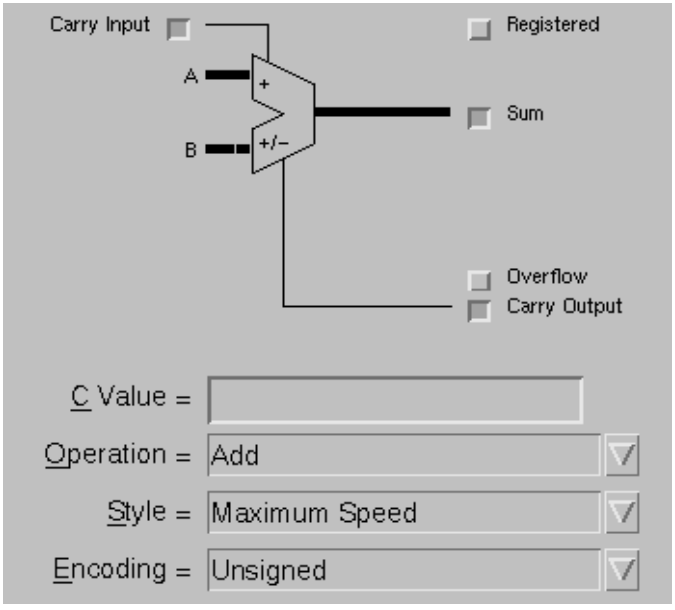

Figure 4-2 The Adder/Subtracter Module (Unregistered)

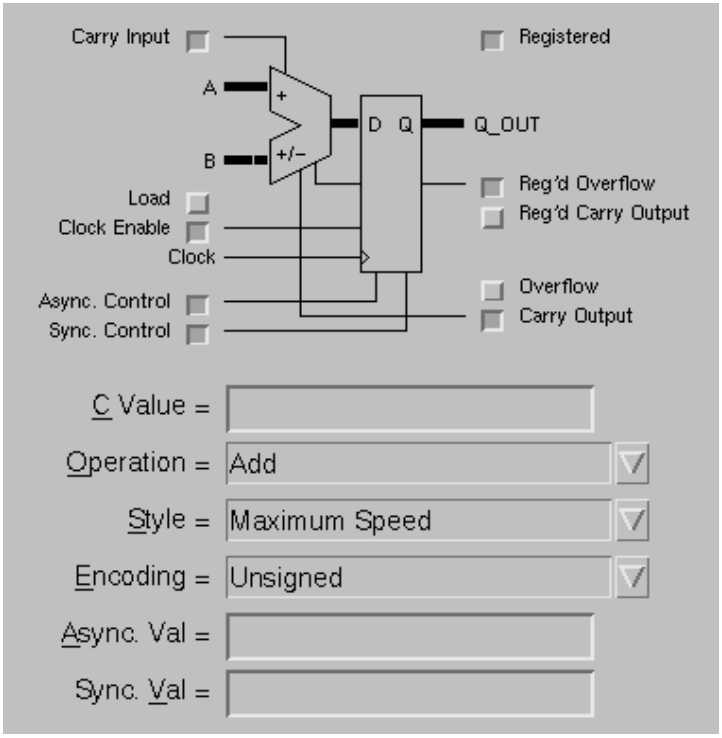

Figure 4-3 The Adder/Subtracter Module (Registered)

Input Pins

A

The data on the A input port, along with the Carry Input port, is added to or subtracted from the data on the B input port (A+B + C_IN or A – B – C_IN).

Connections: Pin A is required.

Add/Sub (ADD_SUB)

The input on this pin determines whether the module should operate as an adder or as a subtracter. If the input is High, the module performs an addition. If the input is Low, the module performs a subtraction.

Connections: This pin is available only if you set the Operation attribute to Add/Subtract. If you select either of the individual Add or Subtract operation modes, the Add/Sub input pin is removed from the module.

Carry Input (C_IN)

The Carry Input port, together with the data on the B input port, is added to or subtracted from the value on port A (A + B + C_IN or A – B – C_IN).

Connections: Carry Input is optional. Its value depends on whether the Operation mode is Add or Subtract. If Carry Input is not specified, the default values are 0 for Add and 1 for Subtract.

B

The data on the B input port, along with the Carry Input port, is added to or subtracted from the data on the A input port (A + B + C_IN or A – B – C_IN).

Connections: The B input is present if a Constant value is not specified. If you set the Constant Value attribute**,** you generate a schematic symbol with a Constant input instead of the B input.

Additional Input Pins for Registered Modules

Load (LOAD)

When the Asynchronous Control and the Synchronous Control pins are Low, the data on the B port is loaded directly into the register on the next active Clock transition.

Connections: Load is optional. If Load is High, the data on the B port is loaded directly into the register. If Load is Low, the output of Adder/Subtracter is registered instead.

Clock Enable (CLK_EN)

Whenever the Clock Enable input is High, either the data on the B input port, the output of the adder/subtracter, or the value assigned to the Synchronous Value attribute is loaded into the register on the next active Clock transition.

If the Clock Enable input is Low, the register contents are unaffected by the Clock and the register holds its current value. This input does

not affect an asynchronous load operation, which occurs when the Asynchronous Control pin is asserted.

Connections: Clock Enable is optional. Use this input when you need to disable the clock temporarily. If you do not use the Clock Enable input, the Clock is always enabled.

Clock (CLOCK)

If the Clock Enable input is High, the rising clock edge loads the selected data into the Adder/Subtracter register. The falling (negative) clock edge can be used for loading data by connecting an inverter to the Clock input.

Connections: The Clock pin is always specified when any of the Adder/Subtracter modules outputs are registered.

Async. Control (ASYNC_CTRL)

The Asynchronous Control input is a level-sensitive input. When this input is High, it loads the value assigned to the Asynchronous Value attribute into the Adder/Subtracter register independently of the Clock and Clock Enable.

Connections: If you specify the Asynchronous Control pin, you can assign a value to the Asynchronous Value attribute. By default, the Asynchronous Value is assigned a value of zero if it is not specified. The Asynchronous Value attribute may also be specified to define the Adder/Subtracter register's power-on value.

Sync. Control (SYNC_CTRL)

Whenever the Synchronous Control and Clock Enable inputs are High, the value assigned to the Synchronous Value attribute is loaded into the Adder/Subtracter register on the next active clock transition. This input has priority over the Load input if both pins are High at the same time.

Connections: If you specify the Synchronous Control pin and do not assign a value to the Synchronous Value attribute, a default value of 0 is used and a warning is issued.

Note: Asserting either Asynchronous Control or Synchronous Control affects all registered elements — the Adder/Subtracter register and potentially the Carry Output and Overflow registers. The Asynchronous Value and Synchronous Value fields may contain

data for all three registered elements. See the "Synchronous and Asynchronous Control" section of the "Understanding Attributes" chapter for more information about specifying the synchronous and asynchronous value of these elements.

Output Pins

One or more of the following output pins must be specified:

Sum (SUM)

Sum contains the result of the arithmetic operation performed on the Adder/Subtracter's inputs. The output is called Sum if the module is not registered. For a registered module, the output is Q_OUT.

Connections: The Sum bus is the output of the Adder/Subtracter and is required if no other output pin is specified.

Overflow (OVFL)

Overflow is the overflow output from the adder/subtracter. The overflow output is High when the result of the operation exceeds the maximum allowed value of the adder/subtracter.

Connections: Overflow is optional. Either the Overflow or the Registered Overflow output can be specified, but not both.

Carry Output (C_OUT)

Carry Output is the carry out from the most significant bit of the adder/subtracter.

Connections: Carry Output is optional. Either the Carry Output or the Registered Carry Output can be specified, but not both.

Additional Output Pins for Registered Modules

Q_OUT

Q_OUT always reflects the current contents of the Adder/Subtracter register.

Connections: Q_OUT is always specified for registered modules.

Reg'd Overflow (OVFL)

Registered Overflow is the overflow from the Adder/Subtracter register. The overflow output is High when the result of the operation exceeds the maximum allowed value of the adder/subtracter.

Connections: Registered Overflow is optional. Either the Overflow or the Registered Overflow output can be specified, but not both.

Reg'd Carry Output (C_OUT)

Registered Carry Output is the carry out from the most significant bit of the Adder/Subtracter register.

Connections: Registered Carry Output is optional. Either the Carry Output or the Registered Carry Output can be specified, but not both.

Attributes

C Value (C_VALUE)

Use the Constant Value attribute to replace the B input port with a constant value. If you do not define this attribute, the B pin is present.

Operation (OPTYPE)

Use the Operation attribute to specify one of the three possible arithmetic operation modes: Add, Subtract, or Add/Subtract. If you select Add/Subtract, an Add/Sub pin is automatically added to the module.

Style (STYLE)

Style defines the implementation style (area or speed preference).

Usage: For more information, see the "Implementation Styles" section of the "Understanding Attributes" chapter.

Encoding (ENCODING)

Encoding defines the encoding scheme assumed in the generation of the overflow output logic. Valid values are Signed and Unsigned.

Additional Attributes for Registered Modules

Async. Val (ASYNC_VAL)

The value of the Asynchronous Value attribute defines the power-on contents of the register. It also determines the data values assigned to the three registered elements: the Adder/Subtracter register, the Registered Carry Out register, and the Registered Overflow register. On assertion of the Asynchronous Control pin, the data values for the specified pins are extracted from the value assigned to the Asynchronous Value attribute.

Usage: Asynchronous Value is always available. You can define it whether or not you use the Asynchronous Control pin. If you do not specify a value, the default value is zero.

Sync. Val (SYNC_VAL)

The Synchronous Value attribute defines the value to which the register returns on assertion of the Synchronous Control pin. It also determines the data values assigned to the three registered elements: the Accumulator register, the Registered Carry Out register, and the Registered Overflow register. On assertion of the Synchronous Control pin, the data values for the specified pins are extracted from the value assigned to the Synchronous Value attribute.

Usage: Synchronous Value is available only if you specify the Synchronous Control pin. The default value is zero.

CLOCK DIVIDER

The Clock Divider module uses a Linear-Feedback-Shift-Register (LFSR) counter and decoder to generate an output pulse train that is a function of the clock input and the control attributes.

The Clock Output period is a multiple of the Clock period specified by the Clock Divisor attribute. Even multiples of the Clock period produce a 50 percent duty cycle on the Clock Output, while odd multiples produce a Low Output for one extra Clock period. You can use the Output Duty Cycle attribute to control the duty cycle if you need values other than 50 percent.

The Bus Width field is not applicable to this module and is therefore disabled.

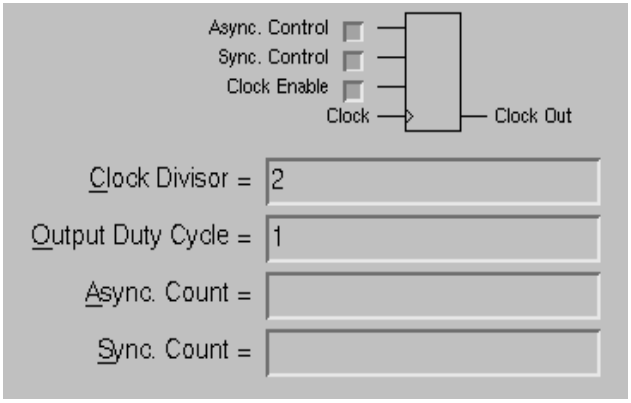

Figure 4-4 The Clock Divider Module

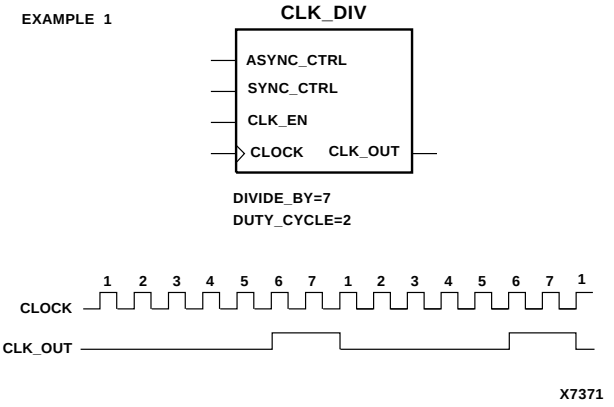

Figure 4-5 Simple Clock Divider Example

Input Pins

Async. Control (ASYNC_CTRL)

The Asynchronous Control input is a level-sensitive input. When this input is High, the Clock Divider's internal counter is reset to the value specified with the Asynchronous Count attribute.

Connections: If you specify the Asynchronous Control pin, you can assign a value to the Asynchronous Value attribute. By default, the Asynchronous Value is assigned a value of zero if it is not specified. The Asynchronous Value attribute may also be specified to define the accumulator register's power-on value.

Sync. Control (SYNC_CTRL)

Whenever the Synchronous Control and Clock Enable inputs are High, the Clock Divider's internal counter is reset to the value specified with the Synchronous Count attribute on the next active clock transition.

Connections: If you specify the Synchronous Control pin and do not assign a value to the Synchronous Value attribute, the default value is the first count in the LFSR count sequence and a warning is issued.

Clock Enable (CLK_EN)

When the Clock Enable input is High, the Clock Divider's internal counter increments on the next active Clock transition. When the Clock Enable is Low, the Clock Divider is unaffected by the Clock.

Connections: Clock Enable is optional. Use this input when you need to disable the clock temporarily. If you do not use the Clock Enable input, the Clock is always enabled.

Clock (CLOCK)

If the Clock Enable input is High, the rising clock edge increments the Clock Divider's internal counter. The falling (negative) clock edge can be used by connecting an inverter to the Clock input.

Connections: The Clock pin is always specified.

Output Pins

Clock Out (CLK_OUT)

The Clock Output port produces a pulse train whose period is a multiple of the period of the Clock input. The Clock Output has a 50 percent duty cycle except when the Clock Divisor attribute is assigned an odd number, in which case the Clock Output is Low for one extra Clock period. Alternatively, the duty cycle can be controlled with the Output Duty Cycle attribute.

Connections: The Clock Output pin is always specified.

Attributes

Clock Divisor (DIVIDE_BY)

The Clock Divisor attribute specifies the number of input Clock cycles for each Output Clock Cycle. This value must be a positive integer.

Usage: The Clock Divisor can be set to a value of 2 or higher. A value must be specified for this parameter.

Output Duty Cycle (DUTY_CYCLE)

The Output Duty Cycle attribute defines the High time of the output clock wave form in terms of multiples of the input clock period. This

value is an integer that is less than the Clock Divisor value. If Output Duty Cycle is not specified, a value of one-half the Clock Divisor value is used. If Clock Divisor is odd and Output Duty Cycle is not specified, the duty cycle is less than 50 percent because the output is High for only (n-1)/2 input clock periods.

Usage: Output Duty Cycle can be set to 1 or more but must be less than the value assigned to the Clock Divisor attribute.

Async. Count (ASYNC_COUNT)

The value of the Asynchronous Count attribute defines the power-on contents of the register. It also defines the value to which the register returns on assertion of the Asynchronous Control pin.

Usage: Asynchronous Count specifies the point in the Clock Divider's count sequence to which the internal LFSR counter returns on assertion of Asynchronous Control. For example, Asynchronous Count=0 causes the LFSR counter to return to the first count in its sequence.

Sync. Count (SYNC_COUNT)

The Synchronous Count attribute defines the value to which the register returns on assertion of the Synchronous Control pin.

Usage: Synchronous Count is available only if you specify the Synchronous Control pin. The Synchronous Count attribute specifies the point in the Clock Divider's count sequence to which the internal LFSR counter returns on assertion of Synchronous Control. For example, Synchronous Count=0 causes the LFSR counter to return to the first count in its sequence.

COMPARATOR

Comparator modules compare the magnitude of two values, their equality, or both their magnitude and equality. There are two types of comparators: those that compare the values applied to the symbol's two bus-pins, and those that compare a constant against the value applied to the A bus-pin.

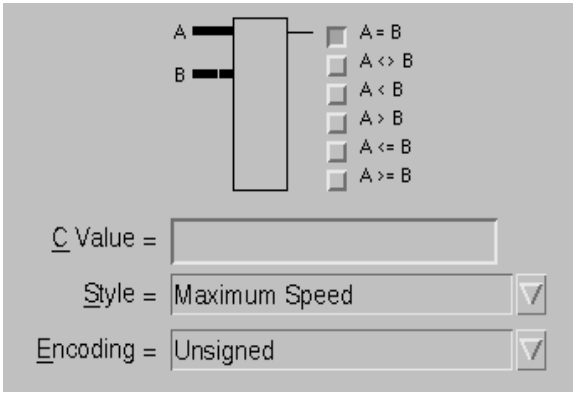

Figure 4-6 The Comparator Module

The following comparisons are available and can be used in any combination.

Table 4-2 COMPARE — Available Comparisons

Equality	Magnitude	
A=B	A<B	A>B
A≠B	A≤B	A≥B

Input Pins

A

The data on the A input port is compared to the data on the B input port.

Connections: A is a required input.

B

The data on the B input port is compared to the data on the A input port.

Connections: The B input is present if a Constant value is not specified. If you set the Constant Value attribute**,** you generate a schematic symbol with a Constant input instead of the B input bus.

Output Pins

At least one of the following outputs must be specified:

A = B (A_EQ_B)

This output is an active High output when the data on the A input port equals the data on the B input port.

A <> B (A_NE_B)

This output is an active High output when the data on the A input port does not equal the data on the B input port.

A < B (A_LT_B)

This output is an active High output when the value on the A input port is less than the value on the B input port.

A > B (A_GT_B)

This output is an active High output when the value on the A input port is greater than the value on the B input port.

A <= B (A_LE_B)

This output is an active High output when the value on the A input port is less than or equal to the value on the B input port.

A >= B (A_GE_B)

This output is an active High output when the value on the A input port is greater than or equal to the value on the B input port.

Attributes

C Value (C_VALUE)

Use the Constant Value attribute to replace the B input port with a constant. If you do not define this attribute, the B pin is present.

Style (STYLE)

Style defines the implementation style (area or speed preference).

Usage: Setting Style to Edge Decode or Wired AND causes all of the outputs to disappear except A = B and A <> B. For more information, see the "Implementation Styles" section of the "Understanding Attributes" chapter.

Encoding (ENCODING)

Encoding defines the encoding scheme used to represent the data type of the module. Valid values are Signed and Unsigned.

CONSTANT

The Constant module is used to force a constant value onto a bus. The Constant value is assigned to the module by means of a LogiBLOX attribute and can only be changed during the LogiBLOX module generation process.

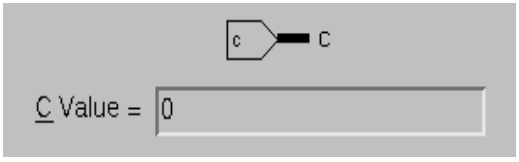

Figure 4-7 The Constant Module

Output Pins

C

The C output port contains the constant value specified by the user.

Connections: The C output pin is always present.

Attributes

C Value (C_VALUE)

Use the Constant Value attribute to define the data value that is forced onto a bus.

Usage: Edit the Constant Value attribute field to assign a value to the constant. Refer to the "Data Values" section of the "Understanding Attributes" chapter for information on how to specify the data values.

COUNTER

The Universal Counter module generates a sequence of count values determined by the selected encoding and the status of the control inputs. The Counter module can be an up counter, down counter, or up / down counter with a predefined asynchronous or synchronous pre-load, and a dynamic synchronous parallel load.

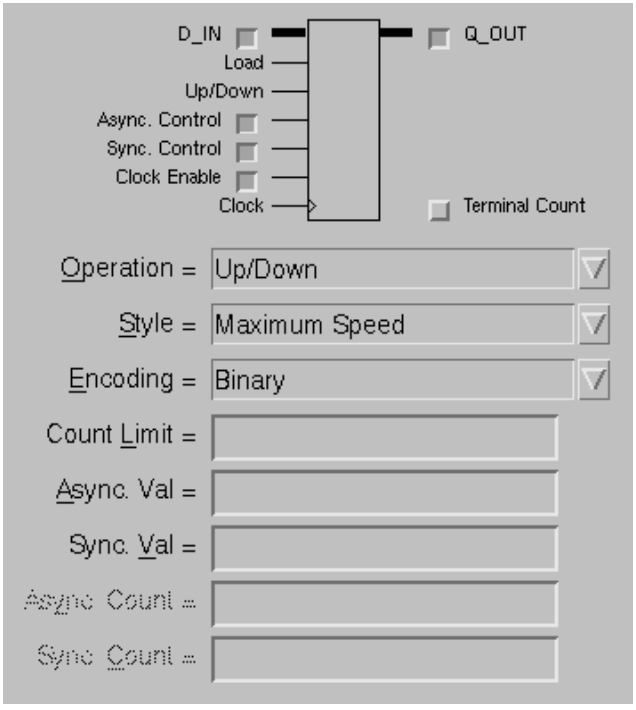

Figure 4-8 The Counter Module

LogiBLOX counter modules can be loaded with a value applied to a bus pin or with a pre-defined constant.

Table 4-3 Universal Counter Truth Table (Binary Style)

UP/DN	Load	Sync. Control	Clock Enable	Clock	Async. Control	Q_OUT	Terminal Count
X	X	X	X	X	H	Async_Value	X[a]
X	X	X	L	↑	L	Q_OUT_{prev}	$TERM\_CNT_{prev}$
X	X	H	H	↑	L	Sync_Value	X
X	H	L	H	↑	L	D_IN	X
H	L	L	H	↑	L	Q+1[b]	L
H	L	L	H	↑	L	Count Limit[c]	H
L	L	L	H	↑	L	Q–1	L
L	L	L	H	↑	L	Count Limit	H

a.An X in the Terminal Count column means this value is undefined, because Terminal Count is a function of the UP/DN and Q_OUT pins.

b.Q is the count value before the clock.

c.The Count Limit value has priority over the maximum (up)-(H...H) or minimum (down)-(L...L) count values (See Count Limit attribute for restrictions).

Input Pins

D_IN

The Parallel Data from the D_IN input port is loaded into the counter during a Parallel Load operation on an active Clock transition.

Connections: The D_IN port is optional. When it is specified, the Load pin is also specified. If the module uses the LFSR Encoding, the maximum bus width for this module is 31, otherwise, the maximum bus width is 64.

Load (LOAD)

When the Parallel Load input is High, the data on the D_IN input port is loaded into the counter on the next active Clock transition. When the Load input is Low, the counter responds to the Up/Down control input. In order for a Load operation to take place, Asynchro-

nous Control and Synchronous Control must both be Low and the Clock Enable must be High.

Connections: The Load input is automatically specified when the D_IN port is specified.

Up/Down (UP_DN)

The Up / Down control input controls the direction of the count on the next active Clock transition. When Up / Down is High, the counter value is increased by one; when Up / Down is Low, the counter value is decreased by one.

Connections: The Up / Down input pin is present only if the Up / Down Operation mode has been selected. Because the LFSR counter does not support down-counting, it must use the Up Operation type.

Async. Control (ASYNC_CTRL)

The Asynchronous Control input is a level-sensitive input. When this input is High, it loads the value assigned to the Asynchronous Value attribute (or Asynchronous Count for an LFSR counter) into the counter independently of the Clock and Clock Enable.

Connections: If you specify the Asynchronous Control pin, you can assign a value to the Asynchronous Value or Asynchronous Count attribute. By default, the attribute is assigned a value of zero if it is not specified. The Asynchronous Value and Asynchronous Control attributes may also be specified to define the counter register's power-on value.

Sync. Control (SYNC_CTRL)

Whenever the Synchronous Control and Clock Enable inputs are High, the value assigned to the Synchronous Value attribute (or Synchronous Count for an LFSR counter) is loaded into the counter on the next active clock transition. This input has priority over the Load input if both pins are High at the same time.

Connections: If you specify the Synchronous Control pin, you must assign a value to the Synchronous Value attribute (or Synchronous Count for an LFSR counter).

Clock Enable (CLK_EN)

When the Clock Enable input is High, the enabled load and count actions take place on the next active Clock transition. When Clock Enable is Low, the counter contents are unaffected by the Clock.

Connections: Clock Enable is optional. Use this input when you need to disable the clock temporarily. If you do not use the Clock Enable input, the Clock is always enabled.

Clock (CLOCK)

If the Clock Enable input is High, the rising clock edge either loads the selected data into the counter or increments / decrements the counter. The falling (negative) clock edge can be used by connecting an inverter to the Clock input.

Connections: The Clock input pin is always specified.

Output Pins

At least one of the output pins, Q_OUT or Terminal Count, must be specified.

Q_OUT

The Counter Output pin (Q_OUT) contains the current value of the counter.

Terminal Count (TERM_CNT)

The Terminal Count output pin goes High for one clock cycle every Count Limit cycles, where Count Limit is either specified by the user or, if not specified by the user, reaches its maximum value. The maximum value of Count Limit is listed in the "Counter — Encodings" table.

- For an Up Counter, the Terminal Count is High during the cycle in which the counter reaches its maximum sequence value.
- For a Down Counter, the Terminal Count is High during the cycle in which the counter reaches its minimum sequence value, typically zero.

For example, for a 4-bit binary Up Counter with no Count Limit attribute specified, the Terminal Count is High when the counter reaches its maximum value of 1111.

The Terminal Count is not qualified with the Clock Enable. To cascade counters, AND the Terminal Count with a common Clock Enable. Refer to the "Cascading Counters with Clock Enable" figure at the end of this section for more information.

Attributes

Operation (OPTYPE)

Use the Operation attribute to specify one of the three possible types of counters: Up, Down, or Up/Down. If you select Up/Down, an Up/Down pin is automatically added to the module. Note that when the Encoding attribute is set to LFSR, the only value allowed for Operation is Up.

Style (STYLE)

Style defines the implementation style (area or speed preference).

Usage: Maximum Speed is the default implementation style and is the only valid style for all counters other than Binary counters. When the Encoding attribute is set to Binary, Style can be set to any of the available values. For more information, see the "Implementation Styles" section of the "Understanding Attributes" chapter.

Encoding (ENCODING)

You can use this parameter to define the encoding of the Q_OUT port. When Encoding is set to LFSR, the only value allowed for Operation is Up, Style must be set to Maximum Speed, and the Asynchronous Count and Synchronous Count attributes are enabled instead of Asynchronous Value and Synchronous Value.

Usage: Encoding defines the count sequence of the counter. Refer to the following table for a list of available encodings.

Table 4-4 Counter — Encodings

Encoding	Counter Configuration	MAX Count Limit
Binary	Binary Counter	2^n-1
Johnson	Johnson Counter	2n
LFSR	Linear Feedback Shift Register	2^n–1
One Hot	Generates a ONE_HOT sequence.	n

Note: n is the width of the counter.

Count Limit (COUNT_TO)

The Count Limit value defines the number of cycles before the counter resets to its initial value, after which the count sequence restarts. Thus, Terminal Count will be High for one cycle every *Count Limit* cycles. You should specify Count Limit only if the length of the count sequence is different than the MAX Count Limit associated with the counter's encoding. The allowed values for Count Limit vary depending on the counter encoding chosen.

- Binary: The Count Limit attribute values are any number between 2 and 2^n-1 inclusive.
- Johnson: The only allowed Count Limit attribute values are 2n or 2n–1
- LFSR: The Count Limit attribute can be any number between 2 and 2^n–1 inclusive.
- One-Hot: The Count Limit attribute may not be set.

When Count Limit is used with a Binary counter, the following applies. If counting down, the counter counts down to 0 and on the next cycle, Q_OUT goes to Count Limit. If counting up, the counter counts up to Count Limit and on the next cycle, Q_OUT goes to 0.

The behavior of a counter loaded with a value outside the range of the Count Limit (with D_IN, Sync_Val, or Async_Val) is undefined. However, Binary and Johnson counters are guaranteed to return to legal count sequences after as many as 2^n (Binary) or 2n (Johnson) clock pulses.

Async. Val (ASYNC_VAL)

The value of the Asynchronous Value attribute defines the power-on contents of the counter. It also defines the value to which the counter returns on assertion of the Asynchronous Control pin.

Usage: Asynchronous Value is always available. You can define it whether or not you use the Asynchronous Control pin. If you do not specify a value, its default value depends on the kind of counter you specify with the Encoding attribute. For Binary and Johnson counters the default value is zero. For One-Hot counters the default value is 1. If the Encoding attribute is set to LFSR, you cannot assign a value to the Asynchronous Value attribute.

Sync. Val (SYNC_VAL)

The Synchronous Value attribute defines the value to which the register returns on assertion of the Synchronous Control pin.

Usage: Synchronous Value is available only if you specify the Synchronous Control pin. Note that when the Encoding attribute is set to LFSR, you cannot assign a value to Synchronous Value.

Async. Count (ASYNC_COUNT)

The Asynchronous Count attribute specifies the counter's reset state on assertion of the Asynchronous Control pin. The number specified for this attribute represents the number of transitions from the counter's initial default state. For example, if Asynchronous Count is set to 5, then the value in the counter after its first five transitions is the value assigned to the counter whenever the Asynchronous Control pin is asserted.

Usage: Asynchronous Count is always available. You can define it whether or not you use the Asynchronous Control pin. If you do not specify a value, the default value is zero. Use Asynchronous Count instead of Asynchronous Value when the Encoding attribute is set to LFSR.

Sync. Count (SYNC_COUNT)

The Synchronous Count attribute specifies the counter's reset state on assertion of the Synchronous Control pin. The number specified for this attribute represents the number of transitions from the counter's initial default state. For example, if Synchronous Count is set to 5,

then the value in the counter after its first five transitions is the value assigned to the counter whenever the Synchronous Control pin is asserted.

Usage: Synchronous Count is available only if you specify the Synchronous Control pin. Use Synchronous Count instead of Synchronous Value when the Encoding attribute is set to LFSR.

Counter Encoding Features and Selection Criteria

Each of the four counter encodings has benefits and limitations that are determined by the available chip resources. The "Counter — Encodings" table lists the criteria that can be used to select the appropriate encoding for each application. A brief description of each encoding follows.

Binary

The Binary Counter produces a predictable binary output pattern and is the recommended encoding for Up / Down counter applications. It is used to produce sequences for address generation, binary arithmetic, or related applications. Variations in count modulo are set by using the Count Limit attribute or Synchronous Load capability. The Binary Counter is synthesized to take advantage of fast carry logic on the XC4000 and XC5200 families. For a binary encoding, the width of the signal connected to D_IN and Q_OUT can be >= $\log_2$ Count Limit.

When the Style is set to Maximum Speed, the bus width determines the implementation of the Binary Counter in the XC4000EX family. If the bus width is greater than four bits, the counter is implemented using carry logic. If the bus width is four bits or less, gates are used in the implementation.

Johnson

The Johnson Counter is the fastest encoding available. This encoding is used to produce very fast state machines and glitchless decoders. It supports Asynchronous and Synchronous Loads, but the loaded values must correspond to the normal count sequence to maintain predictable output results. Valid values for the Synchronous Value and Asynchronous Value attributes include:

- All zeros

- All ones
- Zeros followed by ones
- Ones followed by zeros

For example, a 3-bit Johnson up counter sequence is as follows.

000
100
110
111
011
001

One bit in the count sequence changes per clock cycle if the default Count Limit=2n is used. If Count Limit is assigned by the user to 2n–1, then 2 bits will change during one clock cycle in the middle of the sequence as in the following 3-bit example with Count Limit=5.

000
100
110
011
001

LFSR

The LFSR counter is fast and uses chip resources efficiently. It can be configured to support any Count Limit value, but the output pattern is difficult to determine. It is used for frequency division, such as the CLK_DIV module, modulo x counting, and pseudo-random-pattern generation. It does not support down counting. The width of the LFSR counter must be between 2 and 31 bits. If more bits are needed, several LFSR counters can be cascaded. Refer to the "Cascading Counters" section section for examples of cascaded counters.

In an LFSR counter, the MSB bit toggles first.

Note: Specifying the Asynchronous Count and Count Limit attributes may result in unexpected behavior in an LFSR counter. For example, if Asynchronous Count is set to 2 and Count Limit is set to 5, at startup the counter should start with the second cycle and TERM_CNT should go high at the fifth cycle, that is, after three rising clock edges (this is how a Binary counter works). But TERM_CNT goes high after two rising clock edges. After that, it goes high every

fifth rising edge. This behavior occurs because a Count Limit of 5 specifies a Binary counter with six states (0 to 5), while it specifies an LFSR counter with five states.

One Hot

In this encoding, only one bit is High at a time. Use this encoding to enable a selection of mutually exclusive actions. Do not specify the Count Limit attribute for this encoding.

- An up-counter has the "1" in the least index bit at reset and shifts it one bit toward the greatest index bit at each clock cycle, returning it to the least index bit after n cycles. Terminal Count is High when the high bit is the highest index bit.
- A down-counter has the same initial state, but shifts toward the least bit and has Terminal Count High when the high bit is the least index bit.

Valid One Hot values for loading or assigning to the Asynchronous Value and Synchronous Value attributes may have only a single bit set. Ensure that neither multiple set bits nor no set bits are ever loaded into a One Hot counter, because this will result in improper count behavior.

Cascading Counters

If you need to cascade LogiBLOX counters, exercise caution if the counter module has a Clock Enable control signal. The Terminal Count outputs of all LogiBLOX counters do not incorporate the input Clock Enable control. As a result, the Clock Enable control signal of the first counter in the cascade chain needs to be ANDed with the Terminal Count of each counter. This ANDed term should then be used to enable the next counter in the chain.

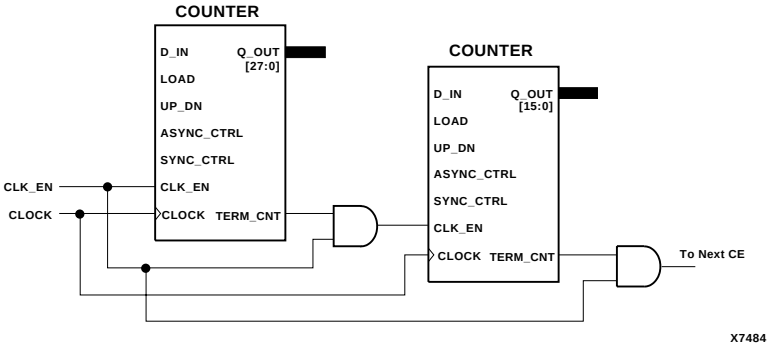

Figure 4-9 Cascading Counters with Clock Enable

LogiBLOX counter chains which do not have an initial Clock Enable control signal can be implemented simply by connecting the Terminal Count from each stage to the subsequent stage.

DATA REGISTER

The Data Register module is used to capture the data applied to its D_IN port on active Clock transitions. The contents of the data register are always present on the Q_OUT port. The module is synthesized as an array of flip-flops that can be loaded with predefined asynchronous and synchronous data.

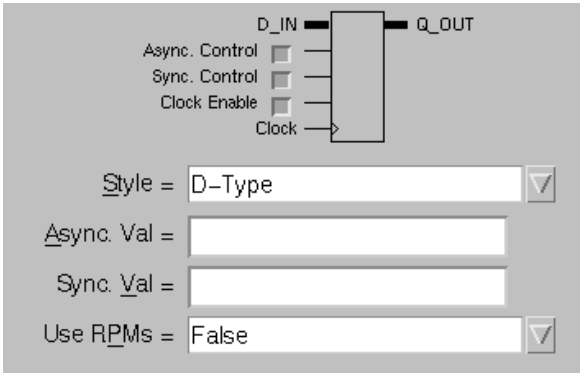

Figure 4-10 The Data Register Module

Table 4-5 Data Register Truth Table

D_IN	Sync. Control	Clock Enable	Clock	Async. Control	Q_OUT
X	X	X	X	H	ASYNC_VAL
X	X	L	↑	L	Q_OUT_{prev}
X	H	H	↑	L	SYNC_VAL
data	L	H	↑	L	data

Input Pins

D_IN

The data applied to the D_IN input port is loaded into the register when the Clock Enable is High and an active Clock transition occurs on the Clock pin.

Connections: The D_IN port is always specified.

Async. Control (ASYNC_CTRL)

The Asynchronous Control input is a level-sensitive input. When this input is High, it loads the value assigned to the Asynchronous Value attribute into the data register independently of the Clock and Clock Enable.

Connections: If you specify the Asynchronous Control pin, you can assign a value to the Asynchronous Value attribute. By default, the Asynchronous Value is assigned a value of zero if it is not specified. The Asynchronous Value attribute may also be specified to define the accumulator register's power-on value.

Sync. Control (SYNC_CTRL)

Synchronous Control is optional. Whenever the Synchronous Control and Clock Enable inputs are High, the value assigned to the Synchronous Value attribute is loaded into the data register on the next active clock transition. This input has priority over the Clock Enable input if both pins are High at the same time.

Connections: If you specify the Synchronous Control pin, you must assign a value to the Synchronous Value attribute.

Clock Enable (CLK_EN)

When the Clock Enable input is High, the D_IN input data or the value assigned to the Synchronous Value attribute is loaded into the register on the next active Clock transition. When the Clock Enable is Low, the register contents are unaffected by the Clock. This input does not affect asynchronous load operations, which occur when the Asynchronous Control pin is asserted.

Connections: Clock Enable is optional. Use this input when you need to disable the clock temporarily. If you do not use the Clock Enable input, the Clock is always enabled. When the Style attribute is set to Latches, this input becomes a Gate Enable.

Clock (CLOCK)

If the Clock Enable input is High, the rising clock edge loads the selected data into the register. You can implement an active falling (negative) clock edge by connecting an inverter to the Clock input.

Connections: The Clock pin is always specified. When the Style attribute is set to Latches, this input becomes a Gate.

Output Pins

Q_OUT

Q_OUT always reflects the Data Register's contents.

Connections: Q_OUT is always specified.

Attributes

Style (STYLE)

Style defines whether the module is implemented as an array of Latches or D-Type flip-flops.

Usage: Set this attribute to D-Type or Latches. The value D-Type is valid for all architectures. The value Latches is only valid for the XC4000EX and XC5200 device families.

Async. Val (ASYNC_VAL)

The value of the Asynchronous Value attribute defines the power-on contents of the register. It also defines the value to which the register returns on assertion of the Asynchronous Control pin.

Usage: Asynchronous Value is always available. You can define it whether or not you use the Asynchronous Control pin. If you do not specify a value, the default value is zero.

Sync. Val (SYNC_VAL)

The Synchronous Value attribute defines the value to which the register returns on assertion of the Synchronous Control pin.

Usage: Synchronous Value is available only if you specify the Synchronous Control pin.

Use RPMs (USE_RPM)

The Use RPMs attribute determines whether the data flip-flops in the register maintain a constant relative location to each other. This attribute applies to the XC4000 and XC5200 device families only.

Usage: Use RPMs can be set to True or False. The default is False.

DECODER

The Decoder module converts unsigned binary data on the Select port to a 1-of-n one-hot output on the D_OUT port. The width (precision) of the Select and D_OUT ports is determined by the Bus Width attribute.

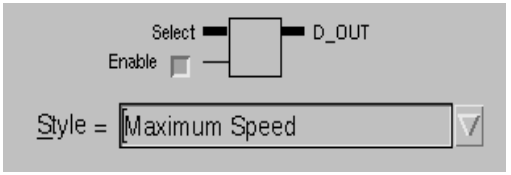

Figure 4-11 The Decoder Module

Input Pins

Select (SEL)

The data on the Select pin is converted to a one-hot value on the D_OUT port.

Connections: The Select pin is always specified. Specify its bus width by selecting the appropriate width from the Bus Width pull-down list box. The valid range of values for the bus width is 2 through 8.

Enable (ENABLE)

When the Enable Input is High, the selected Output is High. When the Enable Input is Low, the Decoder is disabled and all bits of the Output are Low. If the Enable Input is not specified, the Decoder is always enabled.

Connections: The Enable input is optional.

Output Pins

D_OUT

When enabled, one of the lines of the D_OUT port will be High (one-hot encoding). The width of the D_OUT pin is 2^n, where n is the width of the Select pin.

Connections: The D_OUT pin is always specified.

Attributes

Style (STYLE)

The Style attribute is only enabled for the XC5200 device family. You can select Cascade, Maximum Speed, or Normal Gates. For more information, see the "Implementation Styles" section of the "Understanding Attributes" chapter.

For other device families, Style is always set to Normal Gates.

INPUT/OUTPUT

Input/Output modules represent the physical pins on a device that are actually used in a design. Pad modules are connected to the I/O modules to simulate input/output wires.

There are three types of I/O modules. These modules represent all the possible logic combinations that can fit into the Input/Output blocks (IOBs).

Input Modules

Input modules are device-input modules that connect a Pad to an internal pin signal or bus.

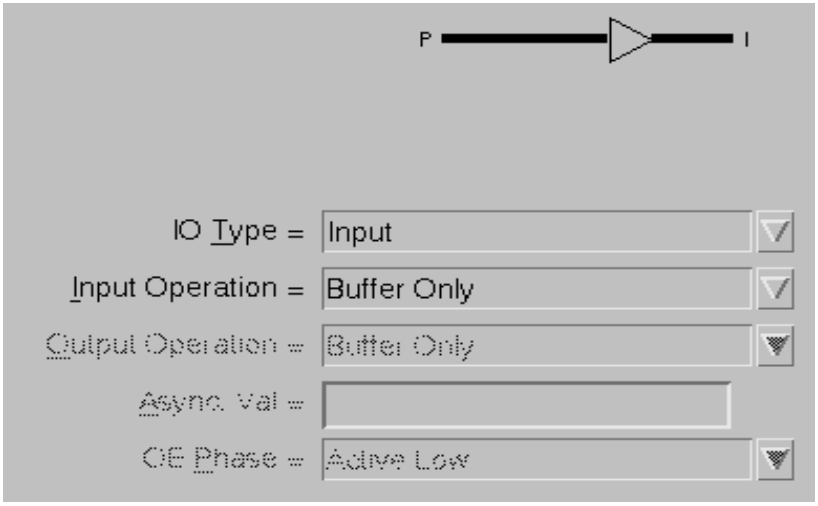

Figure 4-12 The Input Module

Output Modules

Output modules are device-output modules that connect an internal pin signal or bus to a Pad.

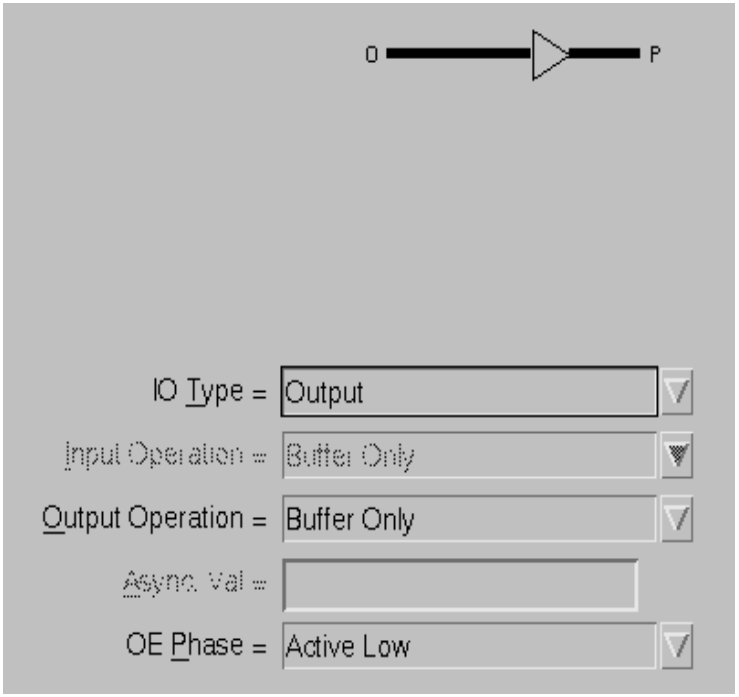

Figure 4-13 The Output Module

Bi-directional Modules

Bi-directional modules combine the function of an Input module with that of an Output module.

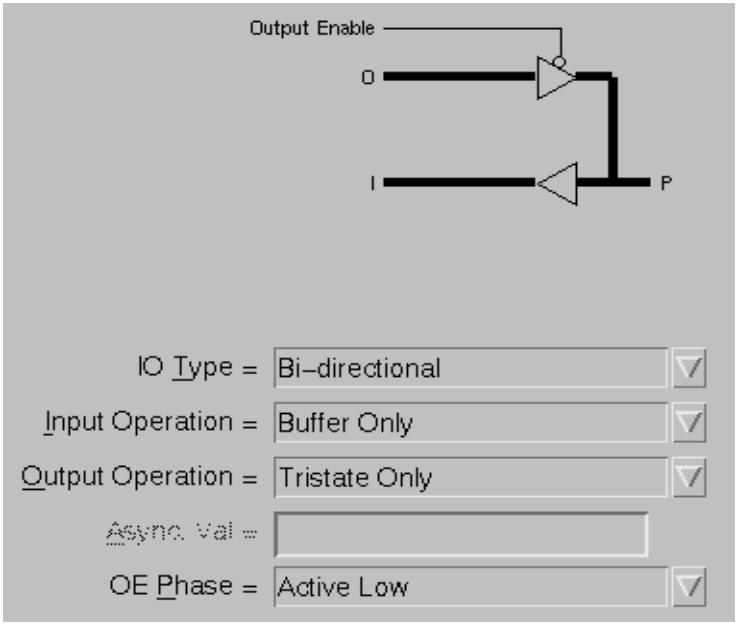

Figure 4-14 The Bi-directional I/O Module

I/O modules can be registered or buffered. Buffered I/O modules simply output whatever appears on the input port. Registered I/O modules are used to store the I/O values and are implemented as IOB registers.

Bi-directional modules expand into one or more input and output data and control signals, plus tristate input/output buffers.

Input Pins

This section describes the input pins of the Input and Output modules. The input pins of the Bi-directional module are just combinations of the same input pins for the Input and Output modules.

Input Module

P

The P pin represents the input to the Input module

Connections: This input is always specified. For Bi-directional modules, P is both the input and output of the I/O module. Connect a Pad module to the P pin to simulate a wire.

Registered Input Module

If you select an input operation that includes a register, the following inputs are also available.

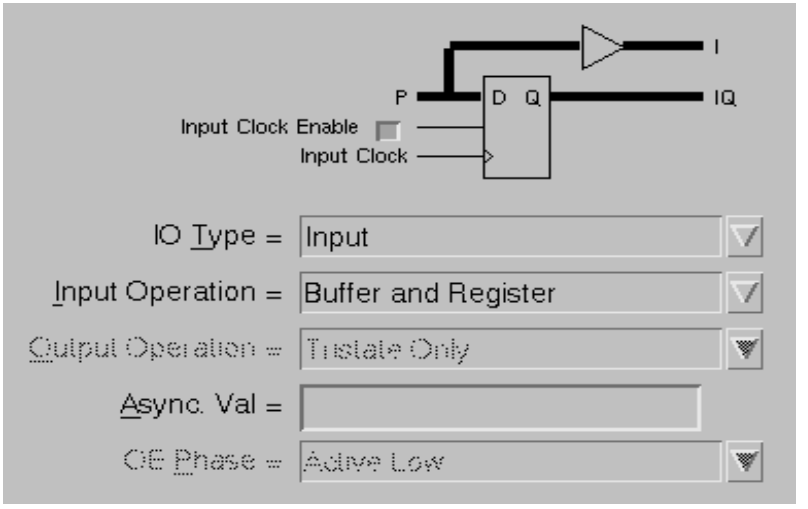

Figure 4-15 Registered Input Module

Input Clock (ICLOCK)

The Clock input loads the selected data into the register on the rising (positive) edge. You can implement an active falling (negative) edge by connecting an inverter to the Clock input.

Connections: The Input Clock pin is always specified on registered modules. If the Clock Enable input is specified and Low, the Clock is temporarily disabled and the register contents remain unchanged.

Input Clock Enable (ICLK_EN)

When it is specified and Low, the Input Clock Enable input temporarily disables the clock, causing the register to hold its previous value. When the Input Clock Enable input is High, the input data is loaded into the register on the next active Clock transition.

Connections: The Input Clock Enable pin is optional. If this input is not specified, the Clock is always enabled.

Output Module

O

The O pin is the input to the Output module.

Connections: The O pin is always connected.

Registered Output Module

If you select an output operation that includes a register or tristate, the following pins are available.

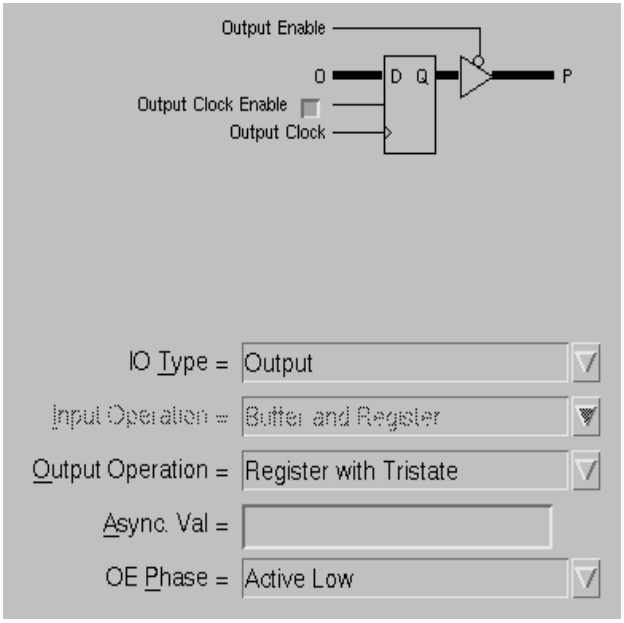

Figure 4-16 Registered Output Module With Tristate

Output Enable (OE)

The Output Enable input appears on a buffered tristate Output module. For all devices except the XC9500 and XC9500XL families, when Output Enable is High, the signal on the P pin is tristated.

When Output Enable is Low, the signal is enabled. For the XC9500 device family, the signal on the P pin is tristated when Output Enable is Low and enabled when Output Enable is High. For the XC9500XL family, the value of the OE Phase attribute determines the behavior of the signal on the P pin when Output Enable is High or Low.

Connections: The Output Enable pin is always specified on a tristate Output module.

Note: In the Unified Libraries this pin is labeled T (for Tristate) rather than OE on the output buffers. Also, the bubble is not present, indicating that the signal is active when T is High.

Output Clock (OCLOCK)

The Clock pin, when enabled, loads the selected data into the register on the rising edge. You can use an active falling edge by connecting an inverter to the Clock input.

Connections: The Output Clock pin is always specified on a registered Output module.

Output Clock Enable (OCLK_EN)

When the Clock Enable pin is High, the input data is loaded into the register on the next active Clock transition. When the Clock Enable is Low, the register contents are unaffected by the active Clock transition (hold).

Connections: The Clock Enable pin is optional. If this pin is not specified, the Clock is always enabled.

Output Pins

This section describes the output pins of the Input and Output modules. The output pins of the Bi-directional module are just combinations of the same output pins for the Input and Output modules.

Input Module

I

The I pin is the output of the buffered Input module.

Connections: The I pin is always connected.

Registered Input Module

IQ

On a registered module, the I pin is replaced by the IQ pin.

On a registered buffer Input module, the output of the register is the IQ pin and the buffer output is the I pin.

Connections: The IQ pin is always specified in a registered Input module.

Output Module

P

The P pin represents the output of the Output module and connects to a signal outside the chip.

Connections: This pin is always specified. You must connect a Pad symbol to the P pin of an Output module.

Attributes

IO Type (MODTYPE)

The IO Type attribute specifies the type of I/O function: Input, Output, or Bi-directional.

Usage: For each mode you select, the Input/Output module graphic is adjusted appropriately. Refer to the appropriate pin descriptions section.

Input Operation (IN_TYPE)

The Input Operation attribute specifies the operation mode of the Input module.

Usage: Valid values include Buffer Only, Register Only, Latch Only, Buffer and Register, and Buffer and Latch.

If you select an input option with a register, the Input Clock pin is automatically added. The Input Clock Enable pin is optional.

If you select an input option with a latch, the Input Gate pin is automatically added. The Input Gate Enable pin is optional.

Output Operation (OUT_TYPE)

The Output Operation attribute specifies the operation mode of the Output module.

Usage: Valid values include Buffer Only, Register Only, Tristate, and Register with Tristate.

If you select an output option with a register, the Output Clock pin is automatically added. The Output Clock Enable pin is optional.

If you select an output option with a tristate, the Output Enable pin is automatically added.

Async. Val (ASYNC_VAL)

The Asynchronous Value attribute controls the power-on state of the registers.

Usage: This attribute applies to the registered I/O modules only. For dual-register modules, you can specify the asynchronous value for either or both registers by using any of the following formats:

```
IN:15
OUT:31
OUT:31.IN:15
```

where OUT is the Output register and IN is the Input register.

OE Phase (OE_PHASE)

This attribute determines whether the signal on the P pin in a tristate Output module is tristated when the Output Enable is High or Low.

Usage: This attribute can only be set by the user for the XC9500XL device family. The attribute is set to Active High for the XC9500 family and to Active Low for all other families by default.

MEMORY

The Memory module allows you to create ROM, RAM, Synchronous RAM, and Dual Port RAM modules. These modules store information in the form of words in a tabular fashion. Use these modules for the following purposes.

- Store information from a user-edited memory file (.mem file) and access that information as needed (ROM)
- Store dynamic information into memory and then read that information from memory (RAM)

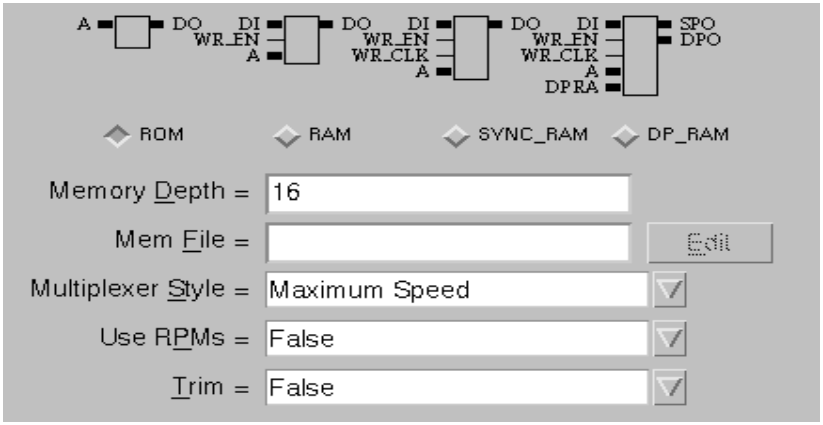

Figure 4-17 The Memory Module

ROM — Read-Only Memory Modules

The Read-Only Memory (ROM) module is a memory for storing information from a user-edited memory file (.mem file). The word storage capacity of the ROM is called the depth. The depth must be a multiple of 16, ranging from 16 to 256 words.

The size of the data stored within the ROM ranges from 1 through 64 bits. The largest word stored in the ROM should be within the boundaries set by the bus width of the Data Out pin, DO.

To read a word from the ROM, specify a binary number on the Address bus that matches the location of the desired word. The data is immediately output at the Data Out pin. Only addressable locations that are within the valid range will be synthesized.

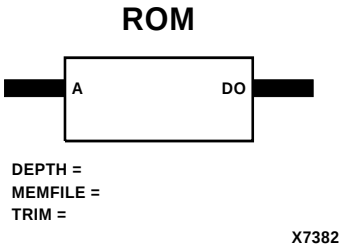

Figure 4-18 The ROM Module Symbol

Input Pins

A

A binary value on the Address port selects a word by pointing to the ROM location in which that word is stored. The Data Output port displays the selected word.

Connections: The Address input is always specified.

Output Pins

DO

The Data Output port reflects the word currently addressed in the ROM.

Connections: The Data Output pin is always specified.

Attributes

Memory Depth (DEPTH)

The Depth attribute defines the number of locations that can be addressed in the ROM module or the number of words that can be stored in the ROM. The depth must be a multiple of 16 and range from 16 to 256 words.

Mem File (MEMFILE)

The Memory File attribute references the name of a memory definition file that defines the contents of the ROM. The name of the memory definition file must have a .mem file extension. The exten-

sion can be omitted from the Memfile attribute. If the memory definition file exists, it will be read; otherwise, you can generate a Memfile template file from LogiBLOX by clicking on the Edit button. The template file created by LogiBLOX does not contain correct data and must be edited before you can use it. Refer to the "Memory Definition File Syntax" section for more information.

Usage: The Memory File attribute must be specified for ROM modules.

Multiplexer Style (STYLE)

The Multiplexer Style attribute determines the way multiplexers are built in memory modules. These multiplexers use address lines as select lines and choose among the outputs of the memory distributed in the CLBs. The Wired AND style uses Tristate buffers and long lines. The Normal Gates style uses CLB logic functions to implement the multiplexers.

Usage: Choose between the Maximum Speed, Normal Gates, and Wired AND values. If Memory Depth is set less than 64, the Maximum Speed value defaults to Normal Gates. If Memory Depth is set to 64 or more, Maximum Speed defaults to Wired AND.

Use RPMs (USE_RPM)

The Use RPMs attribute determines whether the function generators that comprise the ROM maintain a constant relative location to each other. This attribute applies to the XC4000 and XC5200 device families only.

Usage: Use RPMs can be set to True or False. The default is False.

Trim (TRIM)

Empty data columns are preserved by the software unless the Trim attribute is set to TRUE. If the software trims empty ROM primitives, it may yield a smaller design. The trade-off of having a smaller design is that you may not modify the contents of the ROM without rerouting the design. The default value is FALSE.

RAM — Random-Access Memory Modules

Random-Access Memory modules are used to store dynamic data. There are three different types of RAM modules: level-sensitive Asynchronous RAMs, Synchronous RAMs, and Dual Port RAMs.

RAM modules are not supported for the XC3000, XC5200, and XC9500 device families.

There are two ways of using RAM modules. One way consists of storing data dynamically by addressing locations from the Address bus pin and writing binary data furnished by the Data Input port into these locations. The other way consists of initializing the RAM module's contents at power-up by specifying the Memfile attribute. You can access the latter information the same way as you would in a ROM module. Furthermore, you can overwrite the information by writing new data in the RAM locations.

Storing Data Dynamically

To store a new word into the RAM and read it out on the output port, the address of the location to be written is placed on the address bus pin (A), the data to be written is placed on the data bus pin (DI), and Write Enable is driven from low to high.

For Asynchronous RAM modules, when Write Enable goes High, the data on the Data Input port is immediately stored into the currently addressed location and read out at the Data Out port. For Synchronous RAM and Dual Port RAM modules, the Write Enable Clock must also go High. Once a word is stored in the RAM, you can re-access it from the Address pin independently of the Write Enable signal.

If Write Enable stays Low while a value appears at the Data In port, this data is ignored. Only values that occur on a rising Write Enable pulse can be written. The data on the Address bus and on the Data Input port cannot change for the duration of the Write Enable pulse.

Initializing RAM Contents

In XC4000 devices, the contents of the RAM module are undefined at power-up. In XC4000E and XC4000EX devices, the power-on state of the RAM can be controlled by using a Memory Definition File (.mem file).

You can use the RAM module like a ROM to store a memory definition file that fits the depth of the module. The depth of the module refers to the word storage capacity of the RAM. The depth must be a multiple of 16, ranging from 16 to 256 words.

The width of the words contained in the memory definition file must also be within the width of the RAM module. The width of the RAM module ranges from 1 through 64 bits. The width of the module is set by the bus width of the Data Out pin.

You may overwrite the RAM locations using the Data Input pin to feed new data into the locations addressed by the Address bus.

Level-Sensitive Asynchronous RAM Modules (RAM)

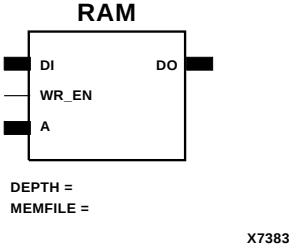

Figure 4-19 The RAM Module Symbol

Each time the Write Enable pin goes High, the data on the Data Input is stored into the addressed RAM location when Write Enable is active High. In addition, to prevent new data from overwriting the current location, there is a time frame during which the data on the Address and the Data In pins cannot change: when the Write Enable pin is Low before going High and triggering a write operation, and when the Write Enable goes Low again immediately after going High. These states are referred to as the Setup and Hold times respectively. The setup and hold time constraints on the Address and Data Input pins with respect to the Write Enable pulse cause the Level-Sensitive RAM module to have lower bandwidth than the synchronous RAM modules.

Synchronous RAM Modules (SYNC_RAM)

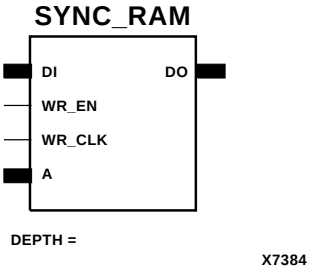

Figure 4-20 The Synchronous RAM Module Symbol

This module is identical to the Level-Sensitive RAM except that writes to the RAM are synchronized to the Write Enable Clock. When the Write Enable input is high and the Write Enable Clock goes from low to high, the data on the Data Input pin is written to the location specified by the Address input pin. The data on the Data Input pin appears on the Data Output pin after it is written to the RAM. New data appearing in the Data Input pin must wait for the next high on the Write Enable pin and a low to high transition on the Write Enable Clock before it is written.

This module is faster than a Level-Sensitive RAM due to the synchronous nature of the write, which allows pipelining of data on the Data Input pin.

Dual Port RAM Modules (DP_RAM)

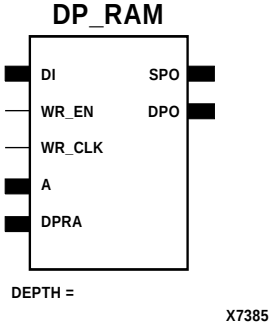

Figure 4-21 The Dual Port RAM Module Symbol

The Dual Port RAM module includes two independent pairs of Address and Data Output pins that share access to the same region of memory, allowing simultaneous read and write access to it. Writes are synchronized by the Write Enable Clock input.

Input Pins

All input pins of the respective RAM modules are always specified.

DI

The Data Input port determines the contents as well as the data that appears at the Data Output port that will be stored into the RAM when the Write Enable input goes High. Only the data currently at the Data Input port is read when Write Enable goes High.

Connections: The Data Input port is always specified.

WR_EN

For Asynchronous RAM modules, when the Write Enable input is High, the data on the Data Input port is written into the currently selected address location. When Write Enable is Low, no new data can be written into the RAM. For Sync_RAM and DP_RAM modules, a High on this input must be accompanied by a rising edge of the Write Enable Clock input to store the data into the memory location.

Connections: Write Enable is always specified.

WR_CLK

When the Write Enable Clock input goes High, the data on the Data Input port is stored into the currently addressed location and immediately read out at the Data Output port.

Connections: This input is available and required for the SYNC_RAM and DP_RAM modules only. For a valid write operation, the Write Enable input must first go High before this input goes High.

A

This input bus addresses a location in the RAM. Its purpose is twofold: 1) it selects the location whose contents are to appear on the Data Output port, or 2) it selects the location where new data is written whenever the Write Enable input goes High (for the level-sensitive RAM) or when the Write Enable Clock goes high while the

Write Enable input is High (for SYNC_RAM and DP_RAM). The width of the Address port is always $\log_2$ Depth.

Connections: The Address input is always specified.

DPRA

The Dual Port Read Address port is used for the secondary address line in a dual port RAM.

Connections: The Dual Port Read Address input is available for Dual Port RAM modules only. It is always specified.

Output Pins

DO

The Data Output port displays the contents of the location being accessed. The width of the Data Output port is set by the Bus Width attribute.

Connections: The Data Output pin is always specified. On DP_RAM modules this output port is called Single Port Output.

SPO

The Single Port Output is used to output the data that appears on the Data Input pin whenever the Write Enable Clock is High.

Connections: This output is only used by DP_RAM modules. It is always specified.

DPO

The Dual Port Output is used to output the data that resides at the address specified by the Dual Port Read Address (DPRA) input of the DP_RAM module. The value of DPO can change independently of the Write Enable Clock going High.

Connections: This output is used only by DP_RAM modules. It is always specified.

Attributes

Memory Depth (DEPTH)

The Depth attribute defines the number of words that can be stored in the RAM module. The depth must be a multiple of 16 and range from 16 to 256 words. The default value is 16. The width of the address port is always $\log_2$ Depth.

Mem File (MEMFILE)

The Memory File attribute references the name of a Memory Definition File that defines the contents of the RAM. The name of the Memory Definition File must have a .mem file extension. The extension can be omitted when specifying the Memfile attribute. If the Memory Definition File exists, it will be read; otherwise, you can generate a template file from LogiBLOX by clicking on the Edit button. The template file created by LogiBLOX does not contain valid data and must be edited before you can use it.

Usage: The Memory File attribute is optional for RAM modules.

Multiplexer Style (STYLE)

The Multiplexer Style attribute determines the way multiplexers are built in memory modules. These multiplexers use address lines as select lines and choose among the outputs of the memory distributed in the CLBs. The Wired AND style uses Tristate buffers and long lines. The Normal Gates style uses CLB logic functions to implement the multiplexers.

Usage: Choose between the Maximum Speed, Normal Gates, and Wired AND values. If Memory Depth is set less than 64, the Maximum Speed value defaults to Normal Gates. If Memory Depth is set to 64 or more, Maximum Speed defaults to Wired AND.

Use RPMs (USE_RPM)

The Use RPMs attribute determines whether the function generators that make up the RAM maintain a constant relative location to each other. This attribute applies to the XC4000 and XC5200 device families only.

Usage: Use RPMs can be set to True or False. The default is False.

Memory Definition File Syntax

A memory definition file, or MEMFILE, consists of two parts — the Header, which describes characteristics of the ROM or RAM module, and the Data portion, which defines the contents of the memory module. The beginning of the Data portion of the file is delimited by the DATA keyword. The memory definition file is not case-sensitive.

Memory Definition File Header

The Header defines characteristics of the memory module, such as its size and radix. Each of the following keywords must exist on a single line in the MEMFILE. Continuation characters are not allowed in the Header.

Depth (optional)

The Depth attribute defines the depth, in words, of the module. The Depth is specified in decimal notation, unless a radix definition precedes it. The value specified must match the value of the Memory Depth attribute.

`depth` *memory_depth*

Width (optional)

The Width definition sets the width of the memory, which is the number of bits in each word. The width must be a positive, non-zero, integer. The Width is specified in decimal, unless a radix definition precedes it. The value specified must match the value of the Bus Width attribute.

`width` *memory_width*

Default

The Default definition sets the value of all memory locations that are not specified in the MEMFILE Data section. If no default value is specified, all unspecified locations are set to zero. The default definition uses the current radix, which is 10, unless a radix definition precedes it.

Radix

This keyword defines the radix (or base) of the numbers following each radix definition in the MEMFILE. Multiple radix definitions can appear in the header and affect all non-radixed numbers up to and including the next Radix definition. A radix definition affects the MEMFILE Header section and the MEMFILE Data section.

The default radix for the MEMFILE Header section is 10. The default radix for the MEMFILE Data section is 16.

radix *integer*

Comments

Comments must be preceded by a semicolon. You can start your comment anywhere on the line. A semicolon at the end of a line generates a blank comment because it does not affect the next line of text.

; commentstring

Example

The following example illustrates the syntactical concepts defined above:

```
; The default radix is 10
Default 10; Defines the default ROM contents = 10(10)=10
Radix 16; Re-defines the default radix = 16(10)=16
Depth 10; Defines the depth = 10(16)=16
Radix 10; Re-defines the default radix = 10(16)=16
Width 12; Defines the width = 12(16)=18
```

Memory Definition File Data Section

The data values specified in the MEMFILE Data Section define the contents of the memory. Data values are specified sequentially, beginning with the lowest address in the memory, as defined. The address of a data value may be specified. The default radix of the data values is 16. If more than one radix definition is given in the MEMFILE Header Section, then the last such definition is the radix used in the Data Section.

data *data values*

Data values may be separated by commas, white space, or both.

Addressing

An address is specified as follows.

```
address:
```

For example, the following definition defines a 16-word memory with the contents 6, 4, 5, 5, 2, 7, 5, 3, 5, 5, 5, 5, 5, 5, 5, 5, starting at address 0. Note that the contents of locations 2, 3, 6, and 8 through 15 are defined via the default definition. Two starting addresses, 4 and 7, are given.

```
depth 16
default 5
data 6,4,
4: 2, 7,
7: 3
```

ASCII Data

You can specify ASCII data values by enclosing a string of characters in double quotes. You can include a double quote by prefacing the character with a backslash (\). A MEMFILE may contain both ASCII strings and numeric values. Each ASCII character gets assigned to one address location.

For instance, the following defines the contents of 16 memory locations. Two ASCII BEL characters (7) are defined here — one before the "R" and one after the two "l" characters.

```
data 7, "Ring the bell", 7, 0.
```

Differences Between the LogiBLOX Memfile and the Memgen/XBLOX Memfile

LogiBLOX imposes some restrictions on the memfiles that were previously supported by Memgen or X-BLOX 5.2.1:

- The LogiBLOX memfile does not allow a PART declaration.
- The radix of the data section is set by the RADIX command in the data section and cannot be changed. The radix cannot be overridden by using notation such as 2#1101#.
- The characters '#' and '_' are not allowed in the memfile.
- The depth value must be a multiple of 16. The valid range is 16 to 256 words.

MULTIPLEXER

Select the type of multiplexer you want: Type 1 or Type 2. Type 1 includes one input bus and one select line. Type 2 includes at least two input buses and one select line.

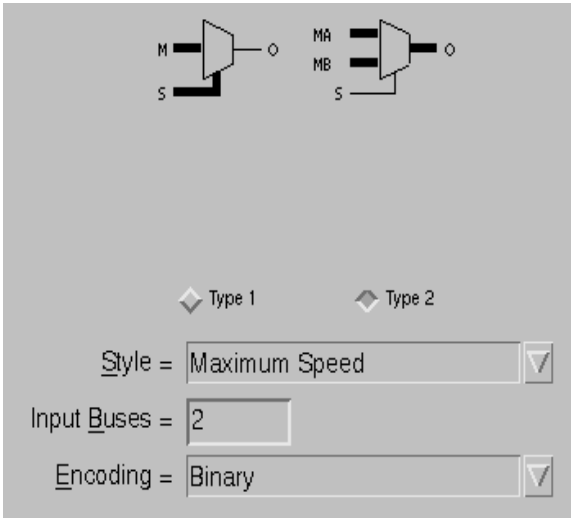

Figure 4-22 The Multiplexer Module

Type 1 (One input bus)

The Type 1 Multiplexer module routes one bit of an n-bit Input to the Output under the control of the Select input port, where the value of n is determined by the width of the input port. The Select input encoding can be binary or one-hot.

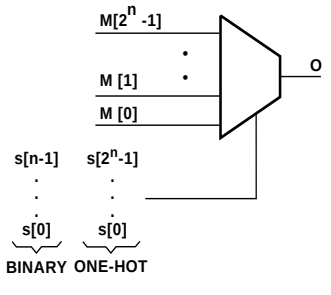

Figure 4-23 The Type 1 Multiplexer Logic Diagram

Type 2 (Two to eight input buses)

The Type 2 Multiplexer module routes one of two or more Input Buses to the Output Bus under the control of the Select input port. The Select input encoding can be binary or one-hot.

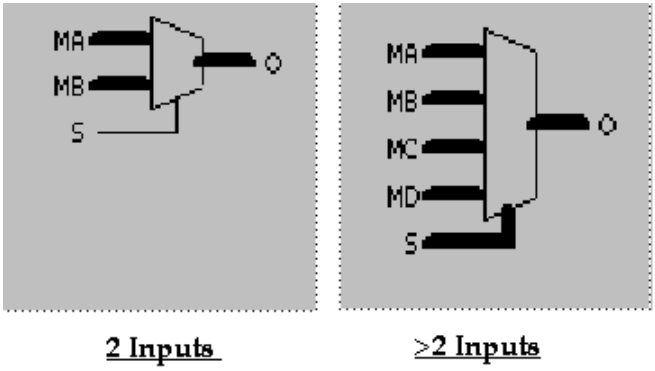

Figure 4-24 The Type 2 Multiplexer Module

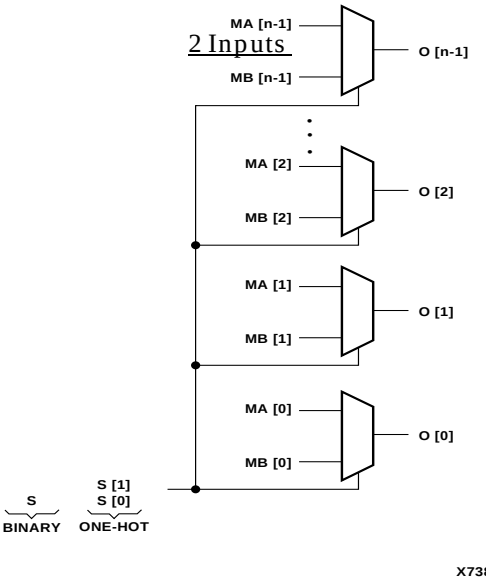

Figure 4-25 The Type 2, 2-input Multiplexer Logic Diagram

Input Pins

M

M is the Mux input bus data port of the Type 1 Multiplexer. The width of this bus is specified via the Bus Width attribute.

Connections: The Mux input is always specified on Type 1 Multiplexers.

MA through MH

MA, MB, ..., MH are the input bus data ports of the Type 2 Multiplexer. For a 3-input bus module with binary encoded Select, MA is selected when the Select input evaluates to zero, MB is selected when the Select input evaluates to 1, and MC is selected when the Select input evaluates to 2. If the value of the Select input is 3, the Multiplexer outputs are all High.

Connections: The MA and MB inputs are always connected on Type 2 Multiplexers. The other inputs are dependent on the value of the INPUT_BUSES attribute. For example, if INPUT_BUSES=4, then MA, MB, MC, and MD would be the input buses shown. The input and output buses must have the same width. The maximum bus width is 64.

S

The Select input port chooses which input bus line is directed to the Multiplexer output line. Depending on the encoding scheme, the Select input can address more lines than are available on the input bus.

Connections: The Select input is required. The Select input port is from 1 to 64 bits wide on a Type 1 Multiplexer. On a Type 2 Multiplexer, the Select input port is from 1 to 8 bits wide, depending on the Select encoding method.

Output Pins

O

The Multiplexer output line (for a Type 1 multiplexer) or Bus port (for a Type 2 multiplexer) reflects the selected input bus port data. If the selected input port is unavailable (out-of-range), the Multiplexer outputs are High.

Connections: The output pin must be connected.

Attributes

Operation (OPTYPE)

The Operation attribute specifies the type of multiplexer module you want: single-bus input or multiple-bus input.

Usage: Valid values are Type 1 (single-bus input) and Type 2 (multiple-bus input).

Style (STYLE)

Style defines the implementation style (area or speed preference).

Usage: For more information, see the "Implementation Styles" section of the "Understanding Attributes" chapter.

Input Buses (INPUT_BUSES)

Use the Input Buses attribute to specify the number of input buses. The number of buses can be any number in the range from 2 to 8.

Usage: This attribute only applies to Type 2 modules.

Encoding (ENCODING)

Use the Encoding attribute to specify the encoding scheme used to represent the Select bus.

Usage: Valid values are Binary and One Hot.

PAD

Connect a Pad module to an Input/Output module to simulate the connection to a pin pad.

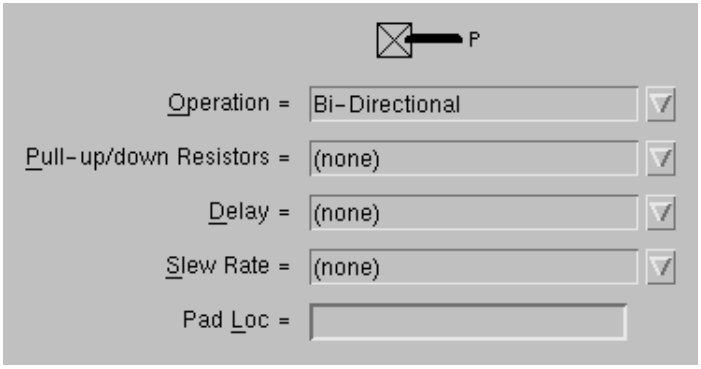

Figure 4-26 The Pad Module

Attributes

Operation (OPTYPE)

This attribute selects the type of I/O module to which you will connect the Pad module. Valid values are Input, Output, and Bi-Directional.

Pull-up/down Resistors (FLOAT_VAL)

This attribute is used to tie all the pads of a symbol to pullup or pull-down resistors. If no external signal is driving an input pad or tristate output pad, and it is not pulled up/down, it will float. You can use a value of Pull Up to drive a value that tends to 1 or a value of Pull Down to drive a value that tends to 0.

This attribute is grayed out for the XC9500 and XC9500XL families.

Usage: This attribute is optional and applies only to Input or tristatable Output modules. By default, no resistor is specified.

See the XC4000E and XC4000EX sections in *The Programmable Logic Data Book* for timing details.

Delay (DELAY)

By default, a delay, representing the internal IOB delay block, is used. The delay block ensures that the data path from Pad to register, also known as the setup time, is always longer than the external clock delay.

Using the IOB delay block prevents external data from overwriting the data currently at the Data Input port of the register before that data has been clocked into the register. Delaying the source data makes it unnecessary to specify a hold time to slow down the external data.

This attribute is grayed out for the XC9500 and XC9500XL families.

Usage: This attribute is only valid for modules that include an input register. The default is to use a Delay block. Select from the following values.

- (none): This is the default. Use a Delay block to slow down the time it takes for the signal to go from the device pin (Pad pin) to the register data input pin.
- Medium Delay: This is only applicable to XC4000EX devices. It is a partial delay block within the IOB. See *The Programmable Logic Data Book* for more details.
- No Delay: Bypass the Delay block. If you get rid of the Delay block, ensure that the external hold time is adequate. Refer to *The Programmable Logic Data Book* for more information.

Slew Rate (SLEWRATE)

The slew rate of each output buffer is, by default, reduced in order to minimize power bus transients when switching non-critical signals. This reduces ground bounce when all outputs are being transitioned at the same time. See *The Programmable Logic Data Book* for more details.

This attribute is grayed out for the XC9500 and XC9500XL families.

Usage: This attribute is only valid for modules that include an output register. Set the Slew Rate to Fast to ensure that the fastest rate available is used for the slew rate of a critical output. The default value is Slow.

Pad Loc (PAD_LOC)

The Pad Loc attribute specifies the pin location for an I/O pad.

Unlike other LogiBLOX attributes, the LogiBLOX Pad Loc setting can be overridden by location attributes specified on the LogiBLOX symbol in the schematic.

Usage: To assign a location to a specific bit, precede the location with a bit identifier. You can assign multiple bits by using a period as a separator. For example, with a bus width of 8 bits, you could have the following assignment:

```
PAD_LOC=0:P44.2:P45.7:P46
```

This specification assigns bit 0 to pad 44, bit 2 to pad 45, and bit 7 to pad 46.

Note: Commas will not work as separators between bit assignments.

SHIFT REGISTER

The Shift Register module is multi-functional, with predefined asynchronous or synchronous pre-load, and dynamic synchronous parallel load.

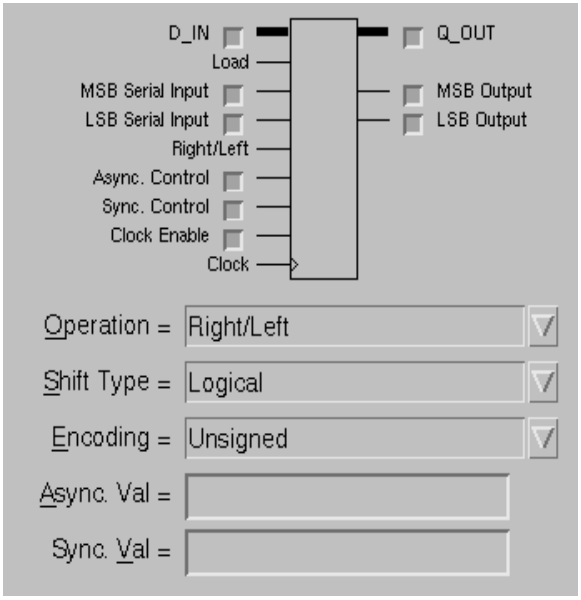

Figure 4-27 The Shift Register Module

The Shift Register can be synthesized in any one of the following configurations. For examples, refer to the figures at the end of this section.

- Serial-in / serial-out shift register (FIFO or LIFO)
- Serial-in / parallel-out shift register
- Parallel-in / parallel-out shift register
- Parallel-in / serial-out shift register

You can also select the shift style of the register. The shift styles include the following.

- Logical — Data is shifted either left or right. For a left shift, the value specified by LS_IN is shifted into the LSB position. For a

right shift, the value specified by MS_IN is shifted into the MSB position.

- Circular — Data is shifted in a circular pattern. For a left shift, the MSB is shifted into the LSB position. For a right shift, the LSB is shifted into the MSB position.
- Arithmetic — Data is shifted left to effectively multiply it by 2 or shifted right to effectively divide it by 2. For a left shift, a value of 0 is stuffed into the LSB. For a right shift, the Sign bit is extended if the data is signed. If the data is unsigned, a value of 0 is stuffed into the MSB.

Table 4-6 Universal Shift Register Truth Table (Logical Style)

RT/LFT	Load	Sync. Control	Clock Enable	Clock	Async. Control	Q_OUT	MS_OUT	
X	X	X	X	X	H	ASYNC_VAL	ASYNC_VAL MSB	ASYN
X	X	X	L	↑	L	Q_{prev}	Q_{prev} MSB	Q_{prev} L
X	X	H	H	↑	L	SYNC_VAL	SYNC_VAL MSB	SYNC_
X	H	L	H	↑	L	D_IN	D_IN MSB	D_IN I
H	L	L	H	↑	L	Q_{prev} / 2	MS_IN	Q_{prev}[I
L	L	L	H	↑	L	Q_{prev}x2 + LS_IN	Q_{prev} [MSB-1]	LS_IN

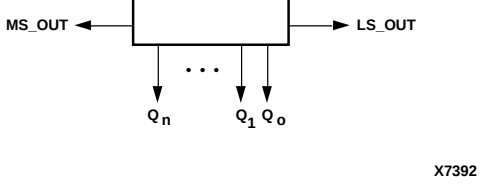

Figure 4-28 The Shift Register Data Flow Diagram

Input Pins

D_IN

The Parallel Data from the Data Input port is loaded into the register during a Parallel Load operation on an active Clock transition.

Connections: The Data Input port is optional. When it is specified, the Load pin is also specified.

Load (LOAD)

When the Parallel Load input is High, the data on the Data Input port is loaded into the Shift register on the next active Clock transition. When the Load input is Low, the counter responds to the Right/Left control input.

Connections: The Load input is automatically specified when the Data Input port is specified.

MSB Serial Input (MS_IN) and LSB Serial Input (LS_IN)

The MSB Serial Input port is only valid for a Logical style Shift Register when the shift direction is to the right. It allows you to specify a value to be stuffed into the MSB when a right shift has taken place. If no value is specified, a value of 0 is used.

The LSB Serial Input port is only valid for an Arithmetic or Logical style Shift Register when the shift direction is to the left. It allows you to specify a value to be stuffed into the LSB when a left shift has taken place. If no value is specified, a value of 0 is used.

Connections: The MSB Serial Input and LSB Serial Input ports are optional.

Right/Left (RIGHT_LEFT)

The Right/Left Shift control input, when High, enables the left-to-right shifting of data (from MSB to LSB); when Low, it enables the right-to-left shifting of data (from LSB to MSB). Inverting this input reverses the active High/Low definition, but does not change the MSB/LSB definitions or the shift direction.

Connections: The Right/Left pin is automatically specified when the Operation attribute is set to Right/Left.

Async. Control (ASYNC_CTRL)

The Asynchronous Control input is a level-sensitive input. When this input is High, it loads the value assigned to the Asynchronous Value attribute into the shift register independently of the Clock and Clock Enable.

Connections: If you specify the Asynchronous Control pin, you can assign a value to the Asynchronous Value attribute. By default, the Asynchronous Value is assigned a value of zero if it is not specified. The Asynchronous Value attribute may also be specified to define the accumulator register's power-on value.

Sync. Control (SYNC_CTRL)

Whenever the Synchronous Control and Clock Enable inputs are High, the value assigned to the Synchronous Value attribute is loaded into the shift register on the next active clock transition. This input has priority over the Load input if both pins are High at the same time.

Connections: If you specify the Synchronous Control pin and do not assign a value to the Synchronous Value attribute, a default value of 0 is used and a warning is issued.

Clock Enable (CLK_EN)

When the Clock Enable input is High, the enabled load and shift actions take place on the next active Clock transition. When Clock Enable is Low, the register contents are unaffected by the Clock.

Connections: Clock Enable is optional. Use this input when you need to disable the clock temporarily. If you do not use the Clock Enable input, the Clock is always enabled.

Clock (CLOCK)

If the Clock Enable input is High, the rising clock edge either loads the selected data into the register or performs a shift on the rising (positive) edge. The falling (negative) clock edge can be used by connecting an inverter to the Clock input.

Connections: The Clock input pin is always specified.

Output Pins

At least one of the output pins must be specified.

Q_OUT

The Parallel Data output port contains the current value of the register.

Connections: If you do not specify this signal, at least one of the MSB Output or LSB Output pins must be specified.

MSB Output (MS_OUT)

The MSB serial-data (left-shift) output port is used for shifting or for parallel-to-serial data conversions. MS_OUT is equal to the MSB of the shift register.

Connections: This pin is optional.

LSB Output (LS_OUT)

The LSB serial-data (right-shift) output port is used for shifting or parallel-to-serial data conversions. LS_OUT is equal to the LSB of the shift register.

Connections: This pin is optional.

Attributes

Operation (OPTYPE)

Use the Operation attribute to specify one of the three possible types of shift registers: Right, Left, or Right/Left. If you select Right/Left, a Right/Left pin is automatically added to the module.

Shift Type (SHIFT_TYPE)

The shift type can be chosen from the values listed in the following table.

Usage: Shift Type defines the operation mode of the module: arithmetic, circular, or logical.

Table 4-7 Shift Types

Type	Behavior of LS_IN and MS_IN pins
Arithmetic	The MSB is the sign bit. MS_IN may not be specified.
Circular	MS_IN and LS_IN are not allowed.
Logical	Shifts in a '0' at the MSB during a right-shift when MS_IN is not specified. Shifts in a '0' at the LSB during a left-shift when LS_IN is not specified.

Encoding (ENCODING)

You can use this parameter to define the encoding scheme of the data type of the Shift register. Valid values are Signed and Unsigned.

Async. Val (ASYNC_VAL)

The value of the Asynchronous Value attribute defines the power-on contents of the shift register. It also defines the value to which the register returns on assertion of the Asynchronous Control pin.

Usage: Asynchronous Value is always available. You can define it whether or not you use the Asynchronous Control pin. If you do not specify a value, the default value is zero.

Sync. Val (SYNC_VAL)

The Synchronous Value attribute defines the value to which the shift register returns on assertion of the Synchronous Control pin.

Usage: Synchronous Value is available only if you specify the Synchronous Control pin.

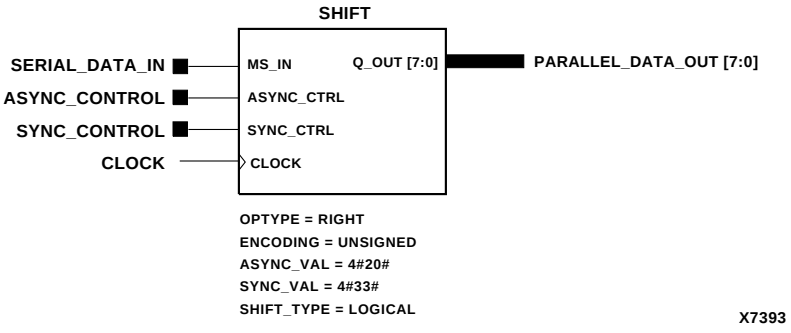

Figure 4-29 Typical Serial-in/Parallel-out Right Shift Register

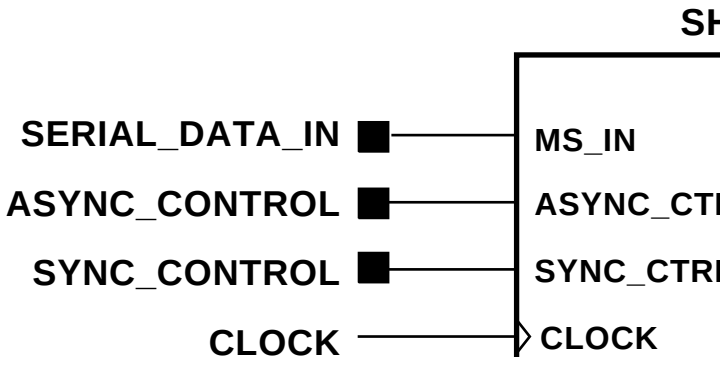

Figure 4-30 Typical Serial-in/Serial-out Right Shift Register

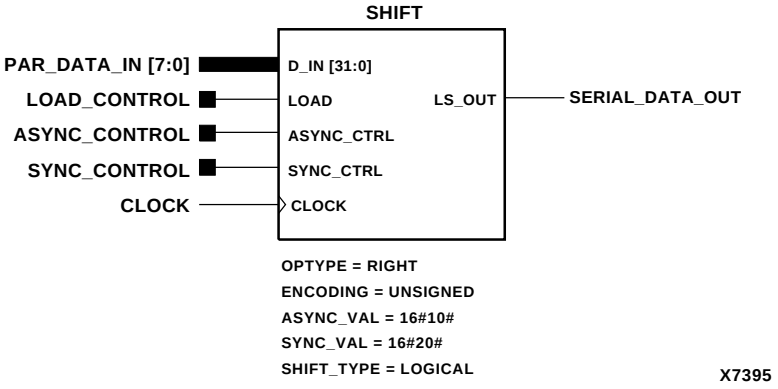

Figure 4-31 Typical Parallel-in/Serial-out Right Shift Register

SIMPLE GATES

This section describes the generic bused gate functions of LogiBLOX. Bused gate functions are defined as generalizations of the common logic primitives: AND, INVERT, NAND, NOR, OR, XNOR, and XOR. Except for the INVERT function, each of these can be implemented in three different ways, depending on the number and type of inputs.

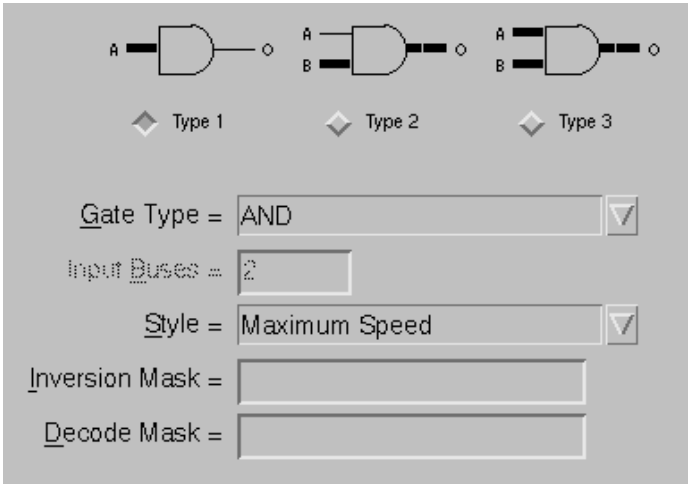

Figure 4-32 The Simple Gates Module

The following functions are described for the AND gate, but the same bus expansion criteria apply to the other functions.

Type 1 AND gate: $O = A_0 \bullet A_1 \bullet \ldots \bullet A_{(n-1)}$

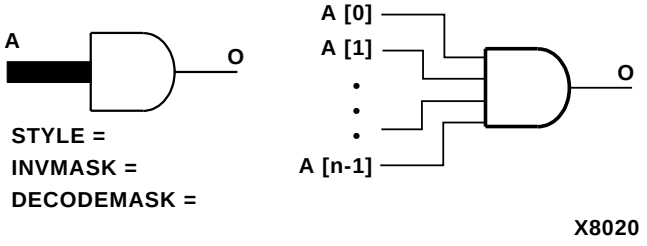

Figure 4-33 Type 1 AND Gate: Symbol and Logic Diagram

Type 2 AND gate: $O_n = A \bullet B_n$

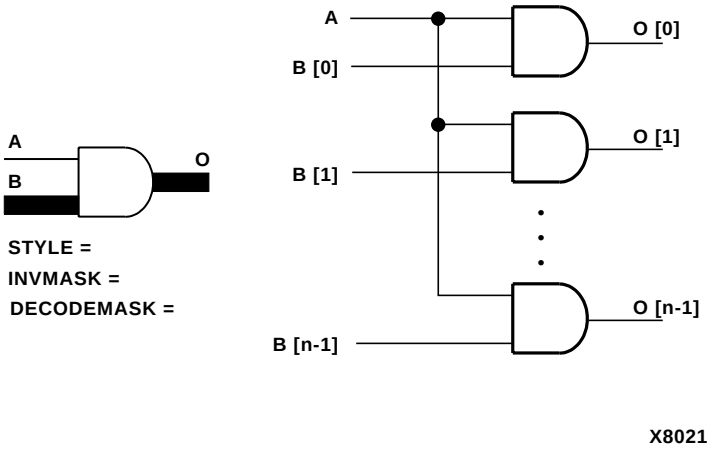

Figure 4-34 Type 2 AND Gate: Symbol and Logic Diagram

Type 3 AND gate: $O_n = A_n \bullet B_n$

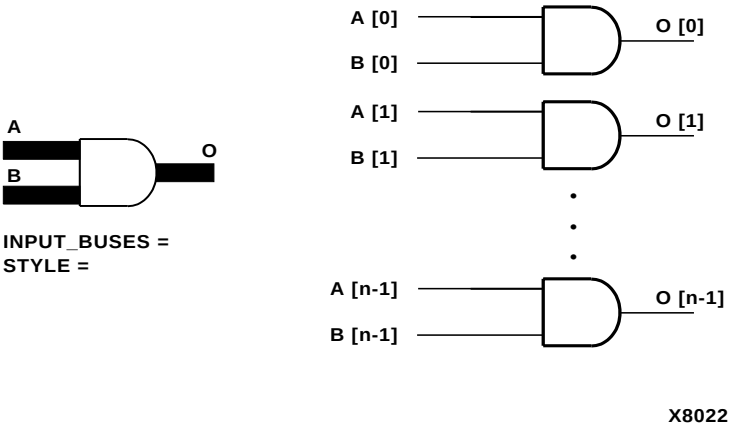

Figure 4-35 Type 3 AND Gate: Symbol and Logic Diagram

You can use the inversions of these logic functions by connecting an INVERT module to the outputs. You can also invert individual inputs (active Low) by using the Inversion Mask attribute associated with each module or specify which individual inputs will be active High with the Decode Mask attribute. See the "Inverting and Decoding

Masks for Gated Modules" section of the "Understanding Attributes" chapter for more information.

INVERT gate: $O_n = \overline{A}_n$

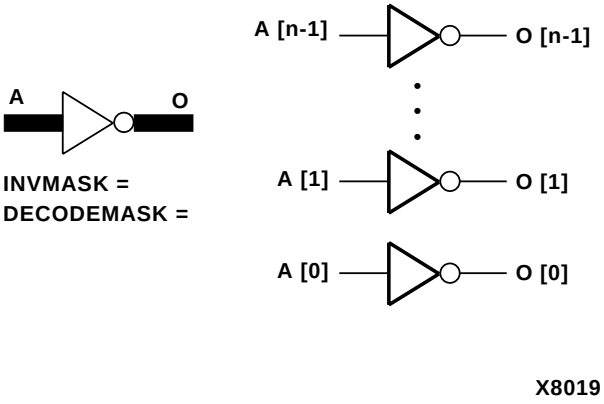

Figure 4-36 INVERT gate: Symbol and Logic Diagram

Attributes

Logic Type (OPTYPE)

To select the type of logic, click on the radio button located under the type of logic you want. The following descriptions use the AND module for an example, but the logic applies to all of the Simple Gates modules except the INVERT module.

- Type_1 (One input bus): The Type 1 AND module logically ANDs the individual signals of an input bus of width (n) to produce a single output signal, where (n) varies from 2-64 bits.
- Type_2 (One input net and one input bus): The Type 2 AND module logically ANDs the individual signals of an input bus of width (n) with a single input to produce an output bus of width (n), where (n) varies from 2-64 bits.
- Type_3 (2-8 input buses): The Type 3 AND module logically ANDs (m) input buses of width (n), where (m) varies from 2-8 buses and (n) varies from 2-64 bits. The output is a single bus of width (n).

Usage: This attribute does not apply to the INVERT module.

Gate Type (module)

Use this attribute to select the type of gate: AND, INVERT, NAND, NOR, OR, XNOR, or XOR.

Input Buses (INPUT_BUSES)

Specifies the number of Input buses (m) of the module.

Usage: This attribute applies to Type 3 modules only. Its value must be in the range 2-8.

Style (STYLE)

Style defines the implementation style (area or speed preference).

Usage: This attribute does not apply to the INVERT module. For more information, see the "Implementation Styles" section of the "Understanding Attributes" chapter.

Inversion Mask (INVMASK)

A High level on this attribute inverts the input of the gate. See the "Inverting and Decoding Masks for Gated Modules" section of the "Understanding Attributes" chapter for more information.

Usage: This attribute does not apply to Type 3 modules.

Decode Mask (DECODEMASK)

A Low level on this attribute inverts the input value. See the "Inverting and Decoding Masks for Gated Modules" section of the "Understanding Attributes" chapter for more information.

Usage: This attribute does not apply to Type 3 modules.

TRISTATE BUFFER

The Tristate Buffer module synthesizes internal non-inverting tristate buffers.

Tristate buffers are not supported for the XC9500XL family.

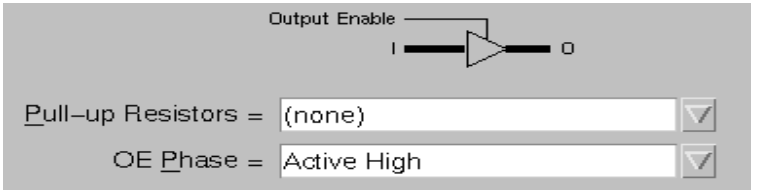

Figure 4-37 The Tristate Buffer Module

Inputs

I

The Input data port has the same width as the Output port.

Connections: The Input pin is always specified.

Output Enable (OE)

If the OE Phase attribute is Active Low, then when the Output Enable input is Low, the Input data passes to the Output. When the Output Enable is High, the Output is in high-impedance. If the OE Phase attribute is Active High, then when the Output Enable input is High, the Input data passes to the Output. When the Output Enable is Low, the Output is in high-impedance.

The Output Enable input must be driven. OE Phase can be Active High for the XC9500 and XC9500XL device families. To synthesize an active-High Output Enable for other device families, add an inverter to this line. This inverter is implemented within the internal tristate buffer block.

Connections: The Output Enable pin is always specified.

Note: In the Unified Libraries this pin is labeled T (for Tristate) rather than OE. Also, the bubble is present when OE Phase is Active Low and is absent otherwise.

Outputs

O

The Output port reflects the state of the Input port when the Output Enable is Low. When the Output Enable is High, the Output is in high-impedance. Although Tristate Buffer output ports can be tied together, only one port at a time can be active.

Connections: The Output pin is always specified.

Attributes

Pull-up Resistors (FLOAT_VAL)

You can use this attribute to add pull-up resistors to Tristate Buffer outputs that are connected to on-chip buses.

This attribute is not supported for the XC9500 family.

Usage: This is an optional parameter for this module. Use this attribute to specify the number of pull-ups: Single or Double. If this attribute is not specified, no pull-up is added. The double pull-up resistor draws more power than a single resistor but supports faster transition times. See *The Programmable Logic Data Book* for timing details.

OE Phase (OE_PHASE)

This attribute determines whether the Input data passes to the Output port when the Output Enable is High or Low.

Usage: This attribute can only be set by the user for the XC9500XL device family. The attribute is set to Active High for the XC9500 family and to Active Low for all other families by default.

Appendix A

LogiBLOX Versus X-BLOX/Memgen

LogiBLOX is a superset of both X-BLOX™ and Memgen. For an HDL flow, the methodology is the same; LogiBLOX generates the HDL template to instantiate a module.

For a schematic design flow, the LogiBLOX design rules are much simpler than the X-BLOX design rules:

- Bus size: In an X-BLOX design, data type and bus width of a module can be specified at a single point in a data path. During module expansion, X-BLOX propagates the information through the entire data path.

  Each LogiBLOX symbol is already sized with the bus_width attribute. The results of this difference are:

  a) BUS_DEF, CAST, SLICE, ELEMENT, and BUS_IFnn are no longer needed on a LogiBLOX schematic. The bus_width and encoding of the LogiBLOX module are implicitly defined.

  b) The bus connected to the LogiBLOX symbol should be labeled with size information as a standard schematic bus, that is, Datain[7:0] in Viewlogic. One of the main benefits of a sized module is that data propagation during module expansion is eliminated, which decreases the LogiBLOX implementation run time.

- The differences between the .mem file of LogiBLOX and the .mem file of Memgen or of the PROM, SRAM, and DPRAM symbols are as follows:

  a) The LogiBLOX memfile does not allow PART declaration.

  b) The radix of data section is determined by the RADIX command. It cannot be overridden by 2#1101#, for example.

  c) '#' and '_' are disallowed.

d) The depth value should be a multiple of 16; the maximum value is 256.

- X-BLOX can perform global design optimization such as implementing flip-flops in IOBs and inserting global buffers. This process, which was run-time consuming, is bypassed in LogiBLOX. (The MAP program has the ability to perform this optimization.) One of the consequences is the X-BLOX Inputs, Outputs and Bidir_IO symbols are replaced by LogiBLOX PAD and I/O modules. The replacement allows you to address a complex IO configuration.
- The symbol attributes are mostly identical for both LogiBLOX and X-BLOX. Some have changed such as:

  ```
  LOC[0] = P19, LOC[1]=P20
  ```

  becomes

  ```
  PAD_LOC = 0:P19.1:P20
  ```

  The constraints (L, LT, TL, T, TR, RT, R, RB, BR, B, BL, LB) are no longer valid for the I/O modules.