

# X-BLOX

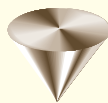
## REFERENCE/USER GUIDE



TABLE OF CONTENTS



INDEX



GO TO OTHER BOOKS

**X S A T C E T<sup>TM</sup> P**

# Contents

---

## Chapter 1 Introduction

X-BLOX Features.....	1-1
X-BLOX Design Examples Directory .....	1-2

## Chapter 2 Creating an X-BLOX Design

Adding an X-BLOX Module to Your Schematic .....	2-1
Customizing an X-BLOX Module .....	2-3
Implementation Styles and Operating Modes.....	2-4
Operating Modes .....	2-6
Data Values .....	2-6
Synchronous and Asynchronous Control .....	2-8
Power-up Reset and Initialization .....	2-9
Inverting and Decoding Masks for Bused Gate Functions.....	2-9
INVMASK and DECODEMASK Attributes.....	2-9
Single-Input Bused Modules.....	2-10
Double-Input Bused Modules .....	2-10
Single-Bus Gated Modules.....	2-11
INVBUS Module .....	2-11
Pull-up and Pull-down Resistors for I/O Pads.....	2-12
FLOAT_VAL Attribute .....	2-12
Out-of-Range Indicators .....	2-13
Representing X-BLOX Buses .....	2-14
Specifying Buses and Bus Labels .....	2-15
Bus Data Types .....	2-16
ENCODING .....	2-16
BOUNDS .....	2-17
Big-Endian vs. Little-Endian .....	2-18
Data Type Propagation.....	2-19
Creating a Hierarchical Symbol .....	2-21
Bus Sizes.....	2-21
Data Types .....	2-21
Data Type Propagation.....	2-21
Signal Aliasing .....	2-22
Bus Manipulation .....	2-25

CAST Symbol .....	2-25
ELEMENT Symbol.....	2-26
FORCE Symbol .....	2-26
SLICE Symbol .....	2-26
MUXBUS Symbol .....	2-27
Location Attributes .....	2-27
Using BUS_IFxx to Connect X-BLOX Buses to non-X-BLOX Logic.....	2-28
Creating a Custom BUS_IFxx Macro.....	2-29
Using XACT-Performance Attributes .....	2-30
TSidentifier Attribute .....	2-31
TNM Attribute.....	2-31

### Chapter 3 Processing Your Design

Step 1: Creating and Modifying Your Design .....	3-3
Step 2: Performing Functional Simulation.....	3-8
Step 3: Implementing a Partial Design.....	3-10
Step 4: Performing Timing Simulation.....	3-12
Step 5: Implementing the Complete Design.....	3-14
Step 6: Downloading Your Design .....	3-14
X-BLOX Design Example.....	3-15
Design Procedure .....	3-17

### Chapter 4 Module Definitions

Bused Gate Functions.....	4-2
ANDBUS Module .....	4-3
ACCUM — Accumulator .....	4-8
Inputs .....	4-9
Outputs .....	4-11
Attributes.....	4-11
ADD_SUB — Adder/Subtractor .....	4-14
Inputs .....	4-14
Outputs .....	4-15
Attributes.....	4-15
BIDIR_IO — Bidirectional I/O Pads with Buffers.....	4-18
Inputs .....	4-19
Outputs .....	4-19
Attributes.....	4-19
Constraints File .....	4-21
BUS_DEF — Bus Data-Type Definition .....	4-22
Bus Connection.....	4-22

Attributes.....	4-22
Example.....	4-23
CAST — Data Type Symbol .....	4-24
Usage .....	4-24
Inputs .....	4-27
Attributes.....	4-27
CLK_DIV — Clock or Frequency Divider .....	4-28
Inputs .....	4-29
Outputs .....	4-29
Attributes.....	4-30
COMPARE — Comparators .....	4-32
Inputs .....	4-32
Outputs .....	4-33
Attributes.....	4-34
RLOC_RANGE .....	4-34
COUNTER — Universal Counter .....	4-36
Inputs .....	4-37
Outputs .....	4-38
Attributes.....	4-39
Counter Style Features and Selection Criteria .....	4-42
LFSR .....	4-43
ONE_HOT .....	4-44
DATA_REG — Data Register .....	4-48
Inputs .....	4-49
Outputs .....	4-50
Attributes.....	4-50
STYLE Attribute .....	4-52
Conditions for Implementation in an IOB .....	4-53
DECODE — 1-of-n Decoder/Demultiplexer .....	4-58
Inputs .....	4-58
Outputs .....	4-59
ELEMENT — Element of a Bus .....	4-60
Connections.....	4-60
Attributes.....	4-60
FORCE — Force Value onto a Bus .....	4-62
Outputs .....	4-62
Attributes.....	4-62
INC_DEC — Increment Decrement Symbol .....	4-64
Inputs .....	4-64
Outputs .....	4-65
Attributes.....	4-65

INPUTS — Input Pads with Buffers .....	4-68
Inputs .....	4-68
Outputs .....	4-68
Attributes .....	4-69
Constraints File .....	4-70
MUXBUS — General n-to-1 Bus Multiplexer .....	4-72
Inputs .....	4-72
Outputs .....	4-73
MUXBUS2 — 2-to-1 Bus Multiplexer .....	4-74
Inputs .....	4-74
Outputs .....	4-75
MUXBUS4 — 4-to-1 Bus Multiplexer .....	4-76
Inputs .....	4-76
Outputs .....	4-77
MUXBUS8 — 8-to-1 Bus Multiplexer .....	4-78
Inputs .....	4-78
Outputs .....	4-79
OUTPUTS — Output Pads with Buffers .....	4-80
Inputs .....	4-80
Outputs .....	4-80
Attributes .....	4-81
Constraints File .....	4-82
PROM — Programmable Read-Only Memories .....	4-84
Inputs .....	4-84
Outputs .....	4-84
Attributes .....	4-85
MEMFILE Syntax .....	4-86
MEMFILE Header .....	4-87
Comments .....	4-88
Example .....	4-88
MEMFILE Data Section .....	4-89
Addressing .....	4-89
ASCII Data .....	4-89
PROM Definition Procedure .....	4-90
SHIFT — Universal Shift Register .....	4-92
Inputs .....	4-94
Outputs .....	4-96
Attributes .....	4-97
SLICE — SLICE of a Bus .....	4-102
Connections .....	4-102
Attributes .....	4-103

SRAM — Static Random-Access Memory .....	4-106
Inputs .....	4-106
Outputs .....	4-107
Attributes .....	4-107
CLB Utilization .....	4-109
TRISTATE — 3-State Buffer .....	4-110
Inputs .....	4-110
Outputs .....	4-110
Attributes (Optional) .....	4-111

## Chapter 5 X-BLOX-Generated Relationally Placed Macros

Implementation Styles for Arithmetic Modules .....	5-1
STYLE=ALIGNED .....	5-2
STYLE=UNALIGNED .....	5-2
FAST3KA and RIPPLE (XC3000A/L and XC3100A) .....	5-3
Controlling the Placement of RPMs .....	5-3
USE_RLOC={TRUE FALSE} .....	5-3
RLOC_ORIGIN=value .....	5-3
RLOC_RANGE=value .....	5-4

## Chapter 6 Understanding X-BLOX Operations

X-BLOX Implementation Flow .....	6-1
Data Type Propagation .....	6-4
Architectural Synthesis and Optimization .....	6-4
Merging Flip-Flops into the I/O Blocks .....	6-4
Global Buffers .....	6-5
Global Set-Reset .....	6-6
Relationally Placed Macros .....	6-6
Computing the Required Number of CLBs .....	6-7
Computing the Number of Rows .....	6-7
Example .....	6-8
Computing the Number of Columns .....	6-9
Example 1 .....	6-10
Example 2 .....	6-10
RLOC_ORIGIN Restrictions .....	6-10
RLOC_RANGE Restrictions .....	6-10
Example .....	6-11
Synthesizing Your Design for Simulation .....	6-11
Synthesizing Simulation Models .....	6-11
Functional Simulation Models for Schematic Entry .....	6-12
Timing Simulation Models .....	6-13

## Appendix A Command and Option Syntax

Usage.....	A-1
Command-line and Xactinit.dat Settings.....	A-1
Options.....	A-1
Xactinit.dat Settings .....	A-3

Index .....	Index-1
-------------	---------

## Trademark Information

## Introduction

---

The X-BLOX (Blocks of Logic Optimized for Xilinx™) synthesis tool consists of a library of modules that you can use to describe a system by means of high-level functions instead of gate-level primitives.

The complex functions provided by the X-BLOX Library complement the macro and gate-level cells that are provided with the Xilinx design entry interfaces. X-BLOX supports the XC4000, XC4000A, XC4000D, XC4000H, XC3000A, XC3000L, and XC3100A FPGA families.

Because X-BLOX modules are customizable, each module can describe thousands of unique functions. You can customize these modules using attributes and by connecting buses and nets to the appropriate pins on the modules.

## X-BLOX Features

X-BLOX offers the following features that simplify and speed up design entry and design modification.

**Block diagram design entry:** X-BLOX allows you to complete the major part of the system design at the block-diagram level using Medium Scale Integration (MSI) and Large Scale Integration (LSI) logic functions, such as adders, counters, comparators, clock dividers, decoders, universal shift-registers, SRAMs, and PROMs.

**Generic data path sizes and encoding:** X-BLOX simplifies design entry and design modification through its unique method of defining bus structures and data formats. In fact, the bus structure of each data path needs to be defined only once, even though the data path might contain many X-BLOX logic modules. The width (precision) of the bus and type of data carried on it are defined as part of one module and automatically propagated throughout the design and through



levels of hierarchy. This enables you to modify the bus size of an entire design by changing just a few fields on the schematic.

**Optimized implementations:** Logic functions are configured by the X-BLOX synthesis tool to fit the desired bus width and target chip family. The X-BLOX module generator tailors the logic implementation to the specific needs of each module and to the specific Xilinx device family. For example, the implementation style of a comparator depends on the size of the data fed to the comparator and on whether the comparison — greater than, less than, or a combination of these output functions, is needed.

The X-BLOX design system uses Xilinx-specific optimization techniques to boost the performance and density of the synthesized design implementation. The X-BLOX software uses expert knowledge of the chip resources coupled with smart logic implementation techniques to produce fast and efficient circuit designs. Thus, the X-BLOX module generators not only save valuable design time, but also produce optimum circuit implementations.

**Compatibility:** X-BLOX supports popular schematic editors, such as Viewlogic, Mentor, OrCAD, and Cadence. It can also be used with Synopsys' FPGA compiler and other third-party synthesis tools that support X-BLOX.

## X-BLOX Design Examples Directory

For your convenience, we have provided a directory for X-BLOX design examples and a directory for the files you should use to run the tutorial. The path for the design examples directory is `$XACT/examples/interface/design`. This directory contains a README file that documents the example designs in that directory. The path for the tutorial files is `$XACT/tutorial/interface/design`. This directory contains a README file that documents the tutorials in that directory.

## Creating an X-BLOX Design

---

Designing with X-BLOX entails adding a module to your schematic using your third-party design tool (Viewlogic, OrCAD, Mentor, or other X-BLOX compatible design tool); customizing the module by adding attributes to the module when the default modes are not desired; specifying X-BLOX buses and connecting them to X-BLOX modules; creating hierarchical symbols to simplify the top level of your design; and interfacing X-BLOX buses with non-X-BLOX buses.

This chapter is structured as follows:

- *Adding an X-BLOX Module to Your Schematic* provides a list of available modules and their functions along with an explanation of what you should know to add an X-BLOX module to your schematic.
- *Customizing an X-BLOX Module* details the different methods of customizing your modules using attributes.
- *Representing X-BLOX Buses* outlines the steps used to specify X-BLOX buses using attributes and the concepts necessary to understand bus manipulation and connection to X-BLOX modules and user-defined hierarchy.
- *Entering XACT-Performance Attributes* summarizes the usage of the TNM and TSidentifier attributes for specifying timing constraints to PPR.

## Adding an X-BLOX Module to Your Schematic

X-BLOX modules are represented by schematic symbols. The X-BLOX symbols are grouped into a library provided with your schematic capture interface. The X-BLOX library must be in your

schematic capture package's library search path for you to access it.

Each X-BLOX module can be considered as a template with a function. This section lists these modules by function.

<b>Counters, Registers, and Arithmetic Modules</b>	
ACCUM	Universal accumulator.
ADD_SUB	Adder and/or subtracter.
BUS_IFxx	Bus interface; used to connect X-BLOX modules to non-X-BLOX components.
COMPARE	Compares the magnitude and/or equality of two values.
COUNTER	Universal counter.
DATA_REG	Universal register.
DECODE	Translates data from any encoding to the one-hot encoding.
INC_DEC	Increments and/or decrements by a constant.
SHIFT	Register that loads and/or shifts data in parallel or serially, and shifts data out.
<b>Buses and Connectivity</b>	
BIDIR_IO	Defines one or more bidirectional I/O pads and buffers.
BUS_DEF	Specify the precision and encoding of a bus.
CAST	Enables multiple interpretations of bus data types (precision and encoding).
ELEMENT	Extracts one net (bit) from a bus.
FORCE	Sets a bus to a constant value.
INPUTS	Defines one or more input pads and input buffers.
MUXBUS	Selects one signal of the input bus.
MUXBUS2	Selects one of two input buses.
MUXBUS4	Selects one of four input buses.
MUXBUS8	Selects one of eight input buses.
OUTPUTS	Defines one or more output pads and output buffers.
SLICE	Extracts a subset of nets from a bus.

<b>Gate-Level Logic</b>	
ANDBUS	ANDs all the input nets of the bus yielding a single-bit result.
ANDBUS1	ANDs each of the bits of a bus with a single bit.
ANDBUS2	ANDs two buses together to yield a bus output.
INVBUS	Inverts the value of selected bits on an input bus.
ORBUS	ORs all the input bits of the bus, yielding a single-bit result.
ORBUS1	ORs each of the bits of a bus with a single bit, yielding a bus.
ORBUS2	ORs two buses together to set a bus output.
TRISTATE	Bus-wide 3-state.
XORBUS	XORs all the input nets of the bus.
XORBUS1	XORs each of the bits of a bus with a single bit.
XORBUS2	XORs two buses together to set a bus output.
<b>Memory</b>	
PROM	Generic programmable read-only memory.
SRAM	Generic static random-access memory.
<b>Clock Divider</b>	
CLK_DIV	Clock divider.

## Customizing an X-BLOX Module

Part of the power of X-BLOX is its ability to customize each module to represent many functions. This is done by connecting only the X-BLOX module control pins that are needed and by specifying module attributes when default modes are not the desired ones. You can customize modules by using attributes some of which are described in this section and by assigning data values to some of these attributes. The pages describing the individual X-BLOX modules, in the chapter “Module Definitions,” list the attributes that are appropriate for each module. This section is structured as follows:

- *Implementation Styles and Operating Modes* describes the different ways that a module can be implemented in the target technology

and the different modes in which a module can operate.

- *Data Values* explains the syntax for specifying the data values for attributes that allow a data value.
- *Synchronous/Asynchronous Control Attributes* explains how to use the ASYNC\_VAL and SYNC\_VAL attributes to initialize, set, or reset a module synchronously and asynchronously. These attributes are associated with modules that include a register.
- *Inverting and Decoding Masks for Bused Gate Functions* provides information on the inversion mask (INVMASK attribute) and decode mask (DECODEMASK), which you can use to specify the individually inverted or masked inputs (active High or active Low inputs) on bused logic modules.
- *Pull-up and Pull-down Resistors for I/O Pads* explains how to use the FLOAT\_VAL attribute to specify whether a pull-up or pull-down resistor is used on I/O and TRISTATE modules.
- *Out-of-Range Indicators* addresses the ADDR\_ERROR and SEL\_ERROR outputs which indicate how some X-BLOX modules can be used to detect out-of-range inputs.

## Implementation Styles and Operating Modes

There are several ways to implement some of the X-BLOX modules within the Xilinx architectures. We call these implementation methods “styles.” Some styles use fewer CLBs at the expense of speed while other styles use more CLBs to achieve faster performance. You can specify styles on specific modules. By default, X-BLOX uses the fastest style when you do not specify a style. A STYLE attribute can be assigned to the following X-BLOX modules:

**Table 2-1 Style Specification for Modules**

Module	Possible Style
ACCUM ADD_SUB INC_DEC	ALIGNED UNALIGNED RIPPLE FAST3KA (3000A only) or none
ANDBUS	WAND DECODE or none
COMPARE	ARITH RIPPLE WIRED TREE or none
COUNTER	BINARY JOHNSON LFSR ONE_HOT or none
DATA_REG	CLB IOB ILD IFD OFD or none
SHIFT	LOGICAL CIRCULAR ARITH or none

For descriptions of these styles, please refer to the description of these modules in the “Module Definitions” chapter.

**Note:** If you do not assign a style, X-BLOX chooses the appropriate style for you. We recommend that you do not specify a style and that you let X-BLOX select the appropriate implementation style, as it will choose the best possible one.

## Operating Modes

Some of the modules use the STYLE attribute to specify the operating mode for the module. For instance, the operating mode of the SHIFT module can be specified with the attribute STYLE=ARITH, STYLE=CIRCULAR, or STYLE=LOGICAL to get an arithmetic, circular, or logical shift operation.

## Data Values

Some module attributes are assigned as numeric values. Data values consist of the arithmetical base, or radix, followed by a representation of the numeric data value in the specified base. The base is a decimal number between 2 and 36 inclusive. Decimal is the default and does not need to be specified. The format is as follows:

**base#value# -or- decimal\_value**

**16#11# -or- 17**

As an example, the decimal value 17 can be expressed in several radices as shown below:

Binary	<b>2#10001#</b>
Octal	<b>8#21#</b>
Decimal	<b>17</b>
Hexadecimal	<b>16#11#</b>
Octadecimal	<b>18#H#</b>

The module attributes for which you can specify data values are listed in the following table along with the modules to which they pertain.

**Table 2-2 Attributes That Can Be Specified As a Data Value**

<b>Module</b>	<b>Attribute Requiring a Data Value</b>
ACCUM, SHIFT, DATA_REG	ASYNC_VAL, SYNC_VAL
COUNTER	ASYNC_VAL, SYNC_VAL, COUNT_TO
CLK_DIV	DIVIDE_BY, DUTY_CYCLE
FORCE	VALUE
ANDBUS, ANDBUS1, ANDBUS2, ORBUS, ORBUS1, ORBUS2, XORBUS, XORBUS1, XORBUS2, INVBUS	INVMASK DECODEMASK
SHIFT, COUNTER, OUTPUTS, INPUTS, CAST, FORCE, BIDIR_IO	BOUNDS
TRISTATE, BIDIR_IO, OUTPUTS, INPUTS	FLOAT_VAL
PROM	DEPTH, MEMFILE
SRAM	DEPTH

Numeric data values must contain only characters valid for the specific radix. A don't care digit (?) can be used with radices 2, 4, 8, 16 and 32 and represents the number of don't care bits normally associated with each digit (for example,  $4\#?2\# = 2\#??10\#$ ).

Negative data values are handled as two's-complement and are represented by a minus sign in front of the data value (for example,  $-2\#0011\# = 2\#1101\# = -3$ ), but don't care digits are not allowed in data values specified in the two's complement base.

To represent fractional data, a radix point (period) can be used. For example,  $2\#100.11\#$  represents 4.75.

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#00010010011100100110011101101001#**



is more legible when it is formatted as follows:

```
2#0001_0010_0111_0010_0110_0111_0110_1001#
```

The following table shows the valid characters for several radices:

**Table 2-3 Valid Characters Using Various Base Values**

Base Type	Base	Valid Data Value Characters
Binary	2	0 1 _ . ?
Octal	8	0-7 _ . ?
Decimal	10	0-9 _ .
Hexadecimal	16	0-9 A-F a-f _ . ?
36	36	0-9 A-Z a-z _ .

## Synchronous and Asynchronous Control

The synchronous/asynchronous control pins and attributes determine how to initialize, set, or reset modules containing flip-flops. When synchronous control is established, flip-flops are set or reset on the rising edge of the clock. When asynchronous control is established, the flip-flops are set or reset independently of the clock.

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

The modules that have synchronous (`SYNC_CTRL`) and asynchronous (`ASYNC_CTRL`) control pins include:

- ACCUM
- CLK\_DIV
- COUNTER
- DATA\_REG
- SHIFT

You can specify the SYNC\_VAL or ASYNC\_VAL attribute values on all the above modules, except the CLK\_DIV module.

The values are loaded into the registers and counters under the control of the ASYNC\_CTRL and SYNC\_CTRL inputs of the module.

An ASYNC\_VAL attribute value is loaded on the rising edge of the ASYNC\_CTRL port on the X-BLOX module and during power-up of the chip. This load has priority over any clock-activated load. The ASYNC\_VAL attribute value overrides all other inputs and is loaded independently of the clock enable and the synchronous control values.

The SYNC\_VAL constant is loaded into the register if the SYNC\_CTRL input on the X-BLOX module 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. If you do not specify an ASYNC\_VAL, all registers and counters except Linear-Feedback-Shift-Register counters (LFSR) are set to zero at power-up. LFSR counters are set to their initial count state at power-up.

## Inverting and Decoding Masks for Bused Gate Functions

The INVMASK and DECODEMASK attributes specify individually inverted or masked inputs (active High or active Low) for certain X-BLOX modules with bused inputs.

### INVMASK and DECODEMASK Attributes

The INVMASK and DECODEMASK attributes are available on X-BLOX modules that perform bused gate functions. These modules include:

- ANDBUS, ANDBUS1, ANDBUS2
- ORBUS, ORBUS1, ORBUS2

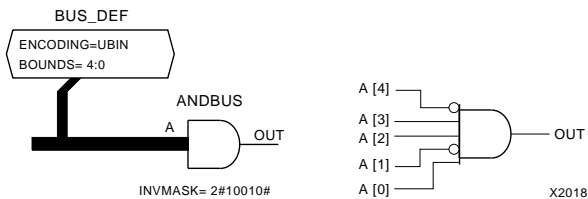
- XORBUS, XORBUS1, XORBUS2
- INVBUS

The inputs for which the  $INVMASK = 1$  or  $DECODEMASK=0$  will be active Low, and those for which the  $INVMASK=0$  or  $DECODEMASK=1$  will be active High. You can specify the inversion mask or the decode mask using any of the radices specified in the section “Data Values” in this chapter.

**Note:** The default value is chosen to be intuitive and is therefore different for some of the modules. On the bused AND, XOR, and OR modules, the default is  $INVMASK=0$  or  $DECODEMASK=1$ , so no inversion is performed on the inputs. On the INVBUS module, the  $INVMASK$  indicates which bits in the bus will be inverted. The default is to invert all the bits in the bus, which is what one would expect from a bus-wide inverter.

## Single-Input Based Modules

As an example, on a 5-bit ANDBUS with  $INVMASK = 2\#10010\#$ , Bits 1 and 4 are inverted (bit 4 is the most significant bit). The same is true of  $DECODEMASK$ . If  $DECODEMASK=2\#01101\#$ , Bits 1 and 4 are inverted.

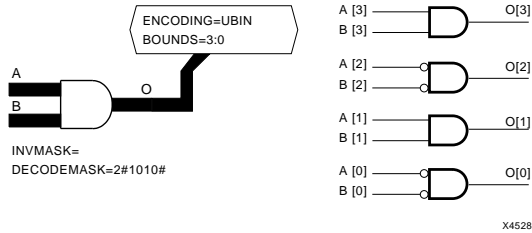


**Figure 2-1 5-Input ANDBUS Using INVMASK**

## Double-Input Based Modules

When you add the  $INVMASK$  or  $DECODEMASK$  attribute to a symbol, this property is applied to all bus inputs on that symbol. For example, for a 2-input module, the same  $INVMASK$  is applied to both inputs (A and B). This rule applies to `ANDBUS2`, `ORBUS2`, and `XORBUS2`. See Figure 2-2.

**Note:** Mathematically, applying one of these masks to both inputs of an XORBUS2 is the same as not applying it to either. If you want the INVMASK or DECODEMASK to apply to only one of the two bused inputs, use an INVBUS symbol in front of the appropriate input.

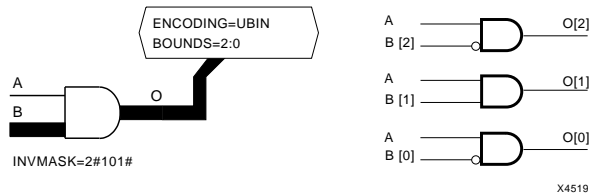


**Figure 2-2 ANDBUS2 Using DECODEMASK**

**Note:** If you need any unique masks for either input, combine the INVMASK of the bus symbol with an INVBUS and its own mask.

### Single-Bus Gated Modules

On an ANDBUS1, only the input bus is affected by the INVMASK. See the following figure.

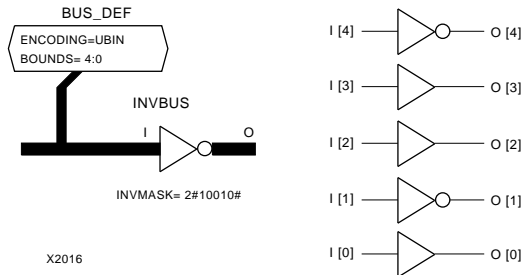


**Figure 2-3 ANDBUS1 Using INVMASK**

### INVBUS Module

The X-BLOX INVBUS module has the opposite default for the INVMASK attribute. When you do not define INVMASK, all signals connected to the INVBUS module are inverted. If you specify INVMASK, the INVBUS outputs are determined by the bit pattern of INVMASK. For each bit in the INVMASK that is 0, the corresponding bit in the bus is not inverted. (See Figure 2-4.) For each bit in the INVMASK that is 1, the corresponding bit in the bus is inverted.

You can specify the value of the INVMASK in any base. (Refer to the “Data Values” section.)



**Figure 2-4 5-Input INVBUS Using INVMASK**

## Pull-up and Pull-down Resistors for I/O Pads

The FLOAT\_VAL attribute determines whether a pull-up or pull-down resistor is used on I/O and TRISTATE modules.

### FLOAT\_VAL Attribute

You can connect pull-up or pull-down resistors to pads of the X-BLOX INPUTS, BIDIR\_IO modules, and to the TBUFs generated by the TRISTATE module. You can specify this value in any of the allowed radices mentioned in the section “Data Values” in this chapter. The value can contain don’t care digits. For example, a module with ENCODING = UBIN, BOUNDS = 7:0, and FLOAT\_VAL = 16#?3# specifies that pads 7 through 4 are not connected to resistors, that pads 3 and 2 have pull-down resistors, and that pads 1 and 0 have pull-up resistors.

Alternately, you can tie all the pads of a symbol to pull-up or pull-down resistors by specifying one pull-up resistor (FLOAT\_VAL=PULLUP), two pull-up resistors (FLOAT\_VAL=PULLUP\_D), or one pull-down resistor (FLOAT\_VAL=PULLDOWN) respectively. The double pull-up resistor draws more power than a single resistor but supports faster transition times. (See the XC4000 Data Sheet for timing details in *The Programmable Logic Data Book*.)

## Out-of-Range Indicators

The modules that select or address logic synthesized by X-BLOX are the DECODE module, the MUXBUS modules, and the PROM and SRAM modules. These modules have out-of-range indicators built into their symbols.

On the DECODE and all MUXBUS modules, this output is called SEL\_ERROR. On the PROM and SRAM symbols, it is called ADDR\_ERROR. The following tables describe how these indicators behave. X-BLOX also considers cases wherein the SEL inputs might index unconnected MUXBUS inputs or DECODE outputs.

**Table 2-4 DECODE Out-of-Range Indicators**

DECODE: Bounds on D_OUT[n:m]		
SEL input value	SEL_ERROR	D_OUT
Value is not between n and m	high	zero
Value indexes an unconnected D_OUT element	high	zero
Value represents all other cases	low	high on SEL <sup>th a</sup> bit <sup>b</sup>

a. SEL<sup>th</sup> represents an ordinal number (1st, 2nd, 3rd, etc.)

b. If EN (enable) is High

**Table 2-5 MUXBUS Out-of-Range Indicators**

MUXBUS: Bounds on MUX_IN [n:m]		
SEL input value	SEL_ERROR	MUX_OUT
Value is not between n and m	high	zero
Value indexes an unconnected MUX_IN element	high	zero
Value represents all other cases	low	SEL <sup>th a</sup> of MUX_IN

a. SEL<sup>th</sup> represents an ordinal number (1st, 2nd, 3rd, etc.)

**Table 2-6 MUXBUSn Out-of-Range Indicators**

<b>MUXBUSn: n=2, 4, or 8</b>		
<b>SEL input value</b>	<b>SEL_ERROR</b>	<b>MUX_OUT</b>
Value is greater than n-1, or <0	high	zero
Value indexes an unconnected D_OUT element	high	zero
Value represents all other cases	low	SEL <sup>th a</sup> of Mx inputs

a. SEL<sup>th</sup> represents an ordinal number (1st, 2nd, 3rd, etc.)

**Table 2-7 PROM, SRAM Out-of-Range Indicators**

<b>PROM, SRAM: DEPTH=n</b>		
<b>ADDR input value</b>	<b>ADDR_ERROR</b>	<b>D_OUT</b>
Value is greater than n-1	high	zero
Value represents all other cases	low	contents of ADDR

## Representing X-BLOX Buses

Another powerful aspect of X-BLOX is how it handles buses. X-BLOX buses include the following special features discussed in the following sections:

- *Specifying Buses and Bus Labels* explains how to define and use generic X-BLOX buses in your design.
- *Bus Data Types* discusses the width and encoding of buses.
- *Data Type Propagation* explains how after specifying bus data types in one location, the data types are propagated along your data path by X-BLOX, thus easing the specification and modification of bused designs.
- *Creating a Hierarchical Symbol* outlines the steps to be followed when creating hierarchical designs with X-BLOX.

- *Signal Aliasing* explains how X-BLOX assigns aliases to signals and buses.
- *Bus Manipulation* is an overview on how to use the SLICE, ELEMENT, FORCE, MUXBUS, and CAST modules to manipulate buses.
- *Location Attributes* documents how to specify locations for a module.
- *Using BUS\_IFxx to Connect X-BLOX Buses to non-X-BLOX Logic* tells you how to connect X-BLOX buses to non-X-BLOX portions of your design.

## Specifying Buses and Bus Labels

All X-BLOX modules and buses are generic in size. X-BLOX represents all bus sizes and encodings with either a bus with no range or a single-bit net, depending on the schematic capture package. For instance, Viewlogic and Mentor use rangeless buses, and OrCAD uses nets.

To label an X-BLOX bus connected to X-BLOX modules, use a label to name the bus without specifying the bus range on the label. Contrast the following specifications for X-BLOX and non-X-BLOX buses.

- Specify an X-BLOX bus as follows:

**DATA**

- Specify a non-X-BLOX bus as follows:

**DATA[15:0]**

After labeling your X-BLOX bus, specify the bounds and encoding for the bus anywhere on the same data path. Refer to the section “Bus Data Types,” which follows, for information on how to specify bus bounds and encoding.

When you run X-BLOX, it synthesizes the buses expanding them to the proper sizes as it generates the simulation models and performs chip-level implementation. Generic-sized buses allow the schematic to be resized quickly because the BOUNDS attribute of a bus needs to be specified only once on each data path. The proper implementation of the modules is achieved using the encoding for the bus. If you



need to change the encoding of a secondary bus, use the CAST module to connect the two buses.

## Bus Data Types

In X-BLOX, a bus is not just a collection of wires: a bus defines the kind of data that travels through the bus. As in structured computer languages, this data has a data type. A bus data type is defined by an encoding scheme and a bus precision. You must assign both attributes, BOUNDS and ENCODING, to establish a data type. The ENCODING attribute defines the data encoding schemes supported by X-BLOX, and the BOUNDS attribute defines the left and right indices of the bits on the bus. For example, an 8-bit unsigned binary bus has eight bits that can be indexed from seven down to zero. In this case, specify the BOUNDS attribute as BOUNDS=7:0.

You must specify the data type of at least one Input, Output, or BUS\_DEF component along each X-BLOX data path to propagate the data type to the other modules on that data path.

### ENCODING

The encoding scheme for a bus is specified with an attribute called ENCODING. The encoding enables X-BLOX to treat unsigned numbers differently from two's-complement or one-hot encoded numbers according to the encoding used. The X-BLOX software uses this information to be sure that functional modules such as adders, comparators, multiplexers, and decoders are synthesized correctly for the data type used.

X-BLOX currently supports three encoding schemes with ENCODING attributes as shown in the following table.

**Table 2-8 X-BLOX Encoding Schemes**

Encoding	Description
BIT or UBIN	Unsigned binary
ONE_HOT	1-of-n encoding — only one wire at a time is set to logical 1
TWO_COMP	Two's-complement

**Note:** X-BLOX does not check the validity of ONE\_HOT data coming from outside the X-BLOX bus environment. X-BLOX modules generate valid ONE\_HOT values when requested unless the module is disabled. For example, the DECODE module is disabled when the Enable is Low, in which case, the output is zero, which is not a valid one-hot value.

**Note:** With encodings of UBIN, ONE\_HOT, or TWO\_COMP, the BOUNDS attribute must be specified; there is no default value for BOUNDS. However, for an encoding of BIT, the attribute BOUNDS is optional. If BOUNDS is specified with an encoding of BIT, it is equivalent to the UBIN encoding. If the attribute BOUNDS is not specified with an encoding of BIT, the result is a scalar net. If BOUNDS are specified but ENCODING is not, it is equivalent to specifying ENCODING=BIT.

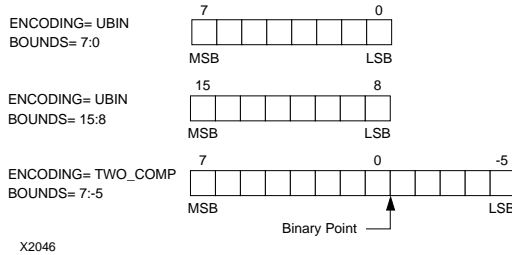
## BOUNDS

To specify the precision (bus bounds) of the data types, use the BOUNDS attribute. The bounds are a pair of integers separated by a colon. The first number is the left bound and represents the most-significant bit (MSB) of a number. The second number is the right bound and represents the least-significant bit (LSB) of a number. An 8-bit port usually has BOUNDS=7:0, indicating that the left-most bit has index 7 and that the right-most bit has index 0. Refer to Figure 2-5.

**Note:** You can use any combination of integers, as long as the difference of the indices, plus one, equals the width of the bus; for example,  $(7-0)+1=8$ . If both bounds are the same, the bus is one bit wide with the index of that bounds number. Note that for big-endian buses, the values of these indices are significant — the indices represent the place-values of the corresponding bits.

A big-endian bus can represent fractional (scaled, not floating point) numbers that require negative bounds or positions to describe the bus. For example, a 13-bit bus with five fractional bits has the bounds 7 down to -5. As shown in Figure 2-5, the binary point is between the bit with index 0 and the bit with index -1 (that is, BOUNDS=7 : -5).

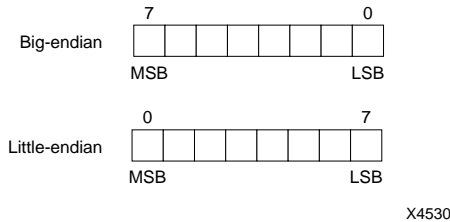
**Note:** Please verify the status of support of negative indices by X-BLOX in the Release Notes.



**Figure 2-5 Example of BOUNDS and ENCODING Representation**

### Big-Endian vs. Little-Endian

When the MSB index of a bus is greater than the LSB index, for example, BOUNDS=9:0, this is called “big-endian”, because the big end (largest bit index) is on the left. When the MSB is less than the LSB, for example, BOUNDS=0:9, this is called “little-endian” because the little end is on the left. See Figure 2-6. Note that the MSB is always on the left and the LSB is always on the right.



**Figure 2-6 Big-Endian vs. Little-Endian**

**Note:** The bounds have an important role in the interpretation of the values on a bus. On a big-endian bus, the indices of the bits specify the weight of the bit. For example, bit 4 has weight  $2^4=16$ , and the 0

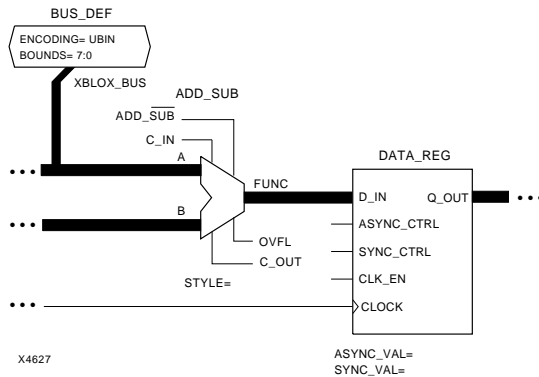
bit (weight  $2^0=1$ ) is just to the left of the binary point. Also, bit  $-3$  has weight  $2^{-3}=1/8=0.125$ . In a little-endian bus, there is no standard place for the binary point, so X-BLOX places the binary point to the right of the right-most bit in a little-endian bus. The value of the number on the bus used as the selector determines the action of the SEL ports on the X-BLOX multiplexers and decoders. The value of the selector depends on the encoding and bounds of the selector.

One-bit arrays are treated as big-endian, that is, BIT 0:0 has the weighting  $2^0=1$ , BIT 4:4 has the weighting  $2^4=16$ , BIT  $-3:-3$  has the weighting  $2^{-3}=1/8=0.125$ . If you wish a single bit with weighting 1, either leave the BOUNDS field empty or set it to 0:0.

Big-endian and little-endian ONE\_HOT data always has the weighting of the corresponding bit index. For example, the ONE\_HOT (3:-1) bit value 10000=3 and the ONE\_HOT (-1:3) bit value 00001=3.

## Data Type Propagation

X-BLOX uses data type propagation to minimize the number of times and locations that the encoding schemes and precision must be specified in a design. X-BLOX propagates the ENCODING and BOUNDS attributes assigned to one module on the bus to all the other connected modules and buses in the data path. This guarantees consistency of bus sizes and encoding schemes, making global modifications much easier. For example, consider Figure 2-7 in which an 8-bit unsigned-binary bus connects to an adder, the output of the adder goes to another 8-bit bus, and that bus goes to an 8-bit register. If you had to specify the data type in eight places — three ports on the adder, two ports on the register, and the sizes of the three buses — changing one of these data types without changing the others might lead to errors. The same design completed with X-BLOX modules requires you to specify the ENCODING=UBIN and the BOUNDS=7:0 attributes just once on the data path.



**Figure 2-7 Simple X-BLOX Data Path Definition**

**Note:** Specify the `ENCODING` and `BOUNDS` attributes in the X-BLOX design system at the periphery of the chip or function; both attributes propagate to the other modules and buses. The X-BLOX modules that use the `ENCODING` and `BOUNDS` attributes are as follows.

- INPUTS
- OUTPUTS
- BIDIR\_IO
- BUS\_DEF
- FORCE
- CAST
- DATA\_REG
- COUNTER
- PROM
- SHIFT
- SRAM

The **INPUTS**, **OUTPUTS**, and **BIDIR\_IO** modules connect the circuit to the pads of the device. The most appropriate place to assign the `ENCODING` and `BOUNDS` attributes is at the point where the buses

interface to other chips. The `BUS_DEF` module is used to define the data type of internal buses and module ports that are not connected to the I/O network.

## Creating a Hierarchical Symbol

This section explains how to create a hierarchical symbol that includes one or more pins connected to X-BLOX buses.

### Bus Sizes

Just as you specify X-BLOX bus labels without specifying a range, specify X-BLOX bus pins without a range when you create a hierarchical symbol. Thus, all bus pins will be one-bit wide.

**Note:** With the Viewlogic editor, to help you visually recognize that these are generic buses, you can draw these interface pins as wide as a bus interface pin by following the instructions below:

1. Add an X-BLOX pin to the current symbol.
2. Draw a box around the pin.
3. Fill the box solid, referring to any X-BLOX symbol as an example.

### Data Types

Within a single schematic sheet, you must specify the data type at least once per data path using the `BOUNDS` and `ENCODING` attributes. The same is true of hierarchical designs: specify the data type at least once per data path, whether or not the data path crosses hierarchical boundaries.

**Note:** If the bus size for the underlying schematic will not change, you can define the data path bounds and encoding at the subcircuit level. However, if you do not specify the data type in the underlying schematic, this schematic becomes generic and can be resized by the data type definitions specified in the main schematic.

### Data Type Propagation

When you define the data type of an X-BLOX bus at a level other than the subcircuit level, any changes to the bounds or encoding of that X-BLOX bus propagate throughout the entire data path, including the subcircuit.

## Signal Aliasing

In an X-BLOX design, all signal lines and buses have unique labels or names. Most of the buses represent unique data paths, but when one or more wires are equated with a CAST symbol or extracted from a bus by means of an ELEMENT or SLICE module, there will be different names assigned to the same logical wire:

- For the two differently named buses connected to the CAST module, each of the corresponding signals in the buses are equivalent to one another.
- The single line extracted from a bus by the ELEMENT module will have multiple names — one is the single line and the other is the equivalent net in the bus.
- Each of the wires included in the slice, or subset of wires selected by the SLICE module from a larger bus will also have multiple names — one from the main bus, the other from the sub-bus.

An alias is the name of a net used to refer to all equivalent nets in your design. Aliases are used because the XNF file supports only one name for any given net in your design. The SLICE, ELEMENT, and CAST modules are implemented merely as buses and nets connected without any logic or buffers. The software combines the net names obtained from these components and retains only the name supplied on the largest containing bus unless the KN attribute (KEEPNAME) is attached to the net or bus. All the other names will not be in the XNF file but will have signal aliases.

X-BLOX determines the signal aliases by assigning the label of the largest bus containing the signal to the lesser bus or signal, whenever possible, removing the original labels assigned to these buses. The signal names that are not kept are aliased by the “signal aliases.” The original labels and aliases of the lesser buses are reported in the .blx file. The example in Figure 2-8 shows several interrelated buses. Table 2-9 shows the resulting signal definitions and aliases

Nets that have not been aliased or that are aliases of other nets, can be referenced directly during simulation.

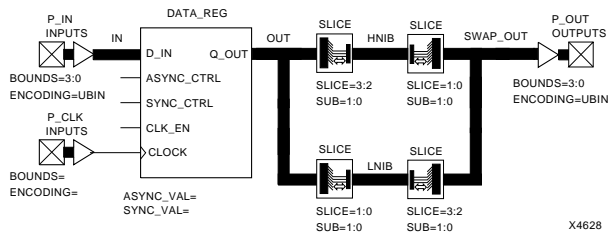
Some schematic capture packages can make use of the signal alias information. Refer to your Interface User Guide and check whether this feature is supported. If it is supported, all of the net names in

your original design that have not been absorbed into CLBs or IOBs will be accessible during timing simulation.

If this feature is not supported in your interface, you must reference an aliased net by its “signal alias.” All the nets and their aliases are listed in the .blx file. Thus, if you cannot reference your X-BLOX bus during timing simulation, it is because you are not referencing an alias or because the net has been absorbed into a CLB. Refer to the .blx file to determine the alias name to use in your simulation stimulus files. The example in Table 2-9 shows a .blx file alias report.

**Note:** To retain a name other than the one chosen by the software, specify the optional KN (KEEPNAME) attribute on the desired bus or signal.

**Warning:** Simulation command files should reference the label of the largest equivalent X-BLOX bus. Otherwise, the command file will not work for timing simulation.



**Figure 2-8 Example of Signal Aliasing**

In the circuit displayed in Figure 2-8, a new bus called SWAP\_OUT is created by re-arranging the bits in the OUT bus with SLICE modules. The first two SLICE modules (on the left) combine the two MSBs and the two LSBs of the OUT bus into SUB buses called HNIB and LNIB, respectively. The next two SLICE modules reassemble the bits into the bus called SWAP\_OUT with the indices relabeled. The X-BLOX software labels the outputs on the pads as P\_OUT/  
SWAP\_OUT<1,0,3,2>, which reflects the swapped pin sequence. The X-BLOX software reports the aliasing assignments as follows in the Data Type Propagation and Signal Aliasing chapter of the .blx file:



**Table 2-9 Sample BLX Report File**

<b>Signal</b>	<b>Data Type</b>	
CLK :	BIT	
GND :	GND	power: GND
HNIB :	ubin(1:0)	bus: OUT<3>, OUT<2>
HNIB<0> :	BIT	aliased by: OUT<2>
HNIB<1> :	BIT	aliased by: OUT<3>
IN :	ubin(3:0)	bus: IN<3>, IN<2>, IN<1>, IN<0>
IN<0> :	BIT	
IN<1> :	BIT	
IN<2> :	BIT	
IN<3> :	BIT	
LNIB :	ubin(1:0)	bus: OUT<1>, OUT<0>
LNIB<0> :	BIT	aliased by: OUT<0>
LNIB<1> :	BIT	aliased by: OUT<1>
OUT :	ubin(3:0)	bus: OUT<3>, OUT<2>, OUT<1>, OUT<0>
OUT<0> :	BIT	alias of: LNIB<0>, SWAP_OUT<2>
OUT<1> :	BIT	alias of: LNIB<1>, SWAP_OUT<3>
OUT<2> :	BIT	alias of: HNIB<0>, SWAP_OUT<0>
OUT<3> :	BIT	alias of: HNIB<1>, SWAP_OUT<1>
SWAP_OUT :	ubin(3:0)	bus: OUT<1>, OUT<0>, OUT<3>, OUT<2>
SWAP_OUT<0> :	BIT	aliased by: OUT<2>
SWAP_OUT<1> :	BIT	aliased by: OUT<3>
SWAP_OUT<2> :	BIT	aliased by: OUT<0>
SWAP_OUT<3> :	BIT	aliased by: OUT<1>
VCC :	VCC	power: VCC

This report indicates the data type and status of every signal in the design. For example, the bus IN has data type ubin(3:0), and is made up of the signals IN<3>, IN<2>, IN<1>, IN<0> (from MSB to LSB). The bus OUT is made up of the signals OUT<3>, OUT<2>, OUT<1>,

OUT<0> (MSB to LSB). The bus SWAP\_OUT has the same data type but is made up of the signals OUT<1>, OUT<0>, OUT<3>, OUT<2> (MSB to LSB). These signals are also in the OUT bus.

By looking at the entry for signal OUT<0>, you can see that this bit is the alias or name chosen for signals LNIB<0> and SWAP\_OUT<2>.

**Note:** Only the bus name OUT will be known during timing simulation.

## Bus Manipulation

The following sections explain how to use special modules to manipulate buses.

### CAST Symbol

Use the CAST module to connect buses that have:

- the same size but different encodings
- the same size but different bounds (ranges)
- the same size but different encodings and bounds

The following table gives examples of how to specify the BOUNDS and ENCODING attributes for pin A and pin B depending on the task you want to do.

**Table 2-10 Specifying the BOUNDS and ENCODING Attributes**

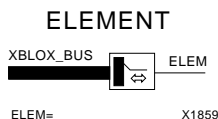
Action	Bounds and Encoding	
	A Pin	B Pin
Change the bounds	7:0 7:0	11:4 4:11
Connect big-endian and little-endian buses	7:0 7:0	0:7 2:9
Change the encoding (No conversion is performed)	BIT, UBIN, TWO_COMP, or ONE-HOT	BIT, UBIN, TWO_COMP, or ONE-HOT

In addition, you can use the CAST module in three different ways by

specifying the data types for both pins on the module (A and B), for one pin only (A or B), or by not specifying the attributes for either of the two pins. Refer to the CAST module description in the chapter “Module Definitions” for more details.

## ELEMENT Symbol

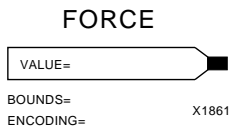
The ELEMENT symbol is used to extract a single wire from a bus or connect a single wire to a bus. To use it, specify the index of the wire to be connected or extracted from the bus using the ELEM attribute.



**Figure 2-9 ELEMENT Symbol**

## FORCE Symbol

The FORCE symbol forces a data value on a bus. Use it to specify a constant input value on the parallel D\_IN of a counter module or to set an input of a comparator module to a constant value. As this value is always present on the bus, you can use a TRISTATE module to control part-time access to the FORCE value. This symbol can also be used to define the data type of the value and the bus to which the FORCE symbol is attached. See the chapter “Module Definitions” for more details.

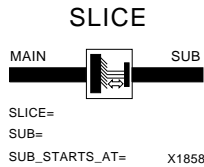


**Figure 2-10 FORCE Symbol**

## SLICE Symbol

The SLICE module extracts a portion of a larger bus or collects smaller buses together into a larger bus. It relabels groups of lines for your convenience. Note that both MAIN and SUB represent X-BLOX buses. The attributes on this symbol are complementary and need not

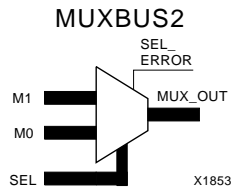
all be used at the same time. See the chapter “Module Definitions” for more details.



**Figure 2-11 SLICE Symbol**

## MUXBUS Symbol

The MUXBUS symbols are multiplexers used to route one of the  $n$  inputs to the output under the control of the Select input port. See the chapter “Module Definitions” for more details.



**Figure 2-12 MUXBUS2 2-to-1 Bus Multiplexer**

## Location Attributes

You can specify the placement of logic synthesized by X-BLOX by placing the LOC or LOC[ $i$ ] attributes on X-BLOX modules. A LOC attribute applies to all of the logic synthesized for an X-BLOX module. A LOC[ $i$ ] attribute applies to the  $i$ th element of the X-BLOX module only, and can be used only when the data type on the module is an array (not BIT). The values assigned to the LOC and LOC[ $i$ ] attributes are the locations of a CLB, TBUF, IOB, or edge decoder on the FPGA.

The LOC attribute can specify the location of a module with data type BIT (no bounds), for example, LOC=AA (CLB location AA), LOC=TL (Top Left edge for I/O, decode logic, or global buffer).

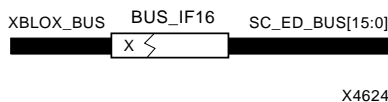
A LOC attribute can also be placed on an X-BLOX module whose data type includes BOUNDS. In this case, the attribute must specify a range of locations. For example, LOC=TBUF\_R\*CO.1 (a TBUF in any row in Column 0).

The LOC[*i*] attribute associates a LOC value with the logic synthesized for the *i*th bit of an X-BLOX module. The ENCODING=BIT with no BOUNDS does not have an index *i*, so LOC[*i*] is inappropriate. Several LOC[*i*] attributes can be placed on an X-BLOX module, each for a different *i*.

For example, LOC[1]=AA, LOC[-1]=AC specifies that Bit 1 goes in CLB location AA and Bit -1 goes in CLB location AC.

## Using BUS\_IFxx to Connect X-BLOX Buses to non-X-BLOX Logic

At times, you might want to connect X-BLOX circuitry to non-X-BLOX circuitry. The problem is that, contrary to X-BLOX buses, which can be resized by modifying the ENCODING and BOUNDS attributes, buses in non-X-BLOX circuitry have a specific size and cannot be connected directly to the generic X-BLOX buses. For this reason, you must use one of the Bus Interface macros (BUS\_IFxx macros) provided in the X-BLOX library to connect the X-BLOX bus and the schematic editor bus. (See Figure 2-13.) These schematic symbols behave like wires with no logic or direction control.



**Figure 2-13 X-BLOX Bus to Schematic-Editor Bus Interface**

Xilinx provides bus interface symbols for bus sizes between two and thirty-two bits wide, named BUS\_IF02 through BUS\_IF32. As the encoding and bounds are **not** specified for these macros, a separate X-BLOX BUS\_DEF symbol must be connected to the X-BLOX bus side of the BUS\_IFxx symbol.

You must use these symbols and connect them during the design phase to enable X-BLOX to complete its data type propagation during the synthesis process.

**Warning:** Ensure you have specified the data type (encoding and bounds) for the bus connected to the XBLOX\_BUS, either through data type propagation from other modules or with a BUS\_DEF module.

## Creating a Custom BUS\_IFxx Macro

If you need a bus interface for a size or bounds combination that is not provided, simply copy one of the existing BUS\_IFxx macros and edit it to suit your needs.

The following example explains how to connect a ubin (4:1) X-BLOX bus to a bus whose bits are numbered from 4 down to 1.

1. Copy the BUS\_IF04 symbol and schematic to a new name, for instance BI\_04\_01.
2. Edit the symbol, changing its name and the declaration for the bus pin. You do not need to change the X-BLOX bus pin.
3. Edit the schematic for BI\_04\_01.

The schematic for the BUS\_IF04 is shown below. It contains four X-BLOX ELEMENT symbols, one for each of the four bits (3 down to 0) in the bus.

4. The ELEMENT for bit 0 is no longer needed, but one is needed for bit 4. Change the index attribute, ELEM=0, to ELEM=4 on this ELEMENT symbol.
5. Change the name of the signal attached to this ELEMENT from B0 to B4.
6. Change the name of the B bus from B[3:0] to B[4:1].
7. Save the result and you are ready to use your BI\_04\_01 macro by connecting it to an X-BLOX bus with Bounds 4:1.

If you need to connect to a bus that has more than 32 bits, copy the BUS\_IF32 macro. Then, add more ELEMENT symbols to the new signals and add the appropriate ELEM indices.

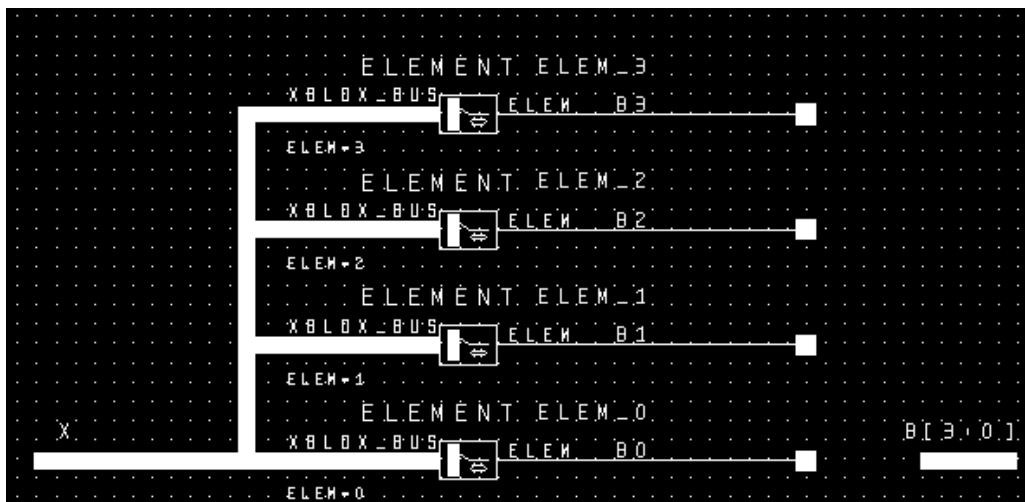


Figure 2-14 BUS\_IF04

## Using XACT-Performance Attributes

XACT-Performance™ enables you to specify precise timing requirements for your Xilinx FPGA designs. Use XACT-Performance to specify the maximum allowable delay for a set of paths in your design. You identify a set of paths by identifying a group of start and end points. The start and end points can be flip-flops, I/O pads, IOB latches, or XC4000 RAMs. You can control the worst-case timing on the set of paths by specifying a single delay requirement for all paths in the set.

The primary method of specifying timing requirements involves entering them on the schematic. However, you can also specify source timing requirements via the constraints file as well as PPR command-line options. These options do not provide as much control and flexibility as entering timing information directly on the schematic. See the “PPR” chapter in the *XACT Reference Guide, Volume 2* for more information about PPR command-line and constraints options.

Once you define timing specifications, PPR can then map, place, and route your design based on these requirements.

To analyze the results of your timing specifications, you can use the XDelay program. Refer to “The XDelay Timing Analysis Program” in the *XACT Reference Guide, Volume 3* for more information.

**Note:** Please refer to the XACT-Performance Utility section of the *XACT Reference Guide* for details on how to use XACT-Performance. You should be familiar with XACT-Performance concepts before continuing with this section. The concepts particular to X-BLOX are described in the present section.

## TSidentifier Attribute

Use the TSidentifier attribute as part of a TIMESPEC symbol. This attribute is used to convey timing constraints to PPR. You can also specify a TSidentifier attribute on nets connected to the modules; however, it is not recommended that you do so. Instead, you are encouraged to use the TNM syntax described below, which provides for more flexibility.

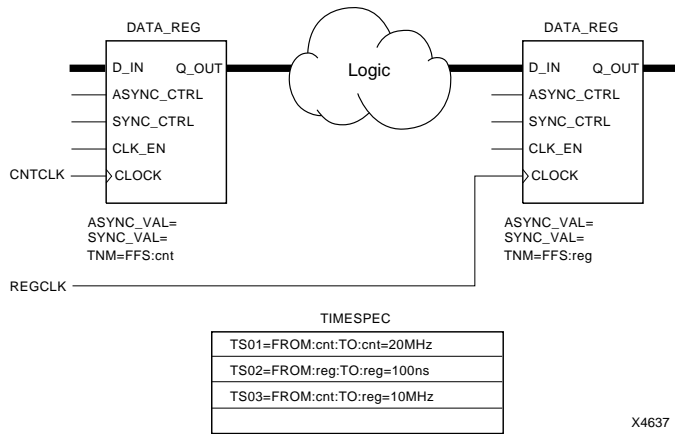
## TNM Attribute

Use TNM attributes to provide timing constraint information to the place and route program (PPR).

When you place them on X-BLOX modules, TNM attributes behave as they do when placed on macros from the Xilinx Unified Libraries: they name groups of XNF primitives in the XNF file output by X-BLOX. When you define them on input signals, TNM and TSidentifier attributes are passed to the signals driving the underlying logic. X-BLOX places the TNM attribute you provided on the appropriate primitives as it translates the design into an XNF file.

**Note:** XACT-Performance attributes are not propagated through the ELEMENT and SLICE modules in this release. Place TNM attributes on the nets or on X-BLOX modules that contain registers, I/Os, or RAMs





**Figure 2-15 Specifying TIMESPEC Statements**

A TNM attribute placed on an X-BLOX symbol might contain “unqualified” TNMs, that is TNMs with none of the qualifiers listed in Table 2-11. However, it is recommended that you use valid qualifiers. Refer to the documentation on XACT-Performance for the definition of these terms. The following table indicates the X-BLOX modules on which you can place a TNM attribute. It also shows the valid combinations of qualifiers and X-BLOX symbols. There are several classes of primitives that TNM attributes can be associated with. For each X-BLOX symbol type, the TNM attribute is propagated to the appropriate class of logic primitives.

A TNM with a qualifier on an X-BLOX symbol in violation of the table below is reported as an error.

**Table 2-11 Valid Combinations of Modules and Qualifiers**

<b>X-BLOX Symbol Type</b>	<b>TNM Class of Primitives</b>
ACCUM, SHIFT, COUNTER, DATA_REG, CLK_DIV	FFS
INPUTS, OUTPUTS, BIDI- R_IO, TRISTATE	PADS
SRAM, PROM	RAMS
DATA_REG <sup>a</sup>	LATCHES

a. On the DATA\_REG module, TNM propagates to flip-flops or latches depending on the STYLE attribute. See the documentation on the DATA\_REG module for more details.



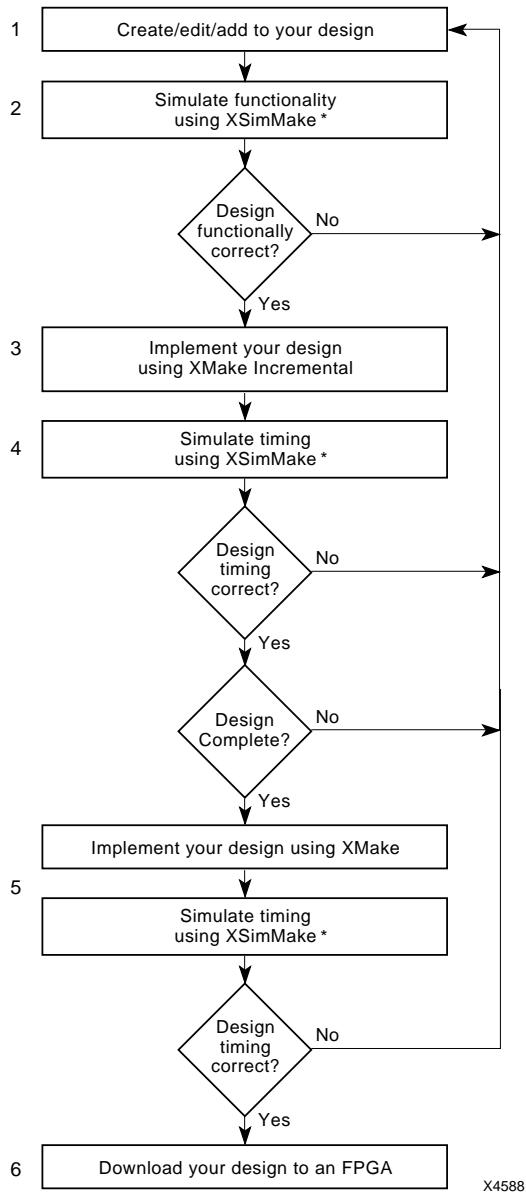
## Processing Your Design

---

This chapter describes the overall flow you should use to create and process an X-BLOX design. Refer to it whenever you need to verify the sequence of tasks you need to enter and process your design.

The flow used to enter and simulate X-BLOX designs is shown in Figure 3-1. The different tasks you must perform are addressed in the following sections:

- *Creating and Modifying Your Design*, outlines the steps used to create a design with your particular interface.
- *Performing Functional Simulation*, explains how to invoke the interface program shell that automatically generates simulation models.
- *Implementing an Incremental Design* guides you through the procedure for running XMake incrementally and processing a partial design.
- *Performing Timing Simulation* includes a procedure for invoking the interface program shell that automatically generates a timing model for your design.
- *Implementing the Complete Design* tells you how to run XMake to produce the bitstream for configuring a Xilinx FPGA.
- *Downloading Your Design* tells you what program to run to download your design into a Xilinx FPGA.
- *X-BLOX Design Example* provides a design example and instructions on how to implement the design.



X4588

Figure 3-1 Incremental Design Flow

\* If you are using the Mentor Graphics design tool, substitute for the XSimMake program mentioned in the preceding illustration the design interface program names used by that design tool: *pld\_fncsim8* for functional simulation and *pld\_timsim8* for timing simulation.

## Step 1: Creating and Modifying Your Design

In this step, you enter your design and specify all the attributes you need to customize the X-BLOX modules.

To create an X-BLOX design, follow the steps outlined in the list and tables below.

1. Configure the environment of your third-party schematic tools.
2. Load the X-BLOX library and other Xilinx libraries as appropriate.
3. Use modules from the X-BLOX library in your schematics.
4. Specify attributes to customize your X-BLOX modules.
5. Attach nets and X-BLOX buses, add bus definition modules where appropriate.

For more details, please refer to your interface user guide.

**Table 3-1 Configuring the Environment**

<b>Mentor</b>	<b>Viewlogic</b>	<b>OrCAD</b>
<p>1. Go to the working directory.</p> <p>2. Set \$MGC_WD to the working directory using the <b>setenv</b> command.</p> <p>3. Invoke PLD_DMGR by typing <b>pld_dmgr</b>.</p>	<p>1. When you create a project, Workview copies a generic viewdraw.ini file from the “workview/standard” directory to your project directory. At this time, edit your viewdraw.ini file to configure your environment.</p> <p>2. Select <b>Window</b> → <b>Open</b> → <b>Viewfile</b>. Then select <b>Project</b> → <b>Create</b> to create a new project directory.</p> <p>3. Specify a directory for your project. Enter <b>/workview/projectname</b> at the Project Directory prompt.</p> <p>4. Select <b>Set</b> → <b>Project</b>.</p> <p>5. Confirm the selection with the middle mouse-button.</p> <p><b>Note:</b> If you are using Viewlogic on a PC, use a backslash (\) instead of a slash (/) when entering path names.</p>	<p>1. Create a design directory and make it your current directory.</p> <p>2. Configure your design directory by running XDRAFT on the specific FPGA family. For example, to configure an XC4000 design, execute the command as follows: <b>xdraft 4 [options]</b></p>

Table 3-2 Loading a Library

Mentor	Viewlogic	OrCAD
<p>1. Select <b>p1d_da</b> by double clicking with the left mouse-button on the p1d_da icon.</p> <p>2. Select <b>Open Sheet</b> from the session palette with the left mouse-button.</p> <p>3. Use the <b>Navigator</b> button to select your design, or specify the path of your design. Then, click <b>OK</b> with the left mouse-button.</p> <p>4. Select the <b>Libraries</b> menu and choose <b>XACT_LIB</b>. From the XACT libs menu, in Design Architect, select <b>UNIFIED LIB</b>, and from the Unified Libraries menu, select <b>X-BLOX LIB</b>.</p>	<p>1. Edit viewdraw.ini or use <b>Project</b> → <b>Search</b> → <b>Viewdraw</b> in the project directory.</p> <p>2. Ensure that the path to the X-BLOX library is included in the viewdraw.ini file.</p>	<p>XDraft automatically adds the X-BLOX libraries to your config.sys file.</p>



**Table 3-3 Bringing Up a Symbol**

<b>Mentor</b>	<b>Viewlogic</b>	<b>OrCAD</b>
<p>1. Select an X-BLOX module from the displayed library menu with the left mouse-button.</p> <p>2. Place the symbol on your schematic using the left mouse-button.</p>	<p>1. Open a schematic sheet for editing by selecting <b>Window</b> → <b>Open</b> → <b>Viewdraw</b> → <b>Schematic</b> from the menu or by typing sch ↵. Press the middle mouse-button to get a prompt and enter a design name.</p> <p>2. Use the <b>Add</b> → <b>Comp</b> command to add X-BLOX symbols to your schematic. Press the middle mouse-button to get a prompt. At the Symbol Name prompt, type a component name.</p> <p>3. Place the component on the worksheet by pressing the middle mouse-button.</p>	<p>1. Invoke the Draft editor and bring up a schematic on your screen.</p> <p>2. Use the <b>Get</b> command from the Draft menu and type the name of the symbol you need at the Get? prompt. For example, MUX-BUS.</p> <p>3. Place the symbol on your schematic using the <b>Place</b> command.</p>

**Table 3-4 Specifying the Module Attributes**

<b>Mentor</b>	<b>Viewlogic</b>	<b>OrCAD</b>
<p>1. Select the component, for example, the X-BLOX BUS_DEF symbol, by clicking the left mouse-button on the symbol. Invoke the properties menu by pressing the right mouse-button on <b>Properties</b> → <b>Modify</b>. Select <b>Encoding</b> from the Properties menu by clicking on the left mouse-button and then clicking on <b>OK</b>.</p> <p>2. Enter a valid property value, for example, <b>ubin</b> for the Encoding property, then click on <b>OK</b>.</p>	<p>1. Select the component using the left mouse-button.</p> <p>2. Use the command <b>Change</b> → <b>Attr</b> → <b>Dialog</b> → <b>All</b> to specify attribute values for your component or type ca↵.</p> <p>3. Move the cursor to the appropriate value field and press the left mouse-button to start entering a value in the field.</p> <p>4. Press enter to submit the value to the system. Then select accept.</p>	<p>1. Use the <b>Edit</b> → <b>Edit</b> → <b>Options</b> → <b>Name</b> command.</p> <p>2. Enter the attribute name followed by the equal sign and a valid value at the prompt.</p>

## Step 2: Performing Functional Simulation

After entering a meaningful part of your schematic, you can simulate the functionality of your design as explained below. There are two main steps. First generate a functional simulation netlist. Then, run the simulation program supported by your third-party design package and simulate your design. The table below summarizes the procedure to follow when performing a functional simulation. Refer to your interface user guide for the complete procedure.

When the Viewlogic program XSimMake or the Mentor program `pld_fncsim8` processes designs that contain X-BLOX modules, it generates a special set of schematics that allow users to back-annotate simulation values to their schematics.

With Viewlogic, if your original schematic was called *design*, XSimMake creates a schematic called *sdesign*. In the *sdesign* schematic, the X-BLOX modules have been expanded to their specific bus widths. This feature enables the Viewsim simulator to back-annotate values to the *sdesign* schematic. This schematic should be used for both functional and timing simulation; however, you cannot use it for FPGA implementation and you must not run XMake on this design.

Similarly, for Mentor users, special schematics are created and placed in the directory called “simdir” to support schematic value back-annotation when using the `-o` or “use original” option.

Since Orcad does not support this feature, no special OrCAD schematics are created by XSimMake. XSimMake creates a directory called “otherxnf,” which contains all the simulation files.

**Note:** If you are using a PC, make sure your design name does not exceed 7 characters. If your design name exceeds 7 characters, XSimMake generates an error.

Table 3-5 Functional Simulation

Mentor	Viewlogic	OrCAD
<p>1. Invoke <b>p1d_fnctsim8</b> from within <b>p1d_dmgr</b> (Mentor's Design Manager that contains Xilinx's program icons) by double-clicking on the icon in the tool window with the left mouse-button.</p> <p>2. Enter the design name and the part type in the form. Fill in the options, select <b>Use Original</b> and click on the <b>OK</b> button.</p> <p>3. Run the Mentor Graphic <b>QuickSimII</b> program on the <b>simdir<sup>a</sup></b> directory to simulate your design.</p>	<p>1. Run <b>XSimMake</b> from the XDM™ Verify menu.</p> <p>2. Choose option -F.</p> <p>3. Select <b>Viewlogic_fpga_func</b> from the list of flows.</p> <p>4. Run the Viewlogic simulation program, <b>ViewSim</b>, to simulate your design. Your design is located in the <i>sdesign<sup>b</sup></i> simulation directory. The simulation netlist is called <i>sdesign.vsm</i>.</p>	<p>1. Run <b>XSimMake</b> from the XDM Verify menu.</p> <p>2. Choose option <b>-F</b>.</p> <p>3. Select <b>OrCAD_fpga_func</b> from the list of flows.</p> <p>4. Run <b>ASCTOVST</b> from the <b>Verify</b> menu.</p> <p>5. Run the OrCAD <b>simulate</b> program to simulate your design.</p>

a. *simdir* refers to the name of the simulation directory automatically created by the Mentor Graphics **p1d\_fnctsim8** program when the -o or "use original" option is selected.

b. *sdesign* refers to the name of the simulation directory automatically generated by XSimMake when it creates a simulation netlist file for a Viewlogic design.

## Step 3: Implementing a Partial Design

The next step is to implement your partial design to verify the timing of this portion (the actual delays) and the size of the design (number of CLBs). If your design is complete, run XMake as explained in step 5.

After you have successfully tested the functionality of your design, you can implement it using XMake in “incremental” mode. Running XMake incrementally means that you use special settings for the core tools automatically invoked by XMake. These settings ensure that the programs do not trim the incomplete logic from your design. They also prevent the program from issuing error and warning messages about this incomplete logic.

Once you feel you have entered a meaningful part of your schematic, follow the instructions below to run XMake from XDM and implement your design. Repeat this step as necessary using the guide file options listed in Table 3-1 for the incremental design flow.

1. Select the **Target** from the XMake pop-up menu. Target refers to the last action you wish to perform. In this case, choose **Placed and Routed Design** as the target.
2. Attach the Save-Signal attribute to all dangling nets and buses.
3. Specify the settings for the programs XMake will run as shown in Table 3-6. These settings are used for a partial design. They prevent the XACT software (XNFPrep, X-BLOX, and PPR) from removing incomplete logic from your design. They also prevent error or warning messages from appearing in your report file (.out file).

**Note:** If you are using the Mentor interface, you need to invoke PLD\_XDM to modify the XMake program settings.

**Table 3-6 Program Settings**

<b>Program</b>	<b>Setting</b>	<b>Function</b>
XNFPrep	savesig=true	Saves open signals.
X-BLOX	archopt=false mergeio=false	Turns off optimization.
PPR	no complete  if guide, guide=filename	Prevents logic from being removed.  If you are using a guide file, specify its name.

**Note:** On a PC, specify these settings from the **Options** selection of the **Profile** menu. Page down to the appropriate program (XNFPrep, X-BLOX, or PPR), select the program, and highlight the required settings.

4. Save your profile. When you run XMake, the program uses the options you specified.
5. Run XMake.

## Step 4: Performing Timing Simulation

Once you are satisfied with your partial design, you can test the timing of your design. There are two steps in this process. First, generate the appropriate simulation files. Then, simulate your design using the simulation program supported by your third-party design package.

X-BLOX supports signal aliasing for your simulation files. Please refer to the section “Signal Aliasing” in the chapter “Creating an X-BLOX Design.”

When the Xilinx XSimMake program operates on Viewlogic designs that contain X-BLOX modules, it generates a special set of schematics that allow Viewlogic users to back-annotate simulation values to their schematics. If your original schematic was called *design*, XSimMake creates a schematic called *sdesign*. In the *sdesign* schematic, the X-BLOX modules have been expanded to their specific bus widths. This enables the Viewsim simulator to back-annotate values to the *sdesign* schematic. This schematic should be used for both functional and timing simulation; however, you cannot use it for FPGA implementation.

Similarly, for Mentor users, special schematics are created and placed in the directory called *design\_tim* to support schematic value back-annotation when using the *-g* or “autogenerate” option.

Since Orcad does not support design back-annotation, no special OrCAD schematics are created by XSimMake. XSimMake creates a directory called “otherxnf,” which contains all the simulation files.

**Note:** If you are using a PC, make sure your design name does not exceed 7 characters. If your design name exceeds 7 characters, XSimMake generates an error.

Table 3-7 Timing Simulation

Mentor	Viewlogic	OrCAD
<p>1. Invoke <b>p1d_timsim8</b> from within <b>p1d_dmgr</b> (Mentor’s Design Manager that contains Xilinx’s program icons) by double-clicking on the <b>p1d_timsim8</b> icon in the tool window with the left mouse-button.</p> <p>2. Enter the design name in the form and fill in the options. Use the “auto-generate” option and click on the OK button. Your design is located in the <i>design_tim</i><sup>a</sup> simulation directory.</p> <p>3. Run <b>QuickSimII</b> on <b>design_tim</b>.</p>	<p>1. Run <b>XSimMake</b> from the XDM Verify menu.</p> <p>2. Choose option <b>-F</b>.</p> <p>3. Select <b>Viewlogic_fpga_timing</b> from the list of flows. Your design is located in the <i>sdesign</i><sup>b</sup> simulation directory. The simulation netlist is called <i>sdesign.vsm</i>.</p> <p>4. Run <b>Viewsim</b>.</p>	<p>1. Run <b>XSimMake</b> from the XDM Verify menu.</p> <p>2. Choose option <b>-F</b>.</p> <p>3. Select <b>OrCAD_fpga_timing</b> from the list of flows.</p> <p>4. Run <b>ASCTOVST</b> from the <b>Verify</b> menu.</p> <p>5. Run <b>Simulate</b>.</p>

a. *design\_tim* refers to the name of the simulation directory automatically created by the Mentor Graphics **p1d\_timsim8** program when the -g or “auto generate” option is selected.

b. *sdesign* refers to the name of the simulation directory automatically generated by XSimMake when it creates a simulation netlist file for a Viewlogic design.



## Step 5: Implementing the Complete Design

When your design is complete and has passed simulation tests successfully, run XMake again to produce a bitstream for chip programming.

1. Select the target from the XMake pop-up menu. Choose **Placed and Routed Design** as the target.
2. Make sure you do not select the MAK file as the file on which you run XMake. MAK is the summary report file generated by XMake.
3. Remove the settings set in step 3 (Implementing a Partial Design) by unselecting them. X-BLOX optimizes the design by merging flip-flops into the IOBs; this might change the timing of the data paths through the optimized IOBs. X-BLOX also tries to optimize high fanout clock nets. Refer to the chapter “Understanding X-BLOX Operations” for information on design optimization.
4. Save your Profile.
5. Run XMake.

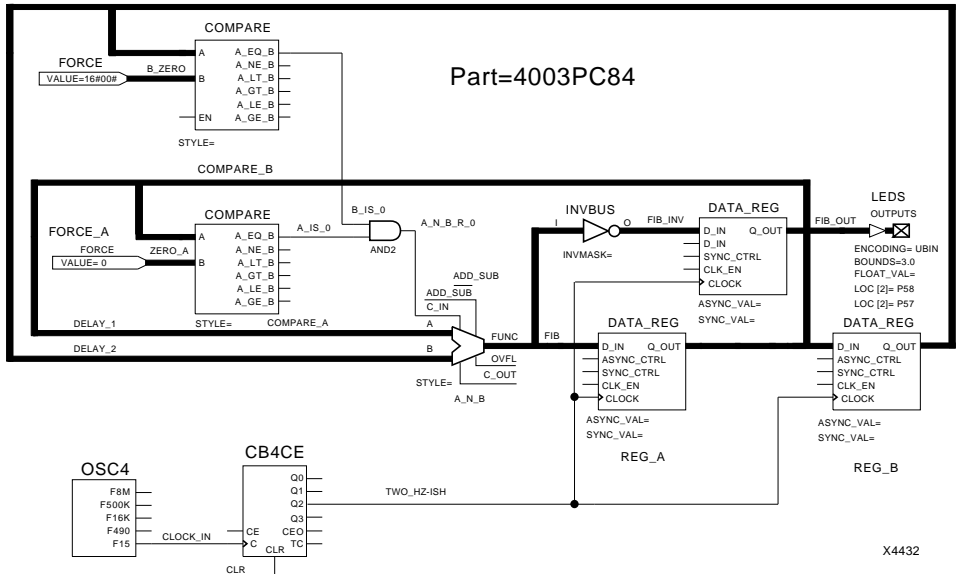
At this point, you should run the timing simulation program again as explained in step 4.

## Step 6: Downloading Your Design

Once your design is complete, you can download your design into an FPGA device using the BIT file generated by XMake. For instructions on how to download your design, refer to the *XACT Reference Guide*.

## X-BLOX Design Example

This section provides a design example to demonstrate the use of X-BLOX in a design using XMake from within the Xilinx Design Manager (XDM). The design example uses components taken from the X-BLOX library and the XC4000 library.



**Figure 3-2 Sample Design — Fibonacci Sequence Generator Using the X-BLOX Design Tool**

The proposed example is a design of a Fibonacci Sequence Generator. A Fibonacci sequence is a sequence in which every value is the sum of the previous two values. The following is a Fibonacci sequence:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \dots$$

$$(0, 1, 0+1=1, 1+1=2, 1+2=3, 2+3=5, \text{etc.})$$

### X-BLOX Modules Used in This Design

- ADD\_SUB (Adder-Subtractor)
- COMPARE (Comparator)
- DATA\_REG (Data Register)

- FORCE (Forces a value)
- OUTPUTS (Output)
- INVBUS (Bus Invertor)

## **XC4000 Library Modules Used in This Design**

- OSC4 (OSCILLATOR)
- CB4CE (COUNTER)
- AND2 (AND gate)

### **Description**

The output of the ADDER is delayed by two clock cycles. An X-BLOX DATA\_REG module, REG\_A, contains the sum delayed by one clock cycle; REG\_B contains the sum delayed by two clock cycles. The sum of the two is the Fibonacci sum.

All the X-BLOX registers are initialized to zero at power-up; therefore, the Fibonacci sequence is 0, 0, 0, . . . unless a one is introduced into the sequence.

If a one is introduced into the sequence, the outputs of REG\_A and REG\_B are compared with zero. The FORCE modules supply the zero values to one input of the COMPARE modules, and the other inputs of the COMPARE modules are connected to the register outputs.

To show the flexibility of X-BLOX, the two FORCE values are defined with different radices. FORCE\_A uses VALUE=0, which is decimal (X-BLOX default radix), and FORCE\_B uses VALUE=16#00#, which is hexadecimal. When the inputs to the comparators are equal (that is, all are zeros), their High A=B outputs are ANDed and fed to the carry-in of the ADD\_SUB module. The carry-in inserts a one into the system and changes the sequence from 0, 0, 0, . . . to 0, 1, 1, 2, . . . etc.

To provide a visible display of the Fibonacci sequence, the outputs of the OUT\_REG are fed to a set of four LEDs. As the LEDs are active Low, an X-BLOX INVBUS module is used to invert the data into the OUT\_REG. Also, to make the output intelligible, some library symbols (OSC4 and CB4CE) are used to divide the clock down to about 2 Hz.

## Design Procedure

Complete the design shown in Figure 3-2 by following the steps outlined below.

1. Place the X-BLOX components and other components from the XC4000 library on your schematic and connect them as shown in Figure 3-2.

Notice that the size and encoding of the design are generic at this point. You can specify the data type and encoding on the data path of any one of the following X-BLOX modules: INPUTS, OUTPUTS, BIDIR\_IO, or BUS\_DEF.

2. Attach the BOUNDS and ENCODING attributes to the OUTPUTS module, specifying the following values:
  - BOUNDS = 3:0
  - ENCODING = UBIN

The first attribute sets the port size, which is the bus size, to four bits wide. The second attribute sets the data encoding to unsigned binary.

As the design has only one data path, you have just defined the data type for the entire design by specifying the attributes on a single module.

3. Label all buses and nets in your design. The messages X-BLOX generates as it processes the design are more meaningful to you when they refer to a bus, a net, or a module by a label you have chosen rather than the label generated by the design entry tool.
4. Save the file.
5. You can run all the steps described in the “Design Flow” section by selecting XMake from the Translate menu in XDM.

X-BLOX processes the design and writes out an XG file, an *xblox.log* file, and a report file (BLX file).

When X-BLOX processes the design, it reports some interesting things:

1. The sizes of the FIB\_OUT, FIB, FIB\_INV, DELAY\_1 and DELAY\_2 buses are all four bits wide and contain unsigned-binary data (that is, BOUNDS=3:0 and ENCODING=UBIN).

2. The nets B\_IS\_0, A\_IS\_0, A\_N\_B\_R\_0, TWO\_HZ\_ISH, and CLOCK\_IN are one bit wide.
3. The ADD\_SUB module is implemented as an RPM that includes the REG\_A module.
4. The OUT\_REG module is placed in the IOBs (not in the CLBs). The TWO\_HZ\_ISH line is a medium fan-out (12) clock line, so it is placed on a longline. Note that if this design needed a higher fan-out signal, it would have been placed on one of the high-speed, low-skew global buffers.
5. The modules are expanded to the desired widths, using the proper arithmetic (unsigned binary) for the ADD\_SUB module.

Two of the IOB placement constraints are specified on the OUTPUTS module labeled LEDS. The attributes LOC[2]=P58 and LOC[3]=P57 specify that bit2 is to be placed on package pin P58, and bit 3 is to be placed on package pin P57. To illustrate the flexibility of the X-BLOX software, the placement of the other two output pins is specified in a constraints file *fibgen.cst*:

```
place instance LEDS/FIB_OUT<0>: p60;  
place instance LEDS/FIB_OUT<1>: p59;
```

PPR produces a routed LCA file for this design, which is intended for an XC4000 FPGA device.

## Module Definitions

---

Each module represents a common logic function and is described in detail in the following pages presented in alphabetical order. The size, or width of a port on a module, is determined in one of four ways.

- By the size of the bus attached to the port of the module. (See “Data Type Propagation” in the previous chapter.)
- By the size of a bus attached to a different port of the same module (converting binary  $2^n$  to  $n$  lines).
- By a data type specification on the INPUTS, OUTPUTS, BIDIR\_IO, BUS\_DEF, SHIFT, COUNTER, SRAM, or PROM modules. (See the description of the individual modules.)
- By an attribute, such as COUNT\_TO on a counter without parallel\_out connected.

The following table lists the different categories of X-BLOX modules.

**Note:** In this chapter, optional attributes appear in blue below the modules to which they apply.

**Table 4-1 X-BLOX Modules Listed by Functional Category**

Arithmetic	Bus	Logic	I/O	Sequential	Storage
ACCUM	BUS_DEF	ANDBUS	BIDIR_IO	CLK_DIV	DATA_REG
ADD_SUB	CAST	ANDBUS1	INPUTS	COUNTER	SHIFT
COMPARE	ELEMENT	ANDBUS2	OUTPUTS	SHIFT	PROM
COUNTER	FORCE	INVBUS			SRAM
INC_DEC	SLICE	ORBUS			
		ORBUS1			
		ORBUS2			
		XORBUS			
		XORBUS1			
		XORBUS2			
		DECODE			
		MUXBUS			
		MUXBUS2			
		MUXBUS4			
		MUXBUS8			
		TRISTATE			

The SHIFT and COUNTER modules are shown in more than one category.

## Bused Gate Functions

This section describes the generic bused gate functions of X-BLOX. Bused gate functions are defined as generalizations of the common logic primitives. These functions act on all the members of a bus in three different ways. For example, the following functions are described for AND but the same bus expansion criteria apply for OR and XOR. You can use the inversions of these logic functions by connecting an INVBUS module to the outputs. You can also invert individual inputs (active Low) by using the INVMASK attribute associated with each module or specify which individual inputs will be active High with the DECODEMASK attribute.

- The ANDBUS module logically ANDs all the individual members of an input bus together to produce a single logic signal.

- The ANDBUS1 module logically ANDs all the individual members of a bus with a single logic signal. The data type on the bused input and output buses must be the same.
- The ANDBUS2 module logically ANDs the corresponding two  $n$ -input buses to produce an  $n$ -output bus. The data type on all the input and output buses must be the same.

## ANDBUS Module

The ANDBUS module supports the following optional attributes on the XC4000 in addition to the INVMASK and DECODEMASK attributes:

### **STYLE=WAND**

The ANDBUS is synthesized with TBUFs configured as a wired-AND with a pull-up resistor.

### **STYLE=DECODE**

The ANDBUS is synthesized with the wide edge-decoders.

With either of these implementation styles, you can use the following location attributes:

### **LOC=*edge-location* or *TBUF-location***

where *edge-location* is the edge for the wide edge-decoder and can be any of the edge-locations listed below.

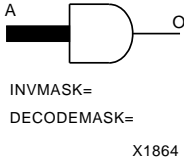
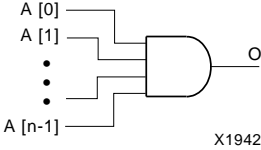
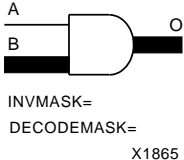
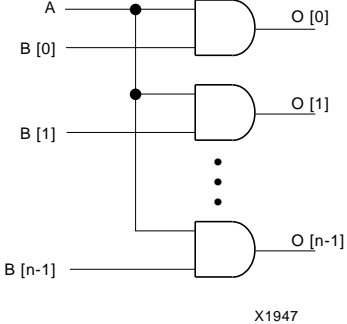
- T — top edge
- B — bottom
- L — left
- R — right
- TL — top left
- TR — top right
- BL — bottom left
- BR — bottom right

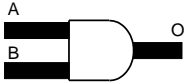
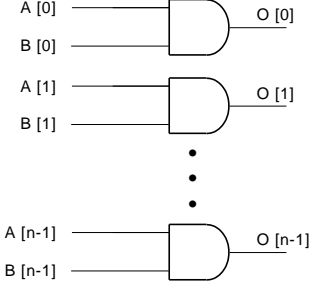
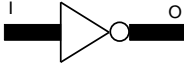
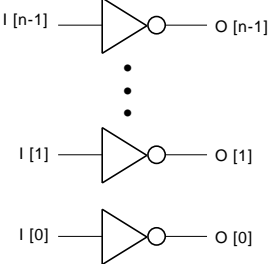

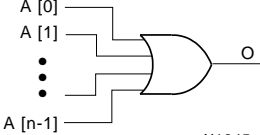



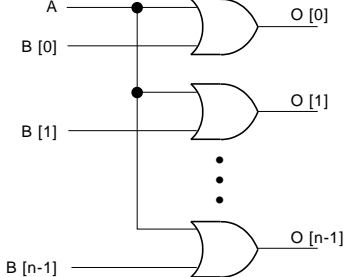

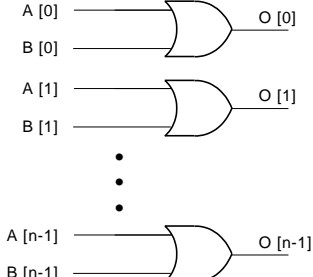
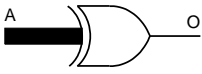
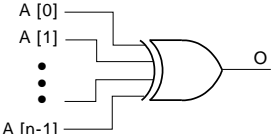
### LOC[i]=TBUF-location


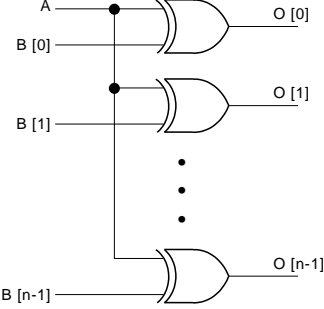

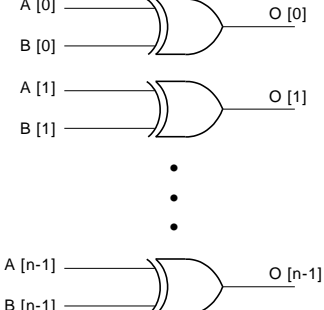
The TBUF location of the individual TBUFs, for instance LOC[1]=TBUF\_R1C1.1. For a definition of location parameters, please refer to the section “Location Attributes” in the chapter “Creating an X-BLOX Design.

Figure 4-1 X-BLOX Gate-Level Functions

Name	Symbol	Logic Diagram
<p>ANDBUS  <math>O = A_0 \bullet A_1 \bullet \dots \bullet A_{(n-1)}</math></p>	 <p>INVMASK=  DECODEMASK=  X1864</p>	 <p>X1942</p>
<p>ANDBUS1  <math>O_n = A \bullet B_n</math></p>	 <p>INVMASK=  DECODEMASK=  X1865</p>	 <p>X1947</p>

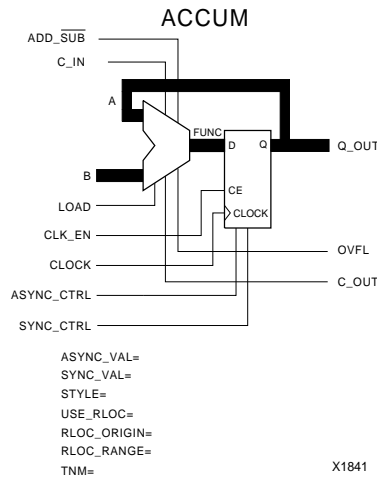
Name	Symbol	Logic Diagram
<p><b>ANDBUS2</b>  <math>O_n = A_n \bullet B_n</math></p>	 <p>INVMASK=  X1866</p>	 <p>X1946</p>
<p><b>INVBUS</b>  <math>O_n = \bar{I}_n</math></p>	 <p>INVMASK=  X1863</p>	 <p>X2044</p>
<p><b>ORBUS</b>  <math>O = A_0 + A_1 + \dots + A_{(n-1)}</math></p>	 <p>INVMASK=  = X1867</p>	 <p>X1945</p>

Name	Symbol	Logic Diagram
<p><b>ORBUS1</b>  <math>O_n = A + B_n</math></p>	 <p>INVMASK= X1868</p>	 <p>X1944</p>
<p><b>ORBUS2</b>  <math>O_n = A_n + B_n</math></p>	 <p>INVMASK= X1869</p>	 <p>X1943</p>
<p><b>XORBUS</b>  <math>O = A_0 \oplus A_1 \oplus \dots \oplus A_{(n-1)}</math></p>	 <p>INVMASK= X1870</p>	 <p>X1948</p>

Name	Symbol	Logic Diagram
<p><b>XORBUS1</b>  <math>O_n = A \oplus B_n</math></p>	 <p>INVMASK=  X1871</p>	 <p>X1953</p>
<p><b>XORBUS2</b>  <math>O_n = A_n \oplus B_n</math></p>	 <p>INVMASK=  X1872</p>	 <p>X1954</p>

## ACCUM — Accumulator

The Accumulator adds or subtracts the data on the B input port and the Carry-in/Borrow-in port to or from the current value stored in the accumulator register, then loads the result back into the register. The Carry-out/Borrow-out and Overflow outputs are provided by the Adder/Subtractor to indicate the status of the present operation (these outputs are not latched by this module). Also, you can load separate predefined values synchronously or asynchronously into the register.



**Figure 4-2 The Accumulator Module Symbol**

**Table 4-2 Accumulator Register Truth Table**

LOAD	SYNC_CTRL	CLK_EN	CLOCK	ASYNC_CTRL	Q_OUT
X	X	X	X	H	ASYNC_VAL
X	X	L	↑	L	Q_OUT <sub>prev</sub> <sup>a</sup>
X	H	H	↑	L	SYNC_VAL
L	L	H	↑	L	FUNC= Q_OUT <sub>prev</sub> ± B
H	L	H	↑	L	FUNC=B

a. Q\_OUT<sub>prev</sub> denotes the previous contents of the register.

## Inputs

### ADD\_SUB

The Add/Subtract control input determines the arithmetic operation: High = add, and Low = subtract. If you do not connect this input, the module is synthesized as an Adder.

### C\_IN

The Carry-in/Borrow-in input port, together with the data on the B input port, is added to or subtracted from the current accumulator value. The Borrow-in function is active Low. If C\_IN is unconnected, the defaults are Carry-in = 0 for Add, and Borrow-in = 1 for Subtract.

### B

The data on the B input port, along with the Carry-in/Borrow-in, is added to or subtracted from the present accumulator value. When Load Enable is High, the data on the B port is loaded directly into the register during the next active Clock transition. You must connect this input. The data type of this port is the same as the Q\_OUT output port.

### LOAD

When the Load Enable input is High, the B input data is loaded directly into the accumulator register during the active Clock

transition. When the Load Enable input is Low, the output of the adder/subtractor is loaded into the accumulator register during the active Clock transition. If you do not connect the Load Enable input, the FUNC output of the adder/subtractor is always selected. (See the definition of SYNC\_CTRL input.)

## **CLK\_EN**

When the Clock Enable input is High, either the data on the B input port, the output of the adder/subtractor, or the SYNC\_VAL attribute is loaded into the accumulator register during the next active Clock transition. When the Clock Enable input is Low, the register contents are unaffected by the active Clock transition (hold). CLK\_EN does not affect asynchronous loading of the register via ASYNC\_CTRL. Hold indicates that the register holds its current value. If you do not connect the Clock Enable input, the Clock is always enabled.

## **CLOCK**

When you enable the Clock input, it loads the selected data into the register on the rising (positive) edge. An active falling (negative) edge can be synthesized by connecting an inverter to the Clock input. The Clock must be connected.

## **ASYNC\_CTRL**

The Asynchronous Control input, when High, loads the value of the ASYNC\_VAL attribute into the accumulator register independently of the Clock and Clock Enable. It is a level-sensitive input. If you do not connect the ASYNC\_CTRL, this function is not synthesized; however, the ASYNC value will be loaded into the accumulator register on power-up.

## **SYNC\_CTRL**

When the Synchronous Control Enable and CLK\_EN inputs are High, the value of the SYNC\_VAL attribute is loaded into the accumulator register during the next active Clock transition. This input has priority over the LOAD input.

## Outputs

### Q\_OUT

Q\_OUT is the output (sum or difference) port of the accumulator register. The data type of this port is the same as the B input port.

### C\_OUT

C\_OUT is the Carry-out/ $\overline{\text{Borrow-out}}$  port from the most significant bit of the adder/subtractor.

### OVFL

OVFL is the Overflow (Underflow) output from the adder/subtractor. The OVFL output is High when the result of the operation exceeds the precision of the adder/subtractor.

## Attributes

### ASYNCR\_VAL

ASYNCR\_VAL is the predefined value that is loaded into the accumulator register when the ASYNCR\_CTRL input is High. This value will also be loaded into the counter at “power up”, whether the ASYNCR\_CTRL is connected or not. If not specified, the default value is “zero.”

### SYNCR\_VAL

SYNCR\_VAL is the predefined value that is loaded into the accumulator register when the SYNCR\_CTRL and CLK\_EN inputs are High during the active Clock transition. If no SYNCR\_VAL was specified, the default value is “zero.”

### STYLE

The default implementation style is ALIGNED for the XC4000. The default implementation style for the XC3000A/L and XC3100A is FAST3KA. Available options include the following:



**Table 4-3 ACCUM — Implementation Styles**

Style	Description
ALIGNED	Aligned RPM (XC4000 only)
UNALIGNED	Unaligned RPM (XC4000 only)
FAST3KA	Gate-level, 1-bit look ahead, fast adder (XC3000A/L and 3100A only)
RIPPLE	Gate-level ripple carry, area-efficient adder

**Note:** The default style for both X-BLOX and RPMs is ALIGNED. Exercise care when mixing ALIGNED and UNALIGNED macros as these two implementation styles are not compatible when you place and route with PPR.

### **USE\_RLOC={TRUE|FALSE}**

If this attribute is set to false, RLOCs are not generated. It is redundant to set this attribute to TRUE.

### **RLOC\_ORIGIN**

Position the upper left corner of the RPM at a particular FPGA location. Do not specify a location that does not allow enough room for the ACCUM. Refer to the section “Computing the Number of CLBs” in the chapter “Understanding X-BLOX Operations” for more information.

### **RLOC\_RANGE**

Use this attribute to specify the range (rectangular area) of FPGA locations that are allowed for the generated RPM. Refer to the chapter “X-BLOX Generated Relationally Placed Macros” for more information.

### **TNM**

Use the TNM attribute to specify timing requirements.

You first place a TNM attribute on this module to identify the start or end point of a path using the following syntax:

```
TNM=FFS: identifier
```

where FFS is the type of primitives to which X-BLOX propagates the TNM attribute in the XNF file and where *identifier* is a user-defined name. Refer to the section “Using XACT-Performance Attributes” in the chapter “Creating an X-BLOX Design” for more information.

You then use all the TNM attributes defined on the schematic in a TIMESPEC statement to specify delay requirements.

## ADD\_SUB — Adder/Subtractor

The Adder/Subtractor module adds or subtracts two data inputs and a carry/borrow input. Use this module as an adder or a subtractor, or switch it between these two modes. The Adder/Subtractor provides the Carry-out/Borrow-out and Overflow ports to indicate the status of the current operation.

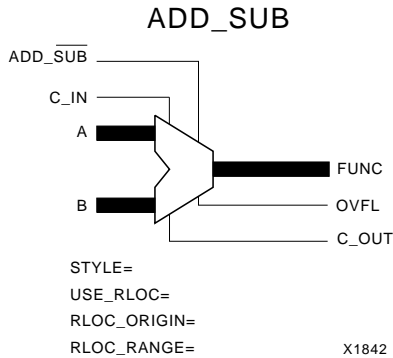


Figure 4-3 The Adder/Subtractor Module Symbol

### Inputs

#### ADD\_SUB

The Add/Subtract control input determines the arithmetic operation: High = add, and Low = subtract. If you do not connect this input, the module is synthesized as an Adder.

#### C\_IN

The Carry-in/Borrow-in input port, along with the data on the B input port, is added to or subtracted from the data on the A input port. The Borrow-in function is synthesized as active Low. If you do not connect C\_IN, the Carry-in = 0 for Add, and Borrow-in = 1 for Subtract.

**A**

The data on the B input port, along with Carry-in/ $\overline{\text{Borrow-in}}$ , is added to or subtracted from the data on the A input port (A+B or A-B). This input must be connected. The data type of this port must be the same as the B input and FUNC output ports.

**B**

The data on the B input port, along with Carry-in/ $\overline{\text{Borrow-in}}$ , is added to or subtracted from the data on the A input port (A+B or A-B). You must connect this input. The data type of this port must be the same as the A input and FUNC output ports.

**Outputs****FUNC**

FUNC is the Functional output (sum or difference) port from the adder/subtractor. The data type of this port is the same as the A and B input ports.

**OVFL**

OVFL is the Overflow (Underflow) output from the adder/subtractor. The OVFL output is High when the result of the operation exceeds the precision of the adder/subtractor.

**Note:** OVFL=C\_OUT if the encoding of the connected buses is UBIN.

**C\_OUT**

C\_OUT is the Carry-output/ $\overline{\text{Borrow-out}}$  port from the most significant bit(s) of the adder/subtractor.

**Attributes****STYLE**

The default implementation style is ALIGNED for the XC4000. The default implementation style for the XC3000A/L and 3100A is FAST3KA. Available options include the ones shown in the following table.

**Table 4-4 ADD\_SUB — Implementation Styles**

Style	Description
ALIGNED	Aligned RPM (XC4000 only)
UNALIGNED	Unaligned RPM (XC4000 only)
FAST3KA	Gate-level, 1-bit look ahead, fast adder (XC3000A/L and 3100A only)
RIPPLE	Gate-level ripple carry, area-efficient adder

**Note:** The default style for both X-BLOX RPMs and the Xilinx Unified Libraries macros is ALIGNED. Exercise care when mixing ALIGNED and UNALIGNED macros as designs that use both styles might be hard to route.

### USE\_RLOC={TRUE|FALSE}

If this attribute is set to false, the RLOCs are not generated. It is redundant to set this attribute to TRUE.

### RLOC\_ORIGIN

Position the upper left corner of the RPM at a particular FPGA location. Do not specify a location that does not allow enough room for the ADD\_SUB. Refer to the section “Computing the Number of CLBs” in the chapter “Understanding X-BLOX Operations” for more information.

### RLOC\_RANGE

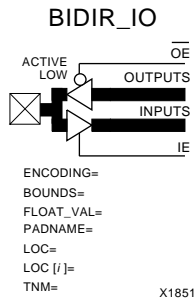
Use this attribute to specify the range (rectangular area) of FPGA locations that are allowed for the generated RPM. Refer to the chapter “X-BLOX Generated Relationally Placed Macros” for more information.



## BIDIR\_IO — Bidirectional I/O Pads with Buffers

The BIDIR\_IO module is a bidirectional chip Input/Output module that connects both input and output buffers from common pads to internal input and output wires or buses. The BIDIR\_IO module expands into one or more bidirectional I/O pads, input and output data and control signals, plus 3-state input/output buffers. You can define the pad locations for a given package in the following ways:

- By adding a LOC attribute on the symbol for a single-bit port
- By adding one or more LOC[i] attributes on the symbol for a multibit port
- By using a Place and Route Constraints file. (See the subsection “Attributes,” which is included in this module’s description.)



**Figure 4-4 The Bidirectional I/O Module Symbol**

You can connect this module to an on-chip bidirectional bus by tying the Input and Output ports together, and using the Input Enable and Output Enable signals to control direction. You can connect a single enable signal to both IE (Input Enable) and  $\overline{OE}$  (Output Enable) to switch between input and output modes.

You can use this module as a 3-stated OUTPUTS module by not connecting the INPUTS and IE pins.

## Inputs

### $\overline{OE}$

$\overline{OE}$  is the active-Low Output Enable for the module. When  $\overline{OE}$  is Low, the data on the OUTPUTS port is available at the I/O pads. If  $\overline{OE}$  is left unconnected, the data on the OUTPUTS port is always available at the pads.

### OUTPUTS

The data on the OUTPUTS bus is available at the I/O pads when  $\overline{OE}$  is Low. This port is connected to a single internal wire or a bus defined by the ENCODING and BOUNDS attributes.

### IE

IE is the active-High Input Enable for the module. When IE is High, the data on the I/O pads is available at the INPUTS port. If you do not connect IE, the data on the pads is always available at the INPUTS port.

## Outputs

### INPUTS

The data on the I/O pads is available on the INPUTS bus when IE is High. This port is connected to a single internal wire or a bus defined by the ENCODING and BOUNDS attributes.

## Attributes

### ENCODING

The available encodings are defined in the “Bus Data Types” section of the chapter “Creating an X-BLOX Design.” All the encodings used on each data path must be the same.

### BOUNDS

The BOUNDS attribute defines the width of the INPUTS and OUTPUTS buses by specifying the MSB and LSB of each bus. Any



pair of numbers can be used, as long as the difference of the indices, plus one, equals the width of the bus. See the “Bus Data Types” section of the chapter “Creating an X-BLOX Design” for more details.

## **FLOAT\_VAL**

Use the `FLOAT_VAL` attribute to connect pull-up or pull-down resistors to the input/output pads defined by this module. You can specify this value in any of the allowed radices and use don't care digits. The bits of the `FLOAT_VAL` number specify which I/O pads are connected to pull-up resistors, pull-down resistors or neither (that is, 1 = pull-up resistor, 0 = pull-down resistor, ? = none). Alternately, you can tie all the pads of a symbol to pull-up or pull-down resistors by specifying `FLOAT_VAL=PULLUP` or `FLOAT_VAL=PULLDOWN` respectively. By default, no resistor is connected. See the section “Pull-up and Pull-down Resistors for I/O Pads” in the chapter “Creating an X-BLOX Design” for examples.

## **PADNAME**

Use this attribute to specify the base name of the I/O pad. With this attribute set to `PADNAME=foo`, the names of the pads will be `SYM/FOO<0>`, `SYM/FOO<1>`, ..., `SYM/FOO<n>`, corresponding to the `BOUNDS` associated with the attached buses and the `BOUNDS` attribute. Without this attribute, the names of the I/O pads will be `SYM/PAD<0>`, `SYM/PAD<1>`, ..., `SYM/PAD<n>`, where `SYM` is the name of the `BIDIR_IO` symbol.

## **LOC**

The `LOC` attribute specifies the pin location for a single input/output pad-buffer pair, for example: `LOC=A1`, or the attribute specifies the placement of all IOBs on a specific edge or corner of the chip, for instance, `LOC=TL` for the top-left corner of the chip.

## **LOC[i]**

The `LOC[i]` attribute specifies the pin locations for multiple input/output pad-buffer pair locations, for example: `LOC[6]=A11`, `LOC[7]=P9`.

## TNM

Use the TNM attribute to specify timing requirements.

You first place a TNM attribute on this module to identify the start or end point of a path using the following syntax:

**TNM=PADS: *identifier***

where PADS is the type of primitives to which X-BLOX propagates the TNM attribute in the XNF file and where *identifier* is a user-defined name. Refer to the section “Using XACT-Performance Attributes” in the chapter “Creating an X-BLOX Design” for more information.

You then use all the TNM attributes defined on the schematic in a TIMESPEC statement to specify delay requirements.

## Constraints File

The BIDIR\_IO module is synthesized into an array of I/O pads with names of the form: SYM/SIG<0>, SYM/SIG<1> . . . SYM/SIG<N-1>. SYM is the schematic-editor label or instance for the BIDIR\_IO symbol, and is the name of the attached OUTPUTS bus, if it is attached; otherwise, it is the name of the INPUTS bus. The names of these I/O pads can be used in a Constraints File as an alternate method of specifying I/O pad locations. Refer to the section on PPR in the *XACT Reference Guide* for more details on writing a constraints file.

## BUS\_DEF — Bus Data-Type Definition

Use the BUS\_DEF symbol to specify the data type (encoding and bounds) of a bus that is otherwise undefined. Use this symbol only for data paths that do not originate from an X-BLOX INPUTS, BIDIR\_IO, FORCE, COUNTER, SHIFT, SRAM, or PROM module, or terminate in an X-BLOX OUTPUTS module. It is used primarily to define the encoding and precision for documentation of your schematic and defining a data type for a data path in a subcircuit.

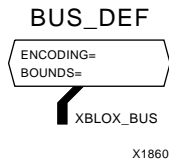


Figure 4-5 The BUS\_DEF Module Symbol

### Bus Connection

#### XBLOX\_BUS

XBLOX\_BUS is the connection from the BUS\_DEF module to the system bus that is being defined.

### Attributes

#### ENCODING

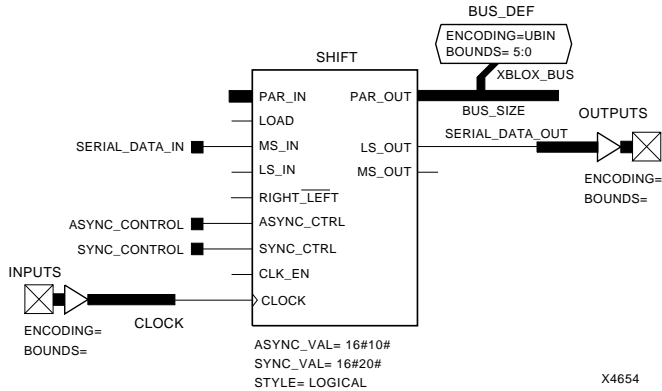
The available ENCODING options are described in the “Bus Data Types” section of the chapter “Creating an X-BLOX Design.” Any restrictions placed on the data type are dependent on the X-BLOX module(s) to which this BUS\_DEF module is connected.

#### BOUNDS

Use the BOUNDS attribute to define the width of the XBLOX\_BUS bus by specifying the MSB and LSB of the bus. You can use any pair of integers, as long as the difference of the indices, plus one, equals the width of the bus. See the “Bus Data Types” section of the chapter “Creating an X-BLOX Design” for details.

## Example

If you use a serial-in/serial-out Shift register, the Parallel-Input and Parallel-Output ports are not used. Because you must specify the width of the Shift register, a dummy bus is connected to the PAR\_IN or PAR\_OUT port, and the BUS\_DEF symbol is used to specify the ENCODING and BOUNDS attributes for the Shift register. (See the figure below.)



**Figure 4-6 Serial-in/Serial-out Shift Register Using a BUS\_DEF Module to Define Its Data Type**

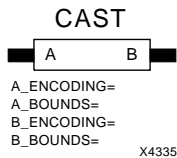
## CAST — Data Type Symbol

The CAST module is used to connect two buses that have:

- the same size but different encodings
- the same size but different bounds (ranges)
- the same size but different encodings and bounds

This feature allows for different interpretations of the same data bits in different parts of your circuit. For example, you might choose to interpret data as unsigned binary for some part of the design and as two's complement or one-hot in another part of the design.

**Warning:** No conversions or data checks are performed on the data carried by the buses.



**Figure 4-7 CAST Module**

A CAST module can connect two buses that have different bounds, as long as the number of bits in each bus is the same. Because the final routed design has only one name for each net in the design, X-BLOX creates aliases for the names of the corresponding nets that are connected by a CAST symbol.

The original names and their aliases appear in the Data Type Propagation and Signal Alias section in the .blx file once X-BLOX has run.

The CAST module has two permutable pins, A and B. As neither of these pins is an electrical driver, data can flow in either direction in the final synthesized circuit.

### Usage

Use the CAST module to connect big-endian and little-endian buses, or to change the bounds or the encoding of your buses.

The following table gives examples of how to specify the BOUNDS and ENCODING attributes for pin A and pin B depending on the task you want to do.

**Table 4-5 Specifying the BOUNDS and ENCODING Attributes**

Action	Bounds and Encoding	
	A Pin	B Pin
Change the bounds	7:0	11:4
	7:0	4:11
Connect big-endian and little-endian buses	7:0	0:7
	7:0	2:9
Change the encoding (No conversion is performed)	BIT, UBIN, TWO_COMP, or ONE-HOT	BIT, UBIN, TWO_COMP, or ONE-HOT

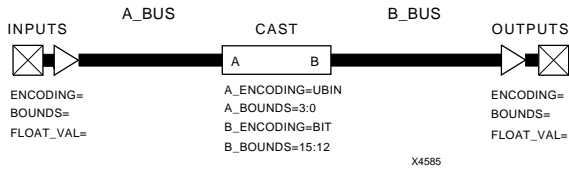
In addition, you can use the CAST module in three different ways by specifying the data types for both pins (A and B), for one pin only (A or B), or by not specifying the attributes for either of the two pins.

- Define the data types of the buses connected to both pins.

In this case you must define all the attributes (A\_ENCODING, B\_ENCODING, A\_BOUNDS, and B\_BOUNDS). Thus, the A\_ENCODING and A\_BOUNDS attributes define the encoding and bounds of the bus connected to the A input pin, and the B\_ENCODING and B\_BOUNDS attributes define the encoding and bounds of the bus connected to the B pin. The A\_BOUNDS and B\_BOUNDS must have the same number of bits.

If you define the data type of either of the two data paths connected to the A or B pin by means of a BUS\_DEF, X-BLOX I/O, or other CAST symbol, then this data type must match the corresponding pin. For example, the data type on the data path connected to the A pin must match the A\_ENCODING and A\_BOUNDS, and the data type on the data path connected to the B pin must match the B\_ENCODING and B\_BOUNDS.

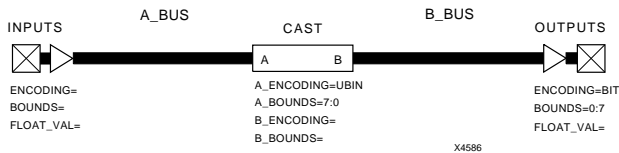
**Note:** Although it is possible to specify the bus attributes as `A_ENCODING=B_ENCODING=BIT` with both `BOUNDS` fields empty, this method makes the symbol redundant.



**Figure 4-8 Defining the CAST Data Type on Both Buses**

- Define the data type for one side of the CAST, but not the other.

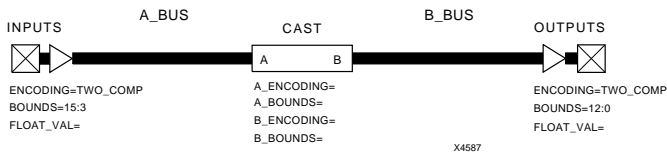
The data type on the second pin must be determined by the data path connected to that second pin. Partially-specified data types are not allowed. The size of the data path must be the same as the `A_BOUNDS` or `B_BOUNDS` defined on the CAST symbol.



**Figure 4-9 Defining the CAST Data Type on One Bus**

- Define no data types on the CAST symbol.

When you do not specify a data type, CAST acts as a firewall for data type propagation, allowing two data paths, whose data types are defined elsewhere on their data paths, to be connected even though the data types are different. The sizes of the two data paths must be the same.



**Figure 4-10 Defining No Data Type for the CAST Module**

## Inputs

### **A One pin of the CAST symbol**

If you define the A\_ENCODING and A\_BOUNDS attributes, they define the encoding and bounds for this pin. The X-BLOX bus connected to this pin must be the same size as the X-BLOX bus connected to the B pin.

### **B Other pin of the CAST symbol**

If the B\_ENCODING and B\_BOUNDS attributes are defined, they define the encoding and bounds for this pin. The X-BLOX bus connected to this pin must be the same size as the X-BLOX bus connected to the A pin.

## Attributes

### **A\_ENCODING**

This attribute represents the encoding for the A pin.

### **A\_BOUNDS**

This attribute represents the bounds for the A pin.

### **B\_ENCODING**

This attribute represents the encoding for the B pin.

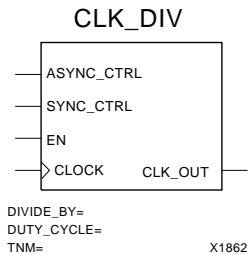
### **B\_BOUNDS**

This attribute represents the bounds for the B pin.

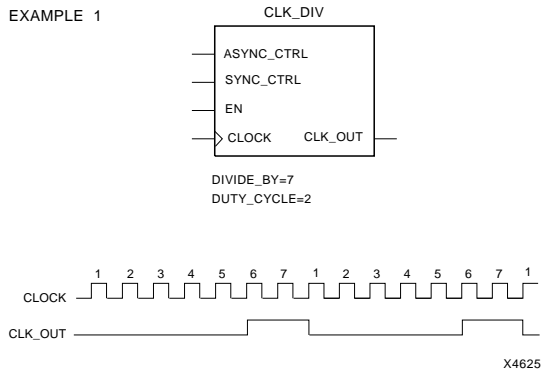


## CLK\_DIV — Clock or Frequency Divider

The CLK\_DIV 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 DIVIDE\_BY attribute. Even multiples of the Clock period produce a 50 percent duty cycle on the Clock-Output, but odd multiples produce a Low Output for one extra Clock period. You can modify the duty cycle by setting the High pulse width with the DUTY\_CYCLE attribute.



**Figure 4-11 The Clock Divider Module Symbol**



**Figure 4-12 Simple Clock Divider Example**

## Inputs

### EN

When the Clock Enable input is High, the Clock Divider is incremented on the next active Clock transition. When EN is Low, the Clock Divider is unaffected by the active Clock transition (hold). If you do not connect EN, the Clock is always enabled.

### CLOCK

The Clock input, when enabled, increments the Divider on the rising (positive) edge. An active falling (negative) edge can be used by connecting an inverter to the Clock input. You must connect the Clock.

### ASYNC\_CTRL

No ASYNC\_VAL can be defined. When the ASYNC\_CTRL input is High, the Clock Divider is reset to the beginning of the duty cycle.

### SYNC\_CTRL

No SYNC\_VAL can be defined so when the SYNC\_CTRL input is High, the Clock Divider is reset to the beginning of the duty cycle.

## Outputs

### CLK\_OUT

The Clock Output produces a pulse train that is the multiple of the input Clock period specified by the DIVIDE\_BY attribute. The Clock Output has a 50 percent duty cycle unless:

1. the divide sequence is an odd number (the Clock Output is Low for an extra Clock period)
2. the DUTY\_CYCLE attribute defines a different High pulse width

## Attributes

### DIVIDE\_BY

The DIVIDE\_BY attribute specifies the number of Clock cycles for each Output period. This value must be a positive integer. You must specify this attribute to synthesize the Clock-Divider module; if you do not specify it, an error is issued.

### DUTY\_CYCLE

The DUTY\_CYCLE attribute sets the output High for a specified number of clock cycles. This value is an integer that is less than the DIVIDE\_BY value. If DUTY\_CYCLE is not specified, a value of one-half the DIVIDE\_BY value is used (but an odd number for the DIVIDE\_BY value produces a Low Output for one extra Clock period).

### TNM

Use the TNM attribute to specify timing requirements.

You first place a TNM attribute on this module to identify the start or end point of a path using the following syntax:

**TNM=FFS: *identifier***

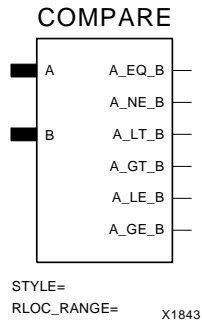
where FFS is the type of primitives to which X-BLOX propagates the TNM attribute in the XNF file and where *identifier* is a user-defined name. Refer to the section “Using XACT-Performance Attributes” in the chapter “Creating an X-BLOX Design” for more information.

You then use all the TNM attributes defined on the schematic in a TIMESPEC statement to specify delay requirements.



## COMPARE — Comparators

The Comparator module compares two input data values A and B. Both magnitude and equality comparators can be synthesized.



**Figure 4-13 The Comparator Module Symbol**

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

**Table 4-6 COMPARE — Available Comparisons**

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

### Inputs

#### A

The data on the A input port is compared to the data on the B input port. The data type of this port must be the same as the B input port.

#### B

The data on the B input port is compared to the data on the A input port. The data type of this port must be the same as the A input port.

**Note:** If the data types of the A and B buses do not match, X-BLOX issues an error message.

## Outputs

### **A\_EQ\_B**

Active High output when the data on the A input port equals the data on the B input port ( $A=B$ ).

### **A\_NE\_B**

Active High output when the data on the A input port does not equal the data on the B input port ( $A\neq B$ ).

### **A\_LT\_B**

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**

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**

Active High output when the value on the A input port is less than or equal to the value on the B input port ( $A\leq B$ ).

### **A\_GE\_B**

Active High output when the value on the A input port is greater than or equal to the value on the B input port ( $A\geq B$ ).

**Note:** If the encoding of the buses attached to the Compare module is ONE\_HOT, then only the A\_EQ\_B and the A\_NE\_B outputs can be connected.

## Attributes

### STYLE

You can choose the implementation style. However, it is recommended that you use the X-BLOX default style (that is, do not specify the STYLE attribute). The X-BLOX tools are optimized to automatically choose the fastest version possible with the existing chip resources. If the style that you have chosen does not match the intended usage, the X-BLOX software issues a Warning message, and halts.

**Note:** If the encoding of the buses attached to the Compare module is specified as two's complement, only the STYLE=ARITH is allowed.

**Table 4-7 COMPARE — Implementation Styles**

Style	Output Connections	Description
ARITH	Any or All	Uses XC4000 fast carry logic in an RPM.
TREE	Any or All	Implements magnitude and equality comparisons in a tree structure. Provides less troublesome placement on the XC4000 than the ARITH style.
RIPPLE	A=B, A≠B	Uses the fewest CLBs for equality comparisons. The results of the comparisons are rippled from MSB to LSB. (3000A/L and 3100A only)
WIRED	A=B, A≠B	Uses a wired-AND to do comparison. Uses the same number of CLBs as RIPPLE, but also uses a horizontal longline. This style is not allowed for the XC3000A/L and 3100A families.

### RLOC\_RANGE

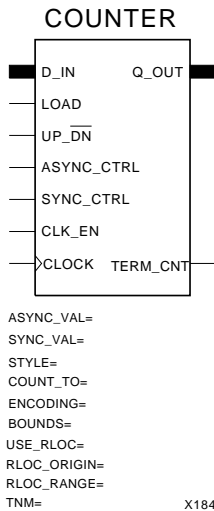
Use this attribute to specify the range (rectangular area) of FPGA locations that are allowed for the generated RPM. Refer to the chapter “X-BLOX Generated Relationally Placed Macros” for more information.





## COUNTER — Universal Counter

The Universal-Counter module generates a sequence of count values defined by the selected style of the Universal-Counter module and the status of the control inputs. The Universal-Counter module can be an up counter, down counter, or up/ $\overline{\text{down}}$  counter with predefined asynchronous or synchronous pre-load, and dynamic synchronous parallel load.



**Figure 4-14 The Counter Module Symbol**

**Table 4-8 Universal Counter Truth Table (for BINARY STYLE)**

UP/ $\overline{\text{DN}}$	LOAD	SYNC_CTRL	CLK_EN	CLOCK	ASYNC_CTRL	Q_OUT	TERM_CNT
X	X	X	X	X	H	ASYNC_VAL	L
X	X	X	L	↑	L	Hold	Hold
X	X	H	H	↑	L	SYNC_VAL	L
X	H	L	H	↑	L	D_IN	L
H	L	L	H	↑	L	<sup>a</sup> Q+1	L
H	L	L	H	↑	L	<sup>b</sup> COUNT_TO	H
L	L	L	H	↑	L	Q-1	L
L	L	L	H	↑	L	COUNT_TO	H

a. Q is the count value before the clock.

b. The COUNT\_TO value has priority over the maximum (up)-(H...H) or minimum (down)-(L...L) count values (See COUNT\_TO attribute for restrictions).

## Inputs

### D\_IN

Parallel Data from the D\_IN input port is loaded into the counter when the Parallel Load Enable is High during the active Clock transition. The data type of this port is the same as that of the Counter Output port. To use the D\_IN port, you must connect the LOAD input.

### LOAD

When this Parallel Load Enable input is High, the data on the D\_IN input port is loaded into the counter on the next active Clock transition. When the Parallel Load Enable is Low, the counter responds to the Up/ $\overline{\text{Down}}$  control input. If Parallel Load Enable is unconnected, the D\_IN port is not synthesized.

### UP\_ $\overline{\text{DN}}$

The Up/ $\overline{\text{Down}}$  control input controls the direction of the count on the next active Clock transition. When Up/ $\overline{\text{Down}}$  is High, the counter value is incremented by one; when Up/ $\overline{\text{Down}}$  is Low, the counter

value is decremented by one. If Up/ $\overline{\text{Down}}$  is unconnected, an up-counter is synthesized. The LFSR STYLE counter does not support down counting.

## **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 active Clock transition (hold). If Clock Enable is left unconnected, the Clock is always enabled.

## **CLOCK**

The Clock input, when enabled, either loads the selected data into the counter or increments/decrements the counter on the rising (positive) edge. You can use an active falling (negative) edge by connecting an inverter to the Clock input. The Clock must be connected.

## **ASYNC\_CTRL**

The Asynchronous Control input, when High, loads the value of the ASYNC\_VAL attribute into the counter independently of the Clock. If unconnected, this function is not synthesized.

## **SYNC\_CTRL**

When the Synchronous Control Enable input is High, the value of the SYNC\_VAL attribute is loaded into the counter during the next active Clock transition. The SYNC\_CTRL has priority over the LOAD input if both are High at the same time. If unconnected, this function is not synthesized.

## **Outputs**

### **Q\_OUT**

The Counter Output pin (Q\_OUT) contains the current value of the counter. The data type of this port is the same as the D\_IN input port.

## TERM\_CNT

The Terminal Count output pin goes High for one clock cycle every COUNT\_TO -1 cycles where COUNT\_TO is provided by the user or is listed in the table below.

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

For example, for a STYLE=BINARY Up Counter with no COUNT\_TO value and connected to a 4-bit UBIN bus, the Term\_Count is High when the counter reaches its maximum value of 1111.

The Terminal Count is not qualified with the Clock Enable (CE). To cascade counters, AND the Terminal Count with a common CE. Refer to Figure 4-15 at the end of this section for more information.

## Attributes

### ASYN\_VAL

The ASYN\_VAL is the predefined value that is loaded into the counter when the ASYN\_CTRL input is High. This value will also be loaded into the counter at power up, whether the ASYN\_CTRL is connected or not. If not specified, a default value of “zero” is used. ASYN\_VAL may not be used with an LFSR counter.

### SYNC\_VAL

The SYNC\_VAL is the predefined value that is loaded into the counter when the SYNC\_CTRL input is High during the active Clock transition. If not specified, a default value of “zero” is used. SYNC\_VAL may not be used with an LFSR counter.

**Note:** When ENCODING=ONE\_HOT, the binary representation for the SYNC\_VAL and ASYN\_VAL attributes must contain only a single “1” character. For example, 1000 and 0100 are both valid values. On the other hand, 1100 is not a valid value, because the “1” digit appears twice in the same data value. When STYLE=JOHNSON, the binary representation must be a valid

Johnson sequence value. Refer to the section “Counter Style Features and Selection Criteria,” following, for more information.

## STYLE

The STYLE attribute specifies the operating mode for the counter. The user can specify one of the following counter modes. If not specified, X-BLOX determines the most efficient style based on the connected pins.

**Table 4-9 COUNTER — Operating Modes**

Style	Counter Configuration	MAX Count
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.	$n$

Where  $n$  is the width of the counter.

## COUNT\_TO

The COUNT\_TO value defines the number of cycles before the counter resets to its initial value, after which the count sequence restarts. Thus, TERM\_CNT will be High for one cycle every COUNT\_TO cycles. You should specify COUNT\_TO only if the length of the count sequence is other than the MAX Count. The allowed values for COUNT\_TO vary depending on the counter style chosen.

- BINARY: The COUNT\_TO attribute values are any number between 2 and  $2^n - 1$  inclusive.
- JOHNSON: The only allowed COUNT\_TO attribute values are  $2n$  or  $2n - 1$
- LFSR: The COUNT\_TO attribute can be any number between 1 and  $2^n - 1$  inclusive, where  $n$  is the width (number of bits) of the counter.

- ONE-HOT: Not available.

When COUNT\_TO is used with a BINARY counter, the following applies. If down counting, the counter counts down to 0. On the next cycle, Q\_OUT goes to COUNT\_TO -1. If up counting, the counter counts up to COUNT\_TO -1. On the next cycle, Q\_OUT goes to 0. The behavior is the same even if the counter is loaded with a value outside the range from 0 to COUNT\_TO (with PAR\_IN, SYNC\_VAL, or ASYNC\_VAL).

## ENCODING

You can use this parameter to define the encoding of the Q\_OUT port. The available encodings are defined in the “Bus Data Types” section of the chapter “Creating an X-BLOX Design.” All the encodings on each data path must be the same.

## BOUNDS

The BOUNDS attribute defines the precision (width) of the XBLOX\_BUS by specifying the MSB and LSB of the bus. Any pair of numbers can be used, as long as the difference of the indices, plus one, equals the width of the bus. See the “Bus Data Types” section of the chapter “Creating an X-BLOX Design” for details. This parameter is optional.

**Note:** The ENCODING and BOUNDS attributes do not automatically appear on this module. Refer to your interface user guide for information on how to add these attributes. If you do not specify the bus data type on the COUNTER module, data type propagation determines the bounds and encoding for the bus ports connected to the module.

## USE\_RLOC={TRUE|FALSE}

If this attribute is set to false, the RLOCs are not generated. It is redundant to set this attribute to TRUE.

## RLOC\_ORIGIN

Position the upper left corner of the RPM at a particular FPGA location. Do not specify a location that does not allow enough room for the COUNTER. Refer to the section “Computing the Number of

CLBs” in the chapter “Understanding X-BLOX Operations” for more information.

## **RLOC\_RANGE**

Use this attribute to specify the range (rectangular area) of FPGA locations that are allowed for the generated RPM. Refer to the chapter “X-BLOX Generated Relationally Placed Macros” for more information.

## **TNM**

Use the TNM attribute to specify timing requirements.

You first place a TNM attribute on this module to identify the start or end point of a path using the following syntax:

```
TNM=FFS: identifier
```

where FFS is the type of primitives to which X-BLOX propagates the TNM attribute in the XNF file and where *identifier* is a user-defined name. Refer to the section “Using XACT-Performance Attributes” in the chapter “Creating an X-BLOX Design” for more information.

You then use all the TNM attributes defined on the schematic in a TIMESPEC statement to specify delay requirements.

## **Counter Style Features and Selection Criteria**

Each of the four counter styles has benefits and limitations that are determined by the available chip resources. Table 4-10 gives a ranking of the criteria that can be used to select the appropriate style for each application. A brief description of each style follows.

### **BINARY**

The Binary Counter produces a predictable binary output pattern and is the recommended style 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\_TO attribute or Synchronous and Parallel Load capability. The Binary Counter is synthesized as an RPM to take advantage of fast carry logic on the XC4000 family. When STYLE=BINARY, the width of the signal connected to D\_IN and

Q\_OUT can be  $\geq \log_2$  COUNT\_TO.

## JOHNSON

The Johnson Counter is the fastest style available. It produces a predictable output pattern. This style is used to produce very fast state machines and glitchless decoders. It supports Asynchronous, Synchronous, and Parallel Load, but the loaded values must correspond to the normal count sequence to maintain predictable output results. Valid values for the SYNC\_VAL and ASYNC\_VAL attributes include:

1. All zeros
2. All ones
3. Zeros followed by ones
4. Ones followed by zeros

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

```

0 0 0
1 0 0
1 1 0
1 1 1
0 1 1
0 0 1

```

One bit in the count sequence changes per clock cycle. If  $\text{COUNT\_TO} = 2n - 1$ , then 2 bits will change on one clock cycle of the sequence. When  $\text{STYLE} = \text{JOHNSON}$ , the width of the signal connected to D\_IN or Q\_OUT must be the greatest integer  $\leq ((\text{COUNT\_TO} + 1)/2)$ .

## LFSR

The LFSR counter is fast and uses chip resources efficiently. It can be configured to support any COUNT\_TO value, but the output pattern is difficult to determine. It is used for frequency division (that is, the CLK\_DIV module), modulo  $x$  counting, and pseudo-random-pattern generation. It does not support down counting. ASYNC\_CNTL or SYNC\_CNTL can be used to reset the counter, but no ASYNC\_VAL or SYNC\_VAL can be specified. The width of the LFSR counter must be between 1 and 30 bits. If more bits are needed, several LFSR counters can be cascaded. Refer to Figure 4-15 and Figure 4-16 for



examples of cascaded counters.

## ONE\_HOT

A single bit is High at any time. 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, TERM\_CNT 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 TERM\_CNT High when the high bit is the least index bit.

The ONE\_HOT style is useful for enabling a sequence of TRISTATE modules or for sequentially accessing the individual signals of a bus by driving the SEL port of the MUXBUS with the counter output.

The COUNT\_TO attribute cannot be specified for this style.

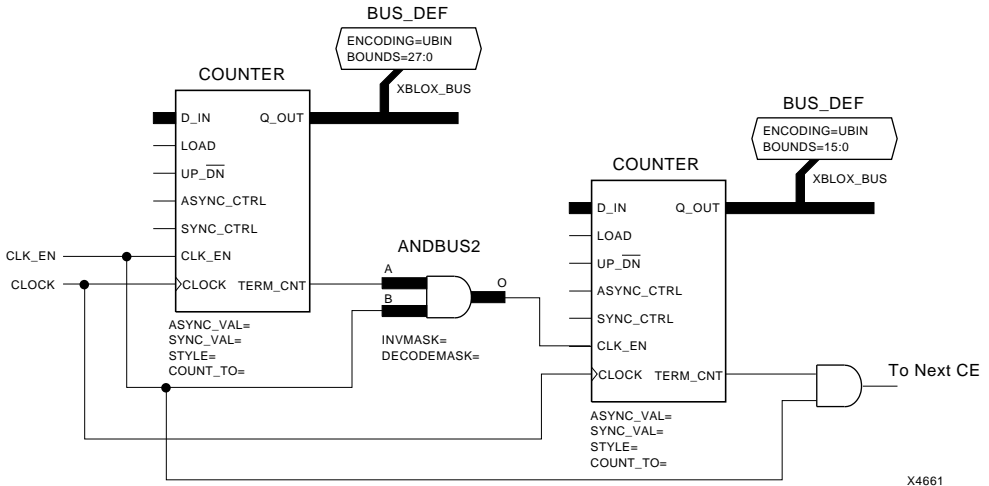
**Table 4-10 Counter Style Selection Criteria**

Counter Application Criteria	BINARY	JOHNSON	LFSR
Fastest clock rate (minimum Clock-to-Output delay)	3	1	2
Up/Down counter	1	1	—
Glitchless type output (one bit changes per clock period)	—	1 <sup>a</sup>	—
Pseudo-random-pattern generation	—	—	1
Frequency divider	3	2	1
Modulo $x$ counter, where $x$ is any number between 1 and $2^n - 1$ and where $n$ is the width of the counter	2	—	1
Easy to Place and Route (flexible layout)	3	1	2

a. Not true when COUNT\_TO=2n-1.

In the table above, “1” = Best and “—” = Not Available.

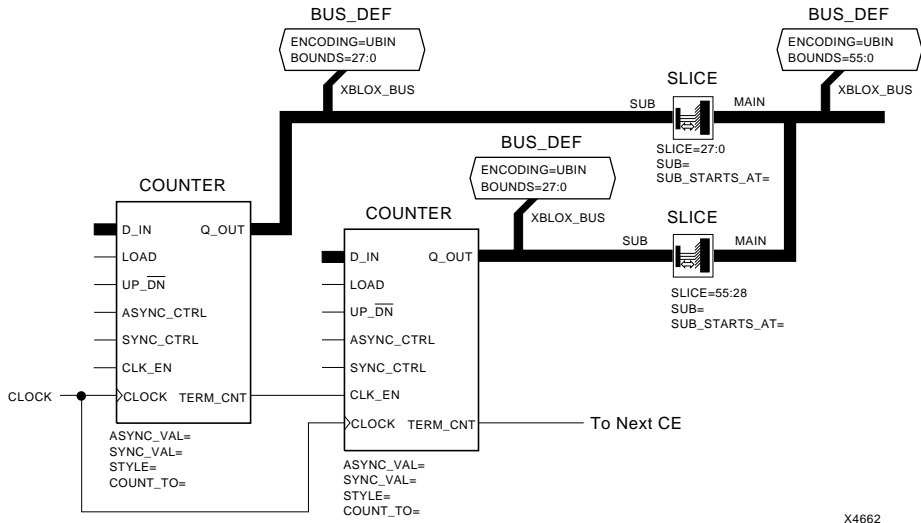
If you need to cascade X-BLOX counters to implement an LFSR counter larger than 30 bits, exercise caution if the counter module contains a Clock Enable control signal. The Terminal Count outputs of all X-BLOX counters qualify on the counter registers, not the input Clock Enable control. As a result, 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 is then used to enable the next counter in the chain.



X4661

**Figure 4-15 Cascading Counters with Clock Enable**

X-BLOX 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, as shown in Figure 4-16.



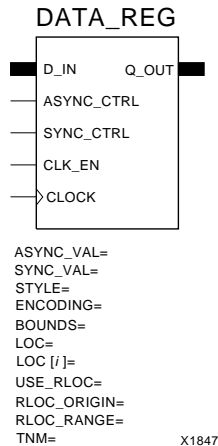
X4662

**Figure 4-16 Cascading Counters without an Initial Clock Enable**



## DATA\_REG — Data Register

The Data Register module stores the input data and passes it to the output on the next active Clock transition. The module is synthesized as an array of flip-flops that can be loaded with predefined asynchronous and synchronous data.



**Figure 4-17 The Data Register Module Symbol**

By default, the DATA\_REG module is synthesized as an ALIGNED RPM in the XC4000 family parts. This means that the registers are placed in a column with the LSB in the bottom-most CLB and the MSB in the upper-most CLB of the group. PPR will then place this column of flip-flops at an appropriate place on the die. While this structure is appropriate for bus-oriented designs, it might not be appropriate for all design structures. If you suspect PPR is having trouble placing the design, you can turn off this behavior by placing the USE\_RLOC=FALSE attribute on the symbol. If the DATA\_REG is taller than the die, the column folds to the right.

**Table 4-11 Data Register Truth Table**

D_IN	SYNC_CTRL	CLK_EN	CLOCK	ASYNC_CTRL	Q_OUT
X	X	X	X	H	ASYNC_VAL
X	X	L	↑	L	(hold)
X	H	H	↑	L	SYNC_VAL
data	L	H	↑	L	data

## Inputs

### D\_IN

Parallel Data from the D\_IN input port is loaded into the register when the Clock Enable is High during the active Clock transition. The data type of this port is the same as the Q\_OUT port.

### CLK\_EN

When the Clock Enable input is High, the D\_IN input data or the SYNC\_VAL 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). If left unconnected, the Clock is always enabled.

### CLOCK

The Clock input, when enabled, loads the selected data into the register on the rising (positive) edge. You can use an active falling (negative) edge by connecting an inverter to the Clock input. The Clock must be connected.

### ASYNC\_CTRL

The Asynchronous Control input, when High, loads the value of the ASYNC\_VAL attribute into the register independent of the Clock. This input, when High, takes priority over the Clock and the SYNC\_CTRL inputs. If unconnected, this function is not synthesized.

## SYNC\_CTRL

When the Synchronous Control Enable input is High, the value of the SYNC\_VAL attribute is loaded into the register during the next active Clock transition. The SYNC\_CTRL, when High, has priority over the parallel data inputs. If unconnected, this function is not synthesized.

## Outputs

### Q\_OUT

Q\_OUT is the output port of the Register. The data type of this port is the same as the D\_IN input port.

## Attributes

### ASYNC\_VAL

The ASYNC\_VAL is the predefined value that is loaded into the register when the ASYNC\_CTRL input is High. This value will also be loaded into the register at power up, whether the ASYNC\_CTRL is connected or not. If not specified, a default value of “zero” is used.

### SYNC\_VAL

The SYNC\_VAL is the predefined value that is loaded into the register when the SYNC\_CTRL input is High during the active Clock transition. If not specified, a default value of “zero” is used.

## ENCODING

You can use this parameter to define the encoding of the Q\_OUT port. The available encodings are defined in the “Bus Data Types” section of the chapter “Creating an X-BLOX Design.” All the encodings on each data path must be the same.

## BOUNDS

The BOUNDS attribute defines the precision (width) of the XBLOX\_BUS by specifying the MSB and LSB of the bus. Any pair of numbers can be used, as long as the difference of the indices, plus

one, equals the width of the bus. See the “Bus Data Types” section of the chapter “Creating an X-BLOX Design” for details.

**Note:** The ENCODING and BOUNDS attributes do not automatically appear on this module. Refer to your interface user guide for information on how to add these attributes. If you do not specify the bus data type on the DATA\_REG module, data type propagation determines the bounds and encoding for the bus ports connected to the module.

## LOC

The LOC attribute specifies the pin location for a single input pad-buffer pair, for example: LOC=A1, or the attribute specifies the placement of all IOBs on a specific edge or corner of the chip, for instance, LOC=TL for the top-left corner of the chip.

## LOC[i]

The LOC[i] attribute specifies the pin locations for multiple input pad-buffer pair locations, for example: LOC[2]=A10, LOC[3]=P15.

## USE\_RLOC={TRUE|FALSE}

If this attribute is set to false, the RLOCs are not generated. It is redundant to set this attribute to TRUE.

## RLOC\_ORIGIN

Position the upper left corner of the RPM at a particular FPGA location. Do not specify a location that does not allow enough room for the DATA\_REG. That is, if the DATA\_REG is connected to a bus that is 10 bits wide, then the RLOC\_ORIGIN must be at least 5 CLBs from the bottom of the die. You can always specify an RLOC\_ORIGIN at R1 (top row) of the die.

## RLOC\_RANGE

Use this attribute to specify the range (rectangular area) of FPGA locations that are allowed for the generated RPM. Refer to the chapter “X-BLOX Generated Relationally Placed Macros” for more information.



## TNM

Use the TNM attribute to specify timing requirements.

You first place a TNM attribute on this module to identify the start or end point of a path using the following syntax:

**TNM=FFS: *identifier***

or

**TNM=LATCHES: *identifier***

where FFS and LATCHES are the types of primitives to which X-BLOX propagates the TNM attribute in the XNF file and where *identifier* is a user-defined name. Refer to the section “Using XACT-Performance Attributes” in the chapter “Creating an X-BLOX Design” for more information.

You then use all the TNM attributes defined on the schematic in a TIMESPEC statement to specify delay requirements.

## STYLE Attribute

The STYLE attribute enables you to control the merging of the DATA\_REG and I/O modules.

You can specify this optional attribute on an X-BLOX DATA\_REG module to control the implementation of the module.

The following list defines all valid STYLE attributes:

- STYLE = none
- STYLE = CLB
- STYLE = IOB
- STYLE = ILD
- STYLE = IFD
- STYLE = OFD

As shown in the table below, when one of the above styles except the CLB style is specified, the DATA\_REG is implemented as specified (if possible). If the style is not specified, X-BLOX can implement the DATA\_REG in either the CLBs or the IOBs.

**Table 4-12 Implementation Styles and Optimization**

Actions	DATA_REG Implementation Styles					
	none	CLB	IOB	ILD	IFD	OFD
<b>Implement in IOB?</b>	May	No	Must	Must	Must	Must
<b>Generated Primitives</b>	FF or Latches	FF	FF or Latches	Latches	FF	FF

## Conditions for Implementation in an IOB

Depending on the implementation style used, X-BLOX can or cannot optimize the DATA\_REG module by moving the generated flip-flops (or latches) into the IOB. Such optimization improves the density and performance of the synthesized design and is possible only if the following conditions are met:

1. The DATA\_REG is connected to an I/O symbol. This can be an X-BLOX INPUTS, OUTPUTS, or BIDIR\_IO, or an IBUF or OBUF symbol.
2. The flip-flop does not use its clock-enable pin. Use of this pin prevents the flip-flop from being implemented as an IOB flip-flop. (This is because IOB flip-flops do not have a clock-enable pin.)
3. The IOBs have input or output flip-flops. The XC4000H architecture has twice as many IOBs as the XC4000 architecture, but does not have input or output flip-flops in the smaller IOBs.
4. There is no CLB External net attribute (X) on the signal between the DATA\_REG or flip-flop and the I/O symbol.
5. The SYNC\_CTRL and ASYNC\_CTRL pins are not connected to the DATA\_REG module.

In addition, the flip-flops and latches are created in the IOBs provided you have specified on the DATA\_REG module only attributes that are compatible with the particular style you specified. The table below summarizes the attribute combinations you should be aware of.

Table 4-13 Impact of Attributes on Implementation Style

Attributes	DATA_REG Implementation Styles					
	none	CLB	IOB	ILD	IFD	OFD
LOC=location or LOC[i]=location	LOC attribute is transferred to the generated IOB or CLB FF					
NODELAY	STYLE=IOB	invalid	associated with input FF or register latch	associated with input latch	associated with input FF	invalid
FAST, MEDFAST, MEDSLOW, or SLOW	STYLE=OFD	invalid	associated with input output register	invalid	invalid	associated with output FF
RLOC_ORIGIN, RLOC_RANGE, USE_RLOC=TRUE	STYLE=CLB	valid	invalid	invalid	invalid	invalid
SYNC_VAL	STYLE=CLB	valid	invalid	invalid	invalid	invalid
TNM=FFS:identifier	valid	valid	valid	invalid	valid	valid
TNM=Latches:identifier	STYLE=ILD	invalid	STYLE=ILD	valid	invalid	invalid

### none

If STYLE is not defined, X-BLOX chooses an implementation style, which might change depending on other attributes you have specified on the module.

If the command-line option `mergeio=false` is specified, X-BLOX is not allowed to perform any optimization.

Specifying the following attributes on a DATA\_REG module with no STYLE attribute changes the style in which the DATA\_REG module is implemented. Note that you can specify only one of these attributes on the module, as each of them assigns a different style to your module.

- NODELAY implements the module as if STYLE=IOB were specified.

- MEDFAST, MEDSLOW, SLOW, or FAST implements the module as if STYLE=OFD were specified. These attributes are associated with the IOB input register/latch or output register created by X-BLOX.
- RLOC\_ORIGIN, RLOC\_RANGE, or USE\_RLOC=TRUE implements the module as if STYLE=CLB were specified.
- TNM=LATCHES:*identifier* implements the module as if STYLE=ILD were specified.

If the attribute LOC=*location* or LOC[i]=*location* is found on a DATA\_REG with no STYLE attribute, the LOC attribute is transferred onto the generated IOB or CLB flip-flop. X-BLOX does not check to ensure that the value of the LOC attribute is consistent with the type of generated flip-flop. We recommend that you specify STYLE=IOB when locations are IOB locations, and STYLE=CLB when locations are internal to the FPGA.

## **STYLE=CLB**

For STYLE=CLB, X-BLOX implements a DATA\_REG module in CLBs only.

Do not use the following attributes with this implementation style:

- NODELAY
- FAST, SLOW, MEDFAST, or MEDSLOW
- TNM=LATCHES:*identifier*

## **STYLE=IOB**

For STYLE=IOB, X-BLOX implements a DATA\_REG module in either the input latches, the input flip-flops or the output flip-flops. You might wish to use this style to control which one of several DATA\_REG modules connected to an I/O symbol is implemented in IOBs when there is a chance that X-BLOX might automatically select the wrong one.

When a TNM=LATCHES:*identifier* attribute is found on a DATA\_REG with STYLE=IOB, the DATA\_REG is implemented as if STYLE=ILD were specified.

If either the NODELAY, the FAST, MEDSLOW, MEDFAST, or SLOW attribute is found on a DATA\_REG with STYLE=IOB, the attribute is associated with the IOB input register/latch or output register created by X-BLOX.

Do not use the following attributes with this implementation style:

- RLOC\_ORIGIN, RLOC\_RANGE, USE\_RLOC=TRUE
- SYNC\_VAL

**Note:** Do not connect ASYNC\_CTRL or SYNC\_CTRL when using this style.

### STYLE=ILD

For STYLE=ILD, X-BLOX implements a DATA\_REG module with input latches in IOBs.

If the NODELAY attribute is found on a DATA\_REG with STYLE=ILD, it is associated with the input latch created by X-BLOX.

Do not use the following attributes with this implementation style:

- FAST, SLOW, MEDFAST, or MEDSLOW
- RLOC\_ORIGIN, RLOC\_RANGE, or USE\_RLOC=TRUE
- SYNC\_VAL
- TNM=FFS:*identifier*

**Note:** Do not connect ASYNC\_CTRL or SYNC\_CTRL when using this style.

### STYLE=IFD

For STYLE=IFD, X-BLOX implements a DATA\_REG module with input flip-flops in IOBs.

If the NODELAY attribute is found on a DATA\_REG with STYLE=IFD, it is associated with the input flip-flop created by X-BLOX.

Do not use the following attributes with this implementation style:

- FAST, SLOW, MEDFAST, or MEDSLOW
- RLOC\_ORIGIN, RLOC\_RANGE, or USE\_RLOC=TRUE

- SYNC\_VAL
- TNM=LATCHES:*identifier*

**Note:** Do not connect ASYNC\_CTRL or SYNC\_CTRL when using this style.

## **STYLE=OFD**

For STYLE=OFD, X-BLOX implements a DATA\_REG module with output flip-flops in IOBs.

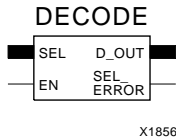
If the FAST, SLOW, MEDFAST, or MEDSLOW attribute is found on a DATA\_REG with STYLE=OFD, it is associated with the output flip-flop created by X-BLOX.

Do not use the following attributes with this implementation style:

- NODELAY
- RLOC\_ORIGIN, RLOC\_RANGE, or USE\_RLOC=TRUE
- SYNC\_VAL
- TNM=LATCHES:*identifier*

## DECODE — 1-of-n Decoder/Demultiplexer

The DECODE module converts binary-encoded two's complement, or one-hot data on the Select port to a 1-of-n output (one-hot) on the D\_OUT port. The width (precision) of the Select and D\_OUT ports is determined by their attached buses.



**Figure 4-18 The Decode Module Symbol**

### Inputs

#### SEL

The Selector is an input port with an encoding of two's complement, unsigned binary, or one-hot that is independent from the D\_OUT encoding. The possible values of the Selector should correspond to at least one of the indices of the D\_OUT port. Depending on the encoding scheme, the Select input can address more lines or fewer lines than are available on the D\_OUT port. If the defined Select input cannot address all the available Decoder outputs, a Warning message is shown when the X-BLOX synthesis software is run. Unaddressable decoder outputs will be tied to Ground. If an unavailable (out of range) D\_OUT line is chosen, the SEL\_ERROR output is High.

#### EN

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 and the SEL\_ERROR are Low. If the Enable Input is left unconnected, the Decoder is always enabled.

**Note:** You must specify the data type (encoding and bounds) for each of the buses connected to the SEL input and D\_OUT ports, either through data type propagation from other modules or with a BUS\_DEF module. The data type is not propagated between or through these two ports.

## Outputs

### **D\_OUT**

When enabled, one of the *n* lines of the D\_OUT port will be High (one-hot encoding). The D\_OUT port must have the one-hot encoding. The bounds must be specified with indices that can be used by the Select input to identify the desired D\_OUT line. If an unavailable (out of range) D\_OUT line is selected by Select, all D\_OUT lines are Low.

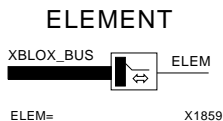
### **SEL\_ERROR**

The SEL\_ERROR output will be High when the value of the Selector input exceeds the available indices of the D\_OUT port. Please refer to the section “Out-of-Range Indicators” in the chapter “Creating an X-BLOX Design” for more information. If the Enable pin is used, it enables the SEL\_ERROR output.



## ELEMENT — Element of a Bus

The ELEMENT module extracts a single wire (net) from a bus or connects a single wire to a bus. Use the ELEM attribute with binary notation to specify the index of the particular wire to be connected to or extracted from the bus. For example, to extract the wire (net) with the index of “4” from a bus, connect the Element module between the bus and the wire, and set the ELEM attribute to “4”. The Element module does not contain any active circuitry.



**Figure 4-19 The Element Module Symbol**

### Connections

The two following ports can be connected to inputs, outputs or bidirectional ports on other modules and components. As the Element module has no active circuitry, there is no real Input or Output; these labels are just used for identification.

#### **XBLOX\_BUS**

The XBLOX\_BUS port is the bus from which you can extract the wire. When you specify one wire (net) of this bus with the ELEM attribute, the wire becomes the ELEM net or ELEM bus.

#### **ELEM**

The ELEM port is a single wire (net) extracted from the XBLOX\_BUS defined by the ELEM attribute.

### Attributes

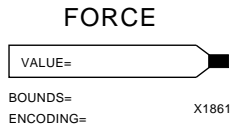
#### **ELEM**

The ELEM attribute specifies the index of the wire (net) to be extracted from or connected to the XBLOX\_BUS.



## FORCE — Force Value onto a Bus

The FORCE module forces a data value onto a bus. For example, you can use it to specify the initialization value on the parallel D\_IN input of a Counter module, or to set one input of a Comparator module to a constant value. This value will always be present on the bus, so a 3-state module is necessary to control part-time access for the Force value.



**Figure 4-20 The Force Module Symbol**

### Outputs

#### FORCE

The FORCE output port has the same data type and width as the bus to which it is attached.

### Attributes

#### ENCODING

The available encodings are described in the “Bus Data Types” section of the chapter “Creating an X-BLOX Design.” Any restrictions placed on the data type are dependent on the X-BLOX module(s) to which this module is connected.

#### BOUNDS

The BOUNDS attribute defines the precision (width) of the XBLOX\_BUS by specifying the MSB and LSB of the bus. You can use any pair of numbers, as long as the difference of the indices, plus one, equals the width of the bus. See the “Bus Data Types” section of the chapter “Creating an X-BLOX Design” for details.

**Note:** The ENCODING and BOUNDS attributes do not automatically appear on this module. Refer to your interface user guide for information on how to add these attributes. If you do not specify the bus data type on the FORCE module, data type propagation determines the bounds and encoding for the bus ports connected to the module.

## VALUE

Use the VALUE attribute to define the data value that is forced onto the bus. 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.

## INC\_DEC — Increment Decrement Symbol

The INC\_DEC symbol allows the data value on the bus to be either incremented or decremented by a constant. The data value changes according to the encoding on the bus. A control signal on the symbol controls whether the value should be incremented or decremented. If this signal is not connected, X-BLOX increments the data value on the bus. You can use an output port, OVFL, to detect overflow or underflow of the data. The INC\_DEC module requires fewer connections than an ADD\_SUB if incrementing by 1. This module is equivalent to an ADD-SUB and FORCE modules if INC\_BY is other than 1. Refer to the INC\_BY attribute below. For a description of the implementation styles available for X-BLOX register modules, refer to the section “Implementation Styles” in the chapter “Creating an X-BLOX Design.”

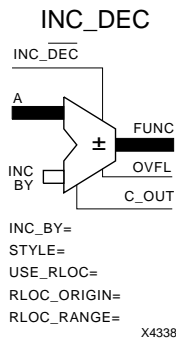


Figure 4-21 The INC\_DEC symbol

### Inputs

#### NC\_DEC

The Increment/Decrement control input determines the arithmetic operation: High = increment, and Low = decrement. If you do not connect this input, the module is synthesized as an Incrementer.

## A

The data defined by INC\_BY is added to or subtracted from the data on the A input port (A+n or A-n) depending on the value of INC\_DEC. If the INC\_BY has no assigned value, 1 is added or subtracted from the A port. The A input must be connected. The data type of this port is the same as the FUNC output port.

## Outputs

### FUNC

FUNC is the Functional output (sum or difference) port from the adder/subtractor. The data type of this port is the same as the A and B input ports.

### OVFL

OVFL is the Overflow (Underflow) output from the adder/subtractor. The OVFL output is High if an overflow occurred while adding. The OVFL output is Low if an overflow occurred while subtracting.  $OVFL=C\_OUT$  if the connected buses have  $ENCODING=UBIN$ .

### C\_OUT

C\_OUT is the Carry-output/Borrow-out port from the most significant bit(s) of the INC\_DEC module.

## Attributes

### INC\_BY

This attribute refers to the value to increment/decrement the input by. By default, it is 1. The value for this attribute should be a positive integer value. If you set another value, the INC\_DEC module is synthesized as an ADD\_SUB with one pin driven by a FORCE with  $VALUE=INC\_BY$ .

## STYLE

The default implementation style is ALIGNED for the XC4000. The default implementation style for the XC3000A/L and XC3100A is FAST3KA. The available options include the ones shown in the following table.

**Table 4-14 INC\_DEC — Implementation Styles**

Style	Description
ALIGNED	Aligned RPM (XC4000 only)
UNALIGNED	Unaligned RPM (XC4000 only)
FAST3KA	Gate-level, 1-bit look ahead, fast adder (XC3000A/L and 3100A only)
RIPPLE	Gate-level ripple carry, area-efficient adder

**Warning:** The default style for both X-BLOX and Xilinx Unified Libraries macros is ALIGNED. Exercise care when mixing ALIGNED and UNALIGNED macros. These two implementation styles are not compatible when attempting to do placement and routing with PPR.

### USE\_RLOC={TRUE|FALSE}

If this attribute is set to false, RLOCs are not generated. It is redundant to set this attribute to TRUE.

### RLOC\_ORIGIN

Position the upper left corner of the RPM at a particular FPGA location. Do not specify a location that does not allow enough room for the INC\_DEC. Refer to the section “Computing the Number of CLBs” in the chapter “Understanding X-BLOX Operations” for more information.

### RLOC\_RANGE

Use this attribute to specify the range (rectangular area) of FPGA locations that are allowed for the generated RPM. Refer to the chapter “X-BLOX Generated Relationally Placed Macros” for more information.

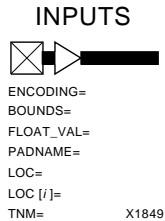




## INPUTS — Input Pads with Buffers

INPUTS is a device-input module that connects one or more input pads to an internal wire or bus. The Inputs module expands into an array of input-pad and input-buffer pairs. The input-pad locations can be defined for a given package in the following ways.

- Adding a LOC attribute on the symbol for a single-bit port
- Adding one or more LOC[i] attributes on the symbol for a multibit port
- In a Place and Route Constraints file (See the subsection on attributes below)



**Figure 4-22 The Inputs Module Symbol**

### Inputs

The inputs are automatically connected between the device input pad(s) and the buffer(s).

### Outputs

#### INPUTS

The INPUTS bus represents the buffered data or control input port(s) of the device. This port is connected to a single wire or a bus defined by the ENCODING and BOUNDS attributes.

## Attributes

### ENCODING

The available encodings are defined in the “Bus Data Types” section of the chapter “Creating an X-BLOX Design.” All the encodings on each data path must be the same.

### BOUNDS

The BOUNDS attribute defines the precision (width) of the XBLOX\_BUS by specifying the MSB and LSB of the bus. Any pair of numbers can be used, as long as the difference of the indices, plus one, equals the width of the bus. See the “Bus Data Types” section of the chapter “Creating an X-BLOX Design” for details. This parameter is optional.

### FLOAT\_VAL

The FLOAT\_VAL attribute is used to connect pull-up or pull-down resistors to the input pads defined by this module. You can specify this value in any of the allowed radices using don't care digits. The bits of the FLOAT\_VAL number specify which Input pads are connected to pull-up resistors, pull-down resistors or neither (that is, 1 = pull-up resistor, 0 = pull-down resistor, ? = none). Alternately, all the pads of a symbol can be tied to pull-up or pull-down resistors by specifying FLOAT\_VAL=PULLUP or FLOAT\_VAL=PULLDOWN, respectively. See the “Pull-up and Pull-down Resistors for I/O Pads” section in the chapter “Creating an X-BLOX Design” for examples.

### PADNAME

Use this attribute to specify the base name of the I/O pad. With this attribute set to PADNAME=foo, the names of the pads will be SYM/FOO<0>, SYM/FOO<1>, ..., SYM/FOO<n>, corresponding to the BOUNDS associated with the attached buses and the BOUNDS attribute. Without this attribute, the names of the I/O pads will be SYM/PAD<0>, SYM/PAD<1>, ..., SYM/PAD<n>, where SYM is the name of the INPUTS symbol.

## LOC

The LOC attribute specifies the pin location for a single input pad-buffer pair, for example: LOC=A1, or the attribute specifies the placement of all IOBs on a specific edge or corner of the chip, for instance, LOC=TL for the top-left corner of the chip.

## LOC[i]

The LOC[i] attribute specifies the pin locations for multiple input pad-buffer pair locations, for example: LOC[2]=A10, LOC[3]=P15.

## TNM

Use the TNM attribute to specify timing requirements.

You first place a TNM attribute on this module to identify the start or end point of a path using the following syntax:

**TNM=PADS:*identifier***

where PADS is the type of primitives to which X-BLOX propagates the TNM attribute in the XNF file and where *identifier* is a user-defined name. Refer to the section “Using XACT-Performance Attributes” in the chapter “Creating an X-BLOX Design” for more information.

You then use all the TNM attributes defined on the schematic in a TIMESPEC statement to specify delay requirements.

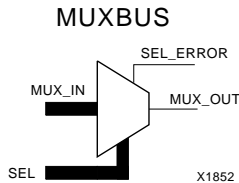
## Constraints File

The INPUTS module is synthesized into an array of input pads with names of the form: SYM/SIG<0>, SYM/SIG<1> . . . SYM/SIG<N-1>. SYM is the schematic-editor label or instance for the INPUTS symbol, and SIG is the name of the attached X-BLOX bus. However, if the PADNAME attribute has been specified, the input pads have the base name given by this attribute. The names of these input pads can be used in a Constraints File as an alternate method of specifying input-pad locations.



## MUXBUS — General n-to-1 Bus Multiplexer

The MUXBUS module routes one of n Inputs to the Output under the control of the Select input port, where n is determined by the width of the input port.



**Figure 4-23 The n-to-1 Bus-Multiplexer Module Symbol**

### Inputs

#### MUX\_IN

MUX\_IN is the Input-Bus data port of the Multiplexer. You can use any data type for the Input-Bus, but you must specify the BOUNDS attribute with indices that can be used by the Select input to identify the desired Multiplexer Input-Bus line.

#### SEL

The Select input port chooses which Input-Bus line is directed to the Multiplexer-Output line. The data type of this port can be a data type independent from the Multiplexer Input-Bus data type. However, the possible values of the Selector must correspond to at least one of the indices of the Input-Bus port. Depending on the encoding scheme, the Select input can address more lines or fewer lines than are available on the Input-Bus port. If the defined Select input cannot address all the available Multiplexer Inputs, a Warning message is shown when the X-BLOX synthesis software is run. If an unavailable (out of range) Input-Bus line is chosen, the SEL\_ERROR output is High.

**Note:** You must specify the data type (encoding and bounds) for each of the buses connected to the SEL input and MUX\_IN ports either through data type propagation from other modules or with a

BUS\_DEF module. The data type does not propagate between or through these two ports.

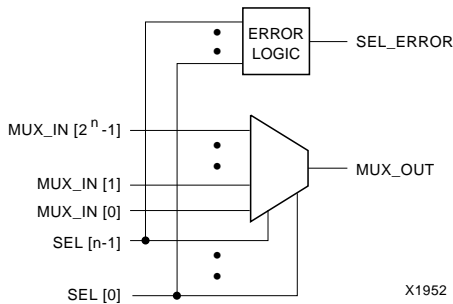
## Outputs

### MUX\_OUT

The Multiplexer Output line reflects the selected Input-Bus data. The data type is BIT without an index. If the selected Input-Bus line is out-of-range, the Multiplexer Output is Low.

### SEL\_ERROR

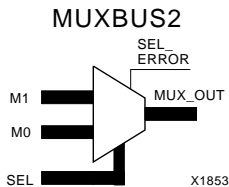
The SEL\_ERROR Output is High when the selected Input-Bus line does not exist; that is, the Selector value is out of the range of the available input indices. Please refer to the section “Out-of-Range Indicators” in the chapter “Creating an X-BLOX Design” for more information.



**Figure 4-24 The n-to-1 Bus-Multiplexer Logic Diagram**

## MUXBUS2 — 2-to-1 Bus Multiplexer

The MUXBUS2 module routes one of two Input Buses to the Output Bus under the control of the Select input port.



**Figure 4-25 The 2-to-1 Bus-Multiplexer Module Symbol**

### Inputs

#### M0, M1

M0 and M1 are the Input-Bus data ports of the Multiplexer. The Input and Output buses must have the same encoding and bounds. The bus width can be any size, as long as the module fits in the device being targeted.

M0 is selected when SEL input evaluates to zero. M1 is selected when SEL input evaluates to +1. For all other values of SEL input, MUX\_OUT bus is Low and SEL\_ERROR is High.

#### SEL

The Select input port chooses which of the two Input-Bus ports is directed to the Output-Bus port. The Select input port will be either one or two bits wide, depending on the Select encoding method. The data type of this port can be a data type independent from the Multiplexer Input-Bus data type. Depending on the encoding scheme, the Select input can address more or fewer ports than are available. If the defined Select input cannot address all the available Multiplexer Inputs, a Warning message is shown when the X-BLOX synthesis software is run. If you choose an unavailable (out of range) port, the SEL\_ERROR output is High.

## Outputs

### MUX\_OUT

The Multiplexer Output-Bus port reflects the selected Input-Bus port data. The Output-Bus port has the same ENCODING and BOUNDS as the Input-Bus ports. If the selected input port is unavailable (out-of-range), the Multiplexer Outputs are Low.

### SEL\_ERROR

The SEL\_ERROR Output is High when the selected Input-Bus port does not exist; that is, the Selector value is outside the range of the available input indices. Please refer to the section “Out-of-Range Indicators” in the chapter “Creating an X-BLOX Design” for more information.

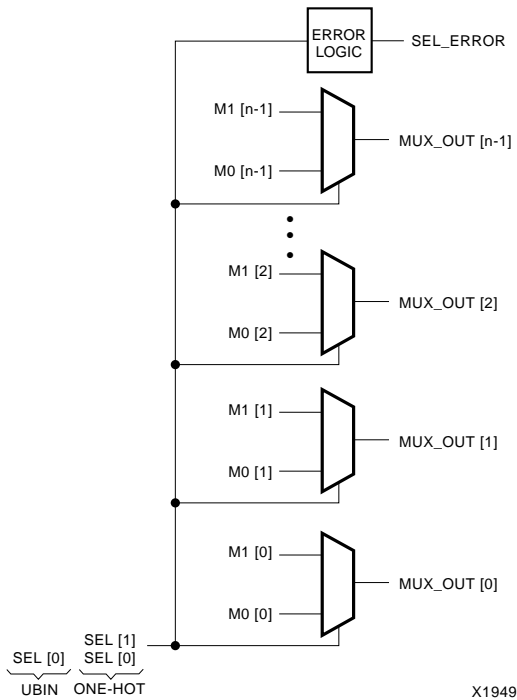
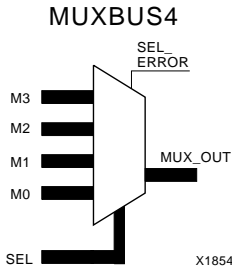


Figure 4-26 The 2-to-1 Bus-Multiplexer Logic Diagram



## MUXBUS4 — 4-to-1 Bus Multiplexer

The MUXBUS4 module routes one of four 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 4-27 The 4-to-1 Bus-Multiplexer Module Symbol**

### Inputs

#### M0 to M3

M0, M1, M2 and M3 are the Input-Bus data ports of the Multiplexer. The Input and Output buses must have the same ENCODING and BOUNDS. The bus width can be of any size, as long as the module fits in the device being targeted.

#### SEL

The Select input port chooses which of the four Input-Bus ports is directed to the Output-Bus port. The Select input port will be either two, three or four bits wide, depending on the Select encoding method. The data type of this port can be a data type independent from the Multiplexer Input-Bus data type. Depending on the encoding scheme, the Select input can address more or fewer ports than are available. If the defined Select input data type cannot address all the available Input-Bus ports, a Warning message is shown when the X-BLOX synthesis software is run. If an unavailable (out-of-range) Input-Bus port is chosen, the SEL\_ERROR output is High.

## Outputs

### MUX\_OUT

The Multiplexer Output-Bus port reflects the selected Input-Bus port data. The Output-Bus port has the same ENCODING and BOUNDS as the Input-Bus ports. If the selected input port is unavailable (out-of-range), the Multiplexer Outputs are Low.

### SEL\_ERROR

The SEL\_ERROR Output is High when the selected Input-Bus port does not exist; that is, the Selector value is outside the range of the available input indices. Please refer to the section “Out-of-Range Indicators” in the chapter “Creating an X-BLOX Design” for more information.

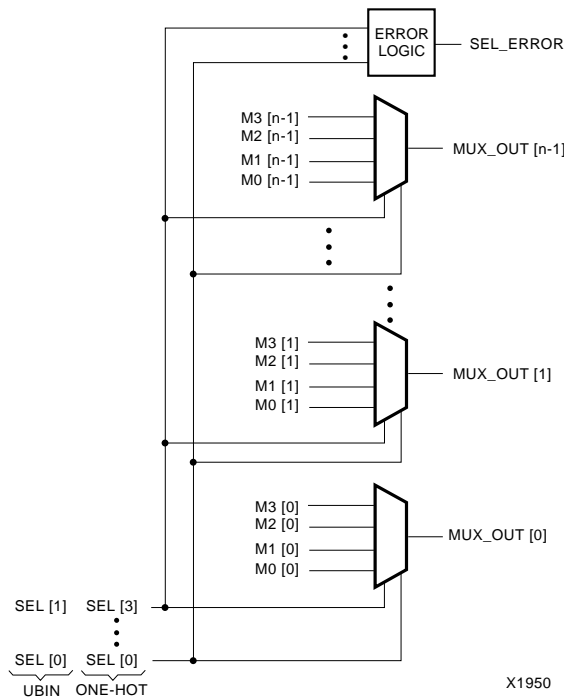


Figure 4-28 The 4-to-1 Bus-Multiplexer Logic Diagram

## MUXBUS8 — 8-to-1 Bus Multiplexer

The MUXBUS8 module routes one of eight Input Buses to the Output Bus under the control of the Select input port.

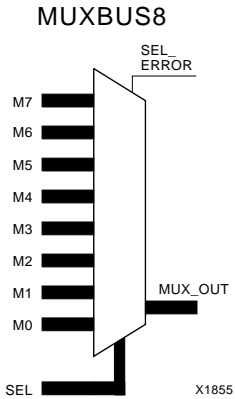


Figure 4-29 The 8-to-1 Bus-Multiplexer Module Symbol

### Inputs

#### M0 to M7

M0, M1, . . . to . . . M7 are the eight Input-Bus data ports of the Multiplexer. The Input and Output buses must have the same ENCODING and BOUNDS. The bus width can be any size, as long as the module fits in the device being targeted.

#### SEL

The Select input port chooses which of the eight Input-Bus ports is directed to the Output-Bus port. The Select input port will be from three to eight bits wide, depending on the Select encoding method. The data type of this port can be a data type independent from the Multiplexer Input-Bus data type. Depending on the encoding scheme, the Select input can address more ports or fewer ports than are available. If the defined Select input cannot address all the available Input-Bus ports, a Warning message is shown when the X-BLOX synthesis software is run. If an unavailable (out of range) Input-Bus port is chosen, the SEL\_ERROR output is High.

## Outputs

### MUX\_OUT

The Multiplexer Output-Bus port reflects the selected Input-Bus port data. The Output-Bus port has the same ENCODING and BOUNDS as the Input-Bus ports. If the selected input port is unavailable (out-of-range), the Multiplexer Outputs are Low.

### SEL\_ERROR

The SEL\_ERROR Output is High when the selected Input-Bus port does not exist; that is, the Selector value is outside the range of the available input indices. Please refer to the section “Out-of-Range Indicators” in the chapter “Creating an X-BLOX Design” for more information.

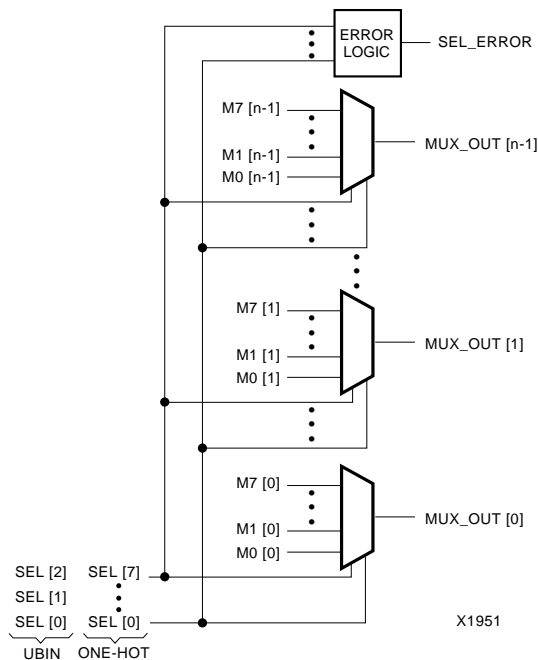
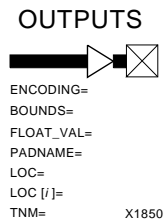


Figure 4-30 The 8-to-1 Bus-Multiplexer Logic Diagram

## OUTPUTS — Output Pads with Buffers

OUTPUTS is a device-output module that connects an internal wire or bus to one or more output pads. The Outputs module expands into an array of output-pad and output-buffer pairs. The output pad locations can be fixed for a given package in the following ways.

- Adding a LOC attribute on the symbol for a single-bit port
- Adding one or more LOC[i] attributes on the symbol for a multibit port
- In a Place and Route Constraints file (see the section on attributes below)



**Figure 4-31 The Outputs Module Symbol**

### Inputs

#### OUTPUTS

The OUTPUTS port represents buffered data or control output port(s) of the device. This port is connected to a single internal wire or a bus defined by the ENCODING and BOUNDS attributes.

### Outputs

The outputs are automatically connected between the device output pad(s) and the buffer(s).

## Attributes

### ENCODING

The available encodings are defined in the “Bus Data Types” section of the chapter “Creating an X-BLOX Design.” All the encodings on each data path must be the same.

### BOUNDS

The BOUNDS attribute defines the precision (width) of the XBLOX\_BUS by specifying the MSB and LSB of the bus. Any pair of numbers can be used, as long as the difference of the indices, plus one, equals the width of the bus. See the “Bus Data Types” section of the chapter “Creating an X-BLOX Design” for details. This parameter is optional.

### FLOAT\_VAL

The FLOAT\_VAL attribute is used to connect pull-up or pull-down resistors to the output pads defined by this module. This value can be specified in any of the allowed radices, and can contain don't care digits. The bits of the FLOAT\_VAL number specify which Output pads are connected to pull-up resistors, pull-down resistors or neither (that is, 1 = pull-up resistor, 0 = pull-down resistor, ? = none). Alternately, all the pads of a symbol can be tied to pull-up or pull-down resistors by specifying FLOAT\_VAL=PULLUP or FLOAT\_VAL=PULLDOWN respectively. See the “Pull-up and Pull-down Resistors for I/O Pads” section for examples.

### PADNAME

Use this attribute to specify the base name of the I/O pad. With this attribute set to PADNAME=foo, the names of the pads will be SYM/FOO<0>, SYM/FOO<1>, ..., SYM/FOO<n>, corresponding to the BOUNDS associated with the attached buses and the BOUNDS attribute. Without this attribute, the names of the I/O pads will be SYM/PAD<0>, SYM/PAD<1>, ..., SYM/PAD<n>, where SYM is the name of the OUTPUTS symbol.

## LOC

The LOC attribute specifies the pin location for a single output pad-buffer pair, for example: LOC=A1, or the attribute specifies the placement of all IOBs on a specific edge or corner of the chip, for instance, LOC=TL for the top-left corner of the chip.

## LOC[i]

The LOC[i] attribute specifies the pin locations for multiple output pad-buffer pair locations, for example: LOC[4]=B10, LOC[5]=P12.

## TNM

Use the TNM attribute to specify timing requirements.

You first place a TNM attribute on this module to identify the start or end point of a path using the following syntax:

**TNM=PADS:*identifier***

where PADS is the type of primitives to which X-BLOX propagates the TNM attribute in the XNF file and where *identifier* is a user-defined name. Refer to the section “Using XACT-Performance Attributes” in the chapter “Creating an X-BLOX Design” for more information.

You then use all the TNM attributes defined on the schematic in a TIMESPEC statement to specify delay requirements.

## Constraints File

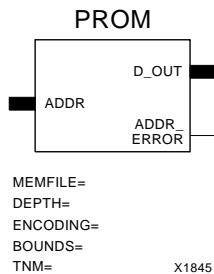
The OUTPUTS module is synthesized into an array of output pads with names of the form: SYM/SIG<0>, SYM/SIG<1> . . . SYM/SIG<N-1>. SYM is the schematic-editor label or instance for the OUTPUTS symbol, and SIG is the name of the attached X-BLOX bus. However, if the PADNAME attribute has been specified, the input pads have the basename given by this attribute. The names of these output pads can be used in a Constraints File as an alternate method of specifying output-pad locations.





## PROM — Programmable Read-Only Memories

The PROM module synthesizes a Programmable Read-Only Memory. The contents of the PROM can be specified either by means of a separate text file or on the symbol. The PROM can be of any size that will fit on a chip. The width of the PROM is determined from the BOUNDS definition of the D\_OUT port. The depth is specified by the DEPTH attribute on the PROM module or the data type of the Address signal. The data type of the Address signal determines maximum and minimum addressable locations. Only the addressable locations that are within the valid DEPTH will be synthesized. The word-locations in the PROM are indexed based on the data type of the signal attached to the ADDR port.



**Figure 4-32 The PROM Module Symbol**

### Inputs

#### ADDR

The Address port selects the word that appears on the Data Output port. The Address port will determine the maximum depth of the PROM if the DEPTH attribute is not specified. The Address port cannot be connected to a bus with ENCODING=ONE\_HOT.

### Outputs

#### D\_OUT

The Data Output port reflects the addressed word of the PROM. The width of the D\_OUT port is set by the BOUNDS of the signal attached

to this port. If an out-of-bounds word is addressed, the D\_OUT port is undefined and the ADDR\_ERROR is High.

## **ADDR\_ERROR**

The Address Error (out-of-bounds) output is High if the value on the Address port is beyond the addressable locations for the PROM. Please refer to the section “Out-of-Range Indicators” in the chapter “Creating an X-BLOX Design” for more information.

## **Attributes**

### **MEMFILE**

The MEMFILE attribute specifies the name of a Memory Definition File that defines the contents of this PROM. The name of the Memory Definition File must have a .mem file extension. The extension can be omitted from the MEMFILE attribute. If the Memory Definition File exists, it will be read; otherwise, a template file with the given name will be created by X-BLOX. The template file created by X-BLOX does not contain valid data and has to be edited before you can re-execute X-BLOX.

### **DEPTH**

The DEPTH attribute defines the depth, in words, of the PROM. Each word is "width" bits wide. The depth is specified in decimal notation, unless a radix definition precedes it. If the DEPTH attribute is not given on the symbol in the MEMFILE, it will be taken from the DEPTH attribute on the PROM symbol, or calculated from the BOUNDS and ENCODING on the ADDR input to the PROM.

### **ENCODING**

You can use this parameter to define the encoding of the D\_OUT port. The available encodings are defined in the “Bus Data Types” section of the chapter “Creating an X-BLOX Design.” All the encodings on each data path must be the same.

## BOUNDS

The BOUNDS attribute defines the precision (width) of the XBLOX\_BUS by specifying the MSB and LSB of the bus. Any pair of numbers can be used, as long as the difference of the indices, plus one, equals the width of the bus. See the “Bounds” section of the chapter “Creating an X-BLOX Design” for details. This parameter is optional.

**Note:** The ENCODING and BOUNDS attributes do not automatically appear on this module. Refer to your interface user guide for information on how to add these attributes. If you do not specify the bus data type on the PROM module, data type propagation determines the bounds and encoding for the bus ports connected to the module.

## TNM

Use the TNM attribute to specify timing requirements.

You first place a TNM attribute on this module to identify the start or end point of a path using the following syntax:

```
TNM=RAMS: identifier
```

where RAMS is the type of primitives to which X-BLOX propagates the TNM attribute in the XNF file and where *identifier* is a user-defined name. Refer to the section “Using XACT-Performance Attributes” in the chapter “Creating an X-BLOX Design” for more information.

You then use all the TNM attributes defined on the schematic in a TIMESPEC statement to specify delay requirements.

## MEMFILE Syntax

A MEMFILE consists of two parts — the Header, which describes characteristics of the PROM, and the Data portion, which defines the contents of the PROM. The Data portion of the file begins as soon as the DATA keyword is encountered. The MEMFILE is not case-sensitive, except for ASCII data. X-BLOX will also read a MEMFILE used by the MEMGEN program.

## MEMFILE Header

The Header defines characteristics of the PROM, 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.

- TYPE (optional)

The Type definition defines the type of memory to be built. The valid types that you can specify are ROM or PROM:

**type ROM | PROM**

- DEPTH

The DEPTH attribute defines the depth, in words, of the PROM. Each word is "width" bits wide. The Depth is specified in decimal notation, unless a radix definition precedes it. If the Depth is not given in the MEMFILE, it will be taken from the DEPTH attribute on the PROM symbol, or calculated from the BOUNDS and ENCODING on the ADDR input to the PROM.

**depth *memory\_depth***

- WIDTH

The Width definition defines the width of the PROM, 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. If the Width is not specified in the MEMFILE, it will be taken from the WIDTH attribute on the PROM symbol (if present) and the BOUNDS of the bus attached to the D\_OUT pin.

**width *memory\_width***

- SYMBOL

The "symbol" keyword is obsolete and will be ignored. No schematic symbols are generated by X-BLOX. The PROM symbol serves as a schematic symbol for all sizes of PROMs.

- DEFAULT

The Default Definition defines the value of all PROM locations that are not specified in the MEMFILE Data section. If no default value is specified, all unspecified locations are zero. The default

definition uses the current radix, which is 10, unless a radix definition was found.

- **RADIX**

You can specify numbers in X-BLOX with or without a radix (base). Refer to the section, “Data Values,” of Chapter 1 for more information. 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 the line generates a blank comment as it does not affect the next line of text.

**;** *commentstring*

## Example

The following example illustrates the syntactical concepts defined above:

```
; Defines the default PROM contents =1010=10
Default 10
Radix 16; Defines default radix as 1610
Depth 10; Defines the depth=1016=16
Radix 10; Re-defines the default Radix=1016=16
Width 12; Defines the width=1216=18
```

## MEMFILE Data Section

The data values specified in the MEMFILE Data Section define the contents of the PROM. Data values are specified sequentially, beginning with the lowest address in the PROM, as defined. The address of a data value may be specified. The default radix of the data values is 16. If Radix Definitions were 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 and/or white space.

## Addressing

An address is specified as follows.

### ***address :***

For example, the following definition defines an 8-word PROM with the contents (starting at address 0) 6,4, 5,5,2,7,5,3. Note that the contents of locations 2, 3, and 6 were defined via the Default Definition. Two starting addresses (4 and 7) are given.

```
depth 8
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.

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 "I" characters.

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

## PROM Definition Procedure

The following sequence should be used when creating a PROM in the X-BLOX environment:

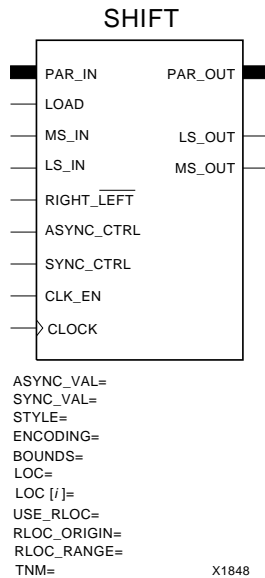
1. Use a text editor to create and edit a Memory Definition File for each different PROM in your design and to verify the proper PROM data. Use the X-BLOX Data Values format, except the default radix, which is hexadecimal for values after the DATA keyword.
2. Insert the X-BLOX PROM symbol into the design using the schematic editor.
3. Use the MEMFILE attribute to specify the name of the Memory Definition File. This file will contain the data for the PROM. If the Memory Definition File does not exist, this name is used by the X-BLOX software to create a shell Memory Definition File when the X-BLOX synthesis program is run. After X-BLOX creates a shell file, X-BLOX halts so that you can return to step 1 to insert the desired values into the file.
4. Run the X-BLOX synthesis software on the XNF file. The PROM module is expanded and a shell created with the pointers set up to the Memory Definition File containing the PROM data.





## SHIFT — Universal Shift Register

The Universal Shift-Register module is a multifunction shift register with predefined asynchronous or synchronous pre-load, and dynamic synchronous parallel load.



**Figure 4-33 The SHIFT Module Symbol**

The Register can be synthesized in any combination of the configurations listed below. Please refer to the figures shown at the end of the descriptive section for this module for configuration examples.

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

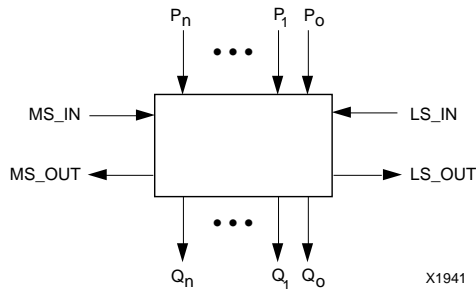
The shifting style of the register can also be selected. The available shifting styles include:

- Logical shift register
- Circular shift register
- Arithmetic shift register

By default, the SHIFT module is synthesized as an ALIGNED RPM in XC4000 family parts if the PAR\_OUT pin is connected. This means that the registers will be placed in a column with the LSB in the bottom-most CLB and the MSB in the upper-most CLB of the group. PPR will then place this column of flip-flops at an appropriate place on the die. This may not be desired if PPR is having trouble placing the design. This behavior may be turned off by placing the USE\_RLOC=FALSE attribute on the symbol. If this behavior is desired, and the PAR\_OUT pin is not connected, then place the USE\_RLOC=TRUE attribute. If the SHIFT is taller than the die, the column will fold to the right.

**Table 4-15 Universal Shift Register Truth Table (for Logical Style)**

RT/ LFT	LOAD	SYNC_ CTL	CLK_ EN	CLOCK	ASYNC_ CTL	PAR_OUT	MS_OUT	LS_OUT
X	X	X	X	X	H	ASYNC_VAL	ASYNC_VAL MSB	ASYNC_VAL LSB
X	X	X	L	↑	L	Q <sub>prev</sub>	Q <sub>prev</sub>	Q <sub>prev</sub>
X	X	H	H	↑	L	SYNC_VAL	SYNC_VAL MSB	SYNC_VAL LSB
X	H	L	H	↑	L	PAR_IN	PAR_IN MSB	PAR_IN LSB
H	L	L	H	↑	L	Q <sub>prev</sub> /2	MS-IN	LSB+1
L	L	L	H	↑	L	Q <sub>prev</sub> x2+ LS-IN	MSB-1	LS-IN



**Figure 4-34 The Shift Register Data Flow Diagram**

## Inputs

### PAR\_IN

Parallel Data from the PAR\_IN input port is loaded into the register when the Parallel Load Enable is High during the active Clock transition. When the PAR\_IN pin is connected, the LOAD pin must always be connected. When the PAR\_IN pin is not connected, the LOAD pin must be grounded or not connected.

If neither this PAR\_IN port nor the PAR\_OUT port are used for parallel data transfers, the encoding and bounds must be defined on this module or on a BUS\_DEF module connected to the PAR\_IN or PAR\_OUT ports.

### LOAD

When the Parallel LOAD Enable input is High, the Parallel Data is loaded into the Shift Register on the next active Clock transition. When LOAD is Low, the Shift Register will respond to the Right/Left control inputs. If LOAD is unconnected, the parallel data entry is not synthesized.

**Note:** If PAR\_IN is connected, LOAD must also be connected; otherwise, X-BLOX reports an error.

## MS\_IN

The MSB serial-data input port provides the input for a right-shifting MSB-to-LSB shift register, or converts serial data to parallel data when the PAR\_OUT output port is used. If unconnected, this port is set Low (0) for logical shifts, set to the MSB for arithmetic shifts, or set to the LSB for circular shifts. If the CIRCULAR STYLE of Shift register is used, this MS\_IN input cannot be connected.

**Note:** If both LS\_IN and MS\_IN are connected, RIGHT\_LEFT must also be connected or the MS\_IN will not participate in the right shift.

## LS\_IN

The LSB serial-data input port provides the input for a left-shifting LSB-to-MSB shift register, or converts serial data to parallel data when the PAR\_OUT output port is used. If unconnected, this port is set to Low (0) for logical or arithmetic shifts, or set to the MSB for circular shifts. If the CIRCULAR STYLE of Shift register is used, this LS\_IN input cannot be connected.

## 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 will reverse the active High/Low definition, but will not change the MSB/LSB definitions nor the shift direction. If left unconnected, the direction is determined by the connections on LS\_IN and MS\_IN. If only LS\_IN is connected, the default direction is Left Shift. If only MS\_IN is connected, or neither LS\_IN or MS\_IN is connected, the default is a Right Shift.

## CLK\_EN

When the Clock Enable input is High, the load and shift actions take place on the active Clock transition. When Clock Enable is Low, the register contents are unaffected by the active Clock transition (hold). If Clock Enable is left unconnected, the Clock is always enabled.

## **CLOCK**

The Clock input, when enabled, either loads the selected data into the register or performs a shift on the rising (positive) edge. An active falling (negative) edge can be used by connecting an inverter to the Clock input. The Clock must be connected.

## **ASYNC\_CTRL**

The Asynchronous Control input, when High, loads the value of the ASYNC\_VAL attribute into the shift register independently from the Clock. If unconnected, this function is not synthesized.

## **SYNC\_CTRL**

When the Synchronous Control Enable input is High, the value of the SYNC\_VAL attribute will be loaded into the shift register during the next active Clock transition. The SYNC\_CTRL has priority over the LOAD input if both are High at the same time. If unconnected, this function is not synthesized.

## **Outputs**

### **PAR\_OUT**

The Parallel Data output port contains the current value of the register. The data type of the PAR\_OUT port is the same as the PAR\_IN port. If unconnected, then one of the MS\_OUT or LS\_OUT outputs must be used. If this port is connected, X-BLOX synthesizes the Shift module as an RPM.

### **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.

### **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.

**Warning:** At least one of the PAR\_OUT, MS\_OUT or LS\_OUT output ports must be connected.

## Attributes

### ASYNC\_VAL

The ASYNC\_VAL is the predefined value that is loaded into the shift register when the ASYNC\_CTRL input is High. This value will also be loaded into the register at power up, whether the ASYNC\_CTRL is connected or not. If not specified, a default value of “zero” is used.

### SYNC\_VAL

The SYNC\_VAL is the predefined value that is loaded into the shift register, when the SYNC\_CTRL input is High during the active Clock transition. If not specified, a default value of “zero” is used.

### STYLE

The shifting style can be chosen from the following table. If no style is specified, the default shifting style is LOGICAL.

**Table 4-16 SHIFT — Shifting Styles**

STYLE	Description
LOGICAL	<ul style="list-style-type: none"> <li>•Shifts in a '0' at the MSB during a right-shift when MS_IN is not connected.</li> <li>•Shifts in a '0' at the LSB during a left-shift when LS_IN is not connected.</li> </ul>
CIRCULAR	MS_IN and LS_IN are not allowed.
ARITH	The MSB is the sign bit. X-BLOX issues a warning if MS_IN is used with a possible right-shift, because the sign bit could change.

### ENCODING

You can use this parameter to define the encoding of the Q\_OUT port. The available encodings are defined in the “Bus Data Types” section of the chapter “Creating an X-BLOX Design.” All the encodings on each data path must be the same.

## BOUNDS

The BOUNDS attribute defines the precision (width) of the XBLOX\_BUS by specifying the MSB and LSB of the bus. Any pair of numbers can be used, as long as the difference of the indices, plus one, equals the width of the bus. See the “Bus Data Types” section of the chapter “Creating an X-BLOX Design” for details. This parameter is optional.

**Note:** The ENCODING and BOUNDS attributes do not automatically appear on this module. Refer to your interface user guide for information on how to add these attributes. If you do not specify the bus data type on the SHIFT module, data type propagation determines the bounds and encoding for the bus ports connected to the module.

## LOC

The LOC attribute specifies the pin location for a single input pad-buffer pair, for example: LOC=A1, or the attribute specifies the placement of all IOBs on a specific edge or corner of the chip, for instance, LOC=TL for the top-left corner of the chip.

## LOC[i]

The LOC[i] attribute specifies the pin locations for multiple input pad-buffer pair locations, for example: LOC[2]=A10, LOC[3]=P15.

## USE\_RLOC={TRUE|FALSE}

If this attribute is set to false, RLOCs are not generated. If no PAR\_OUT pin is connected, but RLOCs are desired, set this attribute to TRUE. RLOCs are produced by default if the PAR\_OUT pin is connected.

## RLOC\_ORIGIN

Position the MSB at a particular FPGA location. Do not specify a location that does not allow enough room for the SHIFT register. That is, if the SHIFT register is connected to a bus that is 10 bits wide, the RLOC\_ORIGIN must be at least 5 CLBs from the bottom of the die. You can always specify an RLOC\_ORIGIN, at R1 (top row) of the die.





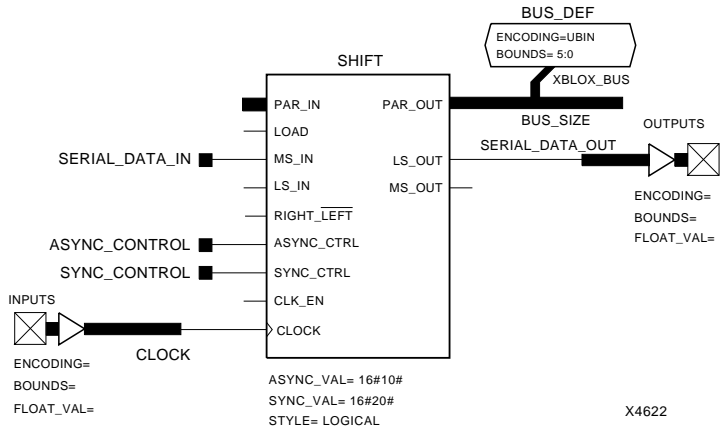


Figure 4-36 Typical Serial-in/Serial-out Shift Register

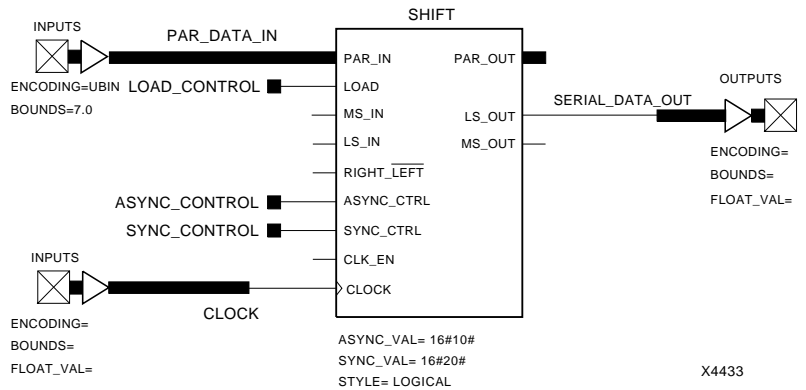
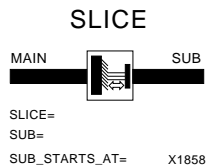


Figure 4-37 Typical Parallel-in/Serial-out Shift Register



## SLICE — SLICE of a Bus

The SLICE module is used to extract a portion of a larger bus or to collect smaller buses together into a larger bus. The SLICE module does not contain any active circuitry; it merely re-labels groups of lines for the convenience of the user. Any contiguous portion of the MAIN bus can be extracted to form the SUB bus by using the SLICE attribute. The SLICE module works the same way that slices work in VHDL.



**Figure 4-38 The Slice Module Symbol**

### Connections

These two ports can be connected to inputs, outputs or bidirectional X-BLOX buses connected to ports on other X-BLOX modules. Because the SLICE module has no active circuitry, there is no real Input or Output.

#### MAIN

The MAIN port is defined as the larger of the two connected buses. A portion of this bus, specified by the SLICE attribute, will become the SUB bus.

#### SUB

The SUB bus is that portion of the MAIN bus defined by the SLICE attribute.

## Attributes

### **SLICE**

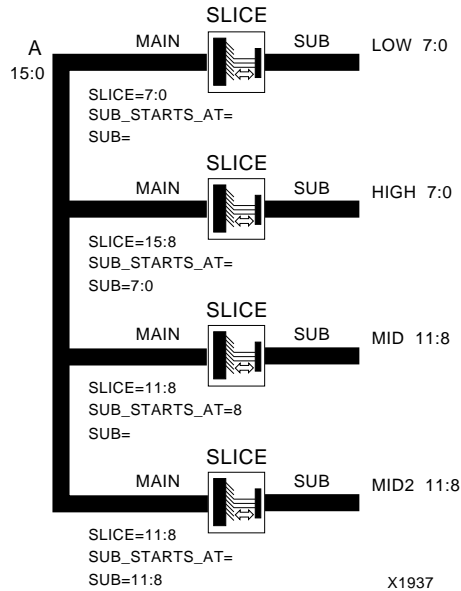
The SLICE attribute specifies the BOUNDS (MSB:LSB) of that portion of the MAIN bus that is mapped to the SUB bus.

### **SUB\_STARTS\_AT**

The SUB\_STARTS\_AT attribute defines the lower BOUND (LSB) of the SUB-bus label, thus allowing the indices of the SUB bus to be different from the indices of the SLICE of the MAIN bus. The upper BOUND is determined either by the width of the SLICE as determined by the SLICE attribute value, or by the SUB attribute. If not specified, the lower BOUND of the SUB-bus index is zero.

### **SUB**

The SUB attribute defines the upper and lower BOUNDS of the SUB bus, thus allowing the indices of the SUB bus to be different from the indices of the SLICE of the MAIN bus. If not specified, the upper and lower BOUNDS of the SUB bus are determined from the SUB\_STARTS\_AT attribute.



Bus A — is a 16-element bus with bounds 15:0.

Bus LOW — is an 8-element bus with bounds 7:0 and represents the same elements as A[7:0].

Bus HIGH — is an 8-element bus with bounds 7:0 and represents the same elements as A[15:8].

Buses MID and MID2 — are 4-element buses with bounds 11:8 and represent the same elements as A[11:8].

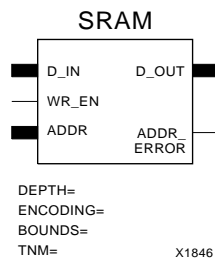
**Figure 4-39 Examples of Three Types of SUB Buses Sliced from One MAIN Bus**



## SRAM — Static Random-Access Memory

The SRAM module synthesizes a read-write Static Random-Access Memory. The SRAM can be of any size that will fit on a chip. The width of the SRAM is specified by the BOUNDS definition of the bus attached to either the D\_IN port or the D\_OUT port. The depth is specified by the DEPTH attribute of the SRAM or the data type of the Address signal. The data type of the Address port determines the maximum and minimum addressable locations, and can be greater than, equal to, or less than the DEPTH attribute. Only the addressable locations that are within the valid DEPTH will be synthesized. The word locations in the SRAM are indexed based on the data type of the signal attached to the ADDR port.

**Note:** The SRAM module is synthesized in different ways depending on the target architecture. On the XC3000A/L, XC3100A, and XC4000D architectures, flips-flops are used. Flip-flop data is stored with an edge-triggered Write-Enable. RAM data is stored with a level-triggered Write-Enable. On the XC4000/A/H architecture, CLB RAM is used. The XC4000/A/H RAM contains 32 bits per CLB. The XC3000A/L and XC3100A CLB can store only 2 bits per CLB.



**Figure 4-40 The SRAM Module Symbol**

### Inputs

#### D\_IN

D\_IN is the Data Input port. The data type of this port is the same as the D\_OUT port.

## ADDR

The Address port selects the word that appears on the Data Output port and the location where new data is written. The Address port determines the maximum depth of the SRAM if the DEPTH attribute is not specified. The Address port cannot be connected to a bus with ENCODING=ONE-HOT.

## WR\_EN

When the Write Enable input is High, the data on the D\_IN port is written into the selected address location. When Write Enable is Low, no new data can be written into the SRAM.

## Outputs

### D\_OUT

The Data Output port reflects the addressed word of the SRAM. The width of the D\_OUT port is set by the BOUNDS of the signal attached to either this port or the signal connected to the D\_IN port, and defines the word width of the SRAM. If an out-of-bounds word is addressed, the D\_OUT port is undefined and the ADDR\_ERROR output is High.

### ADDR\_ERROR

The Address Error (out-of-bounds) output is High, if the value on the Address port is beyond the addressable locations for the SRAM. Please refer to the section “Out-of-Range Indicators” in the chapter “Creating an X-BLOX Design” for more information.

## Attributes

### DEPTH

The DEPTH attribute represents the number of locations to be used in the SRAM, addressed from DEPTH 0 to DEPTH -1.

If the DEPTH attribute is not specified, then the maximum and minimum addressable locations are determined from the data type of the signal attached to the ADDR port.



## ENCODING

You can use this parameter to define the encoding of the D\_OUT port. The available encodings are defined in the “Bus Data Types” section of the chapter “Creating an X-BLOX Design.” All the encodings on each data path must be the same.

## BOUNDS

The BOUNDS attribute defines the precision (width) of the XBLOX\_BUS by specifying the MSB and LSB of the bus. Any pair of numbers can be used, as long as the difference of the indices, plus one, equals the width of the bus. See the “Bounds” section of the chapter “Creating an X-BLOX Design” for details. This parameter is optional.

**Note:** The ENCODING and BOUNDS attributes do not automatically appear on this module. Refer to your interface user guide for information on how to add these attributes. If you do not specify the bus data type on the SRAM module, data type propagation determines the bounds and encoding for the bus ports connected to the module.

## TNM

Use the TNM attribute to specify timing requirements.

You first place a TNM attribute on this module to identify the start or end point of a path using the following syntax:

**TNM=RAMS: *identifier***

where RAMS is the type of primitives to which X-BLOX propagates the TNM attribute in the XNF file and where *identifier* is a user-defined name. Refer to the section “Using XACT-Performance Attributes” in the chapter “Creating an X-BLOX Design” for more information.

You then use all the TNM attributes defined on the schematic in a TIMESPEC statement to specify delay requirements.

## CLB Utilization

The following table shows CLB utilization for 1-bit wide SRAMs for the types of parts that X-BLOX supports for different SRAM depths. To derive the CLB utilization for n-bit wide SRAMs, multiply the CLB utilization for the 1-bit wide SRAM by  $n$ .

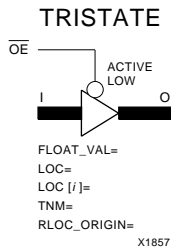
**Table 4-17 CLB Utilization**

Depth	XC3000A	XC4000D	XC4000/A/H
4	6.5	6.5	0.5
8	13.5	13	0.5
16	35.5	34	0.5
32	103	84	1
48	179	126	3
64	238.5	168	3.5

In the XC3000A/XC3100A/XC4000D families, flip-flops are used because these devices do not contain RAM that can be rewritten at non-configuration times.

## TRISTATE — 3-State Buffer

The 3-State (TRISTATE) module synthesizes internal non-inverting 3-State buffers for creating bidirectional or multiplexed data buses.



**Figure 4-41 The 3-State Module Symbol**

### Inputs

#### I

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

#### $\overline{OE}$

When the Output Enable ( $\overline{OE}$ ) input is Low, the Input data passes non-inverted to the Output. When the  $\overline{OE}$  is High, the Output is high-impedance “off”. This  $\overline{OE}$  input must be connected to a signal line. An active-High Output Enable can be synthesized by adding an inverter to this line (equivalent to the existing active-High 3-State buffer Enable used on XC4000).

### Outputs

#### O

The Output port reflects the Input port when the Output Enable is Low. When the Output Enable is High, the Output is high-impedance “off”. Although 3-State Output ports can be tied together, only one port at a time can be active. The Output data port has the same size and encoding as the Input port.

## Attributes (Optional)

### FLOAT\_VAL

You can use a `FLOAT_VAL` attribute to add pull-up resistors to 3-State Buffer Outputs that are connected to on-chip buses. This is an optional parameter for this module, and you must add it from the schematic editor. The allowed specifications are one resistor (`FLOAT_VAL = PULLUP`) or two resistors (`FLOAT_VAL = PULLUP_D`). The double resistor (`PULLUP_D`) draws more power than a single resistor, but it supports faster transition times. (See the XC4000 Data Sheet in *The Programmable Logic Data Book* for timing details).

### LOC

The `LOC` attribute specifies the pin location for a single input/output pad-buffer pair, for example: `LOC=A1`, or the attribute specifies the placement of all IOBs on a specific edge or corner of the chip, for instance, `LOC=TL` for the top-left corner of the chip.

### LOC[i]

The `LOC[i]` attribute specifies the pin locations for multiple input/output pad-buffer pair locations, for example: `LOC[6]=A11`, `LOC[7]=P9`.

### TNM

Use the `TNM` attribute to specify timing requirements.

You first place a `TNM` attribute on this module to identify the start or end point of a path using the following syntax:

**`TNM=PADS:identifier`**

where `PADS` is the type of primitives to which X-BLOX propagates the `TNM` attribute in the XNF file and where *identifier* is a user-defined name. Refer to the section "Using XACT-Performance Attributes" in the chapter "Creating an X-BLOX Design" for more information.

You then use all the `TNM` attributes defined on the schematic in a `TIMESPEC` statement to specify delay requirements.

## **RLOC\_ORIGIN**

Position the upper left corner of the RPM at a particular FPGA location. Do not specify a location that does not allow enough room for the TRISTATE. Refer to the section “Computing the Number of CLBs” in the chapter “Understanding X-BLOX Operations” for more information.

## X-BLOX-Generated Relationally Placed Macros

---

The XC4000 series contains fast-carry logic in its CLBs. The use of the fast-carry logic results in circuits that are up to four times faster and twice as dense. Fast-carry logic uses dedicated routing and is, therefore, restricted to columns on the XC4000 series. X-BLOX synthesizes relationally placed macros (RPMs) from the arithmetic X-BLOX modules (ADD\_SUB, INC\_DEC, ACCUM, and COUNTER modules as well as the COMPARE module when appropriate) to take advantage of the carry logic.

RPMs allow PPR to place related groups of logic in one step. To take advantage of this feature, X-BLOX also generates RPMs for the DATA\_REG and SHIFT modules provided the PAR\_OUT port on these modules is connected or you have specified the USE\_RLOC attribute to force the creation of an RPM, although these modules do not use carry logic. This feature forces the flip-flops generated by X-BLOX modules to be aligned on the chip.

When an RPM uses more CLBs than are available in a single column of the FPGA, X-BLOX synthesizes the RPM so that the logic bends at the top or bottom of the CLB column. X-BLOX knows when to introduce bends in the RPM so that it will fit on the FPGA device that you have selected.

For example, a 32-bit ADD\_SUB requires 16 CLBs plus 1 to start the chain. This is larger than a 4005 die, which is only 14 CLBs high. This RPM starts at the bottom of the die and turns to the right and takes up three CLBs from the top in the next column.

## Implementation Styles for Arithmetic Modules

The STYLE attribute determines the placement of bits in the CLBs of an RPM.

## STYLE=ALIGNED

This style uses the first CLB only to initialize the carry logic and puts the first two bits of the arithmetic function in the second CLB.

Although this requires an extra half-CLB at the bottom of the function, it gives a more logical pairing of bits. Note that only the carry-logic is used in this first CLB and that PPR can still use the function generators and flip-flops in this CLB for non-RPM logic.

## STYLE=UNALIGNED

This style initializes the carry chain in the first bit of the CLB, (only one sum bit is produced in this CLB) and puts two bits in each succeeding CLB. Although this method is the most space-efficient, it results in an odd pairing of bits. To make an UNALIGNED RPM, use the UNALIGNED STYLE attribute.

By default, all relationally placed macros created by X-BLOX follow the ALIGNED scheme. This default ensures that all RPMs in a design are consistent with the Xilinx Unified Libraries macros. The important factor is that all RPMs in a design be the same style and Xilinx's recommendation is that they all be ALIGNED.

If there are more than four inputs required to compute each output bit, the RPM needs to be more than one column wide. A COUNTER with UP\_DN, D\_IN, and LOAD is such an RPM since each bit requires:

- Its current value
- The carry-in from the previous bit
- UP\_DN
- LOAD
- D\_IN

Such an RPM is structured with the carry chain in the first column and the registers and load logic in the second column. If the carry chain is longer than the die, the carry chain travels across the top of the die to skip over the register columns. Be aware that COUNT\_TO on BINARY COUNTERS takes up one CLB input if UP\_DN is not connected, and two if it is connected.

## FAST3KA and RIPPLE (XC3000A/L and XC3100A)

The STYLE=FAST3KA and STYLE=RIPPLE are two implementation styles for the XC3000A/L and XC3100A architectures providing different speed and area trade-offs. Each style uses CLBs for the carry logic in a different way.

The FAST3KA style uses 50 percent more space than the RIPPLE style but is 30 percent faster than the RIPPLE style after routing.

The RIPPLE style is smaller and slower than the FAST3KA implementation style. RIPPLE is also available on the XC4000. It is easier for PPR to place, but it consumes more CLBs and is much slower than the ALIGNED or UNALIGNED styles.

## Controlling the Placement of RPMs

You can control how PPR places RPMs synthesized by X-BLOX modules using the attributes discussed below.

**Note:** The RLOC, U\_SET, and HU\_SET attributes are not supported on any X-BLOX module in this release.

### USE\_RLOC={TRUE|FALSE}

This attribute is recognized in all XC4000 families by the ADD\_SUB, ACCUM, INC\_DEC, SHIFT, and DATA\_REG modules and by the COUNTER module only if STYLE=BINARY.

- When the USE\_RLOC attribute is set to FALSE, the XC4000 carry logic is not used, and only gates are generated.
- When the USE\_RLOC attribute is set to TRUE, the carry logic is used where appropriate. The default is TRUE.

### RLOC\_ORIGIN=value

This attribute determines the absolute position (origin) of an RPM. Without an RLOC\_ORIGIN attribute, the RPM synthesized for an X-BLOX module can be placed in any area of the FPGA, as long as the shape and contents of the RPM remain unchanged. An RLOC\_ORIGIN specifies the absolute location of the upper-left



corner of an RPM. Use the following syntax to specify an RLOC origin:

**RLOC\_ORIGIN=RnCn**

where *Rn* and *Cn* denote the row and column numbers of the FPGA grid array. These numbers must be positive, non-zero integers.

RLOC\_ORIGIN is recognized in XC4000 and XC4000A parts by the ADD\_SUB, ACCUM, INC\_DEC, DATA\_REG, and SHIFT modules and by the COUNTER module only if STYLE=BINARY.

It is an error to give a location for an RLOC\_ORIGIN such that the carry chain is traveling upward (the default) and must wrap at the RLOC\_ORIGIN point, but that point is not at the top of the die. See the section "Restrictions on the RLOC\_ORIGIN" in Chapter 6.

## RLOC\_RANGE=value

You can attach this attribute to certain X-BLOX symbols to specify the range (rectangular area) of FPGA locations that are allowed for the synthesized RPM. Unlike the RLOC\_ORIGIN attribute, which fixes each member of the RPM at an absolute location, the RLOC\_RANGE attribute allows the members of the RPM to be placed anywhere within the rectangular range, as long as the relative location constraints generated by X-BLOX are maintained.

Specify the rectangular region using the following syntax:

**RLOC\_RANGE=RmCm : RnCn**

where *Rn* and *Cn* designate row and column numbers within the FPGA of two opposite corners of the rectangular region. *m* and *n* can be any non-zero, positive integers, or the wildcard character ("\*"). The wildcard character can be associated with either the row or the column on both sides of the range separator character, ":" (colon). It cannot be used for the row character on one side and the column character on the other.

This attribute is recognized in the XC4000/XC4000A parts by only the ADD\_SUB, ACCUM, COMPARE, DATA\_REG, INC\_DEC, SHIFT modules and by the COUNTER module only if STYLE=BINARY. The RLOC\_RANGE attribute is passed to the generated logic, but no error checking is performed by X-BLOX. XNFPprep, which runs after X-BLOX, reports any errors.

## Understanding X-BLOX Operations

---

After entering your design, you must process your design either to implement the design or to prepare for functional simulation. Refer to the chapter “Processing Your Design” for specific information on how to do these tasks. This chapter explains what goes on behind the interface program shell and is intended to provide you with information only on what X-BLOX does. In particular, it describes the special optimization features and synthesis capabilities offered by the X-BLOX software.

The chapter is structured as follows:

- *X-BLOX Implementation Flow* lists the different steps used by the Xilinx software to implement an X-BLOX design.
- *Data Propagation* explains how X-BLOX propagates the bus data types.
- *Architectural Synthesis and Optimization* describes the different types of optimizations that are performed by X-BLOX as it synthesizes your design.
- *Computing the Number of CLBs* discusses how X-BLOX places RPMs on the FPGA chip.
- *Synthesizing Your Design for Simulation* discusses how X-BLOX synthesizes your modules, expands your buses, and synthesizes simulation models for X-BLOX modules in your design.

## X-BLOX Implementation Flow

When you implement a design, that is, you run XMake to place and route the design, the program flow shown in Figure 6-1 is used to process your design.

1. In the first step of the design flow, design entry, you use X-BLOX modules and other components from the XC3000A, XC3000L, XC3100A, or XC4000 library depending on the family that is the target of your design.
2. The netlist obtained from the design entry tool is translated into a Xilinx Netlist File (XNF) by the netlist-to-XNF translator, which is based on your design-entry tool.
3. The XNFMerge program integrates all the XNF files into one (if there is more than one file), resolves all RLOC and TSPEC attributes, and writes out an XFF file. It also writes a report file, called the MRG file.
4. XNFPrep performs a pre-X-BLOX DRC check, trims out all unused or disabled logic, writes an XTG file, which is an XNF file with a .xtg extension, and a report file called PRX file.
5. X-BLOX reads the XNF file (with the .xtg extension) containing the X-BLOX modules and their attributes, propagates the data type (bus size and encoding information) throughout the design, determines operating modes for the modules, synthesizes the modules, creates new relationally-placed macros (RPMs), performs architecture-specific optimization, and creates a standard XNF output file called an XG file and a report file called the BLX file.
6. XNFPrep performs post-X-BLOX DRC checking, trims out the unused or disabled logic connected to the X-BLOX synthesized logic, creates an XTF file, which is an XNF file with a .xtf extension, and a report file called the PRP file.
7. For XC3000A/L and XC3100A designs, run XNFMAP after XNFPrep. XNFMAP creates a MAP file. The report file is called a CRF file.
8. Finally, PPR partitions, places, and routes the design, creating an LCA file on which you run Makebits to create a BIT file that can be downloaded. It also creates a report file called RPT file.

For information on how to enter X-BLOX designs, refer to your schematic or synthesis interface user guide. Continue with the next sections for information on design optimization and space utilization with X-BLOX, and for information on how to prepare for simulation.

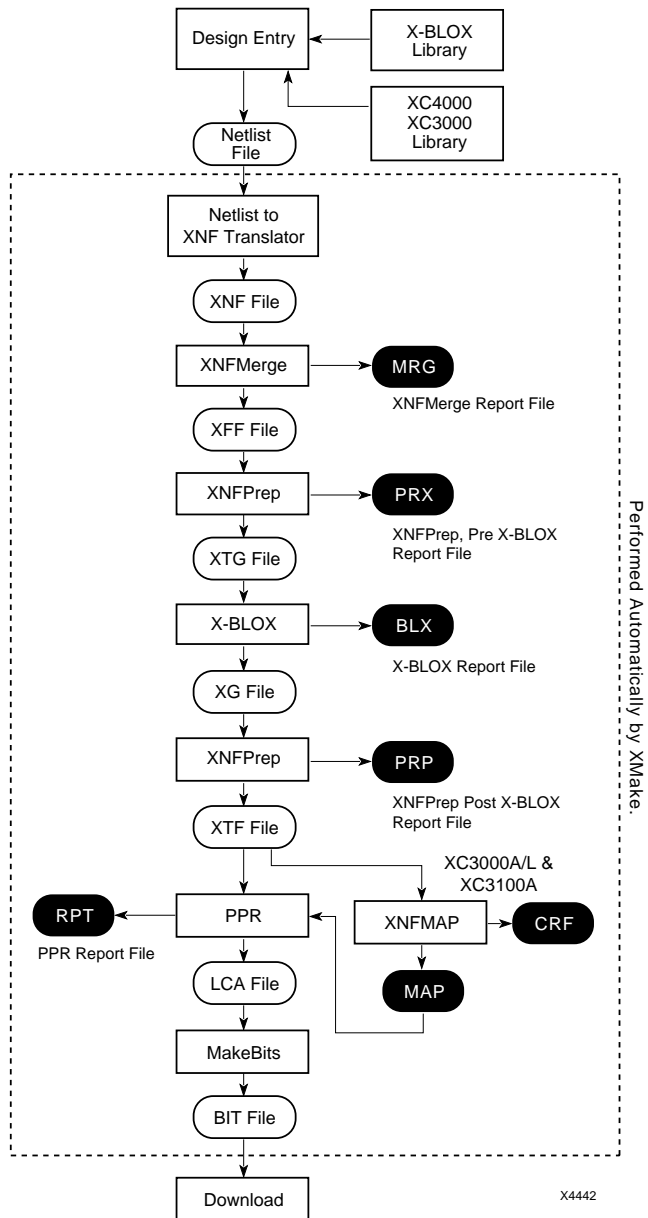


Figure 6-1 Implementation and Generation of a Bitstream File

## Data Type Propagation

During schematic entry, you specify the bus data type only once for all the buses in the same data path and you enter buses as single nets; therefore, all the signals in the buses might not be known to your schematic capture package or your simulator before the X-BLOX program is run.

For this reason, when X-BLOX processes your design, it must propagate the bus data type information to each bus in the same data path so that all these buses have the same encoding and width. The data type bounds determines bus widths, and the data type encoding determines operating modes for some of the X-BLOX modules.

## Architectural Synthesis and Optimization

X-BLOX does several things that are architecture specific so that higher speed and density of your designs can be achieved. These optimizations occur automatically during design implementation unless you disable them. The following paragraphs describe the optimizations and how to counteract them if you need to.

### Merging Flip-Flops into the I/O Blocks

Most of the Xilinx FPGA architectures have flip-flops in the I/O Blocks (IOBs). The IOB flip-flops do not have the same functionality as the flip-flops in the CLBs, but under most circumstances, the IOB flip-flops can be substituted for the CLB flip-flops. The X-BLOX software moves the flip-flops from the internal core to the I/O blocks when it is applicable. The use of these flip-flops instead of the ones in the CLBs can have the following benefits:

1. Using fewer flip-flops in CLBs may mean fewer CLBs are needed, and you can use a smaller, less expensive FPGA device or you can use more logic or flip-flops on your device.
2. The setup and hold times between I/O pads and flip-flops in IOBs are shorter than the setup and hold times between I/O pads and CLB flip-flops.
3. The delay from an output flip-flop to the I/O pad is shorter than the delay from a CLB flip-flop to a pad.

X-BLOX evaluates the potential movement of a flip-flop from a CLB into an IOB. It checks whether the flip-flop is one of the primitive flip-flops, such as FD and FDCE, whether it is part of a macro, or whether it is synthesized from an X-BLOX module, such as DATA\_REG or SHIFT for example.

X-BLOX automatically moves flip-flops from the CLBs into the IOBs if the following conditions are met:

1. Flip-flops are connected to an I/O symbol and only an I/O symbol. This can be an X-BLOX INPUTS, OUTPUTS, or BIDIR\_IO symbol, an IBUF, OBUF, or OBUFT symbol.
2. The CLB flip-flop does not use its clock-enable pin. Indeed, use of this pin prevents the flip-flop from being implemented into an IOB flip-flop. (This happens because IOB flip-flops do not have a clock-enable pin. )
3. The IOBs have input or output flip-flops. The XC4000H architecture has twice as many IOBs as the XC4000 architecture, but does not have input or output flip-flops in the smaller IOBs.
4. There is no CLB External net attribute (X flag) on the signal between the flip-flop and the I/O symbol.
5. The SYNC\_CTRL and ASYNC\_CTRL pins are not connected to the DATA\_REG or SHIFT module.

**Note:** If a flip-flop is connected to a TRISTATE output buffer, and the above conditions have been met, then a TRISTATE output flip-flop will be used.

To keep a flip-flop from being implemented in the I/O Blocks, make sure that one of the above conditions is not met. For example, the Save-Signal attribute can be placed on the signal between the I/O symbol and the flip-flop to prevent the flip-flop from being moved to an IOB.

## Global Buffers

X-BLOX examines your circuit, looking for high-fanout nets connected to clock, clock enable, and set/reset pins, and attempts to use the high-speed global primary and secondary buffers. The use of these global buffers can significantly reduce delays and routing

congestion and can determine whether your circuit completes the routing phase and runs at the desired speed.

If your design already uses a global primary or secondary buffer, the signals that use the buffers are compared to other high-fanout signals in your design. If there are other nets with higher fanout, they may be placed on the global buffer instead of the signals originally placed on the global buffer.

To prevent X-BLOX from replacing the signals already in the global buffers with other high-fanout signals, place the attribute `LOC=LOCKED` on the global buffer. However, designs often change, and a high-fanout net might easily become a net with lower fanout during schematic entry. As a result, if you lock a Global Buffer onto a particular signal, you need to examine your circuit carefully after editing it to ensure that the net still warrants the Global Buffer.

## Global Set-Reset

The XC4000 family chips have a Global Set-Reset line that goes to all flip-flops, both in the CLBs and in the IOBs. If all the flip-flops in your design are set or reset by the same signal, you can use the Global Set-Reset line. The Global Set-Reset line is a high-speed, no-skew, dedicated net on the XC4000 series. By using this net to set or reset your flip-flops, you can reduce delays as well as routing congestion significantly. Use of this net might determine whether your circuit completes the routing phase and runs at the desired speed. See *The Programmable Logic Data Book* for more details on the architecture.

## Relationally Placed Macros

The XC4000 series contains fast-carry logic in its CLBs. The use of the fast-carry logic over CLB logic will result in circuits that are up to four times faster, and twice as dense. Fast-carry logic uses dedicated routing and is, therefore, restricted to columns on the XC4000 series. Typically, two bits of arithmetic logic can occupy a single CLB. X-BLOX synthesizes relationally placed macros (RPMs) from the arithmetic X-BLOX modules to take advantage of the carry logic.

When more CLBs are used than are in a single column on the FPGA, X-BLOX synthesizes the arithmetic logic RPM so that it bends at the top or bottom of CLB column(s). X-BLOX knows when to introduce bends in the RPM so that it will fit on the FPGA device that you have

selected. For example, it is possible for X-BLOX to implement a large COUNTER that spans all of the columns of the FPGA.

Some X-BLOX modules that produce RPMs, such as the Loadable Up/Down COUNTER, might require more than one-half of a CLB per bit. Only up to 4-input functions can be implemented in half of a CLB. When all of the input pins of a function generator are used, a 5-input or larger function might be required. Although this function could be implemented in a single CLB, the fast-carry logic would be sacrificed. Instead, when more than one-half of a CLB is needed, the proper RPM is synthesized using multiple, adjacent columns. Typically, this results in two halves of adjacent CLBs being used for a single bit so that the fast-carry logic can still be used. The logic is synthesized for the fastest throughput.

## Computing the Required Number of CLBs

This section explains how X-BLOX calculates the number of CLB rows and columns required to implement your RPMs. These methods can be used to estimate the space taken by RPMs. The section is divided in two parts. The first part tells you how to calculate the required number of CLB rows, and the second part tells you how to calculate the required number of CLB columns. In the following discussion, the number of rows and columns are calculated independently of the size of the target chip. If the number of rows required is larger than what is available on the target chip, X-BLOX will fold an RPM in a serpentine fashion into multiple columns. Each of these columns will be as wide as the calculation below predicts.

### Computing the Number of Rows

You can compute the number of CLB rows required for an ADD\_SUB, ACCUM, or COUNTER module using the following formulas:



**Table 6-1 Equations for Computing Number of CLB Rows**

STYLE	Equation for Number of CLB Rows
ALIGNED	$1 + \text{CEILING}(\text{number of outputs}/2)$
UNALIGNED	$\text{CEILING}((\text{number of outputs} + 1) / 2)$

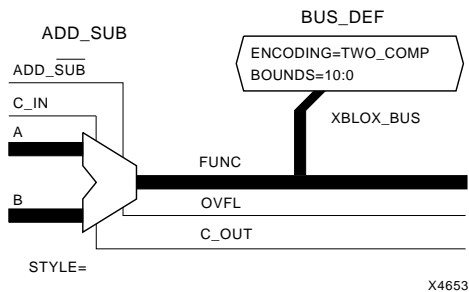
where  $\text{CEILING}(x)$  is the closest integer greater than or equal to  $x$ .

The “1” in the equation for ALIGNED comes from the fact that the ALIGNED style must use one CLB to start the carry chain whether the C\_IN pin is used or not. The “1” in the equation for UNALIGNED comes from the one-half CLB used by the UNALIGNED style to start the carry chain.

Remember that only the carry logic and not the function generators is needed to start the carry chain. PPR uses the unused function generators underneath the carry logic used to start the carry chain.

### Example

The ADD\_SUB shown in Figure 6-2 is connected to a bus with BOUNDS 10:0, which is an 11-bit bus. We also have the C\_OUT and OVFL ports connected. This gives us a total of 13 bits output from this RPM. As no style was specified, the default style of ALIGNED is used. As a result, the number of CLBs required for this RPM is  $1 + \text{CEILING}(13/2) = 8$  CLB rows. For STYLE=UNALIGNED, we can compute the number of CLB rows as  $\text{ceiling}((\text{number of outputs} + 1) / 2)$  which equals 7 for this particular example.

**Figure 6-2 ADD\_SUB Connections**

Counter modules with the STYLE=BINARY attribute are always expanded in the ALIGNED style unless the attribute USE\_RLOC=FALSE is also provided, in which case a gate-level implementation is used. Therefore, a 6-bit counter with STYLE=BINARY and TERM\_CNT connected has 7 output bits and requires  $1 + \text{CEILING}(7/2) = 5$  CLB rows.

DATA\_REG and SHIFT modules that are expanded as RPMs always require  $\text{CEILING}(\text{number of FFs}/2)$  CLB rows.

## Computing the Number of Columns

The number of CLB columns used per row by the RPM for an ACCUM or a STYLE=BINARY counter can be more than one. The number of CLB columns required for an ACCUM and COUNTER can be computed as follows:

**Table 6-2 Equations for Computing Number of CLB Columns**

Module	Equation for Number of CLB Columns
ACCUM	$\text{ceiling}((3 + \text{ADD\_SUB} + \text{SYNC\_CTRL} + \text{LOAD})/4)$
COUNTER	number of pins = $2 + \text{UP\_DN} + \text{SYNC\_CTRL} + (2 * \text{LOAD}) + \text{COUNT\_TO} + (\text{UP\_DN} * \text{COUNT\_TO})$ if (number of pins > 7) then number of columns = 3 else if (number of pins > 4) then number of columns = 2 else number of columns = 1.

where ADD\_SUB, UP\_DN, SYNC\_CTRL, and LOAD are 0 if the corresponding pin is not connected and 1 if the corresponding pin is connected, and COUNT\_TO is 1 if the COUNT\_TO attribute is used. The value 3 in the equation for ACCUM represents the two operands (A and B) and the C\_IN from the previous bit. The value 2 in the equation for COUNTER is the current value and the C\_IN from the previous bit. The factor  $2 * \text{LOAD}$  reflects the two pins required for this option (one for the D\_IN pin, one for LOAD), and the factor  $\text{UP\_DN} * \text{COUNT\_TO}$  represents that the max and min values must be detected for this option to work.

### Example 1

A STYLE=BINARY counter with the UP\_DN, SYNC\_CTRL, and LOAD input pins and a COUNT\_TO attribute has 8 pins. Therefore, the counter requires three CLB columns. If UP\_DN is not connected, it has six pins and only two columns are needed.

### Example 2

Consider an aligned 32-bit ACCUM that uses its ADD\_SUB and SYNC\_CTRL pins but neither its C\_OUT nor its OVFL pins. This RPM will be 2 CLB columns wide and 17 CLB rows high. If the target is the XC4002, which has an 8 x 8 array of CLBs, this RPM will have to be folded into three column-groups. Each column-group will be 2 CLBs wide for a total of 6 CLB columns and 34 occupied CLBs. The unused CLBs in each column will be available for other logic functions.

## RLOC\_ORIGIN Restrictions

An RLOC\_ORIGIN must be specified such that the resulting RPM cannot be placed in the given part type. Take for example the 11-bit ADD\_SUB discussed above. It required 8 CLB rows. If this module were generated for a 4005PC84, which has 14 CLB rows, then the RLOC\_ORIGIN could not be placed any lower than row 7. We cannot place it lower because the upward traveling carry chain would be required to turn right at the RLOC\_ORIGIN point since it is the upper left corner of the RPM. However, the FPGA only allows lateral connection of the carry-chain at the top and bottom of the device. An RLOC\_ORIGIN placed at row 1 is always legal as long as there are enough columns for the RPM to fit.

## RLOC\_RANGE Restrictions

An RLOC\_RANGE must allow enough room for the RPM. Therefore, the RLOC\_RANGE must be at least as tall as the lower of the number of CLB rows on the device or the number of rows required by the RPM. The RLOC\_RANGE must be at least as wide as the number of columns the RPM requires. The number of columns required by an RPM is:

**CEILING (number of CLB rows in RPM / number of CLB rows on the device) \* number of CLB columns per row in the RPM**

### Example

A 16-bit STYLE=BINARY counter with the UP\_DN, SYNC\_CTRL, LOAD ports, and a COUNT\_TO attribute has 8 pins. Therefore, it requires three CLB columns per row. This module requires  $1 + \text{CEILING}(16/2) = 9$  CLB rows.

Therefore, on a 40002APC84 part, which has 8 CLB rows, any RLOC\_RANGE provided for this module must be 8 CLB rows tall and  $\text{CEILING}(9/8) * 3 = 6$  CLB columns wide.

## Synthesizing Your Design for Simulation

Whereas the XACT Unified Libraries include precompiled simulation models for each library primitive and macro, X-BLOX must synthesize new models for its modules at run-time because the X-BLOX modules are flexible and generic. X-BLOX synthesizes a new simulation model for each different way an X-BLOX module is used in your design. You must run X-BLOX to synthesize the simulation models before simulating a design that contains X-BLOX modules.

### Synthesizing Simulation Models

The present section explains how X-BLOX synthesizes simulation models. These details are provided only for your understanding of how the modules are generated. The actual process is entirely performed by an interface-specific simulation program, such as XSimMake.

Depending on the program you run, XMake or XSimMake, X-BLOX optimizes the design or synthesizes simulation models.

- If you run XMake to implement your design, X-BLOX optimizes your design. Please refer to the section “Architectural Optimizations” for more details.
- If you translate the files for simulation by running XSimMake or other interface-specific program, X-BLOX generates simulation models for your design.

- When you translate the design into a functional simulation netlist, X-BLOX does not perform any architectural optimizations.
- When you translate the design into a timing simulation netlist, XSimMake uses the optimized design created by XMake as input.

Because X-BLOX modules are generic — they represent all module sizes and many different operating modes, or functions — there are no underlying simulation models until the X-BLOX modules have been expanded. After X-BLOX has been run in simulation mode, it writes the simulation models in the directory specified by the SIMDIREX command-line parameter. The directory specified by this parameter is specific to your interface. Normally, you do not need to use these files directly. If you need to use them, refer to your design interface user guide for details.

## Functional Simulation Models for Schematic Entry

A functional simulation schematic is derived from your original schematic by a procedure that is specific to your schematic interface. Each interface automatically calls X-BLOX during simulation. X-BLOX expands the buses and synthesizes simulation models. If there are any errors in your design, they are reported and no simulation models are generated.

The interface generates a simulation schematic which looks identical to your original schematic. The simulation schematic differs from the original schematic in that:

- Buses now have their correct widths and indices (subscripts).
- Simulation models now exist for all X-BLOX modules.

All symbols, nets, and design hierarchies are named and placed on the simulation schematic in the same way you entered them on your original schematic. Even sourceless and loadless nets are retained in simulation mode. This means you can simulate partial designs: the components to which the wires are attached are not removed and can be simulated.

The functional simulation models do not contain optimized logic, and you should not pass these models to the place and route software. The logic in each model is not optimized so as to enable

simulation models to be shared whenever possible. Thus, if two X-BLOX modules in your design have the same pins connected, the same attributes, and the same bus widths, X-BLOX generates only one model that the interface uses for both instances.

## **Timing Simulation Models**

Just like functional simulation models, timing simulation models are generated by an interface-specific procedure. Refer to Chapter 3, “Processing Your Design,” for more details. However, timing simulation models must be created from an implemented design.

The wire delays derived from a placed and routed LCA file, or implemented design, are back-annotated into the simulation schematic by the schematic interface.



## Command and Option Syntax

---

### Usage

Use the following command syntax to execute the X-BLOX program from the command line:

```
xblox infile [outfile] [parameter=value...]
```

*infile* — *Input XNF file*: This file should either be *file.xtg*, *file* with an implied *.xtg* extension, or *file* with a non-*xtg* extension. The contents of the file have to conform to the XNF specifications.

*outfile* — *Output XG file*: This file should either be *file.xg*, *file* with an implied *.xg* extension, or *file* with a non-*xg* extension. The default is the *infile* name with a *.xg* extension in place of the *infile* extension. The contents are in the XNF syntax.

*parameter=value*: This variable can be any of the command-line options listed in the next section.

### Command-line and Xactinit.dat Settings

This section lists the options that you can use when issuing commands from the command line or when specifying commands in the *xactinit.dat* file. See your *XACT Reference Guide* for more information on this file.

#### Options

*archopt* — This option allows the architectural optimizations to occur. If set to 0 or False, X-BLOX does not perform any architectural optimizations.

*blxfile* — This option allows you to specify the name of the file into



which X-BLOX writes a report. If provided, no file extension is added. By default, the file is called *infile.blx*. The screen output is always written to *xblox.log*.

*mergeio* — This option allows the flip-flops to be merged, if possible, into the IOBs. If set to 0 or False, *mergeio* prevents the flip-flops from being merged into any IOBs. On a DATA\_REG module, the STYLE attribute overrides the *mergeio=false* option.

*modgen* — This option allows module generation (or synthesis) to occur. If set to 0 or False, it prevents any X-BLOX modules from being expanded or any special symbols from being evaluated. This option should be used only if there are no X-BLOX symbols in the design.

*parttype* — *Target FPGA device*: If specified, this part type takes precedence over the part specified in the input XNF file. The default part type is "4005PQ160-5".

*sim* — If the *sim* option is used, it must be set as SIM=XNF. With this setting, X-BLOX outputs an XGS functional simulation model instead of an optimized simulation model. In this case, X-BLOX does not perform any architectural optimizations, and does not write an XG file. If this option is not used, X-BLOX does not produce any simulation models. This option should only be used by a simulation interface program.

*simdir* — X-BLOX produces many files for functional simulation. The *simdir* option is used to specify the directory or path in which to place the functional simulation models. By default, the directory is *outfile.bsm*. This option is used in combination with SIM=XNF only. You can change the path by setting the *simdir* option to another directory name. For example, 'SIMDIR= .' places the functional simulation models in the current directory. This option should only be used by a simulation interface program.

*subscripts* — When X-BLOX expands buses, it appends subscripts to the bus names. If you set the *subscripts* option to True or 1, this option generates brackets in the bus subscripts. If you set this option to False or 0, no brackets are generated. The default is True.

## Xactinit.dat Settings

The following variables can be set in your xactinit.dat file only. Do not use them when issuing commands from the command line. The xactinit.dat file contains all the options to be used at run-time by PPR and other programs. For more information on this file, see your *XACT Reference Guide*.

### **xblox\_merge\_tristate=true**

If this variable is set to True, X-BLOX merges TRISTATE modules into arithmetic RPM modules that drive them, when it is possible to do so. Merging occurs only if the RLOC\_ORIGIN attribute is present on the RPM driving the TRISTATE. The default is False.

### **xblox\_merge\_reg=false**

If this variable is set to True, X-BLOX merges DATA\_REG modules into arithmetic RPM modules that drive them, when it is possible to do so. The default is True.



# Index

---

## Numerics

3-State

BIDIR\_IO, 4-18

3-State module, 4-110

## A

ACCUM module, 4-2, 4-8

Adder/Subtractor, 4-8, 4-9

ASYNC\_VAL, 4-11

RLOC\_ORIGIN, 4-12

RLOC\_RANGE, 4-12

STYLE

ALIGNED=Default, 4-11

RPM, 4-11

SYNC\_VAL, 4-11

TNM, 4-12

USE\_RLOC, 4-12

Accumulator (ACCUM), 4-8

ADD\_SUB input

ADD\_SUB module, 4-14

ADD\_SUB module, 4-2, 4-14

RLOC\_ORIGIN, 4-16

RLOC\_RANGE, 4-16

STYLE

ALIGNED=Default, 4-15

RPM, 4-15

USE\_RLOC, 4-16

Adder/Subtractor (ADD\_SUB), 4-14

Adding

Module, 2-1

ADDR

PROM, 4-84

SRAM, 4-107

ADDR\_ERROR

PROM, 4-85

SRAM, 4-107

Aliasing, 2-22

ANDBUS module, 4-2

Attributes, 4-3

Based gate functions, 4-4

ANDBUS1 module, 4-2

Based gate functions, 4-4

ANDBUS2 module, 4-5

archopt option, A-1

ARITH style, 4-34, 4-97

ASYNC\_CTRL, 2-9

ACCUM, 4-10

CLK\_DIV, 4-29

COUNTER, 4-38

DATA\_REG, 4-49

SHIFT, 4-96

ASYNC\_VAL attribute, 2-8, 2-9

ACCUM, 4-11

COUNTER, 4-39

DATA\_REG, 4-50

SHIFT, 4-97

Usage, 2-8

Asynchronous control, 2-8

Attributes

ASYNC\_VAL, 2-9

BOUNDS, 2-17

COUNT\_TO, 4-40

DECODEMASK, 2-9

DEPTH, 4-85, 4-107

DIVIDE\_BY, 4-30

DUTY\_CYCLE, 4-30

- ELEM, 4-60
- ENCODING, 2-16
- FAST3KA, 5-3
- FLOAT\_VAL, 2-12
- Implementation style, 2-4
- INC\_BY, 4-65
- INVMASK, 2-9
  - Bus-level functions, 2-9
- KEEPNAME (KN), 2-23
- LOC, 2-27
- MEMFILE, 4-85
- Operating modes, 2-6
- PULLUP, 4-111
- RIPPLE, 5-3
- RLOC\_ORIGIN, 5-3, 6-10
- RLOC\_RANGE, 5-4, 6-10
- SLICE, 4-103
- Specifying attributes, 3-7
- STYLE, 4-3, 5-1
- SUB, 4-103
- SUB\_STARTS\_AT, 4-103
- SYNC\_VAL, 2-9
- TNM, 2-31
- TSidentifier, 2-31
- USE\_RLOC, 5-3
- VALUE, 4-63

## B

- Back-annotation, 3-8, 3-12
- BIDIR\_IO module, 4-2, 4-18
  - BOUNDS, 4-19
  - ENCODING, 4-19
  - FLOAT\_VAL, 4-20
  - LOC, 4-20
  - PADNAME, 4-20
  - TNM, 4-21
- Bidirectional I/O (BIDIR\_IO), 4-18
- Big-endian, 2-18
- Binary, 2-8
- Binary Counter, 4-40
- Binary point, 2-18

- BIT encoding, 2-17
- blxfile option, A-1
- BOUNDS attribute
  - BIDIR\_IO, 4-19
  - BUS\_DEF, 4-22
  - CAST, 4-27
  - COUNTER, 4-41
  - Data type propagation, 2-19
  - DATA\_REG, 4-50
  - FORCE, 4-62
  - INPUTS, 4-69
  - OUTPUTS, 4-81
  - PROM, 4-86
  - SHIFT, 4-98
  - SLICE, 4-103
  - SRAM, 4-108
  - Usage, 2-17

## Bus

- Bidirectional data, 4-110
- Big-endian, Little-endian, 2-18
- Constant, 4-2
- Data type, 2-16
- ELEMENT of a bus, 4-2
- Interface, 2-28
- Labels, 2-15
- Manipulation, 2-25
  - CAST symbol, 2-25
  - ELEMENT symbol, 2-26
  - FORCE symbol, 2-26
  - MUXBUS symbol, 2-27
  - SLICE symbol, 2-26
- Width, 2-17
- Bus HIGH SLICE, example, 4-104
- Bus LOW SLICE, example, 4-104
- Bus MID SLICE, example, 4-104
- BUS\_DEF module, 2-21, 2-29, 4-2, 4-22
  - BOUNDS, 4-22
  - ENCODING, 4-22
- BUS\_IFxx symbol, 2-28
- Bused gate functions, 4-4

- ANDBUS2, 4-5
- INVBUS, 4-5
- ORBUS, 4-5
- ORBUS1, 4-6
- ORBUS2, 4-6
- XORBUS, 4-6
- XORBUS1, 4-7
- XORBUS2, 4-6, 4-7
- Bused gates, 4-2
- C**
- C\_IN
  - ACCUM, 4-9
  - ADD\_SUB, 4-14
- C\_OUT
  - ACCUM, 4-11
  - ADD\_SUB, 4-15, 4-65
- Cascading counters, 4-45
- CAST module, 4-2, 4-24
  - BOUNDS, 4-27
  - ENCODING, 4-27
- CIRCULAR style, 4-97
- CLB
  - Computing number, 6-7
  - SRAM CLB utilization, 4-109
- CLK\_DIV attribute
  - DIVIDE\_BY, 4-30
- CLK\_DIV module, 4-2, 4-28
  - DUTY\_CYCLE, 4-30
  - TNM, 4-30
- CLK\_EN
  - ACCUM, 4-10
  - COUNTER, 4-38
  - DATA\_REG, 4-49
  - SHIFT, 4-95
- CLK\_OUT
  - CLK\_DIV, 4-29
- CLOCK
  - ACCUM, 4-10
  - CLK\_DIV, 4-29
  - COUNTER, 4-38
  - DATA\_REG, 4-49
  - SHIFT, 4-96
  - Clock Divider (CLK\_DIV), 4-2, 4-28
  - Command syntax, A-1
  - COMPARE module, 4-2, 4-32
    - Equality, 4-32
    - Magnitude, 4-32
    - RLOC\_RANGE, 4-34
    - STYLE, 4-34
      - ARITH, 4-34
      - RIPPLE, 4-34
      - TREE, 4-34
      - WIRED, 4-34
  - Configuration
    - Environment, 3-4
  - Constraints file
    - BIDIR\_IO, 4-21
    - Example, 3-18
    - INPUTS, 4-70
    - OUTPUTS, 4-82
  - COUNT\_TO attribute
    - COUNTER, 4-40
  - COUNTER module, 4-2, 4-36
    - ASYNC\_VAL, 4-39
    - BOUNDS, 4-41
    - Cascading counters, 4-45
    - COUNT\_TO, 4-40
    - ENCODING, 4-41
    - RLOC\_ORIGIN, 4-41, 4-112
    - RLOC\_RANGE, 4-42
    - STYLE
      - BINARY, 4-42, 4-43
      - BINARY=Default, 4-40
      - Features, 4-42
      - JOHNSON, 4-40
      - LFSR, 4-40, 4-43
      - ONE\_HOT, 4-44
    - SYNC\_VAL, 4-39
    - TNM, 4-42
    - USE\_RLOC, 4-41

- D**
- D\_IN
  - COUNTER, 4-37
  - DATA\_REG, 4-49
  - SRAM, 4-106
- D\_OUT
  - DECODE, 4-59
  - PROM, 4-84
  - SRAM, 4-107
- Data Register (DATA\_REG), 4-48
- Data type
  - Big-endian, 2-18
  - BOUNDS, 2-16
  - CAST, 4-24
  - COERCE, 4-34
  - ENCODING, 2-16
  - Little-endian, 2-18
  - Usage, 2-16
- Data type propagation, 2-21, 6-4
  - Usage, 2-19
- Data values, 2-6
  - Base, 2-6
  - Binary, 2-8
  - Decimal, 2-8
  - Default, 2-6
  - Dontcaredigits', 2-7
  - Format, 2-8
  - Hexadecimal, 2-8
  - Negative values, 2-7, 2-18
  - Octal, 2-8
  - Radix, 2-6
  - Radix point, 2-7
- DATA\_REG module, 4-2, 4-48
  - ASYNC\_VAL, 4-50
  - BOUNDS, 4-50
  - ENCODING, 4-50
  - LOC, 4-51
  - RLOC\_ORIGIN, 4-51
  - RLOC\_RANGE, 4-51
  - STYLE attribute, 4-52
  - SYNC\_VAL, 4-50
  - TNM, 4-52
  - USE\_RLOC, 4-51
- Decimal, 2-8
- DECODE module, 4-2, 4-58
  - Select warning message, 4-58
- DECODEMASK attribute, 2-9
- DEPTH attribute
  - PROM, 4-85
  - SRAM, 4-107
- Design
  - Creation, 3-3
  - Design example, 3-15
  - Design flow, 6-1
  - Design process, 3-1
  - Downloading, 3-14
  - Functional simulation, 3-8
  - Implementation, 3-10
  - Incremental flow, 3-2
  - Timing simulation, 3-12
- Directories
  - design\_tim, 3-12
  - otherxnf, 3-8, 3-12
  - sdesign, 3-8, 3-12
  - simdir, 3-8
- DIVIDE\_BY attribute
  - CLK\_DIV, 4-30
- Dontcaredigits', 2-7
- DUTY\_CYCLE attribute
  - CLK\_DIV, 4-30
- E**
- Edge location
  - ANDBUS location attribute, 4-3
- ELEM attribute
  - ELEMENT, 4-60
- ELEM port
  - ELEMENT, 4-60
- ELEMENT module, 4-2, 4-60
  - ELEM, 4-60
- EN
  - CLK\_DIV, 4-29
- ENCODING attribute, 2-16

- BIDIR\_IO, 4-19
- BIT, 2-17
- BUS\_DEF, 4-22
- CAST, 4-27
- COUNTER, 4-41
- Data type propagation, 2-19
- DATA\_REG, 4-50
- FORCE, 4-62
- INPUTS, 4-69
- ONE\_HOT, 2-17
- OUTPUTS, 4-81
- PROM, 4-85
- SHIFT, 4-97
- SRAM, 4-108
- TWO\_COMP, 2-17
- UBIN, 2-17
- Usage, 2-16
- Examples directory, 1-2
- F**
- FAST3KA attribute, 4-12, 4-16, 4-66, 5-3
- Fibonacci sequence, 3-15
- FIFO, 4-92
- File
  - .blx, 2-23
- Flip-flops
  - Merging into IOBs, 6-4
- FLOAT\_VAL attribute
  - BIDIR\_IO, 4-20
  - Example, 2-12
  - INPUTS, 4-69
  - OUTPUTS, 4-81
  - TRISTATE, 4-111
  - Usage, 2-12
- FORCE module, 4-62
  - BOUNDS, 4-62
  - ENCODING, 4-62
  - VALUE, 4-63
- Frequency divider, 4-2
- FUNC
  - ADD\_SUB, 4-15, 4-65
- Functional simulation, 3-8
  - Back-annotation, 3-8
- G**
- Global buffers, 6-5
- Global modifications, 2-19
- Global Set-Reset, 6-6
- H**
- Hexadecimal, 2-8
- I**
- I/O, 2-16
  - BIDIR\_IO, 2-20, 4-2
  - INPUTS, 2-20, 4-1, 4-2
  - OUTPUTS, 2-20, 4-1, 4-2
- IE
  - BIDIR\_IO, 4-19
- Implementation
  - Complete design, 3-14
  - Partial design, 3-10
- Implementation styles, 2-4, 5-1
- INC\_BY attribute
  - INC\_DEC, 4-65
- INC\_DEC module, 4-64
  - INC\_BY, 4-65
  - RLOC\_ORIGIN, 4-66
  - RLOC\_RANGE, 4-66
  - STYLE
    - ALIGNED=Default, 4-66
    - RPM, 4-66
    - USE\_RLOC, 4-66
- Incremental design, 3-2
  - Program settings, 3-11
- Initialization, 2-9
- INPUTS module, 4-2, 4-68
  - BOUNDS, 4-69
  - ENCODING, 4-69
  - FLOAT\_VAL, 4-69
  - INPUTS, 4-68
  - LOC, 4-70
  - PADNAME, 4-69
  - TNM, 4-70



INPUTS pin

BIDIR\_IO, 4-19

INVBUS module, 4-2, 4-5

INVMASK attribute, 2-9, 4-2

IOB

Merging flip-flops into IOBs, 6-4

## J

Johnson Counter, 4-40

## K

KEEPNAME (KN) attribute, 2-23

## L

Labels

Bus, 2-15

LFSR counter, 4-40

Cascading counters, 4-45

Library

Loading, 3-5

LIFO, 4-92

Little-endian, 2-18

LOAD

ACCUM, 4-9

COUNTER, 4-37

SHIFT, 4-94

LOC attribute

ANDBUS, 4-3

BIDIR\_IO, 4-20

DATA\_REG, 4-51

INPUTS, 4-70

OUTPUTS, 4-82

SHIFT, 4-98

TRISTATE, 4-111

Location attributes, 2-27

LOGICAL style, 4-97

LS\_IN

SHIFT, 4-95

LS\_OUT

SHIFT, 4-96

## M

MEMFILE attribute

PROM, 4-85

Syntax, 4-86

Memory

PROM, 4-2, 4-84

SRAM, 4-2, 4-106

Memory Definition File

PROM, 4-85

mergeio option, A-2

modgen option, A-2

Module

*see also* X-BLOX modules, Attributes

Data values, 2-6

MS\_IN

SHIFT, 4-95

MS\_OUT

SHIFT, 4-96

Multiplexers

MUXBUS, 4-72

MUXBUS2, 4-74

MUXBUS4, 4-76

MUXBUS8, 4-78

MUX\_IN

MUXBUS, 4-72

MUX\_OUT

MUXBUS, 4-73

MUXBUS2, 4-75

MUXBUS4, 4-77

MUXBUS8, 4-79

MUXBUS

Select warning message, 4-72

MUXBUS module, 4-2, 4-72

MUXBUS2 module, 4-2, 4-74

Select warning message, 4-74

MUXBUS4 module, 4-2, 4-76

Select warning message, 4-76

MUXBUS8 module, 4-2, 4-78

Select warning message, 4-78

## O

Octadecimal, 2-6

Octal, 2-8

OE

BIDIR\_IO, 4-19  
 TRISTATE, 4-110  
 ONE\_HOT encoding, 2-17  
 ONE\_HOT style  
   Counter, 4-40  
   DECODE, 4-59  
 Operating modes, 2-6  
 Optimization, 6-4  
   DATA\_REG module, 4-52  
 Option syntax, *see* X-BLOX options  
 ORBUS module, 4-2  
 ORBUS1 module, 4-2, 4-6  
 ORBUS2 module, 4-2, 4-6  
 Out-of-range indicators, 2-13  
 OUTPUTS module, 4-2, 4-80  
   BOUNDS, 4-81  
   ENCODING, 4-81  
   FLOAT\_VAL, 4-81  
   LOC, 4-82  
   PADNAME, 4-81  
   TNM, 4-82  
 OUTPUTS pin  
   BIDIR\_IO, 4-19  
 OVFL  
   ACCUM, 4-11  
   ADD\_SUB, 4-15, 4-65

**P**

PADNAME attribute  
   BIDIR\_IO, 4-20  
   INPUTS, 4-69  
   OUTPUTS, 4-81  
 PAR\_IN  
   SHIFT, 4-94  
 PAR\_OUT  
   SHIFT, 4-96  
 parttype option, A-2  
 Power-up  
   Reset, 2-9  
 Precision, 2-17  
 PROM module, 4-2, 4-84  
   BOUNDS, 4-86

Default Radix=Hexadecimal, 4-90  
 Definition Procedure, 4-90  
 DEPTH, 4-85  
 ENCODING, 4-85  
 MEMFILE, 4-85  
 TNM, 4-86  
 PULLUP Resistor  
   TRISTATE, 4-111  
 PULLUP\_D Resistor  
   TRISTATE, 4-111

**Q**

Q\_OUT  
   ACCUM, 4-11  
   COUNTER, 4-38  
   DATA\_REG, 4-50

**R**

Register  
   Data  
     Storage, 4-2  
   SHIFT, 4-2  
 Relationally placed macro, *see* RPM  
 Resistors  
   Pull-down, 2-12  
   Pull-up, 2-12  
 RIGHT\_LEFT  
   SHIFT, 4-95  
 RIPPLE attribute, 4-12, 4-16, 4-66  
 RIPPLE style, 4-34, 5-3  
 RLOC\_ORIGIN attribute, 5-3  
   ACCUM, 4-12  
   ADD\_SUB, 4-16  
   COUNTER, 4-41, 4-112  
   DATA\_REG, 4-51  
   INC\_DEC, 4-66  
   Restrictions, 6-10  
   SHIFT, 4-98  
 RLOC\_RANGE attribute, 5-4  
   ACCUM, 4-12  
   ADD\_SUB, 4-16  
   COMPARE, 4-34

- COUNTER, 4-42
- DATA\_REG, 4-51
- INC\_DEC, 4-66
  - Restrictions, 6-10
- SHIFT, 4-99
- RPM, 5-1, 6-2, 6-6
  - Aligned, 4-12, 4-16, 4-66
  - Controlling RPM placement, 5-3
  - Unaligned, 4-12, 4-16, 4-66
- S**
- SEL
  - MUXBUS, 4-72
  - MUXBUS2, 4-74
  - MUXBUS4, 4-76
  - MUXBUS8, 4-78
- SEL\_ERROR
  - DECODE, 4-59
  - MUXBUS, 4-73
  - MUXBUS2, 4-75
  - MUXBUS4, 4-77
  - MUXBUS8, 4-79
- Serial-in/Serial-out
  - BUS\_DEF, 4-23
- SHIFT module, 4-92
  - ASYNC\_VAL, 4-97
  - BOUNDS, 4-98
  - ENCODING, 4-97
  - LOC, 4-98
  - Parallel-in/Serial-out, 4-100
  - Register, 4-2
  - Right/Left Control, 4-95
  - RLOC\_ORIGIN, 4-98
  - RLOC\_RANGE, 4-99
  - Serial-in/Parallel-out, 4-99
  - Serial-in/Serial-out, 4-100
- STYLE
  - ARITH, 4-97
  - CIRCULAR, 4-97
  - LOGICAL=Default, 4-97
  - SYNC\_VAL, 4-97
  - TNM, 4-99
  - USE\_RLOC, 4-98
- Signal aliasing, 2-22
- sim option, A-2
- simdir option, A-2
- Simulation
  - Functional simulation, 3-8
  - Timing simulation, 3-12
- Simulation models, 6-11
  - Functional simulation models, 6-12
  - Timing simulation models, 6-13
- SLICE attribute
  - SLICE, 4-103
- SLICE module, 4-2, 4-102
  - MAIN bus, 4-102
  - SLICE, 4-103
  - SUB, 4-103
  - SUB bus, 4-102
  - SUB\_STARTS\_AT, 4-103
- SRAM module, 4-2, 4-106
  - BOUNDS, 4-108
  - DEPTH, 4-107
  - ENCODING, 4-108
  - TNM, 4-108
- STYLE attribute, 5-1
  - ACCUM, 4-11
  - ADD\_SUB, 4-15
  - ANDBUS, 4-3
  - COMPARE, 4-34
  - COUNTER, 4-40
  - DATA\_REG, 4-52
  - INC\_DEC, 4-66
  - Selection, 2-5
  - SHIFT, 4-97
  - Usage, 5-1
- SUB attribute
  - SLICE, 4-103
- SUB\_STARTS\_AT attribute
  - SLICE, 4-103
- subscripts option, A-2
- SYNC\_CTRL

ACCUM, 4-10  
 CLK\_DIV, 4-29  
 COUNTER, 4-38  
 DATA\_REG, 4-50  
 SHIFT, 4-96  
 SYNC\_VAL attribute, 2-8  
   ACCUM, 4-11  
   COUNTER, 4-39  
   DATA\_REG, 4-50  
   SHIFT, 4-97  
   Usage, 2-8  
 Synchronous control, 2-8  
 Synthesis, 6-4  
**T**  
 TBUF location  
   ANDBUS location attribute, 4-4  
 TERM\_CNT  
   COUNTER, 4-38  
 Timing attributes, *see* XACT-Performance  
 Timing simulation, 3-12  
   Back-annotation, 3-12  
 TNM attribute, 2-31  
   ACCUM, 4-12  
   BIDIR\_IO, 4-21  
   CLK\_DIV, 4-30  
   COUNTER, 4-42  
   DATA\_REG, 4-52  
   INPUTS, 4-70  
   OUTPUTS, 4-82  
   PROM, 4-86  
   SHIFT, 4-99  
   SRAM, 4-108  
   TRISTATE, 4-111  
 TREE structure comparisons, 4-34  
 TREE style, 4-34  
 TRISTATE module, 4-2, 4-110  
   FLOAT\_VAL, 4-111  
   LOC, 4-111  
   TNM, 4-111  
 TSidentifier attribute, 2-31  
 TWO\_COMP encoding, 2-17

**U**

Universal Counter, 4-36  
 Universal Shift Register, 4-92  
 UP\_DN  
   COUNTER, 4-37  
 USE\_RLOC attribute, 5-3  
   ACCUM, 4-12  
   ADD\_SUB, 4-16  
   COUNTER, 4-41  
   DATA\_REG, 4-51  
   INC\_DEC, 4-66  
   SHIFT, 4-98

**V**

VALUE attribute  
   FORCE, 4-63

**W**

Wide edge-decoder  
   ANDBUS DECODE style, 4-3  
 WIRED style, 4-34  
 Wired-AND  
   ANDBUS WAND style, 4-3  
 WR\_EN  
   SRAM, 4-107

**X**

X attribute, 6-5  
 Xactinit.dat settings, A-1, A-3  
 XACT-Performance  
   Attributes, 2-30  
   Logic primitives, 2-32  
 X-BLOX  
   Features, 1-1  
     Block-diagram design, 1-1  
     Compatibility, 1-2  
     Data path, 1-1  
     Optimization, 1-2  
 X-BLOX modules  
   ACCUM, 4-8  
   ADD\_SUB, 4-14  
   ANDBUS, 4-4

ANDBUS1, 4-4  
ANDBUS2, 4-5  
BIDIR\_IO, 4-18  
BUS\_DEF, 4-22  
BUS\_IFxx, 2-28  
CAST, 4-24  
CLK\_DIV, 4-28  
COMPARE, 4-32  
COUNTER, 4-36  
DATA\_REG, 4-48  
DECODE, 4-58  
ELEMENT, 4-60  
FORCE, 4-62  
INC\_DEC, 4-64  
INPUTS, 4-68  
INVBUS, 4-5  
MUXBUS, 4-72  
MUXBUS2, 4-74  
MUXBUS4, 4-76  
MUXBUS8, 4-78  
ORBUS, 4-5  
ORBUS1, 4-6  
ORBUS2, 4-6  
OUTPUTS, 4-80  
PROM, 4-84  
SHIFT, 4-92  
SLICE, 4-102  
SRAM, 4-106  
TRISTATE, 4-110  
XORBUS1, 4-7  
XORBUS2, 4-6, 4-7  
X-BLOX options  
  archopt, A-1  
  blxfile, A-1  
  mergeio, A-2  
  modgen, A-2  
  parttype, A-2  
  sim, A-2  
  simdir, A-2  
  subscripts, A-2  
  Syntax, A-1  
XBLOX\_BUS  
  BUS\_DEF, 4-22  
  ELEMENT, 4-60  
XMake program, 3-10  
XNF files  
  RPM, 6-2  
XORBUS module, 4-2, 4-6  
XORBUS1 module, 4-2  
XORBUS2 module, 4-2, 4-6, 4-7

# Trademark Information

---

**Σ XILINX**<sup>®</sup>, XACT, XC2064, XC3090, XC4005, and XC-DS501 are registered trademarks of Xilinx. All XC-prefix product designations, XACT-Floorplanner, XACT-Performance, XAPP, XAM, X-BLOX, X-BLOX plus, XChecker, XDM, XDS, XEPLD, XPP, XSI, BITA, Configurable Logic Cell, CLC, Dual Block, FastCLK, HardWire, LCA, Logic Cell, LogicProfessor, MicroVia, PLUSASM, SMARTswitch, UIM, VectorMaze, VersaBlock, VersaRing, and ZERO+ are trademarks of Xilinx. The Programmable Logic Company and The Programmable Gate Array Company are service marks of Xilinx.

IBM is a registered trademark and PC/AT, PC/XT, PS/2 and Micro Channel are trademarks of International Business Machines Corporation. DASH, Data I/O and FutureNet are registered trademarks and ABEL, ABEL-HDL and ABEL-PLA are trademarks of Data I/O Corporation. SimuCad and Silos are registered trademarks and P-Silos and P/C-Silos are trademarks of SimuCad Corporation. Microsoft is a registered trademark and MS-DOS is a trademark of Microsoft Corporation. Centronics is a registered trademark of Centronics Data Computer Corporation. PAL and PALASM are registered trademarks of Advanced Micro Devices, Inc. UNIX is a trademark of AT&T Technologies, Inc. CUPL, PROLINK, and MAKEPRG are trademarks of Logical Devices, Inc. Apollo and AEGIS are registered trademarks of Hewlett-Packard Corporation. Mentor and IDEA are registered trademarks and NETED, Design Architect, QuickSim, QuickSim II, and EXPAND are trademarks of Mentor Graphics, Inc. Sun is a registered trademark of Sun Microsystems, Inc. SCHEMA II+ and SCHEMA III are trademarks of Omatation Corporation. OrCAD is a registered trademark of OrCAD Systems Corporation. Viewlogic, Viewsim, and Viewdraw are registered trademarks of Viewlogic Systems, Inc. CASE Technology is a trademark of CASE Technology, a division of the Teradyne Electronic Design Automation Group. DECstation is a trademark of Digital Equipment Corporation. Synopsys is a registered trademark of Synopsys, Inc. Verilog is a registered trademark of Cadence Design Systems, Inc.

Xilinx does not assume any liability arising out of the application or use of any product described or shown herein; nor does it convey any license under its patents, copyrights, or maskwork rights or any rights of others. Xilinx reserves the right to make changes, at any time, in order to improve reliability, function or design and to supply the best product possible. Xilinx will not assume responsibility for the use of any circuitry described herein other than circuitry entirely embodied in its products. Xilinx devices and products are protected under one or more of the following U.S. Patents: 4,642,487; 4,695,740; 4,706,216; 4,713,557; 4,746,822; 4,750,155; 4,758,985; 4,820,937; 4,821,233; 4,835,418; 4,853,626;

4,855,619; 4,855,669; 4,902,910; 4,940,909; 4,967,107; 5,012,135; 5,023,606; 5,028,821; 5,047,710; 5,068,603; 5,140,193; 5,148,390; 5,155,432; 5,166,858; 5,224,056; 5,243,238; 5,245,277; 5,267,187; 5,291,079; 5,295,090; 5,302,866; 5,319,252; 5,319,254; 5,321,704; 5,329,174; 5,329,181; 5,331,220; 5,331,226; 5,332,929; 5,337,255; 5,343,406; 5,349,248; 5,349,249; 5,349,250; 5,349,691; 5,357,153; 5,360,747; 5,361,229; 5,362,999; 5,365,125; 5,367,207; 5,386,154; 5,394,104; 5,399,924; 5,399,925; 5,410,189; 5,410,194; 5,414,377; RE 34,363, RE 34,444, and RE 34,808. Other U.S. and foreign patents pending. Xilinx, Inc. does not represent that devices shown or products described herein are free from patent infringement or from any other third party right. Xilinx assumes no obligation to correct any errors contained herein or to advise any user of this text of any correction if such be made. Xilinx 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.