ModuleWare Reference Manual

for the HDL Designer Series™

Library Version 1.9

June, 2011

Contractor/manufacturer is:
Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777.
Telephone: 503.685.7000
Toll-Free Telephone: 800.592.2210
Website: www.mentor.com
SupportNet: supportnet.mentor.com/
Send Feedback on Documentation: supportnet.mentor.com/user/feedback_form.cfm

Table of Contents

List of Tables

Chapter 1
Introduction

Naming Conventions

ModuleWare parts use the following conventions for port naming and location:

- Data input ports for logical parts usually start with **din**.
- The data output ports for logical parts usually start with **dout**.
- The control (enable/disable) ports for tristates and memory parts include the string **ena** in their names.
- The clock port is named **clk**.
- The clock polarity is controlled by enumerated parameter clk_type which can have the values (Rising, Falling) for Verilog or (Rising, Falling, RisingLast, FallingLast, RisingEdge, FallingEdge) for VHDL.
- The clock enable port (gated clock) is named **clk_en**.
- The polarity of input port **clk_en** is controlled by enumerated parameter clk_en_type (ActiveHigh, ActiveLow).
- The reset port is named **rst**.
- The **rst** port is controlled by enumerated parameter rst_type (SyncActiveHigh, AsyncActiveHigh, SyncActiveLow, AsyncActiveLow).
- The set port is named **set**.
- The **set** port is controlled by enumerated parameter set_type (SyncActiveHigh, AsyncActiveHigh, SyncActiveLow, AsyncActiveLow).
- The clear port is named **clr**.
- The **clr** port is controlled by enumerated parameter clr_type (SyncActiveHigh, AsyncActiveHigh, SyncActiveLow, AsyncActiveLow).
- The preset port is named **pre**.
- The **pre** port is controlled by enumerated parameter pre_type (SyncActiveHigh, AsyncActiveHigh, SyncActiveLow, AsyncActiveLow).
- The enable port is named **enable**.
- The **enable** port is controlled by enumerated parameter enable_type (SyncActiveHigh, AsyncActiveHigh, SyncActiveLow, AsyncActiveLow).
- The load port is named **load**.
- The **load** port is controlled by enumerated parameter load_type (SyncActiveHigh, AsyncActiveHigh, SyncActiveLow, AsyncActiveLow).
- The gate port is named **gate**.

- The **gate** port is controlled by enumerated parameter gate_type (ActiveHigh and ActiveLow).

The behavior of these ports and parameters is explained later in this manual.

There are some exceptions to these conventions. For example, the input port is **d** and the output ports are **q** and **qb** on the D Flip-Flop (adff) part. These are used because they are more familiar to many users when describing D-type flip-flops. The *Register* parts use different names for the input ports depending on their functionality and use **q** and **qb** for the output ports.

A clock port is indicated by  and the presence or absence of an  indicator shows the polarity of a port.

The following convention is observed for the locations of ports on the symbols or instances:

- The input data ports are always on the left.
- The output data ports are always on the right.
- The control input ports may be on the top, bottom or adjacent to the data input ports they control.
- The status output ports may be on the right or the bottom.
- Note that if the symbol is rotated the directional sense changes.

Notes

- The default value for the controlling parameter <*name*>*_type* is usually ActiveHigh. See the individual parts for part-specific defaults.
- The default value for parameter *rst_type* is AsyncActiveHigh.
- The default value for parameter *set_type* is AsyncActiveHigh.
- The default value for parameter *clr_type* is AsyncActiveHigh.
- The default value for parameter *pre_type* is AsyncActiveHigh.
- The default value for parameter *enable_type* is AsyncActiveHigh.
- The default value for parameter *load_type* is AsyncActiveHigh.
- The default value for parameter *gate_type* is ActiveHigh.
- The default value for parameter *clk_type* is Rising.
- The port names are used in the generated HDL and if a module is embedded in-line you may need to manually update changes to the signal names.

Supported VHDL Packages and Types

The ModuleWare parts support the use of either *std_logic_arith* or *numeric_std* with the package *std_logic_1164*.

An error is issued during HDL generation if any other package is used.

The ModuleWare parts also support signals of types (*std_logic*, *std_logic_vector*, *std_ulogic*, *std_ulogic_vector*, *signed*, or *unsigned*) connected to any of its ports. A warning level one is issued if a signal of a different type other than these is connected to any of the ports.

Polarity Control

In general, polarity control is provided for all control and status ports but not for data ports except for all ports on the *Logic* parts.

The polarity for a port is specified by a control parameter. The naming convention for this parameter is as follows:

- If the port is named <*name*>, the parameter is named <*name*>*_type*.
- Most of the polarity-control parameters are enumerated with values of ActiveHigh or Rising (positive polarity) and ActiveLow or Falling (negative polarity).

The polarity of a port is shown by an indicator ● on the symbol or instance. Thus, if the value of the parameter is set to ActiveLow, an indicator appears on the port. If the value is set to ActiveHigh, the indicator disappears on the port.

You are advised not to edit the control parameter of an unconnected port, because this can modify the default operation. The polarity of the port is implemented in the generated HDL code.

Function

<*name*>*_type* = ActiveHigh <*name*> has positive polarity	<*name*>*_type* = ActiveLow <*name*> has negative polarity (shown as ●)

Notes

- The default value for the control parameter <*name*>*_type* is usually ActiveHigh. Please see individual parts for part-specific defaults.
- The control parameters of the **rst** and the **clk** ports on the *Sequential* parts have different enumerated sets of values which are covered in later sections.

Optional Ports

Optional ports are listed in the ModuleWare Parameters dialog box. The HDL for these ports is optimized away when the HDL is generated. You can explicitly remove optional scalar ports by setting the *<name>_type* parameter to None. Optional vector ports may have an enumerated *<name>_type* parameter (Enabled, Disabled) which can be set to Disabled.

Reset Behavior

Reset functionality is available on all of the *Sequential* parts that use the **rst** input port. A part can be reset asynchronously or synchronously. A part can be reset with a value of 1 or 0 on the **rst** port. This behavior of the **rst** port is controlled by the parameter *rst_type*.

- For all parts with a **rst** port with the exception of the Bank of Latches (latch), *rst_type* is enumerated with the values SyncActiveHigh, AsyncActiveHigh, SyncActiveLow and AsyncActiveLow.
- If the value of *rst_type* is SyncActiveHigh, the part gets reset synchronously when the value of input port **rst** is 1.
- If the value of *rst_type* is AsyncActiveHigh, the part gets reset asynchronously when the value of input port **rst** is 1.
- If the value of *rst_type* is SyncActiveLow, the part gets reset synchronously when the value of input port **rst** is 0.
- If the value of *rst_type* is AsyncActiveLow, the part gets reset asynchronously when the value of input port **rst** is 0.
- If the reset is asynchronous (*rst_type* has either the value AsyncActiveHigh or the value AsyncActiveLow), the scalar input port **rst** has a priority higher than the clock port (**clk**).
- The **rst** port is not affected by scalar input port **clk_en** if *rst_type* is asynchronous.
- If the reset is synchronous (*rst_type* is SyncActiveHigh or SyncActiveLow), the scalar input port **rst** has a priority lower than the clock port (**clk**).
- Scalar input port **clk_en** indirectly deactivates the **rst** port if *rst_type* is synchronous.
- The parameter *rst_type* for the Bank of Latches (latch) part is enumerated with values ActiveHigh (positive polarity) and ActiveLow (negative polarity).

The polarity indication (or no) for a **rst** port on a block diagram is dynamically set by the parameter *rst_type*. If the value is set to SyncActiveLow or AsyncActiveLow, a inverted signal indicator appears on the port. If the value is set to SyncActiveHigh or AsyncActiveHigh, the indicator disappears.

You are advised not to edit the value of the parameter *rst_type* if port **rst** is not connected because this can modify the default operation. The polarity of the **rst** port is implemented in the generated HDL code.

Most of the sequential parts have a parameter to specify the value to which the part is reset. This parameter is always named *rst_val*.

Note that this operation deviates from the traditional method of having a **clear** and **set** port, where **clear** puts in the value of 0 and **set** puts the value of 1. The **rst** port in the ModuleWare parts has been parameterized to handle both these functions but the effect of this is that the user is forced to choose one value. This was done to avoid having too many parameters because the majority of designs require only one **rst** port.

The reset function sets the value of the register to a particular value (*rst_val*) which means that, in the post-synthesis netlist, some of these inferred registers could have a **clear** port and others a **set** port. All zeros and all ones are a subset of this value. If the value of the parameter *rst_val* exceeds the maximum allowed by the register, the lower bits are chosen and a warning is issued for such cases. A synchronous reset could also be implemented with logical gates connected to the input of the sequential part.

Function

rst_type = SyncActiveHigh	synchronous, positive polarity
rst_type = AsyncActiveHigh	asynchronous, positive polarity
rst_type = SyncActiveLow	synchronous, negative polarity ( )
rst_type = AsyncActiveLow	asynchronous, negative polarity ( )

Notes

- The default value for the parameter *rst_type* is AsyncActiveHigh.
- Leave the port **rst** unconnected if not used and do not change the value of parameter *rst_type*.
- The **rst** port is always on the top of the symbol.

Reset/Clear Behavior

This functionality is available only on the *Register* parts using the **rst** or **clr** input ports. A part can be reset or cleared asynchronously or synchronously. A part could be reset or cleared with a value of 0 or 1 on the **rst** or **clr** port. This behavior of the **rst** and **clr** ports is controlled by the parameters *rst_type* and *clr_type*.

- For all the parts with a **rst**/**clr** port, the parameters *rst_type* and *clr_type* are enumerated with the values AsyncActiveHigh, SyncActiveHigh, AsyncActiveLow and SyncActiveLow.

- If the value of *rst_type/clr_type* is AsyncActiveHigh, the part gets reset/cleared asynchronously when the value of input port **rst**/**clr** is 1.
- If the value of *rst_type/clr_type* is SyncActiveHigh, the part gets reset/cleared synchronously when the value of input port **rst**/**clr** is 1.
- If the value of *rst_type/clr_type* is AsyncActiveLow, the part gets reset/cleared asynchronously when the value of input port **rst**/**clr** is 0.
- If the value of *rst_type/clr_type* is SyncActiveLow, the part gets reset/cleared synchronously when the value of input port **rst**/**clr** is 0.
- If the value of *rst_type/clr_type* is set to None, the functionality of these ports is disabled.

The polarity indication (or no or hidden) for the port **rst**/**clr** is dynamically set by the *rst_type/clr_type* parameter. If the value is set to AsyncActiveLow or SyncActiveLow, an inverted signal indicator appears on the port.

If the value is set to AsyncActiveHigh or SyncActiveHigh, the indicator disappears. If the value is set to None, the pin is hidden and is not visible in the symbol. You are advised not to edit the value of the *rst_type* or *clr_type* parameter if port **rst** or **clr** is not connected because this can modify the default operation. The polarity of the **rst**/**clr** port is implemented in the generated HDL code.

All the parts that use these ports have a parameter *rst_val/clr_val* to specify the value to which the part is reset/cleared. The reset/clear function sets the value of the register to a particular value (*rst_val/clr_val*). All zeros and all ones are a subset of this value. If the value of the parameter *rst_val/clr_val* exceeds the maximum allowed by the register, the lower bits are chosen and a warning is issued for such cases.

Function

rst_type/clr_type = AsyncActiveHigh	asynchronous, positive polarity
rst_type/clr_type = SyncActiveHigh	synchronous, positive polarity
rst_type/clr_type = AsyncActiveLow	asynchronous, negative polarity ( )
rst_type/clr_type = SyncActiveLow	synchronous, negative polarity ( )

Notes

- The default value for the parameter *rst_type/clr_type* is AsyncActiveHigh.
- Leave the port **rst/clr** unconnected if not used and do not change the value of parameter *rst_type/clr_type*.
- The **rst/clr** port is always on the top of the symbol.

Set/Preset Behavior

This functionality is available only on the *Register* parts that use the **set** or **pre** input port. A part can be set or preset asynchronously or synchronously. A part can be set or preset with a value of 1 or 0 on the **set** or **pre** ports. This behavior of the **set** and **pre** ports is controlled by the *set_type* and *pre_type* parameters.

- For all the parts with a **set/pre** port, the parameters *set_type* and *pre_type* are enumerated with the values AsyncActiveHigh, SyncActiveHigh, AsyncActiveLow and SyncActiveLow.
- If the value of *set_type/pre_type* is AsyncActiveHigh, the part gets set/preset asynchronously when the value of input port **set**/**pre** is 1.
- If the value of *set_type/pre_type* is SyncActiveHigh, the part gets set/preset synchronously when the value of input port **set/pre** is 1.
- If the value of *set_type/pre_type* is AsyncActiveLow, the part gets set/preset asynchronously when the value of input port **set/pre** is 0.
- If the value of *set_type/pre_type* is SyncActiveLow, the part gets set/preset synchronously when the value of input port **set/pre** is 0.
- If the value of *set_type/pre_type* is set to None, the functionality of these ports is disabled.

The polarity indication (or no or hidden) for the port **set**/**pre** is dynamically set by the *set_type/pre_type* parameter. If the value is AsyncActiveLow or SyncActiveLow, an inverted signal indicator appears on the port.

If the parameter is set to AsyncActiveHigh or SyncActiveHigh, the indicator disappears. If the value is set to None, the pin is hidden and is not visible in the symbol. You are advised not to edit the value of the parameter *set_type/pre_type* if port **set**/**pre** is not connected because this can modify the default operation. The polarity of the **set**/**pre** port is implemented in the generated HDL code.

All the parts that use these ports have a parameter *set_val/pre_val* to specify the value to which the part is set/preset. The set/preset function sets the value of the register to a particular value (*set_val/pre_val*). All zeros and all ones are a subset of this value. If the value of the parameter *set_val/pre_val* exceeds the maximum allowed by the register, the lower bits are chosen and a warning is issued for such cases.

Function

set_type/preset_type = AsyncActiveHigh	asynchronous, positive polarity
set_type/preset_type = SyncActiveHigh	synchronous, positive polarity
set_type/preset_type = AsyncActiveLow	asynchronous, negative polarity (●)
set_type/preset_type = SyncActiveLow	synchronous, negative polarity (●)

Notes

- The default value for the parameter *set_type/pre_type* is AsyncActiveHigh.
- Leave the port **set/pre** unconnected if not used and do not change the value of parameter *set_type/pre_type*.
- The **set/pre** port is always on the top of the symbol.

Clock Behavior

Sequential behavior can be triggered on the rising or falling edge of the clock. The behavior of the **clk** port is controlled by the *clk_type* parameter.

For Verilog, the parameter *clk_type* is enumerated with values Rising and Falling.

For VHDL, the extra values RisingLast, FallingLast, RisingEdge, FallingEdge support actions sensitive to the last value or to the rising or falling edge.

If the value of *clk_type* is Rising the part is triggered if an event occurs on the rising edge of the **clk** port. If the value of *clk_type* is Falling, the part is triggered if an event occurs on the falling edge of the **clk** port.

The polarity indication (● or no ● or hidden) for the **clk** port is dynamically set by the parameter *clk_type*. If the value is set to Falling, then an inverted signal indicator appears on the port. If the value is set to Rising, the indicator disappears.

You are advised not to edit the value of the parameter *clk_type* if port **clk** is not connected because this can modify the default operation. The polarity of the **clk** port is implemented in the generated HDL code.

Function

clk_type = Rising	rising edge event, positive polarity
clk_type = Falling	falling edge event, negative polarity ( )
clk_type = RisingLast	rising edge event, last value low, positive
clk_type = FallingLast	falling edge event, last value high, negative polarity ( )
clk_type = RisingEdge	rising edge, positive polarity
clk_type = FallingEdge	falling edge, negative polarity ( )

These functions correspond to the following VHDL code:

Rising	If (clk'EVENT AND clk='1')
Falling	If (clk'EVENT AND clk='0')
RisingLast	If (clk'EVENT AND clk='1' AND clk'LAST_VALUE='0')
FallingLast	If (clk'EVENT AND clk='0' AND clk'LAST_VALUE='1')
RisingEdge	If (RISING_EDGE(clk))
FallingEdge	If (FALLING_EDGE(clk))

Notes

- The default value for the parameter *clk_type* is Rising.
- The **clk** input port is always on the lower part of the left of the symbol and has a clock port indicator .

Gate Behavior

All the latches in the *Register* parts category have a gate port named **gate**. The sequential behavior could be triggered on the high level of the gate or the low level of the gate. The behavior of the **gate** port is controlled by the *gate_type* parameter.

For all the parts with a **gate** port, the parameter *gate_type* is enumerated with values *ActiveHigh* and *ActiveLow*. If *gate_type* is ActiveHigh, the part gets triggered on the high level of the **gate** port. If *gate_type* is ActiveLow, the part gets triggered on the low level of the **gate** port.

The polarity indication (or no or hidden) for the **gate** port is dynamically set by the parameter *gate_type*. If the value is set to ActiveLow, then an inverted signal indicator appears on the port. If the value is set to ActiveHigh, the indicator disappears.

You are advised not to edit the value of the parameter *gate_type* if port **gate** is not connected because this can modify the default operation. The polarity of the **gate** port is implemented in the generated HDL code.

Function

gate_type = ActiveHigh	high level trigger, positive polarity
gate_type = ActiveLow	low level trigger, negative polarity ( )

Notes

- The default value for the parameter *gate_type* is ActiveHigh.
- The **gate** input port is always on the lower part of the left of the symbol and has a clock port indicator .

Clock Enable Behavior

All the *Sequential* parts have a clock enable port named **clk_en**. The behavior of the **clk_en** port is controlled by the *clk_en_type* parameter.

The parameter *clk_en_type* is enumerated with values ActiveHigh and ActiveLow. If the value of *clk_en_type* is ActiveHigh, the **clk** port is enabled if the value of **clk_en** is 1. If the value of *clk_en_type* is ActiveLow, the **clk** port is enabled if the value of **clk_en** is 0. The HDL code for **clk_en** is optimized away if it is not connected.

The polarity indication (or no or hidden) for the **clk_en** port is dynamically set by the parameter *clk_en_type*. If the value is set to ActiveLow, an inverted signal indicator appears on the port. If the value is set to ActiveHigh, the indicator disappears.

You are advised not to edit the value of parameter *clk_en_type* if the port **clk_en** is not connected because this can modify the default operation. The polarity of the **clk_en** port is implemented in the generated HDL code.

The port **clk_en** has special significance for synthesis tools. Usage of this port is extremely costly. Most of the time it forces the synthesis tools to pick up specialized flip-flops that are very large, this reduces the optimization space for synthesis and heavily constrains the synthesis result.

It is usually better to use a clock divider and drive the clock of a section (slower part) of the circuit than to connect up the **clk_en** port. Care should be taken before using **clk_en**. For example, this port should not be used to cascade counters.

As explained in the Configurable Counter (cntr), the ports **load** and **din** should be used to cascade counters. For in-lined HDL code, leaving the port **clk_en** unconnected will suffice because the default driver will optimize this away. For instantiated HDL, you lose the ability to synthesize that instance separately because the default driver is part of the HDL part for the block diagram.

Function

clk_en_type = ActiveHigh	**high level enable,** positive polarity
clk_en_type = ActiveLow	**low level enable**, negative polarity ( )

Notes

- The default value for the parameter *clk_en_type* is ActiveHigh.
- Leave the port **clk_en** unconnected if not used and do not change the value of the parameter *clk_en_type*.
- The **clk_en** port is always on the bottom of the symbol.

Enable/Load Behavior

All the *Register* parts have an enable or load port **enable** or **load** which is used as a control input.

The behavior of the **enable** or **load** port is controlled by parameters *enable_type* or *load_type* which are enumerated with values AsyncActiveHigh, SyncActiveHigh, AsyncActiveLow, SyncActiveLow and None.

The polarity indication (or no or hidden) for the **enable** or **load** port is dynamically set by the *enable_type* or *load_type* parameter. If the value is set to AsyncActiveLow or SyncActiveLow, an inverted signal indicator appears on the port. If the value is set to AsyncActiveHigh or SyncActiveHigh, the indicator disappears. If the value is set to None, the port is hidden and is not visible in the symbol.

You are advised not to edit the value of the *enable_type* or *load_type* parameter if **enable** or **load** is not connected because this can modify the default operation.

The **enable** or **load** port polarity is implemented in the generated HDL code.

Function

enable_type/load_type = AsyncActiveHigh	asynchronous, positive polarity.
enable_type/load_type = SyncActiveHigh	synchronous, positive polarity
enable_type /load_type = AsyncActiveLow	asynchronous, negative polarity ( )
enable_type / load_type = SyncActiveLow	synchronous, negative polarity ( )

Notes

- The default value for the parameters *enable_type* and *load_type* is AsyncActiveHigh.
- Leave the port **enable/load** unconnected or set *enable_type/load_type* to None if not used.

- The **enable/load** port is always on the left of the symbol.

Arithmetic Mode

All of the *Arithmetic* parts support both signed and unsigned operations.

For Verilog, the mode is controlled by an enumerated parameter *sign_type* (Unsigned, Signed). If the value of the parameter *sign_type* is Unsigned, then input data bits are interpreted as unsigned numbers. If the value of the parameter *sign_type* is Signed, then input data bits are interpreted as signed numbers. The Verilog code is written to interpret the bits in the respective modes.

For VHDL, you can use either the *std_logic_arith* or *numeric_std* packages. Mixing and matching of different data types is allowed not only in the arithmetic operations but in all other ModuleWare categories as well. Automatic proper type casting or conversion operations take place during generation to guarantee that the arithmetic operation is done as expected, and depending on the types of the inputs or outputs and the package being used, according to the rules listed in “Mixed Type In/Out Ports” on page 31.

If the package being used does not support an operation between particular operand types, then the mode is controlled by the *sign_type* parameter. All the operands will be type casted or converted using this type before the arithmetic operation takes place.

The value of the parameter *sign_type* (Unsigned, Signed) is shown on the symbol in a block diagram.

Notes

- The default value for the parameter *sign_type* is Unsigned.

Shifter Mode

All the shift operations can be done in three modes. This mode is controlled by an enumerated parameter *mode* (Logical, Arithmetic, Circular). A logical shift operation shifts in 0 (both left and right). An arithmetic shift operation shifts in 0 for left shift and shifts in the most significant bit for right shift. A circular shift operation shifts in the least significant bit for a right shift and the most significant bit for a left shift.

It is important to note that the hardware cost also depends upon the width of the **shift** port (or shift value in the case of the Fixed Shifter (fixshift) part). A logical left shift is the same as an arithmetic left shift.

In general, you are advised to reduce the width of the **shift** port to the exact required number. Shifters may also be implemented outside the synthesis tools. If you have access to silicon compilers that do a better job, you may want to skip synthesis for shifters. No visual feedback is available in the block diagram for the parameter *mode*.

Notes

- The default value for the parameter *mode* is Logical.

VHDL Coding Style

All parts support the nine-value multi-valued logic style (MVL9) which is used by the IEEE *std_ulogic* and *std_logic* types. MVL9 supports the nine values (0, 1, L, H, X, -, U, W and Z).

Many parts can use the alternative four-value (MVL4) style which is used by one particular subtype of *std_ulogic* and supports the four values (0, 1, X and Z) only. The MVL4 style can be set as a preference in the **Style** tab of the VHDL Options dialog box as described in the "Setting Preferences" section of the *HDL Designer Series User Manual*.

The following parts support the alternative VHDL coding styles:

acc	dff	latch	ram2p	shiftps	tribus
addsub	dlatch	lshift	ram	shiftsp	triff
adff	fifo	modcntr	regfile	stack	triinv
alu181	incdec	mux2	rsff	tff	varshift
clkdiv	jkff	mux4	rshift	tlatch	
cntr	jklatch	mux8	rslatch	tribuf	

Signals and Variables

Many of the ModuleWare parts use an internal variable in the generated VHDL. For example, the Multiplier (mult) normally uses the internal variables *dtemp*, *temp0* and *temp1* in the default generated code:

```
I0combo : PROCESS (din0, din1)
VARIABLE dtemp : unsigned(1 DOWNTO 0);
VARIABLE temp0 : unsigned(0 DOWNTO 0);
VARIABLE temp1 : unsigned(0 DOWNTO 0);
BEGIN
   temp0(0) := din0;
   temp1(0) := din1;
   dtemp := (temp0 * temp1);
   prod <= dtemp(0);
END PROCESS I0combo;
```

You can set a VHDL style preference to use the actual signal names instead of internal variables. For the multiplier example, the default generated VHDL would be:

```
I0combo : PROCESS (din0, din1)
BEGIN
   mw_I0dtemp <= (unsigned(conv_std_logic_vector(din0,1)) *
   unsigned(conv_std_logic_vector(din1,1)));
END PROCESS I0combo;
prod <= mw_I0dtemp(0);
```

```
END PROCESS I0combo;
```

This option can be used with the following parts:

Table 1-1. ModuleWare Parts Using Signals

Category	Part
Arithmetic	181 ALU (alu181) Accumulator (acc) Adder (add) Adder Subtractor (addsub) Comparator (cmp) Decrementer (dec) Incrementer (inc) Incrementer Decrementer (incdec) Multiplier (mult) Subtractor (sub) Uni-function Comparator (comp)
Memory	First In First Out (fifo) Stack (stack)
Sequential	Parallel to Serial Shifter (shiftps) Serial to Parallel Shifter (shiftsp)

Verilog Coding Style

You can choose whether the Verilog generated for ModuleWare parts uses blocking or non-blocking assignments in the **Style** tab of the Verilog Options dialog box.

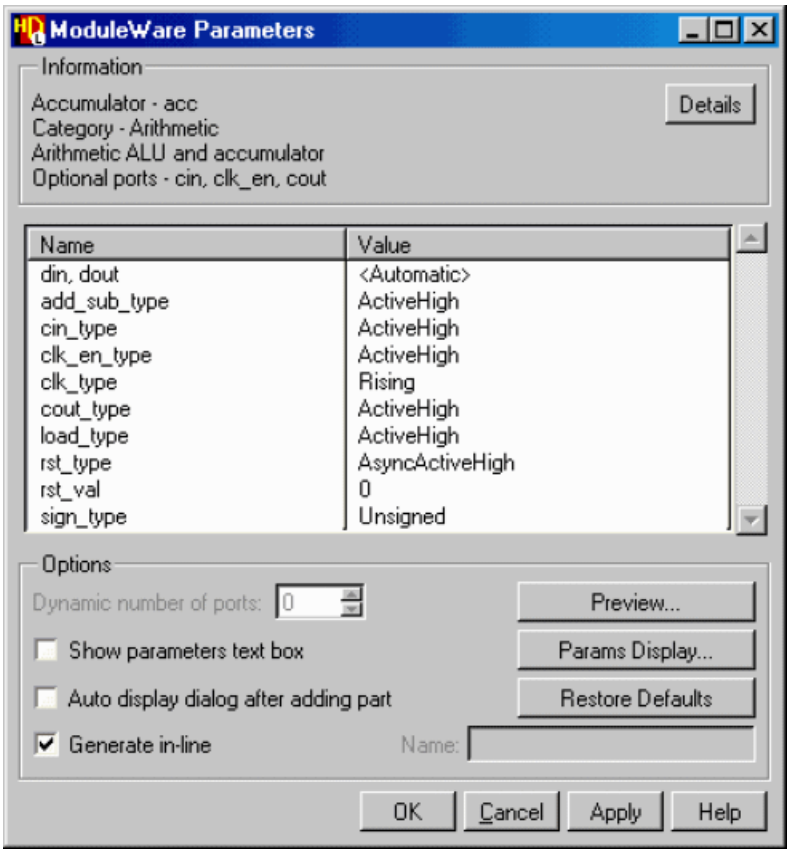
Signal Name Prefix in Generated HDL

You can set VHDL and Verilog style preferences for the signal name prefix used by ModuleWare signals in the generated HDL.

Setting Parameters

You can set parameters for a ModuleWare part in the ModuleWare Parameters dialog box as described in the “Editing ModuleWare Parameters” section of the *Graphical Editors User Manual*.

For example the following dialog box is displayed for the Accumulator (acc) part:



Note that special dialog boxes are used for the Stimulus parts. These dialog boxes are described in “Stimulus Parts” on page 261.

Design Rule Checking

Design rule checks (DRC) are performed when you generate HDL for a view containing instantiated ModuleWare. There are three design rule check categories:

- **Error:** Errors that stops the generation of the whole design.
- **Warning Level 1:** Warnings that stops the generation of the ModuleWare part but do not stop the generation of the whole design.
- **Warning level 2:** Warnings that display messages but do not stop the generation of either the whole design or the ModuleWare part.

Note that checking initially tests that all required ports have been connected and issues warning messages so that other diagram checks can be performed before you have completed connections to the ModuleWare parts. Once, all required ports have been connected, error messages arising from any invalid parameter values may be issued the next time that the HDL is generated.

Automatic Detection of Port Width

ModuleWare parts automatically detect the width of the buses connected to any of their ports.

Explicit setting of the width of the connected port is also allowed through setting the corresponding port parameter in the ModuleWare Parameters dialog box. There is no limitation on the size of the data bus that can be connected to the data ports on a ModuleWare part.

Dynamic Number of Ports

You can dynamically change the number of ports provided on many of the logic and combinatorial parts, for example the N-Input AND Gate (and), by dragging the symbol.

You can also set the number of ports on these parts by setting the **Dynamic number of ports** option in the ModuleWare Parameter dialog box.

Mixed Type In/Out Ports

Mix and match of different data types for input and output ports are allowed. Automatic proper type casting and conversion operations take place during HDL code generation to guarantee that the required operation is done correctly.

The following rules are used for type casting and conversion during the generation of the VHDL code.

Logic Operations

All the input signals are type casted or converted to the most common input type.

If the *numeric_std* package is used, the logic operation is performed directly and the logic expression is type casted or converted from the most common input type to the output type.

If the *std_logic_arith* package is used, the most common type is checked first to be either *std_logic_vector* or *std_ulogic_vector*, then all of the input signals are type casted or converted to this type, the logic operation is performed and the logic expression is type casted or converted to the output type.

Arithmetic Operations

The *numeric_std* package supports the operators (+,-,/,*) between two signals which are both *signed* or both *unsigned*. Thus, if both the arithmetic operation operands are *signed* or both operands are *unsigned*, the arithmetic operation takes place without any type casting ignoring the *sign_type* parameter.

Otherwise the two signals are type casted or converted to the *sign_type* before arithmetic operation takes place. There is no VHDL ambiguity in the output type because the resulting type is signed if both signals are *signed* or *unsigned* if both signals are *unsigned*.

The *std_logic_arith* package supports the operators (+,-,*) between two signals where any one can be signed or unsigned. Thus, if the type of the two signals is either *signed* or *unsigned*, the arithmetic operation takes place without any type casting ignoring the *sign_type* parameter.

Otherwise the two signals are type casted or converted to the *sign_type* before arithmetic operation takes place. If both of the input signals are *unsigned*, the resulting type will be *unsigned* or *std_logic_vector*. Otherwise the resulting type is *signed* or std_logic_vector. To resolve the ambiguity, signed or *std_logic_vector* qualifiers are inserted in the code when appropriate.

Comparison Operations

The *numeric_std* package supports comparison between two *signed* or two *unsigned* signals. Thus, if both of the signals to be compared are *signed* or both are *unsigned*, the comparison operation takes place without any type casting and ignoring the *sign_type* parameter. Otherwise the two signals are type casted or converted to the *sign_type* before comparison takes place.

The *std_logic_arith* package supports comparison between two signals where any one can be *signed* or *unsigned*. Thus, if the type of the two signals is either *signed* or *unsigned*, the comparison operation takes place without any type casting and ignoring the *sign_type* parameter. Otherwise the two signals are type casted or converted to the *sign_type* before comparison takes place.

Slice Support

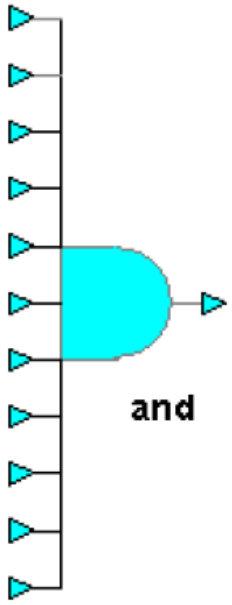
For both VHDL and Verilog, ModuleWare supports connecting one-dimensional slices (of any start/stop range) to any of its models ports as long as the slice width complies with the expected input data width of the part.

Connecting two-dimensional slices to any of the ModuleWare ports is not currently supported.

Chapter 2
Logic Parts

This chapter describes parts supporting the following logic gates and buffers.

N-Input AND Gate (and)



This part performs a logical AND of input ports **din0** to **dinN**. Each bit of output port **dout** is the AND of the corresponding bits of ports **din0** to **dinN**.

Ports are automatically added or removed when you resize the component instance to implement any number of input ports.

This part is equivalent to *n* N-bit AND gates where *n* is the port width and N is the number of input ports.

Setting the **dout** port polarity ActiveLow with all input ports ActiveHigh, results in a N-Input NAND gate.

Function

dout = **din0** AND **din1** AND **din2 …** AND **dinN**

Truth Table

The truth table would contain 2^N rows with **dout** = 0 for all rows except the row with all N bits of the bus = 1. For negative polarity, all rows in the table would have **dout** = 1 except when all N bits of the bus = 0.

Table 2-1. N-Input AND Gate Truth Table

din0(i)	din1(i)	…	dinN(i)	dout(i)
0	0	0	0	0
all other rows				0
1	1	1	1	1

Parameters

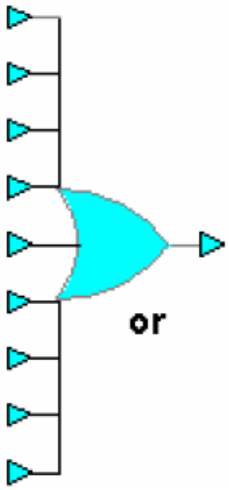
Table 2-2. N-Input AND Gate Parameters

Parameter	Values	Default
din0, dout	Port widths (must be > 0)	Automatic
din0_type to dinN_type, dout_type	ActiveHigh, ActiveLow	ActiveHigh

Design Rule Checks

- An error is issued if any of the ports **din0** to **dinN** and **dout** do not have the same width or if the width of any port cannot be determined.
- A warning is issued and HDL generation fails for this part unless the output port and at least two input ports are connected.

N-Input OR Gate (or)



This part performs a logical OR of input ports **din0** to **dinN**. Each bit of output port **dout** is the OR of the corresponding bits of ports **din0** to **dinN**.

Ports are automatically added or removed when you resize the component instance to implement any number of input ports.

This part is equivalent to *n* N-bit OR gates where *n* is the port width and N is the number of input ports.

Setting the **dout** port polarity ActiveLow with all input ports ActiveHigh, results in an N-Input NOR gate.

Function

dout = **din0** OR **din1** OR **din2 …** OR **dinN**

Truth Table

The truth table would contain 2^N rows with **dout** = 1 for all rows except the row with all N bits of the bus = 0. For negative polarity, all rows in the table would have **dout** = 0 except when all N bits of the bus = 1.

Table 2-3. N-Input OR Gate Truth Table

din0(i)	din1(i)	…	dinN(i)	dout(i)
0	0	0	0	0
All other rows				1
1	1	1	1	1

Parameters

Table 2-4. N-Input OR Gate Parameters

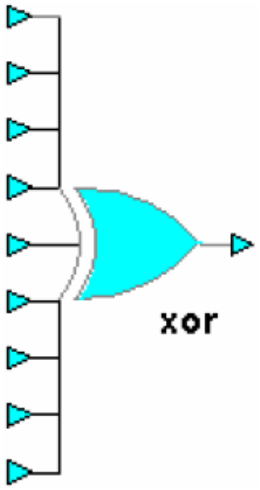
Parameter	Values	Default
din0, dout	Port widths (must be > 0)	Automatic
din0_type to dinN_type, dout_type	ActiveHigh, ActiveLow	ActiveHigh

Design Rule Checks

- An error is issued if any of the ports **din0** to **dinN** and **dout** do not have the same width or if the width of any port cannot be determined.
- A warning is issued and HDL generation fails for this part unless the output port and at least two input ports are connected.

N-Input XOR Gate (xor)

This part performs a logical XOR of input ports **din0** to **dinN**. Each bit of output port **dout** is the XOR of the corresponding bits of ports **din0** to **dinN**.



Ports are automatically added or removed when you resize the component instance to implement any number of input ports.

This part is equivalent to *n* N-bit XOR gates where *n* is the port width and N is the number of input ports.

Setting the **dout** port polarity ActiveLow with all input ports ActiveHigh, results in an N-Input XNOR gate.

Function

dout = **din0** XOR **din1** XOR **din2 …** XOR **dinN**

Parameters

Table 2-5. N-Input XOR Gate Parameters

Parameter	Values	Default
din0, dout	Port widths (must be > 0)	Automatic
din0_type to dinN_type, dout_type	ActiveHigh, ActiveLow	ActiveHigh

Design Rule Checks

- An error is issued if any of the ports **din0** to **dinN** and **dout** do not have the same width or if the width of any port cannot be determined.
- A warning is issued and HDL generation fails for this part unless the output port and at least two input ports are connected.

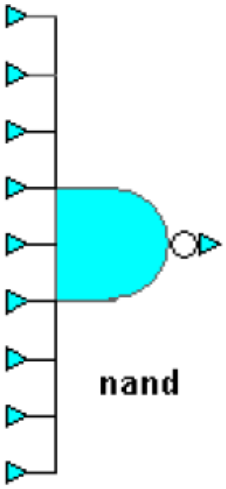
N-Input NAND Gate (nand)

This part performs a logical NAND of input ports **din0** to **dinN**. Each bit of output port **dout** is the NAND of the corresponding bits of ports **din0** to **dinN**.

Ports are automatically added or removed when you resize the component instance to implement any number of input ports.

This part is equivalent to *n* N-bit NAND gates where *n* is the port width and N is the number of input ports.

Setting the **dout** port polarity ActiveLow with all input ports ActiveHigh, results in a N-Input AND gate.



Function

dout = **din0** NAND **din1** NAND **din2 … N**AND **dinN**

Truth Table

The truth table would contain 2^N rows with **dout** = 1 for all rows except the row with all N bits of the bus = 1. For negative polarity, all rows in the table would have **dout** = 0 except when all N bits of the bus = 0.

Table 2-6. N-Input NAND Gate Truth Table

din0(i)	din1(i)	…	dinN(i)	dout(i)
0	0	0	0	1
All other rows				1
1	1	1	1	0

Parameters

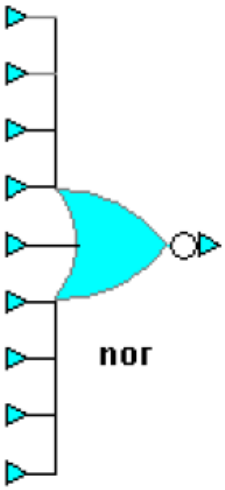
Table 2-7. N-Input NAND Gate Parameters

Parameter	Values	Default
din0, dout	Port widths (must be > 0)	Automatic
din0_type to dinN_type, dout_type	ActiveHigh, ActiveLow	ActiveHigh

Design Rule Checks

- An error is issued if any of the ports **din0** to **dinN** and **dout** do not have the same width or if the width of any port cannot be determined.
- A warning is issued and HDL generation fails for this part unless the output port and at least two input ports are connected.

N-Input NOR Gate (nor)



This part performs a logical NOR of input ports **din0** to **dinN**. Each bit of output port **dout** is the NOR of the corresponding bits of ports **din0** to **dinN**.

Ports are automatically added or removed when you resize the component instance to implement any number of input ports.

This part is equivalent to *n* N-bit NOR gates where *n* is the port width and N is the number of input ports.

Setting the **dout** port polarity ActiveLow with all input ports ActiveHigh, results in an N-Input OR gate.

Function

dout = **din0** NOR **din1** NOR **din2 …** NOR **dinN**

Truth Table

The truth table would contain 2^N rows with **dout** = 0 for all rows except the row with all N bits of the bus = 0. For negative polarity, all rows in the table would have **dout** = 1 except when all N bits of the bus = 1.

Table 2-8. N-Input NOR Gate Truth Table

din0(i)	din1(i)	…	dinN(i)	dout(i)
0	0	0	0	1
All other rows				0
1	1	1	1	0

Parameters

Table 2-9. N-Input NOR Gate Parameters

Parameter	Values	Default
din0, dout	Port widths (must be > 0)	Automatic
din0_type to dinN_type, dout_type	ActiveHigh, ActiveLow	ActiveHigh

Design Rule Checks

- An error is issued if any of the ports **din0** to **dinN** and **dout** do not have the same width or if the width of any port cannot be determined.
- A warning is issued and HDL generation fails for this part unless the output port and at least two input ports are connected.

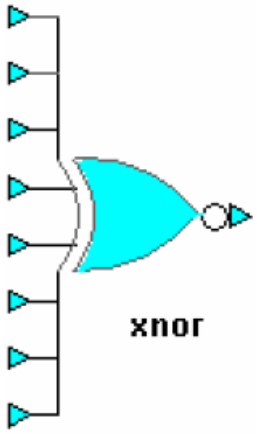
N-Input XNOR Gate (xnor)

This part performs a logical XNOR of input ports **din0** to **dinN**. Each bit of output port **dout** is the XNOR of the corresponding bits of ports **din0** to **dinN**.

Ports are automatically added or removed when you resize the component instance to implement any number of input ports.

This part is equivalent to *n* N-bit XNOR gates where *n* is the port width and N is the number of input ports.

Setting the **dout** port polarity ActiveLow with all input ports ActiveHigh, results in an N-Input XOR gate.



Function

dout = **din0** XNOR **din1** XNOR **din2 …** XNOR **dinN**

Parameters

Table 2-10. N-Input XNOR Gate Parameters

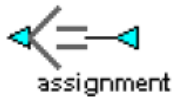
Parameter	Values	Default
din0, dout	Port widths (must be > 0)	Automatic
din0_type to dinN_type, dout_type	ActiveHigh, ActiveLow	ActiveHigh

Design Rule Checks

- An error is issued if any of the ports **din0** to **dinN** and **dout** do not have the same width or if the width of any port cannot be determined.
- A warning is issued and HDL generation fails for this part unless the output port and at least two input ports are connected.

Assign (assignment)

This part assigns the value on the input port **s** to the output port **t** with an optional delay in nanoseconds.



If you are using VHDL a type conversion or type casting string can also be specified.

Function

t = s
When all parameters have their default values, t is assigned the value of s:
t <= s ;(VHDL)
assign t = s ;(Verilog)

t = <type conversion string>**s**
You can specify a VHDL type conversion string or a type conversion function defined in a referenced VHDL package:
t <= type_conversion_string(s) ;

t = <type casting string>**s**
You can specify a VHDL type casting string:
t <= type_casting_string'(s) ;

t = <delay>**s**
You can enter an integer value specifying the delay in nanoseconds:
t <= s AFTER delay NS ;(VHDL)
assign #delay t = s ;(Verilog)

Parameters

Table 2-11. Assign Parameters

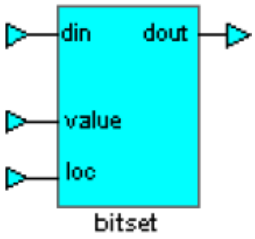
Parameter	Values	Default
type_conversion	Must be a valid type conversion string (for example *unsigned* or *conv_integer*)	null
type_cast	Must be a valid type casting string (for example *std_logic* when s has type *std_ulogic*	null
delay_value	Integer delay value	0

Design Rule Checks

- An error is issued if the width of any port cannot be determined.
- A warning is issued and HDL generation fails for this part if either of ports **s** or **t** are not connected.

Bit Setter (bitset)

This part takes the input port **din** and sets one of its bits to the value of the scalar port **value**. The location of this bit is determined by the input port **loc** and the result is placed in the output port **dout**. If the location given by the input port **loc** is more than the width of input port **din**, the value of port **din** is simply passed to **dout**. The location is zero-indexed.



This part is equivalent to a series of 2-bit multiplexers and decoding logic for port **loc**.

Function

For every bit of the bus:

dout = **din** (if location of the bit not equal to **loc**)
= **value** (if location of the bit = **loc**)

Truth Table

Table 2-12. Bit Setter Truth Table

din(i)	value	loc	dout(i)
0	0	not equal to i	0
1	0	not equal to i	1
0	1	not equal to i	0
1	1	not equal to i	1
0	0	= i	0
1	0	= i	0
0	1	= i	1
1	1	= i	1

Parameters

Table 2-13. Bit Setter Parameters

Parameter	Values	Default
din, dout, loc	Port width (must be > 0)	Automatic
value_type	ActiveHigh, ActiveLow	ActiveHigh

Design Rule Checks

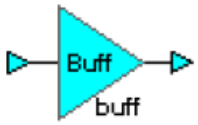
- An error is issued if the width of any port cannot be determined, port **value** has any value other than 1, ports **din** and **dout** do not have the same width or if the width of port **loc** exceeds the required width. For example, if port **din** is four bits wide, a width of two bits is

sufficient for input port **loc**. This reduces the chances of extra area cost in the decoder logic because it cannot be optimized away by synthesis tools.

- A warning is issued and HDL generation fails for this part if any port is not connected.

Buffer (buff)

This part buffers the input data from the input port **din** into the output port **dout**. Both the ports are multi-bit and can be given a width for buffering multiple bits.



This part has no hardware cost in terms of gates except for the inverting behavior if either *din_type* or *dout_type* are set to be ActiveLow.

Function

dout = **din**

Parameters

Table 2-14. Buffer Parameters

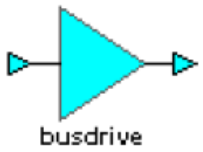
Parameter	Values	Default
din, dout	Port width (must be > 0)	Automatic
din_type, dout_type	ActiveHigh, ActiveLow	ActiveHigh

Design Rule Checks

- An error is issued if the width of any port cannot be determined or ports **din** and **dout** do not have the same width.
- A warning is issued and HDL generation fails for this part if either of ports **din** and **dout** are not connected.

Bus Driver (busdrive)

This part places the bits of input port **din** on output port **dout**. The starting index for the first bit of input port **din** is the value of parameter *loc*. This part has no hardware cost in terms of gates. It adds multiple drivers to a bus. Do not use if your design requires multiplexed bus logic.



Function

This part has multiple drivers. The first driver is:

dout = **'ZZ…'**

and the second driver is:

dout = **din** << *loc*

Any bits not driven by the second driver are set to Z.

Truth Table

Table 2-15. Bus Driver Truth Table

din(i)	loc	dout(i)
0	i < loc	Z
1	i < loc	Z
0	i >= (loc+length of din)	Z
1	i >= (loc+length of din)	Z
0	loc <= i < (loc + length of din)	0
1	loc <= i < (loc + length of din)	1

Parameters

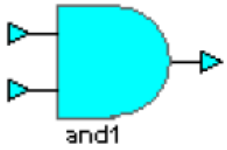
Table 2-16. Bus Driver Parameters

Parameter	Values	Default
din, dout	Port width (must be > 0)	Automatic
loc	Start index for **din** (must be > 0)	0

Design Rule Checks

- An error is issued if the width of any port cannot be determined.
- A warning is issued and HDL generation fails for this part if either of ports **din** and **dout** are not connected.

Gated AND (and1)



This part performs a logical AND of each bit of input port **din** with the scalar input port **ena**. The result is placed on the output port **dout**. This part allows a single control bit to AND an entire bus.

This part is equivalent to *n* 2-bit AND gates where *n* is the port width. If the width of **din** = 1, this part is equivalent to the N-Input AND Gate (and) with 2 inputs.

Function

dout = **din0** AND **ena** (For every bit of the output port **dout**)

Truth Table

This table is for positive polarity. For negative polarity, invert the values.

Table 2-17. Gated AND Truth Table

din(i)	ena	dout(i)
0	0	0
0	1	0
1	0	0
1	1	1

Parameters

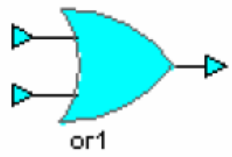
Table 2-18. Gated AND Parameters

Parameter	Values	Default
din, dout	Port width (must be > 0)	Automatic
din_type, ena_type, dout_type	ActiveHigh, ActiveLow	ActiveHigh

Design Rule Checks

- An error is issued if the width of any port cannot be determined, port **ena** does not have a fixed width of 1 or if ports **din0** and **dout** are not the same width.
- A warning is issued and HDL generation fails for this part if either of ports **din** and **dout** are not connected.

Gated OR (or1)



This part performs a logical OR of each bit of input port **din** with the scalar input port **ena**. The result is placed on the output port **dout**. This part allows a single control bit to OR a whole bus.

This part is equivalent to *n* 2-bit OR gates where *n* is the port width. If the width of **din** = 1, this part is equivalent to the N-Input OR Gate (or) with 2 input ports.

Function

dout = **din0** OR **ena** (For every bit of the output port **dout**).

Truth Table

This table is for positive polarity. For negative polarity, invert the values.

Table 2-19. Gated OR Truth Table

din(i)	ena	dout(i)
0	0	0
0	1	1
1	0	1
1	1	1

Parameters

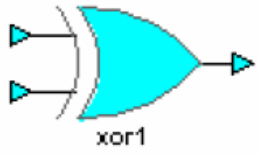
Table 2-20. Gated OR Parameters

Parameter	Values	Default
din, dout	Port widths (must be > 0)	Automatic
din_type, ena_type, dout_type	ActiveHigh, ActiveLow	ActiveHigh

Design Rule Checks

- An error is issued if the width of any port cannot be determined, port **ena** does not have a fixed width of 1 or if ports **din0** and **dout** are not the same width.
- A warning is issued and HDL generation fails for this part if either of ports **din** and **dout** are not connected.

Gated XOR (xor1)



This part performs a logical XOR of each bit of input port **din** with the scalar input port **ena**. The result is placed on the output port **dout**. This part allows a single control bit to XOR a whole bus.

This part is equivalent to *n* 2-bit XOR gates where *n* is the port width. If the width of **din** = 1, this part is equivalent to the N-Input XOR Gate (xor) with 2 input ports.

Function

dout = **din0** XOR **ena** (For every bit of the output port **dout**).

Truth Table

This table is for positive polarity. For negative polarity, invert the values.

Table 2-21. Gated XOR Truth Table

din(i)	ena	dout(i)
0	0	0
0	1	1
1	0	1
1	1	0

Parameters

Table 2-22. Gated XOR Parameters

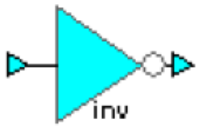
Parameter	Values	Default
din, dout	Port widths (must be > 0)	Automatic
din_type, ena_type, dout_type	ActiveHigh, ActiveLow	ActiveHigh

Design Rule Checks

- An error is issued if the width of any port cannot be determined, port **ena** does not have a fixed width of 1 or if ports **din0** and **dout** are not the same width.
- A warning is issued and HDL generation fails for this part if either of ports **din** and **dout** are not connected.

Inverter (inv)

This part inverts input port **din**. The result is placed on the output port **dout**. Each bit of port **dout** is the inverse of the corresponding bit of port **din**.



This part is equivalent to a buffer if either *din_type* or *dout_type* are set to be **ActiveLow.**

Function

dout = NOT(**din**)

Truth Table

Table 2-23. Inverter Truth Table

din(i)	dout(i)
0	1
1	0

Parameters

Table 2-24. Inverter Parameters

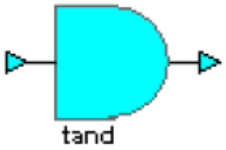
Parameter	Values	Default
din, dout	Port widths (must be > 0).	Automatic
din_type, dout_type	ActiveHigh, ActiveLow	ActiveHigh

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if ports **din0** and **dout** are not the same width.
- A warning is issued and HDL generation fails for this part if either of ports **din** and **dout** are not connected.

Reduction AND (tand)

This part performs a cascading AND of all the bits of input port **din**. The result is placed on the scalar output port **dout**. This part provides a single bit output. Port **dout** is set to 1 only if all the bits of the port **din** are 1.



This part is equivalent to *n*-1 2-bit AND gates where *n* is the port width.

Function

dout = AND (all bits of port **din)**

Truth Table

This table is for positive polarity. For negative polarity, invert the values.

Table 2-25. Reduction AND Truth Table

din	dout
All bits are 1 (NOT(din) = 0)	1
else	0

Parameters

Table 2-26. Reduction AND Parameters

Parameter	Values	Default
din	Port width (must be > 0)	Automatic
din_type, dout_type	ActiveHigh, ActiveLow	ActiveHigh

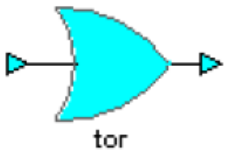
Design Rule Checks

- An error is issued if the width of any port cannot be determined or if ports **dout** does not have a width of 1.
- A warning is issued and HDL generation fails for this part if either of ports **din** and **dout** are not connected.

Reduction OR (tor)

This part performs a cascading OR of all the bits of input port **din**. The result is placed on the scalar output port **dout**. Port **dout** is set to 1 if one or more bits of the port **din** are 1.

This part is equivalent to *n*-1 2-bit OR gates where *n* is the port width.



Function

dout = OR (all bits of port **din**)

Truth Table

This table is for positive polarity. For negative polarity, invert the values.

Table 2-27. Reduction OR Truth Table

din	dout
All bits are 0 (din = 0)	0
else	1

Parameters

Table 2-28. Reduction OR Parameters

Parameter	Values	Default
din	Port width (must be > 0)	Automatic
din_type, dout_type	ActiveHigh, ActiveLow	ActiveHigh

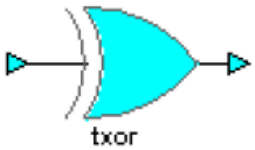
Design Rule Checks

- An error is issued if the width of any port cannot be determined or if ports **dout** does not have a width of 1.
- A warning is issued and HDL generation fails for this part if either of ports **din** and **dout** are not connected.

Reduction XOR (txor)

This part performs a cascading exclusive OR of all the bits of input port **din**. The result is placed on the scalar output port **dout**. Port **dout** is set to value 1 only if an odd number of bits of the port **din** are 1.

This part is equivalent to *n*-1 2-bit XOR gates where *n* is the port width.



Function

dout = XOR (all bits of port **din)**

Truth Table

This table is for positive polarity. For negative polarity, invert the values.

Table 2-29. Reduction XOR Truth Table

din	dout
odd number of 1's	1
else	0

Parameters

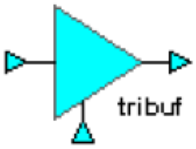
Table 2-30. Reduction XOR Parameters

Parameter	Values	Default
din	Port width (must be > 0)	Automatic
din_type, dout_type	ActiveHigh, ActiveLow	ActiveHigh

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if port **dout** does not have a width of 1.
- A warning is issued and HDL generation fails for this part if either of ports **din** and **dout** are not connected.

Three-state Buffer (tribuf)



When enabled, the value of input port **din** is passed to the output port **dout**. When disabled, all bits of **dout** are set to Z. If the scalar port **ena** has positive polarity, the buffer is enabled when **ena** = 1. If **ena** has negative polarity, the buffer is enabled when **ena** = 0.

This part is equivalent to *n* 1-bit tristate buffers where *n* is the port width. Setting *dout_type* to ActiveLow results in a Three-state Inverter.

Function

For every bit in the bus:

dout = **din** when enabled

dout = Z when disabled

Truth Table

This table is for positive polarity. For negative polarity, invert the values.

Table 2-31. Three-state Buffer Truth Table

din(i)	ena	dout(i)
0	1	0
1	1	1
0	0	Z
1	0	Z

Parameters

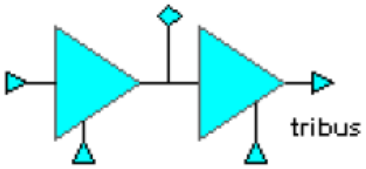
Table 2-32. Three-state Buffer Parameters

Parameter	Values	Default
din, dout	Port widths (must be > 0)	Automatic
ena_type, dout_type	ActiveHigh, ActiveLow	ActiveHigh

Design Rule Checks

- An error is issued if the width of any port cannot be determined, port **ena** does not have a fixed width of 1 or if ports **din0** and **dout** are not the same width.
- A warning is issued and HDL generation fails for this part if any of the ports are not connected.

Three-state bus (tribus)



This part uses two tri-state buffers cascaded together to implement a three-state bidirectional bus.

The input port **din** drives the first tri-state buffer and the result is placed in the inout port **tin**. This port drives the second buffer and the result is placed in the output port **dout**.

The first buffer is enabled, when the scalar port **dena** has a value 1.

The second buffer is enabled when the scalar port **tena** has a value 1.

This part is equivalent to 2*n* 1-bit tristate inverters where *n* is the port width.

Function

For every bit:

dout	= **tin**	(enabled, **tena** = 1)
	= Z	(disabled; **tena** not equal to 1)
tin	= **din**	(enabled, **dena** = 1)
	= Z	(disabled, **dena** not equal to 1)

Truth Tables

Table 2-33. Three-state Bus Truth Table

din(i)	dena	tin(i) as input	tin(i)
0	1	0	0
0	1	1	X
1	1	0	X
1	1	1	1
0	0	0	0
0	0	1	1
1	0	0	0
1	0	1	1

Table 2-34. Three-state Bus Truth Table

tin(i)	tena	dout(i)
0	1	0
1	1	1

Table 2-34. Three-state Bus Truth Table (cont.)

tin(i)	tena	dout(i)
0	0	'Z'
1	0	'Z'

Parameters

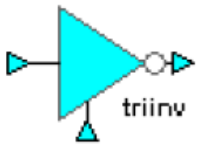
Table 2-35. Three-state Bus Parameters

Parameter	Values	Default
din, tin, dout	Port widths (must be > 0)	Automatic

Design Rule Checks

- An error is issued if the width of any port cannot be determined, port **ena** and **tena** do not have a fixed width of 1 or if ports **din**, **tin** and **dout** are not the same width.
- A warning is issued and HDL generation fails for this part if any of the ports are not connected.

Three-state Inverter (triinv)



When enabled, the value of input port **din** is passed to the output port **dout**. When disabled, all of the bits in output port **dout** are set to Z.

If the scalar port **ena** has positive polarity, the inverter is enabled when the value of port **ena** = 1. If **ena** has negative polarity, the inverter is enabled when the value of port **ena** = 0.

This part is equivalent to *n* 1-bit tristate inverters, where *n* is the port width.

Setting *dout_type* to ActiveLow results in a Three-state Buffer.

Function

For every bit in the bus:

dout = **din** when enabled
= Z when disabled

Truth Table

This table is for positive polarity. For negative polarity, invert the values.

Table 2-36. Three-state Inverter Truth Table

din(i)	ena	dout(i)
0	1	1
1	1	0
0	0	‘Z’
1	0	‘Z’

Parameters

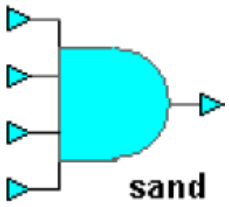
Table 2-37. Three-state Inverter Parameters

Parameter	Values	Default
din, dout	Port width (must be > 0)	Automatic
dout_type, ena_type	ActiveHigh, ActiveLow	ActiveHigh

Design Rule Checks

- An error is issued if the width of any port cannot be determined, port **ena** does not have a fixed width of 1 or if ports **din** and **dout** are not the same width.
- A warning is issued and HDL generation fails for this part if any of the ports are not connected.

Variable Width N-Input AND Gate (sand)



This part performs a logical AND of input ports **din0** to **dinN.** The result is placed on the output port **dout**. Each bit of port **dout** is the AND of the corresponding bits of ports **din0** to **dinN**. Ports **din0** to **dinN** and **dout** may have different widths.

Ports are automatically added or removed when you resize the component instance to implement any number of input ports.

This part is equivalent to *n* N-bit AND gates where *n* is the port width. Setting *dout_type* ActiveLow with all input ports ActiveHigh, results in a variable width N-input NAND gate.

Function

dout = **din0** AND **din1** AND **din2 …** AND **dinN**

Truth Table

This example is for positive polarity. For negative polarity, invert the values.

Table 2-38. Variable Width N-Input AND Gate Truth Table

din0(i)	din1(i)	dout(i)
0	0	0
0	1	0
1	0	0
1	1	1

Parameters

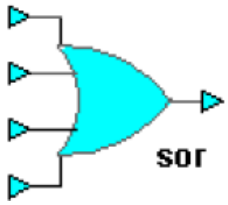
Table 2-39. Variable Width N-Input AND Gate Parameters

Parameter	Values	Default
din0 to dinN, dout	Port width (must be > 0)	Automatic
din0_type to dinN_type, dout_type	ActiveHigh, ActiveLow	ActiveHigh

Design Rule Checks

- An error is issued if the width of any port cannot be determined.
- A warning is issued and HDL generation fails for this part if at least two ports or the output port are not connected.
- A warning is issued but HDL generation succeeds for this part if the width of any of the ports **din0** to **dinN** is greater than the width of **dout**.

Variable Width N-Input OR Gate (sor)



This part performs a logical OR of input ports **din0** to **dinN**. The result is placed on the output port **dout.** Each bit of port **dout** is the OR of the corresponding bits of ports **din0** to **dinN**. Ports **din0** to **dinN** and **dout** may have different widths.

Ports are automatically added or removed when you resize the component instance to implement any number of input ports.

This part is equivalent to *n* N-input OR gates where *n* is the port width. Setting *dout_type* ActiveLow with all input ports ActiveHigh, results in a variable width N-input NOR gate.

Function

dout = **din0** OR **din1** OR **din2 …** OR **dinN**

Truth Table

This example is for positive polarity. For negative polarity, invert the values.

Table 2-40. Variable Width N-Input OR Gate Truth Table

din0(i)	din1(i)	dout(i)
0	0	0
0	1	1
1	0	1
1	1	1

Parameters

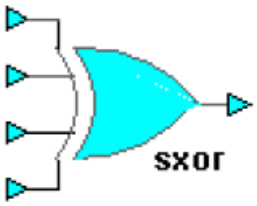
Table 2-41. Variable Width N-Input OR Gate Parameters

Parameter	Values	Default
din0 to dinN, dout	Port width (must be > 0)	Automatic
din0_type to dinN_type, dout_type	ActiveHigh, ActiveLow	ActiveHigh

Design Rule Checks

- An error is issued if the width of any port cannot be determined.
- A warning is issued and HDL generation fails for this part if at least two ports or the output port are not connected.
- A warning is issued but HDL generation succeeds for this part if the width of any of the ports **din0** to **dinN** is greater than the width of **dout**.

Variable Width N-Input XOR Gate (sxor)



This part performs a logical XOR of input ports **din0** to **dinN**. The result is placed on the output port **dout.** Each bit of port **dout** is the XOR of the corresponding bits of ports **din0** to **dinN**. Ports **din0** to **dinN** and **dout** may have different widths.

Ports are automatically added or removed when you resize the component instance to implement any number of input ports.

This part is equivalent to *n* 2-input XOR gates where *n* is the port width. Setting *dout_type* ActiveLow with all input ports ActiveHigh, results in a variable width N-input XNOR gate.

Function

dout = **din0** XOR **din1** XOR **din2 …** XOR **dinN**

Truth Table

This example is for positive polarity. For negative polarity, invert the values.

Table 2-42. Variable Width N-Input XOR Gate Truth Table

din0(i)	din1(i)	dout(i)
0	0	0
0	1	1
1	0	1
1	1	0

Parameters

Table 2-43. Variable Width N-Input XOR Gate Parameters

Parameter	Values	Default
din0 to dinN, dout	Port width (must be > 0)	Automatic
din0_type to dinN_type, dout_type	ActiveHigh, ActiveLow	ActiveHigh

Design Rule Checks

- An error is issued if the width of any port cannot be determined.
- A warning is issued and HDL generation fails for this part if at least two ports or the output port are not connected.
- A warning is issued but HDL generation succeeds for this part if the width of any of the ports **din0** to **dinN** is greater than the width of **dout**.

Chapter 3
Constants

This chapter describes parts supporting the following constant drivers:

Constant Value (constval)

This part is a constant value driver. The parameter *value* must be entered in LNBF format which supports decimal, binary, hexadecimal, and octal radix.

The valid LNBF format is: the radix notation letter (h for hexadecimal, b for binary, o for octal, d for decimal) followed by # followed by a number in the specified radix. If you omit the radix notation letter, the radix will de defaulted to decimal. For example: h#fc (hexadecimal); o#12 (octal); b#10011001 (binary); d#10 or 10 (decimal).

If the *convert_value* parameter is enabled, the **dout** port is assigned the binary value for VHDL or retains the entered value for Verilog.

If the *convert_value* parameter is disabled, the entered value is assigned to the **dout** port for both VHDL and Verilog without any modifications.

This part is equivalent to *n* 1-bit Vdd or Ground, where *n* is the port width.

Function

dout = *value*

Parameters

Table 3-1. Constant Value Parameters

Parameter	Values	Default
dout	Port width (must be > 0)	Automatic
convert_value value	Enabled, Disabled The parameter value (must be > 0)	Enabled 0

Design Rule Checks

- An error is issued if any of the width of the **dout** port cannot be determined, the *value* parameter has no value or if the width of the parameter *value* (after representing it in binary) is greater than the **dout** port width.
- A warning is issued and HDL generation fails for this part if the **dout** port is not connected.

Ground (gnd)

This part is a 0 value driver. A 0 is placed in output port **dout**.

This part is equivalent to *n* 1-bit Ground, where *n* is the port width.

Function

dout = 0 (for every bit)

Parameters

Table 3-2. Ground Parameters

Parameter	Values	Default
dout	Port width (must be > 0)	Automatic

Design Rule Checks

- An error is issued if any of the width of the **dout** port cannot be determined.
- A warning is issued and HDL generation fails for this part if the **dout** port is not connected.

Power (vdd)

This part is a 1 value driver. All ones are placed on output port *dout*.

This part is equivalent to *n* 1-bit Vdd, where *n* is the port width.

Function

For every bit:

dout = 1

Parameters

Table 3-3. Power Parameters

Parameter	Values	Default
dout	Port width (must be > 0)	Automatic

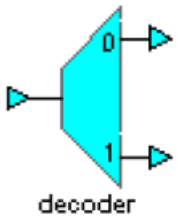
Design Rule Checks

- An error is issued if any of the width of the **dout** port cannot be determined.
- A warning is issued and HDL generation fails for this part if the **dout** port is not connected.

Chapter 4
Combinatorial Parts

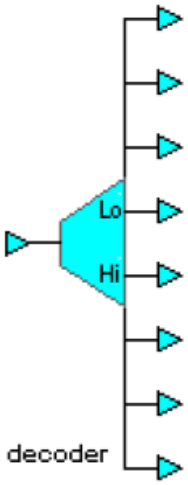
This chapter describes the following decoder, encoder and multiplexer parts:

Decoder, Separate Outputs (decoder)



This part decodes the log2(N)-bit input port **din** and places the result in separate scalar output ports **dout0** to **doutN**.

The default part has two outputs **dout0** and **dout1** and is equivalent to one NOT gate. However, you can resize the component instance to implement parts with four, eight (or any other multiple of two) output ports.



A four-output decoder is equivalent to four 2-bit AND gates and up to four NOT gates.

An eight-output decoder is equivalent to eight 3-bit AND gates and up to twelve NOT gates.

The HDL code for any unused ports is optimized away.

The same functionality is also provided by the Decoder, Combined Output (decoder1) part using a single output bus.

Function

The default two-output decoder has the following function:

dout0 = NOT(**din**)

dout1 = **din**

A four-output decoder has the following function:

dout0 = (NOT(**din**(1))) AND (NOT(**din**(0)))

dout1 = (NOT(**din**(1))) AND **din**(0)

dout2 = **din**(1) AND (NOT(**din**(0)))

dout3 = **din**(1) AND **din**(0)

An eight-output decoder has the following function:

dout0 = (NOT(**din**(2))) AND (NOT(**din**(1))) AND (NOT(**din**(0)))

dout1 = (NOT(**din**(2))) AND (NOT(**din**(1))) AND **din**(0)

dout2 = (NOT(**din**(2))) AND **din**(1) AND (NOT(**din**(0)))

dout3 = (NOT(**din**(2))) AND **din**(1) AND **din**(0)

dout4 = **din**(2) AND (NOT(**din**(1))) AND (NOT(**din**(0)))

dout5 = **din**(2) AND (NOT(**din**(1))) AND **din**(0)

dout6 = **din**(2) AND **din**(1) AND (NOT(**din**(0)))

dout7 = **din**(2) AND **din**(1) AND **din**(0)

Truth Table

The default two-output decoder has the following truth table:

Table 4-1. Decoder Truth Table — Two Separate Outputs

din	dout0	dout1
0	1	0
1	0	1

A four-output decoder has the following truth table:

Table 4-2. Decoder Truth Table — Four Separate Outputs

din	dout0	dout1	dout2	dout3
00	1	0	0	0
01	0	1	0	0
10	0	0	1	0
11	0	0	0	1

An eight-output decoder has the following truth table:

Table 4-3. Decoder Truth Table — Eight Separate Outputs

din	dout0	dout1	dout2	dout3	dout4	dout5	dout6	dout7
000	1	0	0	0	0	0	0	0
001	0	1	0	0	0	0	0	0
010	0	0	1	0	0	0	0	0
011	0	0	0	1	0	0	0	0
100	0	0	0	0	1	0	0	0
101	0	0	0	0	0	1	0	0
110	0	0	0	0	0	0	1	0
111	0	0	0	0	0	0	0	1

Parameters

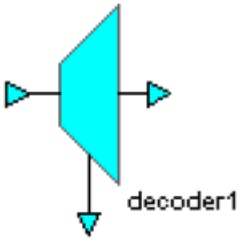
Table 4-4. Decoder Parameters — Separate Outputs

Parameter	Values	Default
din	Port width (must be > 0)	Automatic

Design Rule Checks

- An error is issued if the width of any port cannot be determined or port **din** does not have log2(N) bits (For example: 1, 2 or 3 bits correspond to 2, 4 or 8 output ports).
- A warning is issued and HDL generation fails for this part if port **din** and at least one of the output ports are not connected.

Decoder, Combined Output (decoder1)



This part decodes the input port **din** and places the result on output port **dout**.

Because the widths of **din** and **dout** are both user-definable, an optional scalar output port **selerror** can be used to flag an unsuccessful decode operation. Let *n* be the width of **din** and *m* be the width of **dout**. If *m* >= 2 ** *n* the decoded result is guaranteed to fit into **dout** so **selerror** is set to 0.

If however, *m* < 2 ***n*, only the first *m* bits of the decoded result can be put on the output port **dout** and the decoded result may not fit. In this case, **selerror** is set to 1 if all the bits of **dout** are 0 (indicating that the decoded result could not fit into **dout**) otherwise **selerror** is set to 0. When **selerror** = 0, one and only one bit in **dout** is set to 1. When **selerror** = 1, all the bits of output port **dout** are 0.

This part is equivalent to the gate count of a decoder. The area cost depends upon the connectivity of the output ports **dout** and **selerror**.

If port **selerror** is unconnected, the HDL code for the unused port is optimized away.

The same functions are also provided by the Decoder, Separate Outputs (decoder). Note that the performance is more predictable if you use the non-parameterized decoders.

Function

For every bit:

dout	= 1	if the decoded value of input **din** is the index of the bit
	= 0	if the decoded value of input **din** is not the index of the bit
selerror	= 1	if the decoded value of port **din** >= width of port **dout**
	= 0	if the decoded value of port **din** < width of port **dout**

Truth Table

For every bit i of the bus:

dout(i) = if (i == **din**) then 1 else 0

3 to 8-bit example:

Table 4-5. Decoder Truth Table — Combined Output

din	dout
000	00000001
001	00000010
010	00000100
011	00001000
100	00010000

Table 4-5. Decoder Truth Table — Combined Output (cont.)

din	dout
101	00100000
110	01000000
111	10000000

Parameters

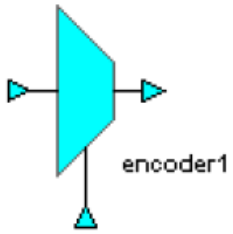
Table 4-6. Decoder Parameters — Combined Output

Parameter	Values	Default
din, dout	Port width (must be > 0)	Automatic
selerror_type	ActiveHigh, ActiveLow, None	ActiveHigh

Design Rule Checks

- An error is issued if the width of any port cannot be determined or port **selerror** does not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if both of ports **din** or **dout** are not connected.

Encoder (encoder1)



This part provides encoder functionality that is the opposite of that provided by a decoder.

The value of the output bus port **dout** is the location of the first bit on input port **din** that is equal to the scalar input port **mode**. This allows the user to encode either the first zero or the first one in **din**.

The most significant bit (MSB) of **dout** is used as an error flag.

The error flag is set if no bit in **din** is equal to the mode (that is, when **mode** = 1 and all the bits of **din** = 0, or when **mode** = 0 and all the bits of **din** = 1). The error flag is also set when the width of **din** is more than **dout** can encode. Because the MSB is reserved for the error flag, the maximum width of **din** should not exceed 2 ** (*n* - 1) where n is the width of **dout**.

The width of **dout** must be at least 2. Encoding starts from the least significant bit (LSB) to MSB.

This part is equivalent to the gate count of an *n* bit encoder. Where *n* is the width of the port **din**.

Function

dout	= i	where i is first location (from LSB) of the bit of **din** = **mode**
	= 100…	if none of the bits in the encodable range of **din** = **mode**

Truth Table

Example (4 to 8 bit):

Table 4-7. Encoder Truth Table

din	mode	dout
00110111	0	0011
00111100	0	0000
10101111	0	0100
01100000	1	0101
01111111	1	0000
00000000	1	1000

Parameters

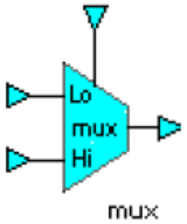
Table 4-8. Encoder Parameters

Parameter	Values	Default
din, dout	Port width (must be > 0)	Automatic

Design Rule Checks

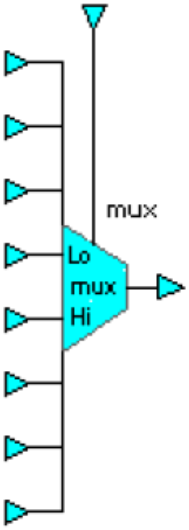
- An error is issued if the width of any port cannot be determined or port **mode** does not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any port is not connected.

N-Input Multiplexer (mux)



This part selects one of the input ports (**din0** to **dinN**) and places it on the output port **dout**. The selection is based on the value of the input port **sel**.

The default part has two inputs **din0** and **din1** and **sel** is a scalar port. However, you can resize the component instance to implement parts with four, eight (or any other multiple of two) input ports with a corresponding width **sel** port.



This part is equivalent to a gate count of N*n(log2(N)+1)-bit AND gates and *n* N-bit OR gates where n is the width of the data ports and N is the number of input ports. For example a four-input mux has the gate count of three 2-bus multiplexers and an eight-input mux has the gate count of seven 2-bus multiplexers.

The HDL code for any unconnected ports is optimized away.

Function

The default two-input multiplexer has the following function:

dout = **din0** (if **sel** = 0)
= **din1** (if **sel** = 1)

A four-input multiplexer has the following function:

dout = **din0** (if **sel** = 00)
= **din1** (if **sel** = 01)
= **din2** (if **sel** = 10)
= **din3** (if **sel** = 11)

An eight-input multiplexer has the following function:

dout = **din0** (if **sel** = 000)
= **din1** (if **sel** = 001)
= **din2** (if **sel** = 010)
= **din3** (if **sel** = 011)
= **din4** (if **sel** = 100)
= **din5** (if **sel** = 101)
= **din6** (if **sel** = 110)
= **din7** (if **sel** = 111)

Truth Table

In the default two-input multiplexer, for every bit i of the bus:

Table 4-9. N-Input Multiplexer Truth Table — Two Inputs

sel	dout
0	din0
1	din1

In a four-input multiplexer, for every bit i of the bus:

Table 4-10. N-Input Multiplexer Truth Table — Four Inputs

sel	dout
00	din0
01	din1
10	din2
11	din3

In an eight-input multiplexer, for every bit i of the bus:

Table 4-11. N-Input Multiplexer Truth Table — Eight Inputs

sel	dout
000	din0
001	din1
010	din2
011	din3
100	din4
101	din5
110	din6
111	din7

Parameters

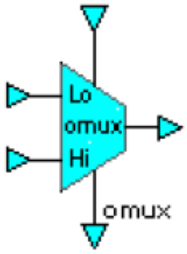
Table 4-12. N-Input Multiplexer Parameters

Parameter	Values	Default
din0 to dinN, dout	Port widths (must be > 0)	Automatic
sel	Port widths (must be > 0)	Automatic

Design Rule Checks

- An error is issued if the width of any port cannot be determined, port **sel** does not have a fixed width of log2(width of **din**) while the number of **din** ports must be a power of 2 or if the input and output ports do not have the same width.
- A warning is issued and HDL generation fails for this part if ports **sel** or **dout** are not connected.

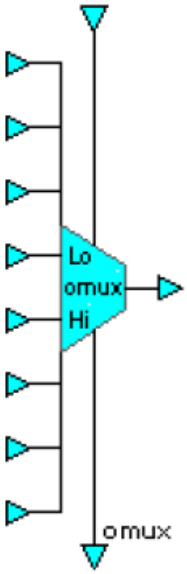
N-Input One-hot Multiplexer (omux)



This part selects one of the input data ports (**din0** to **dinN**) and passes it to the output port **dout**.

The default part has two inputs **din0** and **din1** and **sel** is 2-bits wide. However, you can resize the component instance to implement parts with four, eight (or any other multiple of two) input ports with a corresponding width **sel** port.

The selection is based on the value of the N-bit wide input port **sel** and one of the two select modes controlled by the enumerated parameter *sel_priority* (Enabled, Disabled) associated with this port.



In priority disabled mode, the scalar output port **selerror** will be asserted when the data on the port **sel** is not in one-hot encoded format. The polarity of **selerror** is controlled by the enumerated parameter *selerror_type* (ActiveHigh, ActiveLow).

The parameter *selerror_type* has no effect in priority enabled mode and **selerror** is ignored.

If only one of **dout** and **selerror** is connected, the HDL code for the unconnected port is optimized away.

Function

If *sel_priority* = enabled:

dout	= **din**(i) = 'X…' (all bits are X)	if **sel**(i) = 1 and i = minimum(0,1, .. N-1) otherwise
selerror	= 'X'	always (irrespective of *selerror_type*)

If *sel_priority* = disabled and *selerror_type* = ActiveHigh:

dout	= **din**(i) = 'X…' (all bits are X)	if **sel**(i) = 1 and **sel**(j) = 0 for every j ≠ i Otherwise
selerror	= 0 = 1	if **sel**(i) = 1 and **sel**(j) = 0 for every j ≠ i Otherwise

If *sel_priority* = disabled and *selerror_type* = ActiveLow:

dout	= **din**(i) = 'X... ' (all bits are X)	if **sel**(i) = 1 and **sel**(j) = 0 for every j ≠ i Otherwise
selerror	= 1 = 0	if **sel**(i) = 1 and **sel**(j) = 0 for every j ≠ i Otherwise

Truth Table

The following tables are for positive polarity. For negative polarity, invert the values.

Priority Enabled

The default two-input multiplexer:

Table 4-13. N-Input One-hot Multiplexer Truth Table — Priority Enabled, Two-Input

sel	dout	selerror
00	X...	X
01	din0	X
10	din1	X
11	din0	X

A four-input multiplexer:

Table 4-14. N-Input One-hot Multiplexer Truth Table — Priority Enabled, Four-Input

sel	dout	selerror
0000	X...	X
XXX1	din0	X
XX10	din1	X
X100	din2	X
1000	din3	X

An eight-input multiplexer.

Table 4-15. N-Input One-hot Multiplexer Truth Table — Priority Enabled, Eight-Input

sel	dout	selerror
00000000	X...	X
XXXXXXX1	din0	X
XXXXXX10	din1	X
XXXXX100	din2	X
XXXX1000	din3	X
XXX10000	din4	X
XX100000	din5	X

Table 4-15. N-Input One-hot Multiplexer Truth Table — Priority Enabled, Eight-Input (cont.)

sel	dout	selerror
X1000000	din6	X
10000000	din7	X

Priority Disabled

The default two-input multiplexer:

Table 4-16. N-Input One-hot Multiplexer Truth Table — Priority Disabled, Two-Input

sel	dout	selerror
00	X...	1
01	din0	0
10	din1	0
11	X...	1

A four-input multiplexer:

Table 4-17. N-Input One-hot Multiplexer Truth Table — Priority Disabled, Four-Input

sel	dout	selerror
0000	X...	1
0001	din0	0
0010	din1	0
0100	din2	0
1000	din3	0
others	X...	1

An eight-input multiplexer:

Table 4-18. N-Input One-hot Multiplexer Truth Table — Priority Disabled, Eight-Input

sel	dout	selerror
00000000	X...	1
00000001	din0	0
00000010	din1	0
00000100	din2	0

Table 4-18. N-Input One-hot Multiplexer Truth Table — Priority Disabled, Eight-Input (cont.)

sel	dout	selerror
00001000	din3	0
00010000	din4	0
00100000	din5	0
01000000	din6	0
10000000	din7	0
all others	X...	1

Parameters

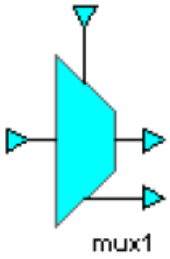
Table 4-19. N-Input One-hot Multiplexer Parameters

Parameter	Values	Default
din0 to dinN, dout sel	Port widths (must be > 0) Port width (must be > 0)	Automatic Automatic
dout_type selerror_type sel_priority	Enabled, Disabled ActiveHigh, ActiveLow Enabled, Disabled	Enabled ActiveHigh Disabled

Design Rule Checks

- An error is issued if the width of any port cannot be determined, port **selerror** does not have a fixed width of 1, port **sel** does not have the same width as the **din0** to **dinN** ports while the number of **input** ports must be a power of 2 or if ports **din0** to **dinN** and **dout** do not have the same width.
- A warning is issued and HDL generation fails for this part if both of ports **dout** and **selerror** are not connected.

W-Bit Multiplexer (mux1)



This part selects one of the bits from the input data bus **din** and places it on the scalar output port **dout**.

The selection is based on the value of input port **sel** which is interpreted as a zero-based index. The output port **selerror** is asserted if the value of the input port **sel** exceeds the width of input bus port **din**.

If only one of **dout** and **selerror** is connected, the HDL code for the unconnected port is optimized away.

This part is equivalent to the gate count of an *n*-bit multiplexer where *n* is the width of the port **din**.

Function

dout	= **din**(**sel**)	if **sel** < width of **din**
	= X	otherwise
selerror	= 0	if **sel** < width of **din**
	= 1	otherwise

Truth Table

Example (8 to 3-bit):

Table 4-20. W-Bit Multiplexer Truth Table

din	sel	dout
00110100	000	0
00110100	001	0
00110100	010	1
00110100	011	0
00110100	100	1
00110100	101	1
00110100	110	0
00110100	111	0

Parameters

Table 4-21. W-Bit Multiplexer Parameters

Parameter	Values	Default
din, sel	Port width (must be > 0)	Automatic
selerror_type	ActiveHigh, ActiveLow, None	ActiveHigh

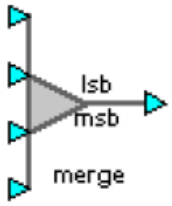
Design Rule Checks

- An error is issued if the width of any port cannot be determined or ports **dout** and **selerror** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if ports **sel** or **din** are not connected or if both of ports **dout** and **selerror** are not connected.

Chapter 5
Bit Manipulation Parts

This chapter describes parts that operate on bits. Most of these parts can also be directly implemented graphically on the block diagram. However, the parts can be useful when explicit drivers are required.

N-Bus Merge (merge)



This part concatenates N input buses **dinN** to **din0** and copies the result onto the output bus **dout**.

Ports are automatically added or removed when you resize the component instance to implement any number of input ports.

Typically, the width of **dout** would be exactly equal to the sum of the widths of the inputs. In that case, the lowest bit of **dinN** is copied to the lowest bit of **dout** and so on until the highest of **din0** is copied to **dout**.

If the width of **dout** is insufficient to hold the result, the lower order bits are put onto **dout** and the highest bits are discarded.

If the width of **dout** is greater than the sum, the value *X* is placed into the extra bits of **dout**.

Function

dout = **dinN,…,din0**

Truth Table (2-Bus examples)

Example merging 4-bit bus **din1** and 3-bit bus **din0** into 7-bit bus **dout**:

Table 5-1. N-Bus Merge Truth Table — Two-Bus

din1	din0	dout
0001	000	0001000
0010	110	0010110
0100	000	0100000
1001	010	1001010

Example merging 4-bit bus **din1** and 3-bit bus **din0** into 8-bit but **dout** showing how the undefined bit in **dout** is filled with an *X*:

Table 5-2. N-Bus Merge Truth Table — Two-Bus

din1	din0	dout
0001	000	X0001000
0010	110	X0010110
0100	000	X0100000
1001	010	X1001010

Example merging 4-bit bus **din1** and 3-bit bus **din0** into 6-bit bus **dout** discarding the highest bit in **din1**:

Table 5-3. N-Bus Merge Truth Table — Two-Bus

din1	din0	dout
0001	000	001000
0010	110	010110
0100	000	100000
1001	010	001010

Truth Table (4-Bus examples)

Example merging 2-bit buses **din3**, **din2** and **din1** with 1-bit bus **din0** into 7-bit bus **dout**:

Table 5-4. N-Bus Merge Truth Table — Four-Bus

din3	din2	din1	din0	dout
00	01	00	0	0001000
00	10	11	0	0010110
01	00	00	0	0100000
10	01	01	1	1001011

Example merging 2-bit buses **din3**, **din2** and **din1** with 1-bit bus **din0** into 8-bit bus **dout** showing how the undefined bit in **dout** is filled with an *X:*

Table 5-5. N-Bus Merge Truth Table — Four-Bus

din3	din2	din1	din0	dout
00	01	00	0	X0001000
00	10	11	0	X0010110
01	00	00	0	X0100000
10	01	01	1	X1001011

Example merging 2-bit buses **din3**, **din2** and **din1** with 1-bit bus **din0** into 6-bit bus **dout** discarding the highest bit in **din3**:

Table 5-6. N-Bus Merge Truth Table — Four-Bus

din3	din2	din1	din0	dout
00	01	00	0	001000
00	10	11	0	010110
01	00	00	0	100000
10	01	01	1	001011

Truth Table (8-Bus examples)

Example merging six 2-bit buses (**din7** to **din2**) and two 1-bit buses (**din1** and **din0**) into a 14-bit output bus (**dout**):

Table 5-7. N-Bus Merge Truth Table — Eight-Bus

din7	din6	din5	din4	din3	din2	din1	din0	dout
00	01	10	00	00	10	0	0	00011000001000
00	10	11	11	11	11	1	0	00101111111110
01	01	00	10	01	00	0	0	01010010010000
10	11	01	11	00	01	0	1	10110111000101

Example merging six 2-bit buses (**din7** to **din2**) and two 1-bit buses (**din1** and **din0**) into a 15-bit output bus (**dout**) showing how the undefined bit in **dout** is filled with an *X*:

Table 5-8. N-Bus Merge Truth Table — Eight-Bus

din7	din6	din5	din4	din3	din2	din1	din0	dout
00	01	10	00	00	10	0	0	X00011000001000
00	10	11	11	11	11	1	0	X00101111111110
01	01	00	10	01	00	0	0	X01010010010000
10	11	01	11	00	01	0	1	X10110111000101

Example merging six 2-bit buses (**din7** to **din2**) and two 1-bit buses (**din1** and **din0**) into a 10-bit output bus (**dout**) discarding the bits in **din6** and **din7**:

Table 5-9. N-Bus Merge Truth Table — Eight-Bus

din7	din6	din5	din4	din3	din2	din1	din0	dout
00	01	10	00	00	10	0	0	1000001000
00	10	11	11	11	11	1	0	1111111110
01	01	00	10	01	00	0	0	10010010000
10	11	01	11	00	01	0	1	0111000101

Parameters

Table 5-10. N-Bus Merge Parameters

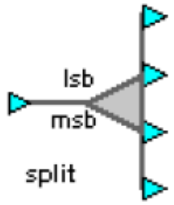
Parameters	Values	Default
din0 to dinN, dout	Port width (must be > 0)	Automatic

Design Rule Checks

- An error is issued if the width of any port cannot be determined.
- A warning is issued and HDL generation fails for this part if any ports are not connected.

N-Way Splitter (split)

This part splits the input bus **din** into N output buses **doutN** to **dout0**.



Ports are automatically added or removed when you resize the component instance to implement any number of output ports.

The lower bits of **din** are placed in **dout0** until it is completely filled, then the remaining bits of **din** are copied to **dout1** to **doutN**.

If the sum of the widths of **doutN** to **dout0** is greater than the width of **din**, extra bits on the output would be filled with X.

If the width of **dout** is less than the sum, then only the lower order bits are copied into **dout0** to **doutN** until all are filled. Any bits remaining in din are ignored.

Function

doutN,…, **dout0** = **din**

Truth Table (2-Bus Examples)

Example splitting the 7-bit bus **din** into 4-bit **dout1** and 3-bit **dout2** buses:

Table 5-11. N-Way Splitter Truth Table — Two-Bus

din	dout1	dout0
0001000	0001	000
0010110	0010	110
0100000	0100	000
1001010	1001	010

Example splitting the 8-bit bus **din** into 4-bit **dout1** and 3-bit **dout2** buses showing how the highest bit of **din** is discarded:

Table 5-12. N-Way Splitter Truth Table — Two-Bus

din	dout1	dout0
10001000	0001	000
00010110	0010	110
10100000	0100	000
11001010	1001	010

Example splitting the 6-bit bus **din** into a 4-bit **dout1** and a 3-bit **dout2** bus showing how the undefined bit in **dout1** is filled with an *X*:

Table 5-13. N-Way Splitter Truth Table — Two-Bus

din	dout1	dout0
001000	X001	000
010110	X010	110
100000	X100	000
001010	X001	010

Truth Table (4-Bus Examples)

Example splitting the 7-bit bus **din** into three 2-bit buses **(dout3** to **dout1)** and one 1-bit bus **dout0**:

Table 5-14. N-Way Splitter Truth Table — Four-Bus

din	dout3	dout2	dout1	dout0
0001000	00	01	00	0
0010110	00	10	11	0
0100000	01	10	00	0
1001011	10	01	01	1

Example splitting the 8-bit bus **din** into three 2-bit buses (**dout3** to **dout1**) and one 1-bit bus **dout0** showing how the highest bit of **din** is discarded:

Table 5-15. N-Way Splitter Truth Table — Four-Bus

din	dout3	dout2	dout1	dout0
00001000	00	01	00	0
10010110	00	10	11	0
00100000	01	10	00	0
01001011	10	01	01	1

Example splitting the 6-bit bus **din** into three 2-bit buses (**dout3** to **dout1**) and one 1-bit bus **dout0** showing how the undefined bit in **dout3** is filled with an *X*:

Table 5-16. N-Way Splitter Truth Table — Four-Bus

din	dout3	dout2	dout1	dout0
001000	X0	01	00	0
010110	X0	10	11	0

Table 5-16. N-Way Splitter Truth Table — Four-Bus (cont.)

din	dout3	dout2	dout1	dout0
100000	X1	10	00	0
001011	X0	01	01	1

Truth Table (8-Bus Examples)

Example splitting the 15-bit bus **din** into seven 2-bit buses (**dout7** to **dout1**) and one 1-bit bus **dout0**:

Table 5-17. N-Way Splitter Truth Table — Eight-Bus

din	dout7	dout6	dout5	dout4	dout3	dout2	dout1	dout0
010011100001100	01	00	11	10	00	01	10	0
001100001110110	00	11	00	00	11	10	11	0
011011010110000	01	10	11	01	01	10	00	0
101110001000011	10	11	10	00	10	00	01	1

Example splitting the 16-bit bus **din** into seven 2-bit buses (**dout7** to **dout1**) and one 1-bit bus **dout0** showing how the highest bit of **din** is discarded:

Table 5-18. N-Way Splitter Truth Table — Eight-Bus

din	dout7	dout6	dout5	dout4	dout3	dout2	dout1	dout0
0010011100001100	01	00	11	10	00	01	10	0
1001100001110110	00	11	00	00	11	10	11	0
1011011010110000	01	10	11	01	01	10	00	0
0101110001000011	10	11	10	00	10	00	01	1

Example splitting the 12-bit bus **din** into seven 2-bit buses (**dout7** to **dout1**) and one 1-bit bus **dout0** showing how the undefined bits in **dout7** and **dout6** are filled with *X*:

Table 5-19. N-Way Splitter Truth Table — Eight-Bus

din	dout7	dout6	dout5	dout4	dout3	dout2	dout1	dout0
011100001100	XX	X0	11	10	00	01	10	0
000001110110	XX	X0	00	00	11	10	11	0
111010110000	XX	X1	11	01	01	10	00	0
110001000011	XX	X1	10	00	10	00	01	1

Parameters

Table 5-20. N-Way Splitter Parameters

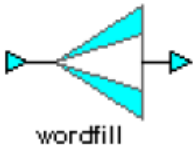
Parameters	Values	Default
din, dout0 to doutN	Port width (must be > 0)	Automatic

Design Rule Checks

- An error is issued if the width of any port cannot be determined.
- A warning is issued and HDL generation fails for this part if none of the output ports are connected.

Bus Fill (wordfill)

This part fills up the bits of the output bus **dout** with the value of the scalar input port **din**.



Function

For every bit i of the bus:

dout(i) = **din**

Truth Table

For every bit i of the bus:

Table 5-21. Bus Fill Truth Table

din	dout(i)
0	0
1	1

Parameters

Table 5-22. Bus Fill Parameters

Parameter	Values	Default
dout	Port width (must be > 0)	Automatic

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if port **din** does not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any ports are not connected.

Bus Tapper (tap)

This part taps bits from the input bus **din** and places them in the output bus **dout**.

The parameter *start_at* is a zero-based index that defines the location of the first bit in **din** that will be copied to the least significant bit of **dout**. The bits of **dout** are filled from the bits of **din** in increasing order until either **dout** has been filled or until there are no more bits in **din**. If there are not enough bits in **din** to fill up **dout**, the remaining bits of **dout** are filled with *X*.

Function

For every bit i of the bus:

dout(i)	= **din**(*start_at*+ i)	if *start_at* + i < width of port **din**
	= X	otherwise

Truth Table

Example tapping 3 bits from a 7-bit bus:

Table 5-23. Bus Trapper Truth Table

din	start	dout0
0100111	0	111
0100111	1	011
0100111	2	001
0100111	3	100
0100111	4	010
0100111	5	X01
0100111	6	XX0

Parameters

Table 5-24. Bus Trapper Parameters

Parameter	Values	Default
din dout	Port width (must be > 0) Port width (must be > 0)	Automatic Automatic
start_at	Start location, must be >= 0 and < width of port **din**	0

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if parameter *start_at* is not greater than or equal to 0 and less than the width of port **din**.
- A warning is issued and HDL generation fails for this part if any ports are not connected.

Fixed Bit Selector (fbitsel)

This part selects a particular bit from the input bus **din** and places this bit in the scalar output port **dout**.

The parameter *sel* is a zero-based index that specifies the location of the bit selected.

Function

dout = **din**(*sel*)

Truth Table

A 4-bit input Example:

Table 5-25. Fixed Bit Selector Truth Table

din<3:0>	**sel**	**dout**
0101	00	1
0101	01	0
0100	10	1
0101	11	0

Parameters

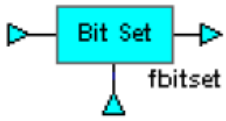
Table 5-26. Fixed Bit Selector Parameters

Parameter	**Values**	**Default**
din	Port width (must be > 1)	Automatic
sel	Bit location, must be >= 0 and < width of port **din**	0

Design Rule Checks

- An error is issued if the width of any port cannot be determined, if parameter *sel* is not greater than or equal to 0 and less than the width of port **din** or if port **dout** does not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any ports are not connected.

Fixed Bit Setter (fbitset)



This part reads the input port **din** and sets one of its bits to the value of scalar input port **value**.

The result is placed on the output port **dout**. The location of the bit to be set is specified by the parameter *loc*. This value is zero-indexed.

This part is equivalent to 1-bit Vdd or Ground.

Function

For every bit:

dout = **din** (if bit location not equal to *loc*)

dout = **value** (if bit location = *loc*)

Truth Table

Table 5-27. Fixed Bit Setter Truth Table

din	value	dout(i)
0	0	0
0	1	if (i not equal to loc) 0, else 1
1	0	if (i not equal to loc) 1, else 0
1	1	1

Parameters

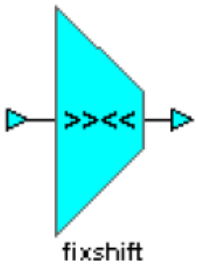
Table 5-28. Fixed Bit Setter Parameters

Parameter	Values	Default
din, dout	Port widths (must be > 0)	Automatic
loc	Bit location (must be >= 0 and < width of port **din**)	0

Design Rule Checks

- An error is issued if the width of any port cannot be determined, if parameter *sel* is not greater than or equal to 0 and less than the width of port **din** or if port **value** does not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any ports are not connected.

Fixed Shifter (fixshift)



This part shifts the bits of the input bus **din** and places the result on the output bus **dout**.

There are three shift modes which are selected by the enumerated parameter *mode* (Logical, Arithmetic, Circular).

The parameter *shift* specifies the number of places the bits will be shifted. The direction of the shift operation is specified by the sign of parameter shift. If the value is < 0, it is a shift right operation. If the value is > 0, it is a shift left operation.

For logical mode, 0 is shifted in for both left and right shift operations. For arithmetic mode, 0 is shifted into the least significant (LSB) bit for left shift but for right shift the most significant bit (MSB) is retained. For circular mode, right shift shifts the LSB to the MSB while the left shift shifts the MSB to the LSB.

Function

If *mode* = logical:

dout	= **din** >> *shift*	if *shift* < 0 (0 is placed in the MSB)
	= **din** << *shift*	if *shift* > 0 (0 is placed in the LSB)
	= **din**	if *shift* = 0

If *mode* = arithmetic:

dout	= **din** >> *shift*	if *shift* < 0 (MSB on **din** is retained)
	= **din** << *shift*	if *shift* > 0 (0 is placed in the LSB)
	= **din**	if *shift* = 0

If *mode* = circular:

dout	= **din** >> *shift*	if *shift* < 0 (LSB of **din** is placed in the MSB)
	= **din** << *shift*	if *shift* > 0 (MSB of **din** is placed in the LSB)
	= **din**	if *shift* = 0

Truth Table

A 4-Bit input example (*mode* = logical):

Table 5-29. Fixed Shifter Truth Table — Mode = Logical

din<3:0>	**shift**	**dout**
1101	5	0000
1101	2	0100
1101	-2	0011
1101	-5	0000

A 4-bit input example (*mode* = arithmetic):

Table 5-30. Fixed Shifter Truth Table — Mode = Arithmetic

din<3:0>	shift	dout
1001	5	0000
1001	2	0100
1001	-2	1110
1001	-5	1111

A 4-bit input example (*mode* = circular):

Table 5-31. Fixed Shifter Truth Table — Mode = Circular

din<3:0>	shift	dout
1001	5	0011
1001	2	0110
1001	-2	0110
1001	-5	1100

Parameters

Table 5-32. Fixed Shifter Parameters

Parameter	Values	Default
din, dout	Port widths (must be > 0)	Automatic
mode shift	Logical, Arithmetic, Circular Number of bits to shift (must be > 0)	Logical 1

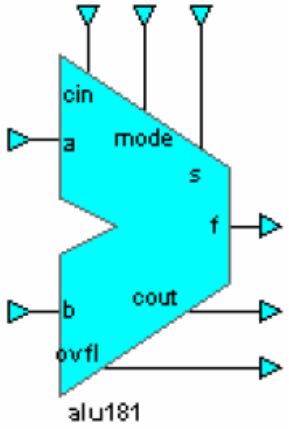
Design Rule Checks

- An error is issued if the width of any port cannot be determined.
- A warning is issued and HDL generation fails for this part if any ports are not connected.
- A warning is issued although generation is successful if parameter *shift* is greater than or equal to the width of port **din** when the mode is logical or arithmetic, if *shift* has a value of 0 or if *shift* is an exact multiple of the width of the input port **din** when the mode of the shifter is circular.

Chapter 6
Arithmetic Parts

This chapter describes parts related to the following arithmetic operations:

181 ALU (alu181)



This part provides the standard 181 Arithmetic Logic Unit (ALU) operations. The input data bus ports are **a** and **b**. The result is placed in the output port **f**.

You can consult vendor data sheets for details of this part (for example, Texas Instruments' TTL Data Book, 74181 series).

The arithmetic operation is controlled by the value of the enumerated parameter *sign_type* (Signed, Unsigned).

The mode of the ALU is controlled by the scalar input port **mode** which can be set to either logical (1) or arithmetic (0). The polarity of port **mode** is controlled by the enumerated parameter *mode_type* (ActiveHigh, ActiveLow).

Scalar input port **cin** is available as a carry in for the arithmetic operations. The polarity of **cin** is controlled by the enumerated parameter *cin_type* (ActiveHigh, ActiveLow).

Two scalar carry ports are available from the ALU: **cout** and **ovfl**. They differ only for the signed arithmetic operations (parameter *sign_type* = signed). The bit after the most significant bit of the output port **f** is placed in the scalar output port **cout**. The result of an exclusive OR between the most significant bit of output port **f** and the bit after the most significant bit of the output port **f** is placed in the scalar output port **ovfl**.

The polarities of **cout** and **ovfl** are controlled by the enumerated parameters *cout_type* (ActiveHigh, ActiveLow) and *ovfl_type* (ActiveHigh, ActiveLow). If the mode of the ALU is logical, both the carry output ports are set to 0.

The operation (arithmetic or logical) is defined by the value of the 4-bit wide input port **s**. Note that in a standard 181 component the functionality of this port is available via four scalar ports (namely **s0**, **s1**, **s2** and **s3**).

This part is very costly in terms of the gate count. Different implementation strategies are needed to achieve better hardware performance. Even though synthesis tools may generate a reasonable implementation other options such as silicon compilers and optimized hard macros should be explored.

If the input port **cin** is unconnected, it is driven by 0.

This part is equivalent to an *n*-bit wide 181 ALU, where *n* is width of input port **a**, **b** and **f**.

Function

If scalar input port mode = 1 and parameter *mode_type* = ActiveHigh or scalar input port mode = 0 and parameter *mode_type* = ActiveLow:

f	= NOT(**a**)	if **s** = 0
	= NOT(**a** AND **b**)	if **s** = 1
	= NOT(**a**) OR **b**	if **s** = 2
	= 1	if **s** = 3
	= NOT(**a** OR **b**)	if **s** = 4
	= NOT(**b**)	if **s** = 5
	= NOT(**a** XOR **b**)	if **s** = 6
	= **a** OR NOT(**b**)	if **s** = 7
	= NOT(**a**) AND **b**	if **s** = 8
	= **a** XOR **b**	if **s** = 9
	= **b**	if **s** = 10
	= **a** OR **b**	if **s** = 11
	= 0	if **s** = 12
	= **a** AND NOT(**b**)	if **s** = 13
	= **a** AND **b**	if **s** = 14
	= **a**	if **s** = 15

cout = 0

ovfl = 0

else (if scalar input port **mode** = 0 and parameter *mode_type* = ActiveHigh or scalar input port **mode** = 1 and parameter *mode_type* = ActiveLow):

f	= **a** - 1 + **cin**	if **s** = 0
	= **a** AND **b** - 1 + **cin**	if **s** = 1
	= **a** AND NOT(**b**) - 1 + **cin**	if **s** = 2
	= -1 + **cin**	if **s** = 3
	= **a** + (**a** OR NOT(**b**) + **cin**	if **s** = 4
	= (**a** AND **b**) + (**a** OR NOT(**b**)) + **cin**	if **s** = 5
	= **a** - **b** - 1 + **cin**	if **s** = 6
	= **a** + NOT(**b**) + **cin**	if **s** = 7
	= **a** + (**a** OR **b**) + **cin**	if **s** = 8
	= **a** + **b** + **cin**	if **s** = 9
	= (**a** AND NOT(**b**)) + (**a** OR **b**) + **cin**	if **s** = 10
	= **a** + **b** + **cin**	if **s** = 11
	= **a** + **a** + **cin**	if **s** = 12
	= (**a** AND **b**) + **a** + **cin**	if **s** = 13
	= (**a** AND NOT(**b**)) + **a** + **cin**	if **s** = 14
	= **a** + **cin**	if **s** = 15

(**a** MSB is the MSB of the arithmetic operation; this is the n+1th bit from the arithmetic operation, where *n* is the width of ports **a**, **b** and **f**)

cout = MSB

ovfl	= MSB	if parameter *sign_type* = unsigned
	= MSB XOR f(n)	if parameter *sign_type* = signed

Truth Table

For positive polarities. (For negative polarities invert the signals, **cin** = 0; **cin** is added only in the arithmetic mode).

Table 6-1. 181 ALU Truth Table

s<3:0>	**f**	
s[3],s[2],s[1],s[0]	logical (mode =1)	arithmetic (mode = 0)
0000	NOT(a)	a - 1
0001	NOT(a AND b)	a AND b - 1
0010	NOT(a) OR b	a AND NOT(b)
0011	1	-1
0100	NOT(a OR b)	a + (a OR NOT(b))
0101	NOT(b)	(a AND b) + (a OR NOT(b))
0110	NOT(a XOR b)	a - b - 1
0111	a OR NOT(b)	a + NOT(b)
1000	NOT(a) AND b	a + (a OR b)
1001	a XOR b	a + b
1010	b	(a AND NOT(b)) + (a OR b)
1011	a OR b	a + b
1100	0	a + a
1101	a AND NOT(b)	(a AND b) + a
1110	a AND b	(a AND NOT(b)) + a
1111	a	a

Parameters

Table 6-2. 181 ALU Parameters

Parameter	Values	Default
a, b, f	Port widths (must be > 0)	Automatic
sign_type cin_type mode_type cout_type ovfl_type	Unsigned, Signed ActiveHigh, ActiveLow ActiveHigh, ActiveLow ActiveHigh, ActiveLow ActiveHigh, ActiveLow	Unsigned ActiveHigh ActiveHigh ActiveHigh ActiveHigh

Design Rule Checks

- An error is issued if the width of any port cannot be determined, if ports **cin**, **cout**, **ovfl** and **mode** do not have a fixed width of 1 or if port **s** does not have a fixed width of 4.
- A warning is issued and HDL generation fails for this part if any of ports **a**, **b**, **mode**, **s** and **f** are not connected.

Absolute Value (absval)

This part takes the absolute value of the input port **din** and places the result in the output port **dout**. **din** is assumed to be a signed number in 2's complement format.

This part is equivalent to n of 2-bit multiplexers, *n* inverters and an *n*-bit wide incrementer; where *n* is the width of the ports.

Function

dout	= **din**	if the MSB of the input port **din** = 0
	= (NOT(**din**)) + 1	Otherwise

Truth Table

A 3-bit example:

Table 6-3. Absolute Value Truth Table

din<2:0>	Value of din	dout<2:0>	Value of dout
000	0	000	0
001	1	001	1
010	2	010	2
011	3	011	3
100	-4	100	4
101	-3	011	3
110	-2	010	2
111	-1	001	1

Parameters

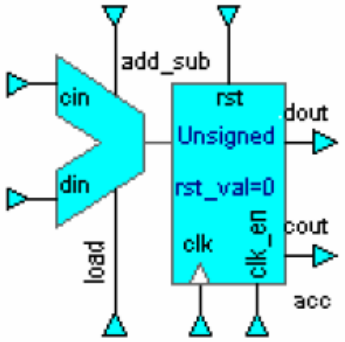
Table 6-4. Absolute Value Parameters

Parameter	Values	Default
din, dout	Port widths (must be > 1)	Automatic
dout_type	ActiveHigh, ActiveLow	ActiveHigh

Design Rule Checks

- An error is issued if the width of any port cannot be determined.
- A warning is issued and HDL generation fails for this part if any of the ports are not connected.

Accumulator (acc)



This part accumulates the value of the input data port **din** with the value of the register during each clock cycle. The output port **dout** gets the current value of the register. The register value accumulates by addition or subtraction depending on the status of the scalar input port **add_sub**.

The part can be triggered on the rising edge or the falling edge of the scalar input port **clk**. The polarity of this port is controlled by the enumerated parameter *clk_type* (Rising, Falling).

The **clk** can optionally be enabled by the scalar input port **clk_en**. The polarity of **clk_en** is controlled by the enumerated parameter *clk_en_type* (ActiveHigh, ActiveLow). Do not change the value of the parameter *clk_en_type* if the **clk_en** port is not used.

The register can be reset to the value of the parameter *rst_val* by activating the scalar input port **rst**. The mode is set by the enumerated parameter *rst_type* (SyncActiveHigh, AsyncActiveHigh, SyncActiveLow, AsyncActiveLow). Do not change the value of the parameter *rst_type* if the **rst** port is not used.

If the scalar input port **load** is activated when the **rst** port not active, the value of the input port **din** can be loaded into the register (as opposed to accumulation). The polarity of port **load** is controlled by the enumerated parameter *load_type* (ActiveHigh, ActiveLow).

The scalar input port **cin** is the carry in port for the arithmetic operation. The polarity of the **cin** port is controlled by the enumerated parameter *cin_type* (ActiveHigh, ActiveLow).

The scalar output port **cout** is the carry out of the arithmetic operation. The polarity of port **cout** is controlled by enumerated parameter *cout_type* (ActiveHigh, ActiveLow). The carry out is available one cycle before the value gets registered. If the timing for carry-out needs to be synchronized with the register output, a separate flip-flop needs to be used to account for the delay. This part was designed not to have an in-built flip-flop for the output port **cout** so as to save a flip-flop unless it is required.

If the arithmetic operation is deactivation (scalar input ports **rst** or **load** are active or **clk** is disabled) the value of the scalar output port **cout** must be ignored because it reflects a carry-out of an operation not selected by the multiplexer.

The arithmetic operation can be done in unsigned or signed mode. This operation is controlled by the enumerated parameter *sign_type* (Unsigned, Signed).

The arithmetic operation is controlled dynamically by the scalar input port **add_sub**. The polarity of port **add_sub** is controlled by the enumerated parameter *add_sub_type* (ActiveHigh, ActiveLow).

If only the adder or the subtractor is required, the scalar input port **add_sub** can be driven by a constant.

If **add_sub** is driven by a 1 with positive polarity (*add_sub_type* = ActiveHigh) an adder is implemented or a subtractor with negative polarity, *add_sub_type* = ActiveLow). If only an

adder or a subtractor is used, do not synthesize this part separately and use flatten for the instance of this part.

If input ports **clk** and **rst** (for asynchronous behavior) are not enabled, the output ports of the flip-flops retain their values.

This part is equivalent to a *n*-bit adder_subtractor, an *n*-bit multiplexer and an *n*-bit register; where *n* is the width of the ports **din** and **dout**. The hardware cost depends upon the usage of specific scalar input ports. The implementation can be controlled via the synthesis script.

If **clk_en** is not connected, it is driven by 1. If the clock enable feature is not used, the port **clk_en** must be unconnected and the parameter *clk_en_type* must be set to ActiveHigh. Port **clk_en** can also be driven by an active driver, in which case the *clk_en_type* must be correctly adjusted. The HDL code for **clk_en** is optimized away if it is not used.

If port **cin** is unconnected, it is driven by 0.

Function

An enabled **clk** trigger is a trigger that occurs when input port **clk_en** is active.

If **rst** is synchronous, then at every enabled **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

dout	= *rst_val*	if **rst** is active, else
	= **din**	if **load** is active, else
	= **dout** + **din** + **cin**	if addition
	= **dout** - **din** - **cin**	otherwise

If **rst** is asynchronous:

dout	= *rst_val*	if **rst** is active (irrespective of the **clk** trigger), else at every enabled **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):
	= **din**	if **load** is active, else
	= **dout** + **din** + **cin**	if addition
	= **dout** - **din** - **cin**	otherwise

Truth Table

Asynchronous high **rst** and positive polarities (for negative polarities invert the values)

Table 6-5. Accumulator Truth Table — Asynchronous high rst, Positive Polarities

clk	rst	load	addsub	clk_en	creg
-	1	-	-	-	rst_val
-	0	-	-	0	creg
Posedge	0	1	-	1	din
Posedge	0	0	1	1	creg+din+cin

Table 6-5. Accumulator Truth Table — Asynchronous high rst, Positive Polarities (cont.)

clk	rst	load	addsub	clk_en	creg
Posedge	0	0	0	1	creg-din-cin

Synchronous high **rst** and positive polarities (for negative polarities invert the values)

Table 6-6. Accumulator Truth Table — Synchronous high rst, Positive Polarities

clk	rst	load	addsub	clk_en	creg
-	-	-	-	0	creg
Posedge	1	-	-	1	rst_val
Posedge	0	1	-	1	din
posedge	0	0	1	1	creg+din+cin
posedge	0	0	0	1	creg-din-cin

creg is the internal register.

dout = *creg*

Parameters

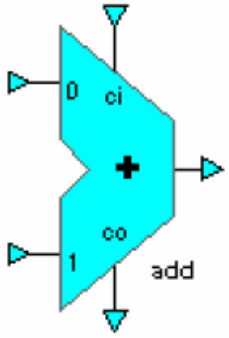
Table 6-7. Accumulator Parameters

Parameter	Values	Default
din, dout	Port widths (must be > 0)	Automatic
rst_val	Reset value (must be >= 0)	0
sign_type	Unsigned, Signed	Unsigned
load_type	ActiveHigh, ActiveLow	ActiveHigh
cin_type	ActiveHigh, ActiveLow	ActiveHigh
add_sub_type	ActiveHigh, ActiveLow	ActiveHigh
rst_type	SyncActiveHigh, AsyncActiveHigh, SyncActiveLow, AsyncActiveLow	AsyncActiveHigh
clk_type	Polarity (Rising, Falling)	Rising
clk_en_type	ActiveHigh, ActiveLow	ActiveHigh
cout_type	ActiveHigh, ActiveLow	ActiveHigh

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if any of the ports **clk**, **clk_en**, **add_sub**, **load**, **cin**, **cout** and **rst** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any of the ports **din**, **load**, **add_sub**, **rst**, **clk** or **dout** are not connected.

Adder (add)



This part adds the values of the input buses **din0**, **din1** and scalar input port **cin** and places the result in the output port **dout**.

The carry out from the addition is placed in scalar output port **cout**.

The addition can be performed in unsigned or signed mode by setting the parameter *sign_type*. Parameters *cin_type* and *cout_type* control the polarity of ports **cin** and **cout**.

This part is equivalent to *n*-bit adder; where *n* is the maximum width of the ports **din** and **dout**. The implementation of the adder is controlled by your synthesis script.

If input port **cin** is unconnected, it is driven by 0.

Function

dout = **din0** + **din1** + **cin**

The addition is unsigned or signed depending of the value of *sign_type*.

If *cout_type* = ActiveHigh:

cout = 1 if there is a carry out during addition
= 0 otherwise

If *cout_type* = ActiveLow:

cout = 0 if there is a carry out during addition
= 1 otherwise

Truth Tables

All the following examples are for positive polarities and **cin** = 0.

Unsigned, 3-bit **din0**, **din1** and **dout**:

Table 6-8. Adder Truth Table — Unsigned, 3-bit din0, din1 and dout

din0<2:0>	din1<2:0>	dout<2:0>	cout
000	010	010	0
001	010	011	0
010	010	100	0
011	010	101	0
100	010	110	0
101	010	111	0
110	010	000	1

Table 6-8. Adder Truth Table — Unsigned, 3-bit din0, din1 and dout (cont.)

din0<2:0>	din1<2:0>	dout<2:0>	cout
111	010	001	1

Unsigned, 3-bit **din0**, 4-bit **din1**, 4-bit **dout**:

Table 6-9. Adder Truth Table — Unsigned, 3-bit din0, 4-bit din1, 4-bit dout

din0<2:0>	din1<3:0>	dout<3:0>	cout
000	1111	1111	0
001	1111	0000	1
010	1111	0001	1
011	1111	0010	1
100	1111	0011	1
101	1111	0100	1
110	1111	0101	1
111	1111	0110	1

Unsigned, 3-bit **din0**, 4-bit **din1**, 5-bit **dout**:

Table 6-10. Adder Truth Table — Unsigned, 3-bit din0, 4-bit din1, 5-bit dout

din0<2:0>	din1<3:0>	dout<4:0>	cout
000	1111	01111	0
001	1111	10000	0
010	1111	10001	0
011	1111	10010	0
100	1111	10011	0
101	1111	10100	0
110	1111	10101	0
111	1111	10110	0

Signed, 3-bit **din0**, **din1** and **dout**:

Table 6-11. Adder Truth Table — Signed, 3-bit din0, din1 and dout

din0<2:0>	din1<2:0>	dout<2:0>	cout
000	010	010	0
001	010	011	0

Table 6-11. Adder Truth Table — Signed, 3-bit din0, din1 and dout (cont.)

din0<2:0>	din1<2:0>	dout<2:0>	cout
010	010	100	1
011	010	101	1
100	010	110	0
101	010	111	0
110	010	000	0
111	010	001	0

Signed, 3-bit **din0**, 4-bit **din1**, 4-bit **dout**:

Table 6-12. Adder Truth Table — Signed, 3-bit din0, 4-bit din1, 4-bit dout

din0<2:0>	din1<3:0>	dout<3:0>	cout
000	0111	0111	0
001	0111	1000	1
010	0111	1001	1
011	0111	1010	1
100	0111	0011	0
101	0111	0100	0
110	0111	0101	0
111	0111	0110	0
100	1000	0100	1

Signed, 3-bit **din0**, 4-bit **din1**, 5-bit **dout**:

Table 6-13. Adder Truth Table — Signed, 3-bit din0, 4-bit din1, 5-bit dout

din0<2:0>	din1<3:0>	dout<4:0>	cout
000	0111	00111	0
001	0111	01000	0
010	0111	01001	0
011	0111	01010	0
100	0111	00011	0
101	0111	00100	0
110	0111	00101	0
111	0111	00110	0

Table 6-13. Adder Truth Table — Signed, 3-bit din0, 4-bit din1, 5-bit dout

din0<2:0>	din1<3:0>	dout<4:0>	cout
100	1000	10100	0

Parameters

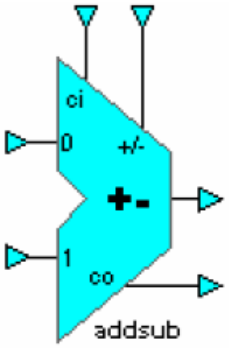
Table 6-14. Adder Parameters

Parameter	Values	Default
din0, din1, dout	Port widths (must be > 0)	Automatic
sign_type cin_type cout_type	Unsigned, Signed ActiveHigh, ActiveLow ActiveHigh, ActiveLow	Unsigned ActiveHigh ActiveHigh

Design Rule Checks

- An error is issued if the width of any port cannot be determined, if either of the ports **cin** and **cout** do not have a fixed width of 1 or if the width of **dout** is less than the maximum width of both input ports.
- A warning is issued and HDL generation fails for this part if any of the ports **din0**, **din1** or **dout** are not connected.

Adder Subtractor (addsub)



This part adds or subtracts the values of the input bus **din0**, input bus **din1** and scalar input port **cin** and places the result in the output bus **dout**. The carry out from this result is placed in scalar output port **cout**.

The addition or subtraction is decided by the value of the scalar input port **add_sub**. The polarity of this port is controlled by the enumerated parameter *add_sub_type*.

The parameter *sign_type* selects the mode of the arithmetic operation: unsigned and signed.

Parameters *cin_type* and *cout_type* control the polarity of ports **cin** and **cout**.

This part is equivalent to an *n*-bit Adder Subtractor; where *n* is the maximum width of the ports **din** and **dout**. The implementation of the Adder Subtractor is controlled via the synthesis script.

The default action is addition. You are advised to use the Adder (add) if only the addition operation is required. Synthesis of Adder Subtractor (addsub) infers both an adder and subtractor.

If input port **cin** is unconnected, it is driven by 0.

Function

dout	= **din0** + **din1** + **cin**	if **add_sub** = 1 and *add_sub_type* = ActiveHigh
	= **din0** - **din1** - **cin**	if **add_sub** = 0 and *add_sub_type* = ActiveHigh
	= **din0** + **din1** + **cin**	if **add_sub** = 0 and *add_sub_type* = ActiveLow
	= **din0** - **din1** - **cin**	if **add_sub** = 1 and *add_sub_type* = ActiveLow

The addition or subtraction is unsigned or signed depending of the value of the parameter *sign_type*.

If parameter *cout_type* = ActiveHigh:

cout	= 1	if there is a carry out in the arithmetic operation
	= 0	otherwise

Truth Table

A 3-bit Example (*sign_type* = unsigned, positive polarities and **cin** = 0)

Table 6-15. Adder Subtractor Truth Table — Three-bit, Unsigned, Positive Polarities, cin=0

din0<2:0>	din1<2:0>	addsub	dout<2:0>	cout
000	010	1	010	0
001	010	1	011	0
010	010	1	100	0

Table 6-15. Adder Subtractor Truth Table — Three-bit, Unsigned, Positive Polarities, cin=0 (cont.)

din0<2:0>	din1<2:0>	addsub	dout<2:0>	cout
011	010	1	101	0
100	010	1	110	0
101	010	1	111	0
110	010	1	000	1
111	010	1	001	1
000	010	0	110	1
001	010	0	111	1
010	010	0	000	0
011	010	0	001	0
100	010	0	110	0
101	010	0	011	0
110	010	0	00	1
111	010	0	101	1

A 3-bit example (*sign_type* = signed, **cin** = 0):

Table 6-16. Adder Subtractor Truth Table — Three-bit, Signed, cin=0

din0<2:0>	din1<2:0>	addsub	dout<2:0>	cout
000	010	1	010	0
001	010	1	011	0
010	010	1	100	1
011	010	1	101	1
100	010	1	110	0
101	010	1	111	0
110	010	1	000	0
111	010	1	001	0
000	010	0	110	1
001	010	0	111	1
010	010	0	000	0
011	010	0	001	0
100	010	0	110	1

Table 6-16. Adder Subtractor Truth Table — Three-bit, Signed, cin=0 (cont.)

din0<2:0>	din1<2:0>	addsub	dout<2:0>	cout
101	010	0	011	1
110	010	0	00	0
111	010	0	101	0

Parameters

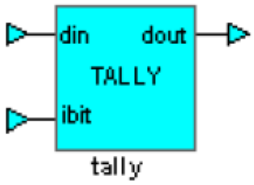
Table 6-17. Adder Subtractor Parameters

Parameter	Values	Default
din0, din1, dout	Port widths (must be > 0)	Automatic
sign_type cin_type add_sub_type cout_type	Unsigned, Signed ActiveHigh, ActiveLow ActiveHigh, ActiveLow ActiveHigh, ActiveLow	Unsigned ActiveHigh ActiveHigh ActiveHigh

Design Rule Checks

- An error is issued if the width of any port cannot be determined, if any of the ports **cin**, **add_sum** and **cout** do not have a fixed width of 1 or if the width of **dout** is less than the maximum width of both input ports.
- A warning is issued and HDL generation fails for this part if any of the ports **din0**, **din1**, **add_sum** or **dout** are not connected.

Bit Tally (tally)



This part counts the number of bits of input data on bus port **din** matching the scalar input port **ibit**. The result is placed in the output data port **dout**.

If the width of the output port **dout** is greater than the width required to store the result, 0 is padded in the result. If the width of the output port is less than the width required to store the result, extra most significant bits from the result are ignored.

This part is equivalent to *(n-1)* single-bit incrementers, where *n* is width of input port **din**. Note that the hardware performance depends upon the width of the output port **dout**.

Function

dout = N where N is the number of bits of **din** equal to **ibit**

Truth Table

An 8-bit input, 3-bit output Example

Table 6-18. Bit Tally Truth Table

din<7:0>	**ibit**	**dout<2:0>**
00100010	0	110
01001111	0	011
01110100	1	100
10111011	1	110

Parameters

Table 6-19. Bit Tally Parameters

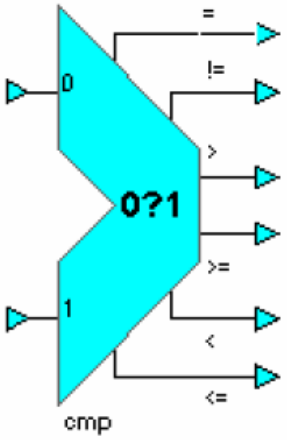
Parameter	**Values**	**Default**
din, dout	Port widths (must be > 1)	Automatic

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if port **ibit**, does not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any of the ports are not connected.

Comparator (cmp)

This part compares the input ports **din0** and **din1**.



The result is placed on the scalar output ports **eq** (=), **ne** (!=), **gt** (>), **ge** (>=), **lt** (<) and **le** (<=).

If both **din0** and **din1** are signed, the comparison is a signed operation. If both are unsigned, the comparison is an unsigned operation, otherwise the *sign_type* parameter is used to determine the type of the comparison operation.

This part is equivalent to an *n*-bit comparator where *n* is the width of the ports **din0** and **din1**. Using more than one output port can increase the hardware cost. The implementation is controlled via the synthesis script.

Function

eq = 1 if **din0** value is equal to **din1** value
= 0 otherwise

ne = 1 if **din0** value is not equal to **din1** value
= 0 otherwise

gt = 1 if **din0** value is greater than **din1** value
= 0 otherwise

ge = 1 if **din0** value is greater than or equal to **din1** value
= 0 otherwise

lt = 1 if **din0** value is less than **din1** value
= 0 otherwise

le = 1 if **din0** value is less than or equal to **din1** value
= 0 otherwise

Truth Table

A 3-bit example:

Table 6-20. Comparator Truth Table

din0<2:0>	din1<2:0>	eq	ne	gt	ge	lt	le
000	010	0	1	0	0	1	1
001	010	0	1	0	0	1	1
010	010	1	0	0	1	0	1
011	010	0	1	1	1	0	0
100	010	0	1	1	1	0	0
101	010	0	1	1	1	0	0

Table 6-20. Comparator Truth Table (cont.)

din0<2:0>	din1<2:0>	eq	ne	gt	ge	lt	le
110	010	0	1	1	1	0	0
111	010	0	1	1	1	0	0

Parameters

Table 6-21. Comparator Parameters

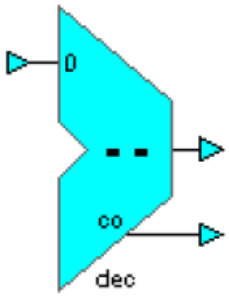
Parameter	Values	Default
din0, din1	Port widths (must be > 0)	Automatic
eq_type ne_type gt_type ge_type lt_type le_type sign_type	ActiveHigh, ActiveLow, None ActiveHigh, ActiveLow, None ActiveHigh, ActiveLow, None ActiveHigh, ActiveLow, None ActiveHigh, ActiveLow, None ActiveHigh, ActiveLow, None Unsigned, Signed	ActiveHigh ActiveHigh ActiveHigh ActiveHigh ActiveHigh ActiveHigh Unsigned

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if any of the ports **eq**, **ne**, **gt**, **ge**, **lt** and **le** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if either of the ports **din0** or **din1** are not connected or unless at least one port of **eq**, **ne**, **gt**, **ge**, **lt**, **le** is connected.

Decrementer (dec)

This part decrements the value of the input port **din** and places the result in the output port **dout**.



The carry out from this result is placed in scalar output port **cout**.

The decrement can be performed in unsigned or signed mode by setting the parameter *sign_type*. Parameter *cout_type* controls the polarity of port **cout**.

This part is equivalent to an *n*-bit decrementer; where *n* is the width of the ports **din** and **dout**. The implementation of the decrementer is controlled via the synthesis script.

Function

dout = **din** - 1

The decrement is unsigned or signed depending of the value of *sign_type*.

cout = 1 if there is a carry out during decrement
= 0 otherwise

Truth Table

A 3-bit example (*sign_type* = unsigned, positive polarities):

Table 6-22. Decrementer Truth Table — 3-bit, Unsigned, Positive Polarities

din<2:0>	dout<2:0>	cout
000	111	1
001	000	0
010	001	0
011	010	0
100	011	0
101	100	0
110	101	0
111	110	0

A 3-bit example (*sign_type* = signed):

Table 6-23. Decrementer Truth Table — 3-bit, Signed

din<2:0>	dout<2:0>	cout
000	111	0
001	000	0

Table 6-23. Decrementer Truth Table — 3-bit, Signed (cont.)

din<2:0>	dout<2:0>	cout
010	001	0
011	010	0
100	011	1
101	100	0
110	101	0
111	110	0

Parameters

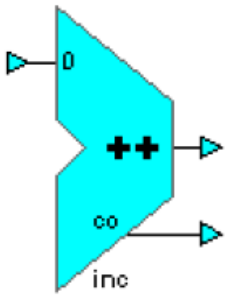
Table 6-24. Decrementer Parameters

Parameter	Values	Default
din, dout	Port widths (must be > 0)	Automatic
sign_type cout_type	Unsigned, Signed ActiveHigh, ActiveLow	Unsigned ActiveHigh

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if port **cout**, does not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if either of the ports **din** or **dout** are not connected.

Incrementer (inc)



This part increases the value of the input data port **din** and places the result in the output data port **dout**.

The carry out from this result is placed in scalar output port **cout**.

The increment can be performed in unsigned or signed mode by setting the parameter *sign_type*. Parameter *cout_type* controls the polarity of port **cout**.

This part is equivalent to an *n*-bit incrementer; where *n* is the width of the ports **din** and **dout**. The implementation of the incrementer is controlled via the synthesis script.

Function

dout = **din** + 1

The increment is unsigned or signed depending of the value of *sign_type*.

cout = 0 if there is a carry out during increment
= 1 otherwise

Truth Table

A 3-bit example (sign_type = unsigned, positive polarities):

Table 6-25. Incrementer Truth Table — 3-bit, Unsigned, Positive Polarities

din<2:0>	dout<2:0>	cout
000	001	0
001	010	0
010	011	0
011	100	0
100	101	0
101	110	0
110	111	0
111	000	1

A 3-bit example (sign_type = signed):

Table 6-26. Incrementer Truth Table — 3-bit, Signed

din<2:0>	dout<2:0>	cout
000	001	0
001	010	0

Table 6-26. Incrementer Truth Table — 3-bit, Signed (cont.)

din<2:0>	dout<2:0>	cout
010	011	0
011	100	1
100	101	0
101	110	0
110	111	0
111	000	0

Parameters

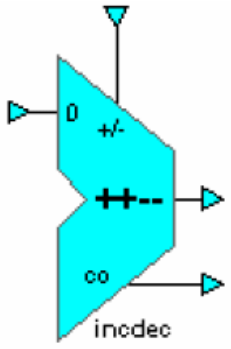
Table 6-27. Incrementer Parameters

Parameter	Values	Default
din, dout	Port widths (must be > 0)	Automatic
sign_type cout_type	Unsigned, Signed ActiveHigh, ActiveLow	Unsigned ActiveHigh

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if port **cout**, does not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if either of the ports **din** or **dout** are not connected.

Incrementer Decrementer (incdec)



This part increments or decrements the value of the input port **din** and places the result in the output port **dout**. The carry out from this result is placed in scalar output port **cout**.

The addition or subtraction is decided by the value of the scalar input port **inc_dec**. The polarity of this port is controlled by the enumerated parameter *inc_dec_type*.

The parameter *sign_type* selects one of two decrement modes: Unsigned and Signed.

Parameter *cout_type* controls the polarity of port **cout**.

This part is equivalent to an *n*-bit Incrementer or Decrementer; where *n* is the width of the ports **din** and **dout**. The implementation of the increment or decrement is controlled via the synthesis script.

The default action is increment. Use the Incrementer (inc) part if only the increment operation is needed. Synthesis tools may infer both an increment and decrement for the Incrementer Decrementer (incdec) part.

Function

dout = **din0** + 1 if **inc_dec** = 1 and *inc_dec_type* = ActiveHigh
= **din0** - 1 if **inc_dec** = 0 and *inc_dec_type* = ActiveHigh
= **din0** + 1 if **inc_dec** = 0 and *inc_dec_type* = ActiveLow
= **din0** - 1 if **inc_dec** = 1 and *inc_dec_type* = ActiveLow

The arithmetic is unsigned or signed, depending on the value of the parameter *sign_type*.

cout = 1 if there is a carry out during the arithmetic operation
= 0 otherwise

Truth Table

A 3-bit example (*sign_type* = unsigned, positive polarities):

Table 6-28. Incrementer Decrementer Truth Table — 3-bit, Unsigned, Positive Polarities

din<2:0>	incdec	dout<2:0>	cout
000	1	001	0
001	1	010	0
010	1	011	0
011	1	100	0
100	1	101	0
101	1	110	0

Table 6-28. Incrementer Decrementer Truth Table — 3-bit, Unsigned, Positive Polarities (cont.)

din<2:0>	incdec	dout<2:0>	cout
110	1	111	0
111	1	000	1
000	0	111	1
001	0	000	0
010	0	001	0
011	0	010	0
100	0	011	0
101	0	100	0
110	0	101	0
111	0	110	0

A 3-bit example (*sign_type* = signed):

Table 6-29. Incrementer Decrementer Truth Table — 3-bit, Signed

din<2:0>	incdec	dout<2:0>	cout
000	1	001	0
001	1	010	0
010	1	011	0
011	1	100	1
100	1	101	0
101	1	110	0
110	1	111	0
111	1	000	0
000	0	111	0
001	0	000	0
010	0	001	0
011	0	010	0
100	0	011	1
101	0	100	0
110	0	101	0
111	0	110	0

Parameters

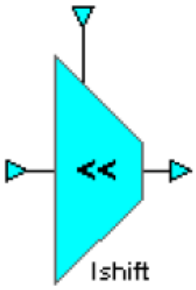
Table 6-30. Incrementer Decrementer Parameters

Parameter	Values	Default
din, dout	Port widths (must be > 0)	Automatic
sign_type inc_dec_type cout_type	Unsigned, Signed ActiveHigh, ActiveLow ActiveHigh, ActiveLow	Unsigned ActiveHigh ActiveHigh

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if either of ports **cout** or **inc_dec** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any of the ports **din**, **inc_dec** or **dout** are not connected.

Left Shifter (lshift)



This part left shifts the input port **din** by the value given on the input port **shift**.

The result of this shift operation is placed in the output port **dout**. The enumerated parameter *mode* (Logical, Arithmetic, Circular) selects one of three shift modes.

For logical and arithmetic mode, 0 is shifted in the least significant bit.

For circular mode, the most significant bit of the input port **din** is shifted into the least significant bit.

This part is equivalent to an *n*-bit left shifter; where *n* is the width of the ports **din** and **dout**. The implementation is controlled via the synthesis script. Silicon compilers perform better with this part in terms of hardware performance.

Function

dout = **din** << **shift** if *mode* = logical or arithmetic, 0 is placed in the LSB
if mode = circular, MSB of **din** is placed in the LSB

Truth Table

A 4-bit input example (mode = logical or arithmetic):

Table 6-31. Left Shifter Truth Table — 4-bit Input, Mode = Logical/Arithmetic

din<3:0>	shift	dout
1101	101 (5)	0000
1101	010 (2)	0100

A 4-bit input example (mode = circular):

Table 6-32. Left Shifter Truth Table — 4-bit Input, Mode = Circular

din<3:0>	shift	dout
1001	101 (5)	0011
1001	010 (2)	0110

Parameters

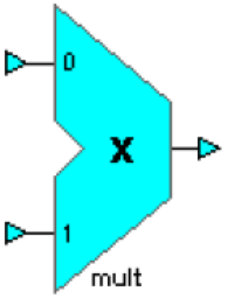
Table 6-33. Left Shifter Parameters

Parameter	Values	Default
din, dout shift	Port widths (must be > 1) Port width (must be > 0)	Automatic Automatic
mode	Logical, Arithmetic, Circular	Logical

Design Rule Checks

- An error is issued if the width of any port cannot be determined or the width of port **din** is less than 2 bits.
- A warning is issued and HDL generation fails for this part if any of the ports are not connected.
- A warning is issued but the generation succeeds if the shift value is greater than the **din** port size in the circular mode.

Multiplier (mult)



The multiplier part inputs data bus ports **din0** and **din1** and places the result in the output port **prod**.

The multiplication can be performed in two modes: unsigned and signed controlled by the value of the enumerated parameter *sign_type* (Unsigned, Signed).

If the width of the output port **prod** is greater than the sum of the widths of the input ports, 0 is padded in the result.

If the width of the output port is less than the sum of the widths of the input ports, the most significant bits from the result are ignored.

This part is equivalent to an *m***n*-bit multiplier where *m* and *n* are the widths of the ports **din0** and **din1**. The implementation of the multiplier is controlled via the synthesis script. The hardware performance depends on the width of the output port **prod**. This is a very large part and it is difficult to estimate the hardware performance. This part significantly effects the place and route of a design.

Function

prod = **din0** * **din1**

The multiplication is unsigned or signed, depending on the value of the parameter *sign_type*.

Truth Table

A 3-bit input, 6-bit output example (*sign_type* = unsigned):

Table 6-34. Multiplier Truth Table — 3-bit Input, 6-bit Output, Unsigned

din0<2:0>	din1<2:0>	prod<5:0>
001	011	000011 (3)
100	011	001100 (12)
011	011	001001 (9)
111	011	010101 (21)
001	101	000101 (5)
100	101	010100 (20)
011	101	001111 (15)
111	101	100011 (35)

A 3-bit input, 6-bit output example (*sign_type* = signed):

Table 6-35. Multiplier Truth Table — 3-bit Input, 6-bit Output, Signed

din0<2:0>	din1<2:0>	prod<5:0>
001	011	000011 (3)
100	011	110100 (-12)
011	011	001001 (9)
111	011	111101 (-3)
001	101	111101 (-3)
100	101	001100 (12)
011	101	110111 (-9)
111	101	000011 (3)

Parameters

Table 6-36. Multiplier Parameters

Parameter	Values	Default
din0 din1 prod	Port width (must be > 0) Port width (must be > 0) Port width (must be > 0)	Automatic Automatic Automatic

Design Rule Checks

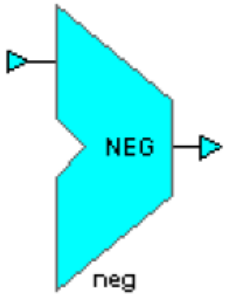
- An error is issued if the width of any port cannot be determined.
- A warning is issued and HDL generation fails for this part if any of the ports are not connected.

Negate (neg)

This part negates the value of the input data port **din** and puts the result in the output data port **dout**.

The data is assumed to be in 2's complement mode.

This part is equivalent to an *n*-bit incrementer and *n* inverters; where *n* is the width of the ports **din** and **dout**. The implementation can be controlled via the synthesis script.



Function

dout = 0 - **din**

Truth Table

A 3-bit example:

Table 6-37. Negate Truth Table

din<2:0>	dout<2:0>
000	000
001	111
010	110
011	101
100	100
101	011
110	010
111	001

Parameters

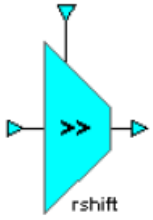
Table 6-38. Negate Parameters

Parameter	Values	Default
din, dout	Port widths (must be > 0)	Automatic

Design Rule Checks

- An error is issued if the width of any port cannot be determined.
- A warning is issued and HDL generation fails for this part if any of the ports are not connected.

Right Shifter (rshift)



This part right shifts the input port **din** by the value given by the input port **shift**.

The result of this shift operation is placed in the output port **dout**. The enumerated parameter *mode* (Logical, Arithmetic, Circular) selects one of three shift modes.

For logical mode, 0 is shifted in the most significant bit.

For arithmetic mode, the most significant bit of input port **din** is retained.

For circular mode, the least significant bit of the input port **din** is shifted into the most significant bit.

This part is equivalent to an *n*-bit right shifter; where *n* is the width of the ports **din** and **dout**. The implementation is controlled via the synthesis script. Silicon compilers perform better at this part in terms of hardware performance.

Function

dout = **din** >> **shift**

if mode = logical, 0 is placed in the MSB
if mode = arithmetic, the MSB is retained
if mode = circular, LSB of **din** is placed in the MSB

Truth Table

A 4-bit input example (mode = logical):

Table 6-39. Right Shifter Truth Table — 4-bit Input, Mode = Logical

din<3:0>	**shift**	**dout**
1101	101 (5)	0000
1101	010 (2)	0011

A 4-bit input example (mode = arithmetic):

Table 6-40. Right Shifter Truth Table — 4-bit Input, Mode = Arithmetic

din<3:0>	**shift**	**dout**
1001	101 (5)	1111
1001	010 (2)	1110

A 4-bit input example (mode = circular):

Table 6-41. Right Shifter Truth Table — 4-bit Input, Mode = Circular

din<3:0>	**shift**	**dout**
1001	101 (5)	1100
1001	010 (2)	0110

Parameters

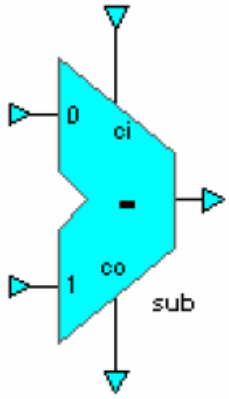
Table 6-42. Right Shifter Parameters

Parameter	Values	Default
din, dout shift	Port widths (must be > 1) Port width (must be > 0)	Automatic Automatic
mode	Logical, Arithmetic, Circular	Logical

Design Rule Checks

- An error is issued if the width of any port cannot be determined or the width of **din** is less than 2.
- A warning is issued and HDL generation fails for this part if any of the ports are not connected.
- A warning is issued but the generation succeeds if the shift value is greater than the **din** port size in the circular mode.

Subtractor (sub)



This part subtracts the input values of port **din1** and scalar port **cin** from the port **din0** and places the result in the output data port **dout**.

The carry out from this result is placed in the scalar output port **cout**. The subtraction can be performed in signed and unsigned mode. The mode is selected by the parameter *sign_type*. Parameters *cin_type* and *cout_type* control the polarity of ports **cin** and **cout**.

This part is equivalent to an *n*-bit subtractor; where *n* is the width of the ports **din** and **dout**. The implementation of the subtractor is controlled via the synthesis script.

If input port **cin** is unconnected, it is driven by 0.

Function

dout = **din0 - din1 - cin**

The subtraction is signed or unsigned depending of the value of *sign_type*.

If *cout_type* = ActiveHigh:

cout = 1 if there is a carry out during subtraction
= 0 otherwise

If *cout_type* = ActiveLow:

cout = 0 if there is a carry out during subtraction
= 1 otherwise

Truth Tables

All the following examples are for positive polarities and **cin** = 0.

Unsigned, 3-bit **din0**, **din1** and **dout**:

Table 6-43. Subtractor Truth Table — Unsigned, 3-bit din0, din1 and dout

din0<2:0>	din1<2:0>	dout<2:0>	cout
000	010	110	1
001	010	111	1
010	010	000	0
011	010	001	0
100	010	010	0
101	010	011	0
110	010	100	0
111	010	101	0

Unsigned, 3-bit **din0**, 4-bit **din1**, 4-bit **dout**:

Table 6-44. Subtractor Truth Table — Unsigned, 3-bit din0, 4-bit din1, 4-bit dout

din0<2:0>	din1<3:0>	dout<3:0>	cout
000	0010	1110	1
001	0010	1111	1
010	0010	0000	0
011	0010	0001	0
100	0010	0010	0
101	0010	0011	0
110	0010	0100	0
111	0010	0101	0

Unsigned, 3-bit **din0**, 4-bit **din1**, 5-bit **dout**:

Table 6-45. Subtractor Truth Table — Unsigned, 3-bit din0, 4-bit din1, 5-bit dout

din0<2:0>	din1<3:0>	dout<4:0>	cout
000	0010	11110	1
001	0010	11111	1
010	0010	00000	0
011	0010	00001	0
100	0010	00010	0
101	0010	00011	0
110	0010	00100	0
111	0010	00101	0

Signed, 3-bit **din0**, **din1** and **dout**:

Table 6-46. Subtractor Truth Table — Signed, 3-bit din0, din1 and dout

din0<2:0>	din1<2:0>	dout<2:0>	cout
000	010	110	0
001	010	111	0
010	010	000	0
011	010	001	0

Table 6-46. Subtractor Truth Table — Signed, 3-bit din0, din1 and dout (cont.)

din0<2:0>	din1<2:0>	dout<2:0>	cout
100	010	010	1
101	010	011	1
110	010	100	0
111	010	101	0

Signed, 3-bit **din0**, 4-bit **din1**, 4-bit **dout**:

Table 6-47. Subtractor Truth Table — Signed, 3-bit din0, 4-bit din1, 4-bit dout

din0<2:0>	din1<3:0>	dout<3:0>	cout
000	0010	1110	0
001	0010	1111	0
010	0010	0000	0
011	0010	0001	0
100	0010	1010	0
101	0010	1011	0
110	0010	1100	0
111	0010	1101	0
000	1000	1000	1

Signed, 3-bit **din0**, 4-bit **din1**, 5-bit **dout**:

Table 6-48. Subtractor Truth Table — Signed, 3-bit din0, 4-bit din1, 5-bit dout

din0<2:0>	din1<3:0>	dout<4:0>	cout
000	0010	11110	0
001	0010	11111	0
010	0010	00000	0
011	0010	00001	0
100	0010	11010	0
101	0010	11011	0
110	0010	11100	0
111	0010	11101	0
000	1000	01000	0

If sign_type is unsigned:

Then both the inputs (din0 an din1) and the main output (dout) of the subtractor should be interpreted as unsigned numbers.

The cout bit may be interpreted a sign bit and hence when taken together with the dout signal may be used to from a word that is one bit longer than dout which may be interpreted as a signed number. In this scenario overflow is impossible provided dout has at least as many bits as the largest input.

If sign_type is signed:

Then both the inputs (din0 an din1) and the main output (dout) of the subtractor should be interpreted as signed numbers.

The cout bit must be interpreted as an overflow indicator. When cout = 1 then the output "dout" must be interpreted as having overflowed its signed bounds.

Example 1: for the case of Table 6-46: din0=011 "3" and din1=111 "-1" the difference should be ""4" but 4 is too large for a 3 bit signed representation. Therefore cout =1 in this case to indicate that an overflow has occurred. cout cannot be interpreted as a sign bit as in the unsigned scenario above.

Example 2: for the case of Table 6-48, where dout has been oversized to make overflow impossible, if din0=110 "-2" and din1=0010 "2" then the result is "-4" but notice that cout=0! - meaning that it cannot be interpreted as a sign bit as with unsigned arithmetic. It must be interpreted as an overflow and in this case no overflow occurred.

Parameters

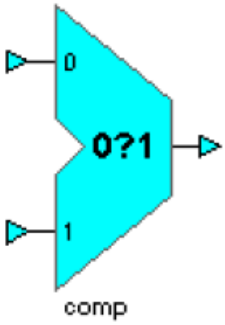
Table 6-49. Subtractor Parameters

Parameter	Values	Default
din0,din1,dout	Port widths (must be > 0)	Automatic
sign_type cin_type cout_type	Unsigned, Signed ActiveHigh, ActiveLow ActiveHigh, ActiveLow	Unsigned ActiveHigh ActiveHigh

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if either or ports **cin** or **cout** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any of the ports **din0**, **din1** or **dout** are not connected.

Uni-function Comparator (comp)



This part is a highly parameterized comparator which compares the input ports **din0** and **din1**. The result is placed in the scalar output port **fout**.

The output is controlled by an enumerated parameter *Fout_function* which supports the comparison functions: EqualTo, NotEqualTo, GreaterThan, GreaterOrEqualTo, LessThan and LessOrEqualTo.

If both **din0** and **din1** are signed, the comparison is a signed operation. If both are unsigned, the comparison is an unsigned operation, otherwise the *sign_type* parameter is used to determine the type of the comparison operation. The polarity of the output port is controlled by an enumerated parameter *fout_type* (ActiveHigh, ActiveLow).

This part is equivalent to an *n*-bit comparator; where *n* is the width of the ports **din0** and **din1**. The implementation is controlled via the synthesis script.

Function

If *fout_function* is EqualTo:

fout	= 1	if **din0** value is equal to **din1** value
	= 0	otherwise

If *fout_function* is NotEqualTo:

fout	= 1	if **din0** value is not equal to **din1** value
	= 0	otherwise

If *fout_function* is GreaterThan:

fout	= 1	if **din0** value is greater than **din1** value
	= 0	otherwise

If *fout_function* is GreaterOrEqualTo:

fout	= 1	if **din0** value is greater than or equal to **din1** value
	= 0	otherwise

If *fout_function* is LessThan:

fout	= 1	if **din0** value is less than **din1** value
	= 0	otherwise

If *fout_function* is LessOrEqualTo:

fout	= 1	if **din0** value is less than or equal to **din1** value
	= 0	otherwise

Truth Table

A 3-bit example (**fout** with positive polarity, invert values for negative polarity):

Table 6-50. Uni-function Comparator Truth Table

din0<2:0>	din1<2:0>	fout (Equal to)	fout (Not Equal to)	fout (Greater than)	fout (Greater than or Equal to)	fout (Less than)	fout (Less than or Equal to)
000	010	0	1	0	0	1	1
001	010	0	1	0	0	1	1
010	010	1	0	0	1	0	1
011	010	0	1	1	1	0	0
100	010	0	1	1	1	0	0
101	010	0	1	1	1	0	0
110	010	0	1	1	1	0	0
111	010	0	1	1	1	0	0

Parameters

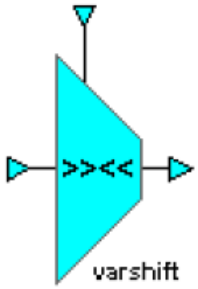
Table 6-51. Uni-function Comparator Parameters

Parameter	Values	Default
din0, din1	Port widths (must be > 0)	Automatic
fout_type	ActiveHigh, ActiveLow	ActiveHigh
fout_function	EqualTo, NotEqualTo, GreaterThan, GreaterOrEqualTo, LessThan, LessOrEqualTo	EqualTo
sign_type	Unsigned, Signed	Unsigned

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if port **fout** does not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any of the ports are not connected.

Variable Shifter (varshift)



This part right or left shifts the input data port **din** by the value given by the input port **shift**. The result of this shift operation is placed in the output port **dout**.

Logical and Arithmetic left shift behave the same way because the negative data is assumed to be in 2's complement format Thus the generated code in both cases are the same.

The direction is decided by interpreting the bits of the input port **shift** in 2’s compliment. A negative number implies right shift while a positive number implies left shift.

The shift operation can be performed in one of three modes set by the enumerated parameter *mode* (Logical, Arithmetic, Circular).

For logical mode in right shift, 0 is shifted in the most significant bit.

For arithmetic mode in right shift, the most significant bit of input port **din** is retained.

For circular mode in right shift, the least significant bit of the input port **din** is shifted into the most significant bit. For logical and arithmetic mode in left shift, 0 is shifted in the least significant bit.

For circular mode in left shift, the most significant bit of the input port **din** is shifted into the least significant bit.

This part is very costly in terms of the gate count. Different implementation strategies are needed to achieve better hardware performance. Even though synthesis tools may generate a reasonable implementation other options such as silicon compilers and optimized hard macros should be explored.

This part is equivalent to an *n*-bit right and left shifter; where *n* is the width of the ports **din** and **dout**. The implementation can be controlled via the synthesis script.

Function

dout = **din** << **shift** if **shift** > 0
if *mode* = logical or arithmetic, 0 is placed in the LSB
if *mode* = circular, MSB of **din** is placed in the LSB

dout = **din** >> **shift** if **shift** < 0
if *mode* = logical, 0 is placed in the MSB
if *mode* = arithmetic, the MSB is retained
if *mode* = circular, LSB of din is placed in the MSB

dout = **din** if **shift** = 0

Truth Table

A 4-bit input example (*mode* = logical):

Table 6-52. Variable Shifter Truth Table — 4-bit Input, Mode = Logical

din<3:0>	**shift**	**dout**
1101	000 (0)	1101
1101	001 (1)	1010
1101	010 (2)	0100
1101	011 (3)	1000
1101	100 (-4)	0000
1101	101 (-3)	0001
1101	110 (-2)	0011
1101	111 (-1)	0110

A 4-bit input example (*mode* = arithmetic):

Table 6-53. Variable Shifter Truth Table — 4-bit Input, Mode = Arithmetic

din<3:0>	**shift**	**dout**
1101	000 (0)	1101
1101	001 (1)	1010
1101	010 (2)	0100
1101	011 (3)	1000
1101	100 (-4)	1111
1101	101 (-3)	1111
1101	110 (-2)	1111
1101	111 (-1)	1110

A 4-bit input example (*mode* = circular):

Table 6-54. Variable Shifter Truth Table — 4-bit Input, Mode = Circular

din<3:0>	**shift**	**dout**
1101	000 (0)	1101
1101	001 (1)	1011
1101	010 (2)	0111
1101	011 (3)	1110
1101	100 (-4)	1101
1101	101 (-3)	1011
1101	110 (-2)	0111
1101	111 (-1)	1110

Parameters

Table 6-55. Variable Shifter Parameters

Parameter	**Values**	**Default**
din, dout shift	Port width (must be > 1) Port width (must be > 0)	Automatic Automatic
mode	Logical, Arithmetic, Circular	Logical

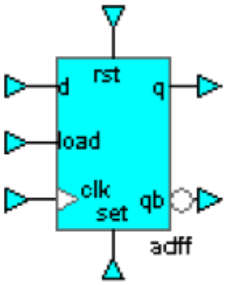
Design Rule Checks

- An error is issued if the width of any port cannot be determined.
- A warning is issued and HDL generation fails for this part if any of the ports are not connected.
- A warning is issued but the generation succeeds if the shift value is greater than the **din** port size in the circular mode.

Chapter 7
Register Parts

This chapter describes the following register parts which implement flip-flops and latches:

D Flip-Flop (adff)



This part is a highly parameterized D-type flip-flop. Upon a trigger of an enabled scalar input port **clk**, the data from the input port **d** is propagated into the output ports **q** and **qb**.

The optional output port **qb** has a bitwise inverted value of the output port **q** and can be enabled by setting the enumerated parameter *qb_type* (Enabled, Disabled). Note that using **qb** does not imply extra inverters, these inverters are usually absorbed by inferred flip-flops because most have a **qb** port.

The part can be triggered on the rising or the falling edge of the scalar input port **clk**. The polarity of **clk** is controlled by the enumerated parameter *clk_type* (Rising, Falling, RisingLast, FallingLast, RisingEdge, FallingEdge) where RisingLast, FallingLast, RisingEdge, FallingEdge are supported for VHDL only.

The flip-flop has a scalar input port **load** with its mode controlled by the enumerated parameter *load_type* (ActiveHigh, ActiveLow, None).

The register can be set or reset to the value of the parameter *set_val* and *rst_val* by activating the scalar input ports **set** and **rst**. The modes for these inputs are controlled by enumerated parameters *set_type* and *rst_type* (AsyncActiveHigh, SyncActiveHigh, AsyncActiveLow, SyncActiveLow, None). If the mode for these inputs is set to None or if they are unconnected, then the respective pins are not visible in the symbol and the functionality for that pin is disabled. The modes for these ports can be synchronous or asynchronous.

The enumerated parameter *initialization* (Enabled, Disabled) determines whether the local registers have an initial value.

This part is equivalent to n flip-flops, where *n* is the width of the ports **d**, **q** and **qb**. The inferred register depends upon the type of **rst** and **set** behavior. The gate count greatly changes if the synthesis tool is able to pick up smaller flip-flops.

If **set**, **rst** or **load** are not connected, the code for unused ports is removed.

Parameters *set_val* and *rst_val* can take LNBF format as described in the Constant Value (constval) model, with no limitation on the size of the **q** and **qb** ports.

Function

If **rst** and **set** are synchronous, then at every **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

q	= *rst_val*	if **rst** is active
	= *set_val*	If **set** is active
	= **d**	If **load** is active or not connected
qb	= NOT(*rst_val*)	If **rst** is active
	= NOT(*set_val*)	If **set** is active
	= NOT(**d**)	If **load** is active or not connected

If **rst** and **set** are asynchronous:

q	= *rst_val*	if **rst** is active (irrespective of the **clk** trigger)
	= *set_val*	if **set** is active (irrespective of the **clk** trigger)
qb	= NOT(*rst_val*)	if **rst** is active (irrespective of the **clk** trigger)
	= NOT(*set_val*)	if **set** is active (irrespective of the **clk** trigger)

and at every **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

q	= d	if **load** is active or not connected
qb	= NOT(**d**)	if **load** is active or not connected

Truth Tables

The tables are for positive polarities. For negative polarities, invert the values.

Asynchronous **rst** and **set**:

Table 7-1. D Flip-Flop Truth Table — Asynchronous rst and set

load	clk	rst	set	q	qb
-	-	1	- / NC	*rst_val*	NOT(*rst_val*)
-	-	0 / NC	1	set_val	NOT(*set_val*)
NC	posedge	0 / NC	0 / NC	d	NOT(d)
1	posedge	0 / NC	0 / NC	d	NOT(d)
0	posedge	0 / NC	0 / NC	q	qb

Synchronous **rst** and **set**:

Table 7-2. D Flip-Flop Truth Table — Synchronous rst and set

load	clk	rst	set	q	qb
-	-	1	- / NC	q	qb
-	-	0 / NC	1	q	qb
-	posedge	1	0 / NC	rst_val	NOT(*rst_val*)
-	posedge	0 / NC	1	set_val	NOT(*set_val*)
NC	posedge	0 / NC	0 / NC	d	NOT(d)
0	posedge	0 / NC	0 / NC	q	qb
1	posedge	0 / NC	0 / NC	d	NOT(d)

Parameters

Table 7-3. D Flip-Flop Parameters

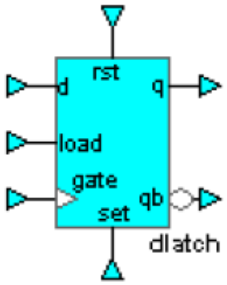
Parameter	Values	Default
d, q, qb	Port widths (must be > 0)	Automatic
clk_type	Rising,Falling,RisingLast,FallingLast,RisingEdge,FallingEdge	Rising
load_type	ActiveHigh,ActiveLow,None	ActiveHigh
qb_type	Enabled,Disabled	Enabled
rst_type	AsyncActiveHigh,SyncActiveHigh,AsyncActiveLow,SyncActiveLow,None	AsyncActiveHigh
set_type	AsyncActiveHigh,SyncActiveHigh,AsyncActiveLow,SyncActiveLow,None	AsyncActiveHigh
initialization	Enabled,Disabled	Disabled
rst_val	Register reset value (must be >= 0)	0
set_val	Register set value (must be >= 0)	1

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if ports **clk**, **load**, **rst**, and **set** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if either ports **d** and **clk** are not connected or if at least one of ports **q** and **qb** are not connected.

D Latch (dlatch)

This part is a highly parameterized D-type latch. Upon a trigger of an enabled scalar input port **gate**, the data from the input port **d** is latched into the output ports **q** and **qb**.



The optional output port **qb** has a bitwise inverted value of the output port **q** and can be enabled by setting the enumerated parameter *qb_type* (Enabled, Disabled). Note that using **qb** does not imply extra inverters, these inverters are usually absorbed by the inferred latches, because most have a **qb** port.

The part can be triggered on the high or the low level of the scalar input port **gate**. The polarity of **gate** is controlled by the enumerated parameter *gate_type* (ActiveHigh, ActiveLow).

The latch has a scalar input port **load** with its mode controlled by the parameter *load_type* (AsyncActiveHigh, SyncActiveHigh, AsyncActiveLow, SyncActiveLow, None). In asynchronous mode, the **load** input has a higher priority over the **gate** input and in synchronous mode, the **gate** input has a higher priority over the **load** input.

The register can be set or reset to the value of the parameter *set_val* and *rst_val* by activating the scalar input ports **set** and **rst**. The modes for these inputs are controlled by enumerated parameters *set_type* and *rst_type* (AsyncActiveHigh, SyncActiveHigh, AsyncActiveLow, SyncActiveLow, None). If the mode for these inputs is set to None or if they are unconnected, then the respective pins are not visible in the symbol and the functionality for that pin is disabled. The modes for these ports can be synchronous or asynchronous.

The enumerated parameter *initialization* (Enabled, Disabled) determines whether the local registers have an initial value.

This part is equivalent to *n* latches where *n* is the width of the ports **d**, **q** and **qb**. The inferred register depends upon the type of **rst** and **set** behavior. The gate count greatly changes if the synthesis tool is able to pick up smaller latches.

If **set**, **rst** or **load** are not connected, the code for unused ports is removed.

Parameters *set_val* and *rst_val* can take LNBF format as described in the Constant Value (constval) model, with no limitation on the size of the **q** and **qb** ports.

Function

If **rst** and **set** are synchronous and **load** is synchronous, then at every **gate** trigger (high level if *gate_type* = ActiveHigh or low level if *gate_type* = ActiveLow):

q	= *rst_val*	If **rst** is active
	= *set_val*	If **set** is active
	= **d**	If **load** is active or not connected
qb	= NOT(*rst_val*)	If **rst** is active
	= NOT(*set_val*)	If **set** is active
	= NOT(**d**)	If **load** is active or not connected

If **load** is set to asynchronous, then at every **load** trigger:

q	= *rst_val*	If **rst** is active and **gate** trigger is present
	= *set_val*	If **set** is active and **gate** trigger is present
	= **d**	If **gate** trigger is present
qb	= NOT(*rst_val*)	If **rst** is active and **gate** trig
	= NOT(*set_val*)	If **set** is active and **gate** trigger is present
	= NOT(**d**)	If **gate** trigger is present

If **rst** and **set** are asynchronous and **load** is set to synchronous:

q	= *rst_val*	if **rst** is active (irrespective of the **gate** trigger)
	= *set_val*	if **set** is active (irrespective of the **gate** trigger)
qb	= NOT(*rst_val*)	if **rst** is active (irrespective of the **gate** trigger
	= NOT(*set_val*)	if **set** is active (irrespective of the **gate** trigger)

and at every **gate** trigger (high level if *gate_type* = ActiveHigh or low level if *gate_type* = ActiveLow):

q	= **d**	if **load** is active or not connected
qb	= NOT(**d**)	if **load** is active or not connected

If **load** is set to asynchronous:

q	= *rst_val*	if **rst** is active (irrespective of the **gate** & **load** trigger)
	= *set_val*	if **set** is active (irrespective of the **gate** & **load** trigger)
qb	= NOT(*rst_val*)	if **rst** is active (irrespective of the **gate** & **load** trigger)
	= NOT(*set_val*)	if **set** is active (irrespective of the **gate** & **load** trigger)

and at every **load** trigger:

q	= **d**	if **gate** trigger is present
qb	= NOT(**d**)	if **gate** trigger is present

Truth Tables

The tables are for positive polarities. For negative polarities, invert the values.

Asynchronous **rst** and **set** and synchronous **load**:

Table 7-4. D Latch Truth Table — Asynchronous rst and set, Synchronous load

load	gate	rst	set	q	qb
-	-	1	- / NC	rst_val	NOT(*rst_val*)
-	-	0 / NC	1	set_val	NOT(*set_val*)
NC	1	0 / NC	0 / NC	d	NOT(d)

Table 7-4. D Latch Truth Table — Asynchronous rst and set, Synchronous load (cont.)

load	gate	rst	set	q	qb
1	1	0 / NC	0 / NC	d	NOT(d)
0	1	0 / NC	0 / NC	q	qb

Asynchronous **rst** and **set** and asynchronous **load**:

Table 7-5. D Latch Truth Table — Asynchronous rst, set and load

load	gate	rst	set	q	qb
-	-	1	- / NC	rst_val	NOT(*rst_val*)
-	-	0 / NC	1	set_val	NOT(*set_val*)
NC	1	0 / NC	0 / NC	d	NOT(d)
0	-	0 / NC	0 / NC	q	qb
1	1	0 / NC	0 / NC	d	NOT(d)

Synchronous **rst** and **set** and synchronous **load**:

Table 7-6. D Latch Truth Table — Synchronous rst, set and load

load	gate	rst	set	q	qb
-	-	1	- / NC	q	qb
-	-	0 / NC	1	q	qb
-	1	1	0 / NC	rst_val	NOT(*rst_val*)
-	1	0 / NC	1	set_val	NOT(*set_val*)
NC	1	0 / NC	0 / NC	d	NOT(d)
0	1	0 / NC	0 / NC	q	qb
1	1	0 / NC	0 / NC	d	NOT(d)

Synchronous **rst** and **set** and asynchronous **load**:

Table 7-7. D Latch Truth Table — Synchronous rst and set, Asynchronous load

load	gate	rst	set	q	qb
-	-	1	- / NC	q	qb
-	-	0 / NC	1	q	qb
1 / NC	1	1	0 / NC	rst_val	NOT(*rst_val*)
1 / NC	1	0 / NC	1	set_va	NOT(*set_val*)

Table 7-7. D Latch Truth Table — Synchronous rst and set, Asynchronous load (cont.)

load	gate	rst	set	q	qb
NC	1	0 / NC	0 / NC	d	NOT(d)
0	1	0 / NC	0 / NC	q	qb
1	1	0 / NC	0 / NC	d	NOT(d)

Parameters

Table 7-8. D Latch Parameters

Parameter	Values	Default
d, q, qb	Port widths (must be > 0)	Automatic
gate_type	ActiveHigh,ActiveLow	ActiveHigh
load_type	AsyncActiveHigh,SyncActiveHigh,AsyncActiveLow,SyncActiveLow,None	SyncActiveHigh
qb_type	Enabled,Disabled	Enabled
rst_type	AsyncActiveHigh,SyncActiveHigh,AsyncActiveLow,SyncActiveLow,None	AsyncActiveHigh
set_type	AsyncActiveHigh,SyncActiveHigh,AsyncActiveLow,SyncActiveLow,None	AsyncActiveHigh
initialization	Enabled,Disabled	Disabled
rst_val	Register reset value (must be >= 0)	0
set_val	Register set value (must be >= 0)	1

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if ports **clk**, **load**, **rst**, and **set** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if either ports **d** and **clk** are not connected or if at least one of ports **q** and **qb** are not connected.

JK Flip-Flop (jkff)

This is a highly parameterized JK flip-flop. Upon a trigger of an enabled scalar input port **clk**, the outputs for the **q** and **qb** ports are generated according to the following truth table:

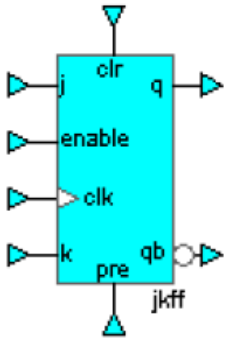


Table 7-9. JK Flip-Flop Truth Table

j	k	clk	Qn+1
0	0	Rising	Qn
0	1	Rising	0
1	0	Rising	1
1	1	Rising	QnB

The polarities of the **j** and **k** input ports are controlled by the enumerated parameters, *j_type* and *k_type* (ActiveHigh, ActiveLow). The **j** and **k** input ports can be single bit or buses which controls the corresponding bits of output ports **q** and **qb**.

The optional output port **qb** has a bitwise inverted value of the output port **q** and can be enabled by setting the enumerated parameter *qb_type* (Enabled, Disabled). Note that using **qb** does not imply any extra inverters; these inverters are usually absorbed by inferred flip-flops, because most have a **qb** port.

The part can be triggered on the rising or the falling edge of the scalar input port **clk**. The polarity of **clk** is controlled by the enumerated parameter *clk_type* (Rising, Falling, RisingLast, FallingLast, RisingEdge, FallingEdge) where RisingLast, FallingLast, RisingEdge, FallingEdge are supported for VHDL only.

The flip-flop has a scalar input port **enable** with its mode controlled by the enumerated parameter *enable_type* (ActiveHigh, ActiveLow, None).

The register can be preset or clear to the value of the parameters *pre_val* and *clr_val* by using the scalar input ports **pre** and **clr**. The modes for these ports are controlled by enumerated parameters *pre_type* and *clr_type* (AsyncActiveHigh, SyncActiveHigh, AsyncActiveLow, SyncActiveLow, None). If the mode for these inputs is set to None or if they are unconnected, then the respective pins are not visible in the symbol and the functionality for that pin is disabled. The modes for these ports can be synchronous or asynchronous.

The inferred register depends upon the type of **clr** and **pre** behavior. The gate count greatly changes if the synthesis tool is able to pick up smaller flip-flops.

If **clr**, **pre** or **enable** are not connected, the code for unused ports is removed.

Parameters *pre_val* and *rst_val* can take LNBF format as described in the Constant Value (constval) model, with no limitation on the size of the **q** and **qb** ports.

The enumerated parameter *initialization* (Enabled, Disabled) determines whether the local registers have an initial value.

Function

If **clr** and **pre** are synchronous, then at every **clk** trigger (rising edge if *clk_type* = Rising, falling edge if *clk_type* = Falling):

q	= *clr_val*	If **clr** is active
	= *pre_val*	If **pre** is active
	= JK Truth Table	If **enable** is active or not connected
qb	= NOT(*clr_val*)	If **clr** is active
	= NOT(*pre_val*)	If **pre** is active
	= NOT(JK Truth Table)	If **enable** is active or otherwise

If **clr** and **pre** are asynchronous:

q	= *clr_val*	if **clr** is active (irrespective of the **clk** trigger)
	= *pre_val*	if **pre** is active (irrespective of the **clk** trigger)
qb	= NOT(*clr_val*)	if **clr** is active (irrespective of the **clk** trigger)
	= NOT(*pre_val*)	if **pre** is active (irrespective of the **clk** trigger)

and at every **clk** trigger (rising edge if *clk_type* = Rising or falling edge if *clk_type* = Falling):

q	= JK Truth Table	if **enable** is active or not connected
qb	= NOT(JK Truth Table)	if **enable** is active or not connected

Truth Tables

The tables are for positive polarities. For negative polarities, invert the values.

Asynchronous **clr** and **pre**:

Table 7-10. JK Flip-Flop Truth Table — Asynchronous clr and pre

enable	clk	clr	pre	q	qb
-	-	1	- / NC	clr_val	NOT(*clr_val*)
-	-	0 / NC	1	pre_val	NOT(*pre_val*)
NC	posedge	0 / NC	0 / NC	JK TT	NOT(JK TT)
1	posedge	0 / NC	0 / NC	JK TT	NOT(JK TT)
0	posedge	0 / NC	0 / NC	q	qb

Synchronous **clr** and **pre**:

Table 7-11. JK Flip-Flop Truth Table — Synchronous clr and pre

enable	clk	clr	pre	q	qb
-	-	1	- / NC	q	qb
-	-	0 / NC	1	q	qb

Table 7-11. JK Flip-Flop Truth Table — Synchronous clr and pre (cont.)

enable	clk	clr	pre	q	qb
-	posedge	1	0 / NC	clr_val	NOT(*clr_val*)
-	posedge	0 / NC	1	pre_val	NOT(*pre_val*)
NC	posedge	0 / NC	0 / NC	JK TT	NOT(JK TT)
0	posedge	0 / NC	0 / NC	q	qb
1	posedge	0 / NC	0 / NC	JK TT	NOT(JK TT)

Parameters

Table 7-12. JK Flip-Flop Parameters

Parameter	Values	Default
q, qb	Port widths (must be > 0)	Automatic
clk_type	Rising,Falling,RisingLast,FallingLast,RisingEdge,FallingEdge	Rising
clr_type	AsyncActiveHigh,SyncActiveHigh,AsyncActiveLow,SyncActiveLow,None	AsyncActiveHigh
enable_type	ActiveHigh,ActiveLow,None	ActiveHigh
j_type	ActiveHigh,ActiveLow	ActiveHigh
k_type	ActiveHigh,ActiveLow	ActiveHigh
pre_type	AsyncActiveHigh,SyncActiveHigh,AsyncActiveLow,SyncActiveLow,None	AsyncActiveHigh
qb_type	Enabled,Disabled	Enabled
clr_val	Register clear value (must be >= 0)	0
initialization	Enabled,Disabled	Disabled
pre_val	Register preset value (must be >= 0)	1

Design Rule Checks

- An error is issued if the width of any port cannot be determined, if ports **clk**, **clr**, **enable** and **pre** do not have a fixed width of 1or if ports **j**, **k q** and **qb** do not have the same width.
- A warning is issued and HDL generation fails for this part if any of ports **j**, **k** and **clk** are not connected or if at least one of ports **q** and **qb** are not connected.

JK Latch (jklatch)

This is a highly parameterized JK latch. Upon a trigger of an enabled scalar input port **gate**, the outputs for the **q** and **qb** ports are generated according to the following truth table:

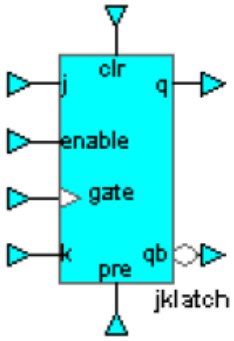


Table 7-13. JK Latch Truth Table

j	k	gate	Qn+1
0	0	High	Qn
0	1	High	0
1	0	High	1
1	1	High	QnB

The polarities of the **j** and **k** input ports are controlled by the enumerated parameters, *j_type* and *k_type* (ActiveHigh, ActiveLow). The **j** and **k** input ports can be single bit or buses which controls the corresponding bits of output ports **q** and **qb**.

The optional output port **qb** has a bitwise inverted value of the output port **q** and can be enabled by setting the enumerated parameter *qb_type* (Enabled, Disabled). Note that using **qb** does not imply extra inverters, these inverters are usually absorbed by inferred flip-flops because most have a **qb** port.

The part can be triggered on the high or the low level of the scalar input port **gate**. The polarity of the **gate** port is controlled by the enumerated parameter *gate_type* (ActiveHigh, ActiveLow).

The latch has a scalar input port **enable** with its mode controlled by the parameter *enable_type* (AsyncActiveHigh, SyncActiveHigh, AsyncActiveLow, SyncActiveLow, None). In asynchronous mode, the **enable** input has a higher priority over the **gate** input and in synchronous mode, the **gate** input has a higher priority over the **enable** input.

The register can be preset or cleared to the value of the parameters *pre_val* and *clr_val* by using the scalar input ports **pre** and **clr**. The modes for these ports are controlled by enumerated parameters *pre_type* and *clr_type* (AsyncActiveHigh, SyncActiveHigh, AsyncActiveLow, SyncActiveLow, None). If the mode for these inputs is set to None or if they are unconnected, then the respective pins are not visible in the symbol and the functionality for that pin is disabled. The modes for these ports can be synchronous or asynchronous.

The inferred register depends upon the type of **clr** and **pre** behavior. The gate count greatly changes if the synthesis tool is able to pick up smaller flip-flops.

If **clr**, **pre** or **enable** are not connected, the code for unused ports is removed.

Parameters *pre_val* and *rst_val* can take LNBF format as described in the Constant Value (constval) model, with no limitation on the size of the **q** and **qb** ports.

The enumerated parameter *initialization* (Enabled, Disabled) determines whether the local registers have an initial value.

Function

If **clr** and **pre** are synchronous and **enable** is synchronous, then at every **gate** trigger (high level if *gate_type* = ActiveHigh or low level if *gate_type* = ActiveLow):

q	= *clr_val*	If **clr** is active
	= *pre_val*	If **pre** is active
	= JK Truth Table	If **enable** is active or otherwise
qb	= NOT(*clr_val*)	If **clr** is active
	= NOT(*pre_val*)	If **pre** is active
	= NOT(JK Truth Table)	If **enable** is active or otherwise

If **enable** is set to asynchronous, then at every **enable** trigger:

q	= *clr_val*	If **clr** is active and **gate** trigger is present
	= *pre_val*	If **pre** is active and **gate** trigger is present
	= JK Truth Table	If **gate** trigger is present
qb	= NOT(*clr_val*)	If **clr** is active and **gate** trigger is present
	= NOT(*pre_val*)	If **pre** is active and **gate** trigger is present
	= NOT(JK Truth Table)	If **gate** trigger is present

If **clr** and **pre** are asynchronous and **enable** is set to synchronous:

q	= *clr_val*	if **clr** is active (irrespective of the **gate** trigger)
	= *pre_val*	if **pre** is active (irrespective of the **gate** trigger)
qb	= NOT(*clr_val*)	if **clr** is active (irrespective of the **gate** trigger)
	= NOT(*pre_val*)	if **pre** is active (irrespective of the **gate** trigger)

and at every **gate** trigger (high level if *gate_type* = ActiveHigh or low level if *gate_type* = ActiveLow):

q	= JK Truth Table	if **enable** is active or not connected
qb	= NOT(JK Truth Table)	if **enable** is active or not connected

If **enable** is set to asynchronous:

q	= *clr_val*	if **clr** is active (irrespective of the **gate** & **enable** trigger)
	= *pre_val*	if **pre** is active (irrespective of the **gate** & **enable** trigger)
qb	= NOT(*clr_val*)	if **clr** is active (irrespective of the **gate** & **enable** trigger)
	= NOT(*pre_val*)	if **pre** is active (irrespective of the **gate** & **enable** trigger)

and at every **enable** trigger:

q	= JK Truth Table	if **gate** trigger is present
qb	= NOT(JK Truth Table)	if **gate** trigger is present

Truth Tables

The tables are for positive polarities. For negative polarities, invert the values.

Asynchronous **clr** and **pre** and synchronous **enable**:

Table 7-14. JK Latch Truth Table — Asynchronous clr and pre, Synchronous enable

enable	gate	clr	pre	q	qb
-	-	1	- / NC	clr_val	NOT(*clr_val*)
-	-	0 / NC	1	pre_val	NOT(*pre_val*)
NC	1	0 / NC	0 / NC	JK TT	NOT(JK TT)
1	1	0 / NC	0 / NC	JK TT	NOT(JK TT)
0	1	0 / NC	0 / NC	q	qb

Asynchronous **clr** and **pre** and asynchronous **enable**:

Table 7-15. JK Latch Truth Table — Asynchronous clr, pre and enable

enable	gate	clr	pre	q	qb
-	-	1	- / NC	clr_val	NOT(*clr_val*)
-	-	0 / NC	1	pre_val	NOT(*pre_val*)
NC	1	0 / NC	0 / NC	JK TT	NOT(JK TT)
0	-	0 / NC	0 / NC	q	qb
1	1	0 / NC	0 / NC	JK TT	NOT(JK TT)

Synchronous **clr** and **pre** and synchronous **enable**:

Table 7-16. JK Latch Truth Table — Synchronous clr, pre and enable

enable	gate	clr	pre	q	qb
-	-	1	- / NC	q	qb
-	-	0 / NC	1	q	qb
-	1	1	0 / NC	clr_val	NOT(*clr_val*)
-	1	0 / NC	1	pre_val	NOT(*pre_val*)
NC	1	0 / NC	0 / NC	JK TT	NOT(JK TT)
0	1	0 / NC	0 / NC	q	qb
1	1	0 / NC	0 / NC	JK TT	NOT(JK TT)

Synchronous **clr** and **pre** and asynchronous **enable**:

Table 7-17. JK Latch Truth Table — Synchronous clr and pre, Asynchronous enable

enable	gate	clr	pre	q	qb
-	-	1	- / NC	q	qb
-	-	0 / NC	1	q	qb
1 / NC	1	1	0 / NC	clr_val	NOT(*clr_val*)
1 / NC	1	0 / NC	1	pre_val	NOT(*pre_val*)
NC	1	0 / NC	0 / NC	JK TT	NOT(JK TT)
0	1	0 / NC	0 / NC	q	qb
1	1	0 / NC	0 / NC	JK TT	NOT(JK TT)

Parameters

Table 7-18. JK Latch Parameters

Parameter	Values	Default
q, qb	Port widths (must be > 0)	Automatic
clr_type	AsyncActiveHigh,SyncActiveHigh,AsyncActiveLow,SyncActiveLow,None	AsyncActiveHigh
enable_type	AsyncActiveHigh,SyncActiveHigh,AsyncActiveLow,SyncActiveLow,None	SyncActiveHigh
gate_type	ActiveHigh,ActiveLow	ActiveHigh
j_type	ActiveHigh,ActiveLow	ActiveHigh
k_type	ActiveHigh,ActiveLow	ActiveHigh
pre_type	AsyncActiveHigh,SyncActiveHigh,AsyncActiveLow,SyncActiveLow,None	AsyncActiveHigh
qb_type	Enabled,Disabled	Enabled
clr_val	Register clear value (must be >= 0)	0
initialization	Enabled,Disabled	Disabled
pre_val	Register preset value (must be >= 0)	1

Design Rule Checks

- An error is issued if the width of any port cannot be determined, if ports **gate**, **clr**, **enable** and **pre** do not have a fixed width of 1 or if ports **j**, **k q** and **qb** do not have the same width.
- A warning is issued and HDL generation fails for this part if any of ports **j**, **k** and **gate** are not connected or if at least one of ports **q** and **qb** are not connected.

RS Flip-Flop (rsff)

This is a highly parameterized RS flip-flop. Upon a trigger of an enabled scalar input port **clk**, the outputs **q** and **qb** are generated according to the following truth table:

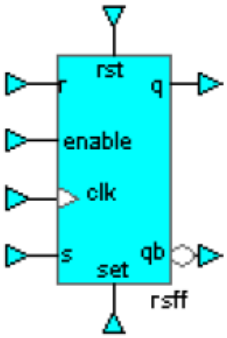


Table 7-19. RS Flip-Flop Truth Table

r	s	clk	Qn+1
0	0	Rising	Qn
0	1	Rising	1
1	0	Rising	0

r and **s** are the scalar input ports, the polarity of these ports is controlled by the enumerated parameters *r_type* and *s_type* (ActiveHigh, ActiveLow).

The optional output port **qb** has a bitwise inverted value of the output port **q** and can be enabled by setting the enumerated parameter *qb_type* (Enabled, Disabled). Note that using **qb** does not imply extra inverters, these inverters are usually absorbed by inferred flip-flops because most have a **qb** port.

The part can be triggered on the rising or the falling edge of the scalar input port **clk**. The polarity of **clk** is controlled by the enumerated parameter *clk_type* (Rising, Falling, RisingLast, FallingLast, RisingEdge, FallingEdge) where RisingLast, FallingLast, RisingEdge, FallingEdge are supported for VHDL only.

The flip-flop has a scalar input port **enable** with its mode controlled by the enumerated parameter *enable_type* (ActiveHigh, ActiveLow, None).

The register can be set or reset to the value of the parameters *set_val* or *rst_val* by using the scalar input ports **set** and **rst**. The modes for these ports are controlled by the enumerated parameters *set_type* and *rst_type* (AsyncActiveHigh, SyncActiveHigh, AsyncActiveLow, SyncActiveLow, None). If the mode for these inputs is set to None or if they are unconnected, then the respective pins are not visible in the symbol and the functionality for that pin is disabled. The modes for these ports can be synchronous or asynchronous.

The inferred register depends upon the type of **rst** and **set** behavior. The gate count greatly changes if the synthesis tool is able to pick up smaller flip-flops.

If **rst**, **set** or **enable** are not connected, the code for unused ports is removed.

The enumerated parameter *initialization* (Enabled, Disabled) determines whether the local registers have an initial value.

The parameter *rs_priority* controls the priority of **r** over **s** in the generated code, when *rs_priority* equals R-S, **r** takes a higher priority over **s**, while if *rs_priority* equals S-R, **s** takes a higher priority over **r**.

Parameters *set_val* and *rst_val* can take LNBF format as described in the Constant Value (constval) model, with no limitation on the size of the **q** and **qb** ports.

Function

If **rst** and **set** are synchronous, then at every **clk** trigger (rising edge if *clk_type* = Rising, falling edge if *clk_type* = Falling):

q	= *rst_val*	If **rst** is active
	= *set_val*	If **set** is active
	= RS Truth Table	If **enable** is active or not connected
qb	= NOT(*rst_val*)	If **rst** is active
	= NOT(*set_val*)	If **set** is active
	= NOT(RS Truth Table)	If **enable** is active or not connected

If **rst** and **clr** are asynchronous:

q	= *rst_val*	if **rst** is active (irrespective of the **clk** trigger)
	= *set_val*	if **set** is active (irrespective of the **clk** trigger)
qb	= NOT(*rst_val*)	if **rst** is active (irrespective of the **clk** trigger)
	= NOT(*set_val*)	if **set** is active (irrespective of the **clk** trigger)

and at every **clk** trigger (rising edge if *clk_type* = Rising or falling edge if *clk_type* = Falling):

q	= RS Truth Table	if **enable** is active or not connected
qb	= NOT(RS Truth Table)	if **enable** is active or not connected

Truth Tables

The tables are for positive polarities. For negative polarities, invert the values.

Asynchronous **rst** and **set**:

Table 7-20. RS Flip-Flop Truth Table — Asynchronous rst and set

enable	clk	rst	set	q	qb
-	-	1	- / NC	rst_val	NOT(*rst_val*)
-	-	0 / NC	1	set_val	NOT(*set_val*)
NC	posedge	0 / NC	0 / NC	RS TT	NOT(RS TT)
1	posedge	0 / NC	0 / NC	RS TT	NOT(RS TT)
0	posedge	0 / NC	0 / NC	q	qb

Synchronous **rst** and **set**:

Table 7-21. RS Flip-Flop Truth Table — Synchronous rst and set

enable	clk	rst	set	q	qb
-	-	1	- / NC	q	qb
-	-	0 / NC	1	q	qb

Table 7-21. RS Flip-Flop Truth Table — Synchronous rst and set (cont.)

enable	clk	rst	set	q	qb
-	posedge	1	0 / NC	rst_val	NOT(*rst_val*)
-	posedge	0 / NC	1	set_val	NOT(*set_val*)
NC	posedge	0 / NC	0 / NC	RS TT	NOT(RS TT)
0	posedge	0 / NC	0 / NC	q	qb
1	posedge	0 / NC	0 / NC	RS TT	NOT(RS TT)

Parameters

Table 7-22. RS Flip-Flop Parameters

Parameter	Values	Default
q, qb	Port widths (must be > 0)	Automatic
clk_type	Rising,Falling,RisingLast,FallingLast,RisingEdge,FallingEdge	Rising
enable_type	ActiveHigh,ActiveLow,None	ActiveHigh
qb_type	Enabled,Disabled	Enabled
r_type	ActiveHigh,ActiveLow	ActiveHigh
rst_type	AsyncActiveHigh,SyncActiveHigh,AsyncActiveLow,SyncActiveLow,None	AsyncActiveHigh
s_type	ActiveHigh,ActiveLow	ActiveHigh
set_type	AsyncActiveHigh,SyncActiveHigh,AsyncActiveLow,SyncActiveLow,None	AsyncActiveHigh
initialization	Enabled,Disabled	Disabled
rs_priority	R-S,S-R	R-S
rst_val	Register reset value (must be >= 0)	0
set_val	Register set value (must be >= 0)	1

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if ports **clk**, **enable**, **rst**, **set**, **r** and **s** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any of ports **r**, **s** and **clk** are not connected or if at least one of ports **q** and **qb** are not connected.

RS Latch (rslatch)

This is a highly parameterized RS latch. Upon a trigger of an enabled scalar input port **gate**, the outputs **q** and **qb** are generated according to the following truth table:

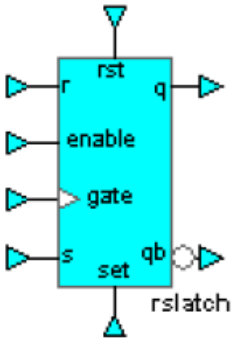


Table 7-23. RS Latch Truth Table

r	s	gate	Qn+1
0	0	High	Qn
0	1	High	1
1	0	High	0

r and **s** are the scalar input ports, the polarity of these ports is controlled by the enumerated parameters *r_type* and *s_type* (ActiveHigh, ActiveLow).

The optional output port **qb** has a bitwise inverted value of the output port **q** and can be enabled by setting the enumerated parameter *qb_type* (Enabled, Disabled). Note that using **qb** does not imply extra inverters, these inverters are usually absorbed by inferred flip-flops because most have a **qb** port.

The part can be triggered on the high or the low level of the scalar input port **gate**. The polarity of the **gate** port is controlled by the enumerated parameter *gate_type* (ActiveHigh, ActiveLow).

The latch has a scalar input port **enable** with its mode controlled by the parameter *enable_type* (AsyncActiveHigh, SyncActiveHigh, AsyncActiveLow, SyncActiveLow, None). In asynchronous mode, the **enable** input has a higher priority over the **gate** input and in synchronous mode, the **gate** input has a higher priority over the **enable** input.

The register can be set or reset to the value of the parameters *set_val* or *rst_val* by using the scalar input ports **set** and **rst**. The modes for these ports are controlled by the enumerated parameters *set_type* and *rst_type* (AsyncActiveHigh, SyncActiveHigh, AsyncActiveLow, SyncActiveLow and None). If the mode for these inputs is set to None or if they are unconnected, then the respective pins are not visible in the symbol and the functionality for that pin is disabled. The modes for these ports can be synchronous or asynchronous.

The inferred register depends upon the type of **rst** and **set** behavior. The gate count greatly changes if the synthesis tool is able to pick up smaller flip-flops.

If **set**, **rst** or **enable** are not connected, the code for unused ports is removed.

The enumerated parameter *initialization* (Enabled, Disabled) determines whether the local registers have an initial value.

The parameter *rs_priority* controls the priority of **r** over **s** in the generated code, when *rs_priority* equals R-S, **r** takes a higher priority over **s**, while if *rs_priority* equals S-R, **s** takes a higher priority over **r**.

Parameters *set_val* and *rst_val* can take LNBF format as described in the Constant Value (constval) model, with no limitation on the size of the **q** and **qb** ports.

Function

If **rst** and **set** are synchronous and if **enable** is synchronous, then at every **gate** trigger (high level if *clk_type* = ActiveHigh or low level if *gate_type* = ActiveLow):

q	= *rst_val*	If **rst** is active
	= *set_val*	If **set** is active
	= RS Truth Table	If **enable** is active or not connected
qb	= NOT(*rst_val*)	If **rst** is active
	= NOT(*set_val*)	If **set** is active
	= NOT(RS Truth Table)	If **enable** is active or not connected

If **enable** is set to asynchronous, then at every **enable** trigger:

q	= *rst_val*	If **rst** is active and **gate** trigger is present
	= *set_val*	If **set** is active and **gate** trigger is present
	= RS Truth Table	If **gate** trigger is present
qb	= NOT(*rst_val*)	If **rst** is active and **gate** trigger is present
	= NOT(*set_val*)	If **set** is active and **gate** trigger is present
	= NOT(RS Truth Table)	If **gate** trigger is present

If **rst** and **clr** are asynchronous and **enable** is set to synchronous:

q	= *rst_val*	if **rst** is active (irrespective of the **gate** trigger)
	= *set_val*	if **set** is active (irrespective of the **gate** trigger)
qb	= NOT(*rst_val*)	if **rst** is active (irrespective of the **gate** trigger)
	= NOT(*set_val*)	if **set** is active (irrespective of the **gate** trigger)

and at every *gate* trigger (high level if *gate_type* = ActiveHigh, low level if *gate_type* = ActiveLow):

q	= RS Truth Table	if **enable** is active or not connected
qb	= NOT(RS Truth Table)	if **enable** is active or not connected

If **enable** is set to asynchronous:

q	= *rst_val*	if **rst** is active (irrespective of the **gate** & **enable** trigger)
	= *set_val*	if **set** is active (irrespective of the **gate** & **enable** trigger)
qb	= NOT(*rst_val*)	if **rst** is active (irrespective of the **gate** & **enable** trigger)
	= NOT(*set_val*)	if **set** is active (irrespective of the **gate** & **enable** trigger)

and at every **enable** trigger:

q	= RS Truth Table	if **gate** trigger is present
qb	= NOT(RS Truth Table)	if **gate** trigger is present

Truth Tables

The tables are for positive polarities. For negative polarities, invert the values.

Asynchronous **rst** and **set** and synchronous **enable**:

Table 7-24. RS Latch Truth Table — Asynchronous rst and set, Synchronous enable

enable	gate	rst	set	q	qb
-	-	1	- / NC	rst_val	NOT(*rst_val*)
-	-	0 / NC	1	set_val	NOT(*set_val*)
NC	1	0 / NC	0 / NC	RS TT	NOT(RS TT)
1	1	0 / NC	0 / NC	RS TT	NOT(RS TT)
0	1	0 / NC	0 / NC	q	qb

Asynchronous **rst** and **set** and asynchronous **enable**:

Table 7-25. RS Latch Truth Table — Asynchronous rst, set and enable

enable	gate	rst	set	q	qb
-	-	1	- / NC	rst_val	NOT(*rst_val*)
-	-	0 / NC	1	set_val	NOT(*set_val*)
NC	1	0 / NC	0 / NC	RS TT	NOT(RS TT)
0	-	0 / NC	0 / NC	q	qb
1	1	0 / NC	0 / NC	RS TT	NOT(RS TT)

Synchronous **rst** and **set** and synchronous **enable**:

Table 7-26. RS Latch Truth Table — Synchronous rst, set and enable

enable	gate	rst	set	q	qb
-	-	1	- / NC	q	qb
-	-	0 / NC	1	q	qb
-	1	1	0 / NC	rst_val	NOT(*rst_val*)
-	1	0 / NC	1	set_val	NOT(*set_val*)
NC	1	0 / NC	0 / NC	RS TT	NOT(RS TT)
0	1	0 / NC	0 / NC	q	qb
1	1	0 / NC	0 / NC	RS TT	NOT(RS TT)

Synchronous **rst** and **set** and asynchronous **enable**:

Table 7-27. RS Latch Truth Table — Synchronous rst and set, Asynchronous enable

enable	gate	rst	set	q	qb
-	-	1	- / NC	q	qb
-	-	0 / NC	1	q	qb
1 / NC	1	1	0 / NC	rst_val	NOT(*rst_val*)
1 / NC	1	0 / NC	1	set_val	NOT(*set_val*)
NC	1	0 / NC	0 / NC	RS TT	NOT(RS TT)
0	1	0 / NC	0 / NC	q	qb
1	1	0 / NC	0 / NC	RS TT	NOT(RS TT)

Parameters

Table 7-28. RS Latch Parameters

Parameter	Values	Default
q, qb	Port widths (must be > 0)	Automatic
enable_type	AsyncActiveHigh,SyncActiveHigh,AsyncActiveLow,SyncActiveLow,None	SyncActiveHigh
gate_type	ActiveHigh,ActiveLow	ActiveHigh
qb_type	Enabled,Disabled	Enabled
r_type	ActiveHigh,ActiveLow	ActiveHigh
rst_type	AsyncActiveHigh,SyncActiveHigh,AsyncActiveLow,SyncActiveLow,None	AsyncActiveHigh
s_type	ActiveHigh,ActiveLow	ActiveHigh
set_type	AsyncActiveHigh,SyncActiveHigh,AsyncActiveLow,SyncActiveLow,None	AsyncActiveHigh
initialization	Enabled,Disabled	Disabled
rs_priority	R-S,S-R	R-S
rst_val	Register reset value (must be >= 0)	0
set_val	Register set value (must be >= 0)	1

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if ports **gate**, **enable**, **rst**, **set**, **r** and **s** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any of ports **r**, **s** or **gate** are not connected or if at least one of ports **q** and **qb** are not connected.

T Flip-Flop (tff)

This is a highly parameterized T-type flip-flop. Upon a trigger of an enabled scalar input port **clk**, the outputs **q** and **qb** are generated according to the following truth table:

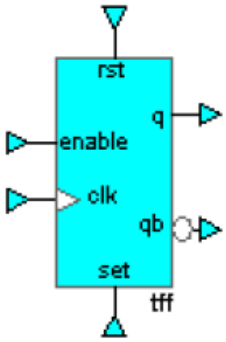


Table 7-29. T Flip-Flop Truth Table

enable	clk	Qn+1
0	Rising	Qn
1	Rising	QnB

The optional output port **qb** has a bitwise inverted value of the output port **q** and can be enabled by setting the enumerated parameter *qb_type* (Enabled, Disabled). Note that using **qb** does not imply extra inverters, these inverters are usually absorbed by inferred flip-flops because most have a **qb** port.

The part can be triggered on the rising or the falling edge of the scalar input port **clk**. The polarity of **clk** is controlled by the enumerated parameter *clk_type* (Rising, Falling, RisingLast, FallingLast, RisingEdge, FallingEdge) where RisingLast, FallingLast, RisingEdge, FallingEdge are supported for VHDL only.

The flip-flop has a scalar input port **enable** with its mode controlled by the enumerated parameter *enable_type* (ActiveHigh, ActiveLow, None).

The register can be set or reset to the value of the parameter *set_val* and *rst_val* by using the scalar input ports **set** and **rst**. The modes for these ports are controlled by the enumerated parameters *set_type* and *rst_type* (AsyncActiveHigh, SyncActiveHigh, AsyncActiveLow, SyncActiveLow, None). If the mode for these inputs is set to None or if they are unconnected, then the respective pins are not visible in the symbol and the functionality for that pin is disabled. The modes for these ports can be synchronous or asynchronous.

This part is equivalent to *n* flip-flops where *n* is the width of the ports **q** and **qb**. The inferred register depends upon the type of **rst** and **set** behavior. The gate count greatly changes if the synthesis tool is able to pick up smaller flip-flops.

If **rst**, **set** or **enable** are not connected, the code for unused ports is removed.

The enumerated parameter *initialization* (Enabled, Disabled) determines whether the local registers have an initial value.

Parameters *set_val* and *rst_val* can take LNBF format as described in the Constant Value (constval) model, with no limitation on the size of the **q** and **qb** ports.

Function

If **rst** and **set** are synchronous, then at every **clk** trigger (rising edge if *clk_type* = Rising or falling edge if *clk_type* = Falling):

q	= *rst_val*	if **rst** is active
	= *set_val*	If **set** is active
	= T Truth Table	If **enable** is active or not connected
qb	= NOT(*rst_val*)	If **rst** is active
	= NOT(*set_val*)	If **set** is active
	= NOT(T Truth Table)	If **enable** is active or not connected

If **rst** and **set** are asynchronous:

q	= *rst_val*	if **rst** is active (irrespective of the **clk** trigger)
	= *set_val*	if **set** is active (irrespective of the **clk** trigger)
qb	= NOT(*rst_val*)	if **rst** is active (irrespective of the **clk** trigger)
	= NOT(*set_val*)	if **set** is active (irrespective of the **clk** trigger)

and at every **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

q	= T Truth Table	if **enable** is active or not connected
qb	= NOT(T Truth Table)	if **enable** is active or not connected

Truth Tables

The tables are for positive polarities. For negative polarities, invert the values.

Asynchronous **rst** and **set** and synchronous **enable**:

Table 7-30. T Flip-Flop Truth Table — Asynchronous rst and set, Synchronous enable

enable	clk	rst	set	q	qb
-	-	1	- / NC	rst_val	NOT(*rst_val*)
-	-	0 / NC	1	set_val	NOT(*set_val*)
NC	posedge	0 / NC	0 / NC	T TT	NOT(T TT)
1	posedge	0 / NC	0 / NC	T TT	NOT(T TT)
0	posedge	0 / NC	0 / NC	q	qb

Synchronous **rst** and **set**:

Table 7-31. T Flip-Flop Truth Table — Synchronous rst and set

enable	clk	rst	set	q	qb
-	-	1	- / NC	q	qb
-	-	0 / NC	1	q	qb

Table 7-31. T Flip-Flop Truth Table — Synchronous rst and set (cont.)

enable	clk	rst	set	q	qb
-	posedge	1	0 / NC	rst_val	NOT(*rst_val*)
-	posedge	0 / NC	1	set_val	NOT(*set_val*)
NC	posedge	0 / NC	0 / NC	T TT	NOT(T TT)
0	posedge	0 / NC	0 / NC	q	qb
1	posedge	0 / NC	0 / NC	T TT	NOT(T TT)

Parameters

Table 7-32. T Flip-Flop Parameters

Parameter	Values	Default
q, qb	Port widths (must be > 0)	Automatic
clk_type	Rising,Falling,RisingLast,FallingLast,RisingEdge,FallingEdge	Rising
enable_type	ActiveHigh,ActiveLow,None	ActiveHigh
qb_type	Enabled,Disabled	Enabled
rst_type	AsyncActiveHigh,SyncActiveHigh,AsyncActiveLow,SyncActiveLow,None	AsyncActiveHigh
set_type	AsyncActiveHigh,SyncActiveHigh,AsyncActiveLow,SyncActiveLow,None	AsyncActiveHigh
initialization	Enabled,Disabled	Disabled
rst_val	Register reset value (must be >= 0)	0
set_val	Register set value (must be >= 0)	1

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if ports **clk**, **enable**, **rst** and **set** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if port **clk** is not connected or if at least one of ports **q** and **qb** are not connected.

T Latch (tlatch)

This is a highly parameterized T-type latch. Upon a trigger of an enabled scalar input port **gate**, the outputs **q** and **qb** are generated according to the following truth table:

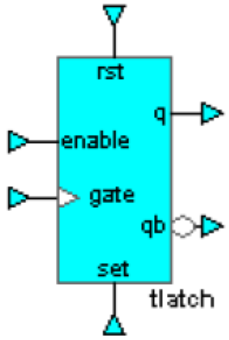


Table 7-33. T Latch Truth Table

enable	gate	Qn+1
0	High	Qn
1	High	QnB

The optional output port **qb** has a bitwise inverted value of the output port **q** and can be enabled by setting the enumerated parameter *qb_type* (Enabled, Disabled). Note that using **qb** does not imply extra inverters, these inverters are usually absorbed by inferred flip-flops because most have a **qb** port.

The part can be triggered on the high or the low level of the scalar input port **gate**. The polarity of the **gate** port is controlled by the enumerated parameter *gate_type* (ActiveHigh, ActiveLow).

The latch has a scalar control input port **enable** with its mode controlled by the parameter *enable_type* (AsyncActiveHigh, SyncActiveHigh, AsyncActiveLow, SyncActiveLow, None). In asynchronous mode, the **enable** input has a higher priority over the **gate** input and in synchronous mode, the **gate** input has a higher priority over the **enable** input.

The register can be set or reset to the value of the parameters *set_val* or *rst_val* by using the scalar input ports **set** and **rst**. The modes for these inputs are controlled by the enumerated parameters *set_type* and *rst_type* (AsyncActiveHigh, SyncActiveHigh, AsyncActiveLow, SyncActiveLow, None). If the mode for these inputs is set to None or if they are unconnected, then the respective pins are not visible in the symbol and the functionality for that pin is disabled. The modes for these ports can be synchronous or asynchronous.

This part is equivalent to *n* latches where *n* is the width of the ports **q** and **qb**. The inferred register depends upon the type of **rst** and **set** behavior. The gate count greatly changes if the synthesis tool is able to pick up smaller latches.

If **set**, **rst** or **enable** are not connected, the code for unused ports is removed.

The enumerated parameter *initialization* (Enabled, Disabled) determines whether the local registers have an initial value.

Parameters *set_val* and *rst_val* can take LNBF format as described in the Constant Value (constval) model, with no limitation on the size of the **q** and **qb** ports.

Function

If **rst** and **set** are synchronous and if **enable** is synchronous, then at every **gate** trigger (high level if *gate_type* = ActiveHigh or low level if *gate_type* = ActiveLow):

q	= *rst_val*	if **rst** is active
	= *set_val*	If **set** is active
	= T Truth Table	If **enable** is active or not connected
qb	= NOT(*rst_val*)	If **rst** is active
	= NOT(*set_val*)	If **set** is active
	= NOT(T Truth Table)	If **enable** is active or not connected

If **enable** is set to asynchronous, then at every **enable** trigger:

q	= *rst_val*	if **rst** is active and **gate** trigger is present
	= *set_val*	If **set** is active and **gate** trigger is present
	= T Truth Table	If **gate** trigger is present
qb	= NOT(*rst_val*)	If **rst** is active and **gate** trigger is present
	= NOT(*set_val*)= NOT(T Truth Table)	If **set** is active and **gate** trigger is present If **gate** trigger is present

If **rst** and **set** are asynchronous and **enable** is set to synchronous:

q	= *rst_val*	if **rst** is active (irrespective of the **gate** trigger)
	= *set_val*	if **set** is active (irrespective of the **gate** trigger)
qb	= NOT(*rst_val*)	if **rst** is active (irrespective of the **gate** trigger)
	= NOT(*set_val*)	if **set** is active (irrespective of the **gate** trigger)

and at every **gate** trigger (low level if *gate_type* = ActiveHigh, low level if *gate_type* = ActiveLow):

q	= T Truth Table	if **enable** is active or not connected
qb	= NOT(T Truth Table)	if **enable** is active or not connected

If **enable** is set to asynchronous:

q	= *rst_val*	if **rst** is active (irrespective of the **gate** & **enable** trigger)
	= *set_val*	if **set** is active (irrespective of the **gate** & **enable** trigger)
qb	= NOT(*rst_val*)	if **rst** is active (irrespective of the **gate** & **enable** trigger)
	= NOT(*set_val*)	if **set** is active (irrespective of the **gate** & **enable** trigger)

and at every **load** trigger:

q	= T Truth Table	if **gate** trigger is present
qb	= NOT(T Truth Table)	if **gate** trigger is present

Truth Tables

The tables are for positive polarities. For negative polarities, invert the values.

Asynchronous **rst** and **set** and synchronous **enable**:

Table 7-34. T Latch Truth Table — Asynchronous rst and set, Synchronous enable

enable	gate	rst	set	q	qb
-	-	1	- / NC	rst_val	NOT(*rst_val*)
-	-	0 / NC	1	set_val	NOT(*set_val*)
NC	1	0 / NC	0 / NC	T TT	NOT(T TT)
1	1	0 / NC	0 / NC	T TT	NOT(T TT)
0	1	0 / NC	0 / NC	q	qb

Asynchronous **rst** and **set** and asynchronous **enable**:

Table 7-35. T Latch Truth Table — Asynchronous rst, set and enable

enable	gate	rst	set	q	qb
-	-	1	- / NC	rst_val	NOT(*rst_val*)
-	-	0 / NC	1	set_val	NOT(*set_val*)
NC	1	0 / NC	0 / NC	T TT	NOT(T TT)
0	-	0 / NC	0 / NC	q	qb
1	1	0 / NC	0 / NC	T TT	NOT(T TT)

Synchronous **rst** and **set** and synchronous **enable**:

Table 7-36. T Latch Truth Table — Synchronous rst, set and enable

enable	gate	rst	set	q	qb
-	-	1	- / NC	q	qb
-	-	0 / NC	1	q	qb
-	1	1	0 / NC	rst_val	NOT(*rst_val*)
-	1	0 / NC	1	set_val	NOT(*set_val*)
NC	1	0 / NC	0 / NC	T TT	NOT(T TT)
0	1	0 / NC	0 / NC	q	qb
1	1	0 / NC	0 / NC	T TT	NOT(T TT)

Synchronous **rst** and **set** and asynchronous **enable**:

Table 7-37. T Latch Truth Table — Synchronous rst and set, Asynchronous enable

enable	gate	rst	set	q	qb
-	-	1	- / NC	q	qb
-	-	0 / NC	1	q	qb
1 / NC	1	1	0 / NC	rst_val	NOT(*rst_val*)
1 / NC	1	0 / NC	1	set_val	NOT(*set_val*)
NC	1	0 / NC	0 / NC	T TT	NOT(T TT)
0	1	0 / NC	0 / NC	q	qb
1	1	0 / NC	0 / NC	T TT	NOT(T TT)

Parameters

Table 7-38. T Latch Parameters

Parameter	Values	Default
q, qb	Port widths (must be > 0)	Automatic
enable_type	AsyncActiveHigh,SyncActiveHigh,AsyncActiveLow,SyncActiveLow,None	SyncActiveHigh
gate_type	ActiveHigh,ActiveLow	ActiveHigh
qb_type	Enabled,Disabled	Enabled
rst_type	AsyncActiveHigh,SyncActiveHigh,AsyncActiveLow,SyncActiveLow,None	AsyncActiveHigh
set_type	AsyncActiveHigh,SyncActiveHigh,AsyncActiveLow,SyncActiveLow,None	AsyncActiveHigh
initialization	Enabled,Disabled	Disabled
rst_val	Register reset value (must be >= 0)	0
set_val	Register set value (must be >= 0)	1

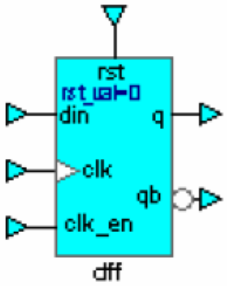
Design Rule Checks

- An error is issued if the width of any port cannot be determined or if ports **gate**, **enable**, **rst** and **set** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if port **gate** is not connected or if at least one of ports **q** and **qb** are not connected.

Chapter 8
Sequential Parts

This chapter describes the following sequential blocks:

Bank of Flip-Flops (dff)



This part provides a bank of D-type flip-flops (registers). When the scalar input port **clk** is triggered, the data from input port **din** is shifted into the output port **q**. The output port **qb** has the bitwise inverted value of output port **q**.

The part can be triggered on the rising edge or the falling edge of the scalar input port **clk**. The polarity of the scalar input port **clk** is controlled by the enumerated parameter *clk_type* (Rising, Falling). Note that RisingLast, FallingLast, RisingEdge and FallingEdge are also supported for VHDL.

The **clk** can optionally be enabled by the scalar input port **clk_en**. The polarity of port **clk_en** is controlled by the enumerated parameter *clk_en_type* (ActiveHigh, ActiveLow). Do not change the value of *clk_en_type* if the **clk_en** port is not used.

The register can be reset to the value of the parameter *rst_val* by activating the scalar input port **rst**. The mode of the **rst** is controlled by the enumerated parameter *rst_type* (SyncActiveHigh, AsyncActiveHigh, SyncActiveLow, AsyncActiveLow). If input ports **clk** and **rst** (for asynchronous behavior) are not enabled, the output ports of the flip-flops retain their values.

This part is equivalent to *n* flip-flops where *n* is the width of the ports **din**, **q** and **qb**.

The inferred register depends upon the type of reset behavior. The gate count changes significantly if the synthesis tool is able to pick up smaller flip-flops. Note that using **qb** does not imply extra inverters; these inverters are usually absorbed by inferred flip-flops because most have a **qb** port.

If **clk_en** is not connected, it is driven by 1. If the clock enable feature is not used, the scalar input port **clk_en** must be unconnected and the parameter *clk_en_type* must be set to ActiveHigh. Port **clk_en** can also be driven by an active driver, in which case the *clk_en_type* must be correctly adjusted.

The HDL code for unconnected optional ports is optimized away.

Parameter *rst_val* can take LNBF format as described in the Constant Value (constval) model, with no limitation on the size of the **q** and **qb** ports.

Function

An enabled **clk** trigger is a trigger that occurs when input port **clk_en** is active.

If **rst** is synchronous then at every enabled **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

q	= *rst_val*	if **rst** is active
	= **din**	otherwise
qb	= NOT(*rst_val*)	if **rst** is active
	= NOT(**din**)	otherwise

If **rst** is asynchronous (*rst_type* = AsyncActiveHigh or *rst_type* = AsyncActiveLow):

q = *rst_val* if **rst** is active (irrespective of the **clk** trigger)

else at every enabled **clk** trigger (rising edge if *clk_type* = Rising or falling edge if *clk_type* = Falling):

q = **din** otherwise

qb = NOT(*rst_val*) if **rst** is active (irrespective of the **clk** trigger)

else at every enabled **clk** trigger (rising edge if *clk_type* = Rising or falling edge if *clk_type* = Falling):

qb = NOT(**din**) otherwise

Truth Table

The following tables are for positive polarities. For negative polarities, invert the values.

Asynchronous high reset and positive polarities:

Table 8-1. Bank of Flip-Flops Truth Table — Asynchronous High Reset, Positive Polarities

clk_en	clk	rst	q	qb
-	-	1	rst_val	NOT(rst_val)
0	posedge	0	q	qb
1	posedge	0	din	NOT(din)

Synchronous high reset and positive polarities:

Table 8-2. Bank of Flip-Flops Truth Table — Synchronous High Reset, Positive Polarities

clk_en	clk	rst	q	qb
0	-	-	q	qb
1	posedge	1	rst_val	NOT(rst_val)
1	posedge	0	din	NOT(din)

Parameters

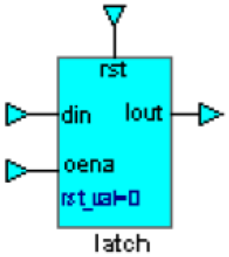
Table 8-3. Bank of Flip-Flops Parameters

Parameter	Values	Default
din, q, qb	Port widths (must be > 0)	Automatic
clk_en_type	ActiveHigh,ActiveLow	ActiveHigh
clk_type	Rising,Falling,RisingLast,FallingLast,RisingEdge,FallingEdge	Rising
qb_type	Enabled,Disabled	Enabled
rst_type	SyncActiveHigh,AsyncActiveHigh,SyncActiveLow,AsyncActiveLow	AsyncActiveHigh
rst_val	Register reset value (must be >= 0)	0

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if ports **clk**, **clk_en** and **rst** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any of ports **rst**, **clk** and **din** are not connected or if at least one of ports **q** and **qb** are not connected.

Bank of Latches (latch)



This part provides a bank of latches. This part has a level-sensitive trigger. The scalar input port **oena** is the trigger for the latch. If the scalar input port **oena** is activated the value of the input data bus port **din** is placed in the output data bus port **lout**.

The activation of scalar input port **oena** is controlled by the enumerated parameter *oena_type* (ActiveHigh, ActiveLow). If the value of the parameter *oena_type* is ActiveHigh, the scalar input port **oena** is Active on a value of 1. If the value of the parameter *oena_type* is ActiveLow, the scalar input port **oena** is Active on a value of 0.

The scalar input port **rst** has precedence over the input port **oena**. If the **rst** port is activated, the value of the parameter *rst_val* is placed in the output port **lout**. The **rst** port is controlled by the enumerated parameter *rst_type* (ActiveHigh or ActiveLow). If the value of parameter *rst_type* is ActiveHigh, the scalar input port **rst** is active on a value of 1. If the value of *rst_type* is ActiveLow, the scalar input port **rst** is active on a value of 0. If none of the controlling input ports (**rst** and **oena**) are active, the output port **lout** retains its values.

This part is equivalent to *n* latches; where *n* is the width of the ports **din** and **lout**. The inferred latch depends upon the type of input ports **rst** and **oena**.

Parameter *rst_val* can take LNBF format as described in the Constant Value (constval) model.

Function

lout	= *rst_val*	if **rst** is active, else
	= **din**	if **oena** is active
	= **lout**	otherwise

Truth Table

Table 8-4. Bank of Latches Truth Table

oena	rst	lout
-	1	rst_val
1	0	din
0	0	lout

Parameters

Table 8-5. Bank of Latches Parameters

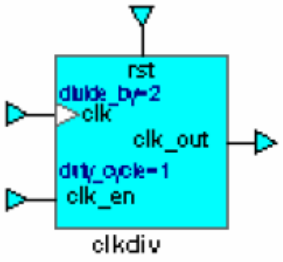
Parameter	Values	Default
din, lout	Port widths (must be > 0)	Automatic
rst_val rst_type oena_type	Register reset value (must be >= 0) ActiveHigh,ActiveLow ActiveHigh,ActiveLow	0 ActiveHigh ActiveHigh

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if ports **rst**, and **oena** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any of ports are not connected.

Clock Divider (clkdiv)

This part allows you to divide a clock into smaller frequency clocks.



The frequency of the output clock (scalar output port **clk_out**) is the frequency of the scalar input clock port (**clk**) divided by the value of the parameter *divide_by*. For example, a value of 4 for parameter *divide_by* provides an output clock that is 4 times slower.

The value of the parameter *duty_cycle* controls the position of the rising edge. In every cycle (time period of **clk** * *divide_by*), the output begins with a value 0 and has one rising edge during the cycle. The value of the parameter *duty_cycle* gives the number of time periods of scalar input port **clk** for which the output is 1.

The part can be reset by the scalar input port **rst**. The reset operation resets the internal counter to a 0. The reset operation always puts the clock divider back to the start of the output clock period.

The part can be triggered on the rising edge or the falling edge of the scalar input port **clk**. The polarity of port **clk** is controlled by the enumerated parameter *clk_type* (Rising, Falling). Note that RisingLast, FallingLast, RisingEdge and FallingEdge are also supported for VHDL.

The **clk** can optionally be enabled by the scalar input port **clk_en**. The polarity of **clk_en** is controlled by the enumerated parameter *clk_en_type* (ActiveHigh, ActiveLow). Do not change the value of the parameter *clk_en_type* if the **clk_en** port is not used.

The register can be reset to the value of the parameter *rst_val* by activating the scalar input port **rst**. The reset mode is controlled by the parameter *rst_type* (SyncActiveHigh, AsyncActiveHigh, SyncActiveLow, AsyncActiveLow). If the reset functionality is not used, do not change the value of the parameter *rst_type*.

This part is equivalent to a clock divider. The performance cannot be estimated because it depends on the synthesis tools. A clock divider has many possible counter implementations such as a binary counter, LFSR counter or one-hot counter. You are advised to carefully control the implementation via synthesis scripts based on the requirements.

If **clk_en** is not connected, it is driven by 1. If the clock enable feature is not used, the scalar input port **clk_en** must be unconnected and the parameter *clk_en_type* must be set to ActiveHigh. Port **clk_en** can also be driven by an active driver, in which case the *clk_en_type* must be correctly adjusted. The HDL code for **clk_en** is optimized away if it is not used.

A reset operation must be performed on this part for proper function. If the part is not reset, the simulation part will work but the post synthesis part does not. This is because reset is the only way to load a value in the internal register.

Function

internalreg the internal register.

If reset is synchronous, then at every enabled **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

internalreg	= 0	if **rst** is active
	= 0	if *internalreg* >= (divide_by - 1)
	= *internalreg* +1	otherwise

If reset is asynchronous (*rst_type* = AsyncActiveHigh, or *rst_type* = AsyncActiveLow):

internalreg = 0 if **rst** is active (irrespective of the **clk** trigger), else

at every enabled **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

internalreg	= 0	if *internalreg* >= (divide_by - 1), else
	= *internalreg* +1	otherwise
clk_out	= *internalreg* >	= (*divide_by* - *duty_cycle*)

Truth Table

creg internal register.

The following tables are for positive polarities. For negative polarities, invert the values.

This table is for asynchronous high reset and positive polarities:

Table 8-6. Clock Divider Truth Table — Asynchronous High Reset, Positive Polarities

clk_en	clk	rst	creg
-	-	1	0
0	-	0	creg
1	Posedge	0	if (creg >= divide_by - 1) = 0; else creg+1

This table is for synchronous high reset and positive polarities:

Table 8-7. Clock Divider Truth Table — Synchronous High Reset, Positive Polarities

clk_en	clk	rst	creg
0	-	-	creg
1	Posedge	1	0
1	Posedge	0	if (creg >= divide_by - 1) = 0; else creg+1

clk_out = *creg* >= (*divide_by* - *duty_cycle*)

Parameters

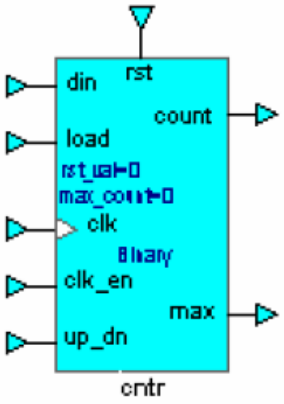
Table 8-8. Clock Divider Parameters

Parameter	Values	Default
clk_en_type	ActiveHigh,ActiveLow,None	ActiveHigh
clk_type	Rising,Falling,RisingLast,FallingLast,RisingEdge,FallingEdge	Rising
rst_type	SyncActiveHigh,AsyncActiveHigh,SyncActiveLow,AsyncActiveLow	AsyncActiveHigh
divide_by	Divider value	2
duty_cycle	Location of the output rising edge	1

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if ports **clk**, **clk_en**, **clk_out** and **rst** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any of ports **rst**, **clk** and **clk_out** are not connected.

Configurable Counter (cntr)



This part provides a highly configurable and highly parameterized counter functionality.

When the scalar input port **clk** is triggered, the counter is activated and the new value if placed in the output port **count**. The *maxcntr* part has been obsoleted and the configurable counter in binary mode gives the same functionality. A value of 1 is placed in the scalar output port **max** if the maximum value is reached.

Dynamic control is provided to increment or decrement the counter during the counter operation. A new value can be loaded into the counter.

This part provides four modes for counting: Binary, Johnson, Linear Feedback Shift Register and One-hot controlled by the enumerated parameter *style* (Binary, Johnson, LFSR, Onehot).

The part can be triggered on the rising edge or the falling edge of the scalar input port **clk**. The polarity of port **clk** is controlled by the enumerated parameter *clk_type* (Rising, Falling). Note that RisingLast, FallingLast, RisingEdge and FallingEdge are also supported for VHDL.

The **clk** can optionally be enabled by the scalar input port **clk_en**. The polarity of port **clk_en** is controlled by the enumerated parameter *clk_en_type* (ActiveHigh, ActiveLow). Do not change the value of the parameter *clk_en_type* if the **clk_en** port is not used.

The counter can be reset to the value of the parameter *rst_val* by activating the scalar input port **rst**. If the style is LFSR, the actual reset value is derived. The reset mode is controlled by the enumerated parameter *rst_type* (SyncActiveHigh, AsyncActiveHigh, SyncActiveLow, AsyncActiveLow).

The value of the input data bus port **din** is loaded into the counter if the scalar input port **load** is active. The scalar input port **rst** has a priority over port **load**. If the **rst** port is active, no value can be loaded.

Note

Both “load” and “din” ports are not optional and have to be connected otherwise a warning will be issued by DRC.

The load mode is controlled by the enumerated parameter *load_type* (ActiveHigh, ActiveLow). If load_type = ActiveHigh, the **load** port is activated with a value of 1. If port **load** is unconnected, it is driven by 0 (disabled). The values are reversed if load_type = ActiveLow.

This part is equivalent to an *n* bit counter; where *n* is the width of the ports **din** and **count**. The implementation can be controlled by the synthesis scripts. This part is very heavily parameterized and the performance is affected by the configuration of the counter.

If **clk_en** is not connected, it is driven by 1. If the clock enable feature is not used, the scalar input port **clk_en** must be unconnected and the parameter *clk_en_type* must be set to

ActiveHigh. Port **clk_en** can also be driven by an active driver, in which case the *clk_en_type* must be correctly adjusted. The HDL code for **clk_en** is optimized away if it is not used.

If the enumerated parameter *up_dn_type* (ControlledByPort, Up, Down) is set to ControlledByPort, the direction of the counter is controlled by the scalar input port **up_dn**. If the value of the port is 1 (or the parameter is set to Up), the count is upwards on an enabled active **clk** edge. If the value of the port is 0 (or the parameter is set to Down), the count is downwards on an enabled active **clk** edge.

The effect of counting upwards or downwards depends of the mode of counting (Binary, Johnson, LFSR or Onehot) as described in the sections for each style.

The input ports **rst** and **load** have a higher priority than the port **up_dn** and the counter counts only when the control ports **rst** and **load** are not active. The result is placed in the output port **count**.

If the counter is counting up and the enumerated parameter *max_type* is set to ActiveHigh, a value of 1 is placed on the output port **max** if the counter value is equal to the value of parameter *max_count*.

If the counter is counting down and the enumerated parameter *max_type* is set to ActiveHigh, a value of 1 is placed on the output port **max** if the counter value is equal to lowest possible value (all bits are o). In either case, if the enumerated parameter *max_type* is set to ActiveLow, a value of 0 is placed on the output port **max**.

Parameter *sync_rst_priority* has no effect when the *rst_type* is AsyncActiveHigh or AsyncActiveLow. When the value of *rst_type* is SyncActiveHigh or SyncActiveLow and *sync_rst_priority* is enabled, a synchronous reset will always reset synchronously regardless of the clock enable. When *sync_rst_priority* is disabled, the clock enable takes priority over the reset.

The counter behavior, the maximum count and the reset value depends upon the value of the parameter *style* as explained in the following sections.

Binary Counter (parameter *style* = Binary)

In this style, the count is executed by increasing or decreasing the counter value by 1. Upon reset, the value of the parameter *rst_val* is placed in the counter.

If the parameter *max_count* has a value of 0, the implied value used for maximum count is the maximum possible value ($2^{(\text{width of output port count})} - 1$). This saves users from having to calculate the maximum possible value but is only available for the binary style.

For all other values of the parameter *max_count*, the implied value is exactly equal to its given value.

If the value of the input port **up_dn** is 1 (counting up), the counter counts until the implied value of *max_count* and upon reaching this value it is set back to 0. If the value of the input port **up_dn** is 0 (counting down), the counter counts down to 0 and is set to the implied value of the parameter *max_count*.

If a new value is loaded at any time (input port **load** = 1), the counter resumes counting from this new value.

In the case of a up counter, if the loaded value is more than the value of the parameter *max_count*, the counter counts up until all bits are 1, wraps back to 0 and counts up until the value of the parameter *max_count*.

In the case of a down counter, if the loaded value is more than the value of the parameter *max_count*, the counter counts down until 0 and wraps back to the value of the parameter *max_count*.

The output port **max** always has a value of 1 when the implied value of the parameter *max_count* is reached. Note that if the counter is counting to the maximum possible value it can hold, it automatically wraps back to 0.

Johnson Counter (parameter *style* = Johnson)

In this style, the counting operation is executed by shifting by one bit all except one of the bits to the left (counting upwards) or right (counting downwards). When counting upwards, the LSB is filled with NOT(MSB) of the counter.

When counting downwards, the most significant bit is filled with NOT(LSB) of the counter. Only two choices are available for the parameter *max_count*; namely all but the MSB is 0 (10…) or all but the MSB and MSB-1 are 0 (110…).

A HDL generation warning is issued if the value of the parameter *max_count* is not equal to a valid choice. Note that the value 10… automatically wraps back to 0. It is important to note that the value of the parameter *max_count* does not signify the number of times the counter counts but it signifies the final bit pattern reached by the counter before wrapping around.

The parameter *rst_val* can be any positive value including a value that the Johnson counter starting to count from 0 cannot reach.

In such a case, the Johnson counter operates with wrong numbers. Any number with bit patterns 101 in it is a wrong number for the parameter *rst_val*. Checking is not performed, allowing the users to operate in this range of numbers. It is important to note that the behavior of the output port **max** does not account for this. Note that an incorrect value can be loaded into the counter through the input port **din** (input port **load** = 1).

If the counter is counting upwards (input port **up_dn** = 1), a value of 1 is placed in the output port **max** if the final bit pattern is reached. If the counter is counting downwards (input port **up_dn** = 0), a value of 1 is placed in the output port **max** if the value of the counter is 0.

LFSR Counter (parameter *style* = LFSR)

In this style of counting, the counter shifts to the right by 1. The MSB is derived by an algorithm that depends upon the width of the counter.

This algorithm is used to cover all the possible values available as far as possible. The algorithm can be deciphered by setting the appropriate width on the instance of this counter in the block diagram and viewing the HDL. The algorithm usually uses XOR and NOT operations on certain bits of the counters.

This style does not support down counting and the input port **up_dn** is ignored.

This counter is very efficient and fast compared to the binary counter. The counter can be used for pseudo-random generator, modulo-x counting, or for frequency division. The width of the counter must be less than 31. Two LFSR counters can be cascaded to achieve larger widths. The reset value of the counter in this style is statically derived from the parameter *rst_val*. The output port **max** is asserted when the value of the counter is 0.

One-Hot Counter (parameter *style* = Onehot)

In this style, the counting operation is executed by shifting all the bits to the left by one bit (counting upwards; input port **up_dn** = 1) or right by one bit (counting downwards; input port **up_dn** = 0). A HDL generation warning is issued if the value of the parameter *rst_val* does not conform to one-hot encoding.

Note that an illegal value can always be loaded dynamically through input port **din** (input port **load** = 1). In such a case, checking is not performed.

The behavior of all the other features in the counter assumes that the value of the counter is a valid one-hot value. If the counter is counting upwards, the MSB of the counter is placed in the output port **max**.

If the counter is counting downwards, the LSB of the counter is placed in the output port **max**. One-hot style is very useful to control bus driving logic where only one driver is to be activated at a time.

The value of the parameter *rst_val* cannot exceed 2^{31} - 1. This means that if the width of the counter is more than 31, bits over 31 are reset to 0. Counters can be cascaded to work around this. The default value for the parameter *max_count* is 0.

Cascading two counters can be done using the value of the output port **max**. Connect the scalar output port **max** of the first counter to the scalar input port **load**. Connect the output port **count** of the second counter to the input port **din** of the second counter (to feedback the counter value). Adjust the polarity of the output port **max** of the first counter to ActiveLow (*max_type* = ActiveLow). In this configuration, the second counter is counting only when the output port **max** is 0, that is when the maximum possible value is reached by the first counter, at all other times the second counter just feeds back its value to itself.

You are advised not to connect the output port **max** of the first counter to the scalar input port **clk_en** of the second counter. This connection has a severe hardware cost and can severely constrain synthesis tools. The same strategy can be used to cascade more than two counters. Note that cascaded counters can give better performance in the hardware. The maximum size of this counter can be set to 2**31.

If input ports **clk** (for asynchronous behavior) is not enabled, the output port **count** retains its value. The value of the output port **max** can change with the scalar input port **up_dn**.

Note that, if only the **max** output is required, the output port **count** can be disabled using the enumerated parameter *count_type* (Enabled, Disabled).

Function for Binary Counter

If *max_count* = 0 then (*mcount* = all bit 1's (*width* = width of port **count**)) else *mcount* = *max_count*)

where *mcount* is the implied value of parameter *max_count*.

If the reset is synchronous, then at every enabled **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

count	= *rst_val*	if **rst** is active, else
	= **din**	if **load** = 1, else
	= **count** + 1	if **up_dn** = 1 and **count** not equal to *mcount*, else
	= 0	if **up_dn** = 1 and **count** = *mcount*, else
	= **count** - 1	if **up_dn** = 0 and **count** not equal to 0
	= *mcount*	otherwise

If the reset is asynchronous:

count = *rst_val* if **rst** is active (irrespective of the **clk** trigger)

else at every enabled **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

count	= **din**	if **load** = 1, else
	= **count** + 1	if **up_dn** = 1 and **count** not equal to *mcount*, else
	= 0	if **up_dn** = 1 and **count** = *mcount*, else
	= **count** - 1	if **up_dn** = 0 and **count** not equal to 0
	= *mcount*	otherwise
max	= 1	if **up_dn** = 1 and **count** = *mcount*
	= 1	if **up_dn** = 0 and **count** = 0
	= 0	otherwise

Function for Johnson counter

If *max_count* = 100...; mcount = *max_count*;
else if *max_count* = 100... - 1 mcount = 1100....)

If reset is synchronous, then at every enabled **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

count	= *rst_val*	if **rst** is active, else
	= **din**	if **load** = 1, else
	= {**count**, *nmsb*} << 1	if **up_dn** = 1 and **count** not equal to *mcount*, else
	= 0	if **up_dn** = 1 and **count** = *mcount*, else
	= {*nlsb*, **count**} >> 1	if **up_dn** = 0 and **count** not equal to 0
	= *mcount*	otherwise

where *nmsb* is the NOT of the most significant bit of the output port **count**; *nlsb* is the NOT of the least significant bit of the output port **count** and *mcount* is the value derived from *max_count*;

If reset is asynchronous:

count = *rst_val* if **rst** is active (irrespective of the **clk** trigger)

else at every enabled **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

count	= **din**	if **load** = 1, else
	= {**count**, *nmsb*} << 1	if **up_dn** = 1 and **count** not equal to *mcount*, else
	= 0	if **up_dn** = 1 and **count** = *mcount*, else
	= {*nlsb*, **count**} >> 1	if **up_dn** = 0 and **count** not equal to 0
	= *mcount*	otherwise

max	= 1	if **up_dn** = 1 and **count** = *mcount*
	= 1	if **up_dn** = 0 and **count** = 0
	= 0	otherwise

Function for LFSR Counter

If reset is synchronous, then at every enabled **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

count	= *rval*	if **rst** is active, else
	= **din**	if **load** = 1, else
	= {*nmsb*, **count**} >> 1	otherwise

where *nmsb* is a bit derived from the algorithm; and *rval* is the reset value derived from a similar algorithm using the value of the parameter *rst_val*.

If reset is asynchronous (*rst_type* = AsyncActiveHigh, or *rst_type* = AsyncActiveLow):

count = *rval* if **rst** is active (irrespective of the **clk** trigger)

else at every enabled **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

count	= **din**	if **load** = 1
	= {*nmsb*, **count**} >> 1	otherwise

max	= 1	if the **count** = 0
	= 0	otherwise

Function for One-Hot Counter

Here *nmsb* is the most significant bit of the output port **count**; *nlsb* is the least significant bit of the output port **count**.

If reset is synchronous, then at every enabled **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

count	= *rst_val*	if **rst** is active, else
	= **din**	if **load** = 1, else
	= **count** << 1	if **up_dn** = 1
	= **count** >> 1	otherwise

If reset is asynchronous:

count = *rst_val* if **rst** is active (irrespective of the **clk** trigger)

else at every enabled **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

count	= **din**	if **load** = 1, else
	= **count** << 1	if **up_dn** = 1, else
	= **count** >> 1	otherwise
max	= *nmsb*	if **up_dn** = 1
	= *nlsb*	otherwise

All the following truth tables are for positive polarities. For negative polarities, invert the values.

Truth Table for Binary Counter

This table is for asynchronous high reset.

Table 8-9. Configurable Counter Truth Table — Binary Counter, Asynchronous High Reset

max_count	mcount
0	all bits are 1 (width = width of port count)
else	max_count

Table 8-10. Configurable Counter Truth Table — Binary Counter, Asynchronous High Reset

load	clk_en	clk	rst	up_dn	count
-	-	-	1	-	rst_val
-	0	-	0	-	count
1	1	posedge	0	-	din
0	1	posedge	0	1	if (**count** <= mcount) **count**+1; else 0
0	1	posedge	0	0	if (**count** not equal to 0) **count**-1; else mcount

This table is for synchronous high reset:

Table 8-11. Configurable Counter Truth Table — Binary Counter, Synchronous High Reset

load	clk_en	clk	rst	up_dn	count
-	0	-	-	-	count
-	1	posedge	1	-	rst_val
1	1	posedge	0	-	din
0	1	posedge	0	1	if (**count** <= mcount) **count**+1; else 0
0	1	posedge	0	0	if (**count** not equal to 0) **count**-1; else mcount

Table 8-12. Configurable Counter Truth Table — Binary Counter, Synchronous High Reset

up_dn	max
1	**count** = mcount
0	**count** = 0

Truth Table for Johnson Counter

This table is for asynchronous high reset. Here *width* is the width of port **count**.

Table 8-13. Configurable Counter Truth Table — Johnson Counter, Asynchronous High Reset

max_count	mcount
10...	max_count
10... - 1	1100...

Table 8-14. Configurable Counter Truth Table — Johnson Counter, Asynchronous High Reset

load	clk_en	clk	rst	up_dn	count
-	-	-	1	-	rst_val
-	0	-	0	-	count
1	1	posedge	0	-	din
0	1	posedge	0	1	if (**count** not equal to mcount) {count, NOT(count[width-1])} << 1; else 0
0	1	posedge	0	0	if (**count** not equal to 0) {NOT(count[0]),count} >> 1; else mcount

This table is for synchronous high reset.

Table 8-15. Configurable Counter Truth Table — Johnson Counter, Synchronous High Reset

load	clk_en	clk	rst	up_dn	count
-	0	-	-	-	count
-	1	posedge	1	-	rst_val
1	1	posedge	0	-	din
0	1	posedge	0	1	if (**count** not equal to mcount) {count, NOT(count[width-1])} << 1; else 0
0	1	posedge	0	0	if (**count** not equal to 0) {NOT(count[0]),count} >> 1; else mcount

Table 8-16. Configurable Counter Truth Table — Johnson Counter, Synchronous High Reset

up_dn	max
1	**count** = mcount
0	**count** = 0

Truth Table for LFSR Counter

This table is for asynchronous high reset. Here *nmsb* is a bit derived from the port **count** using the LFSR derivation polynomial (XOR coefficients), *rval* is a value derived from the parameter *rst_val* using the same algorithm.

Table 8-17. Configurable Counter Truth Table — LFSR Counter, Asynchronous High Reset

load	clk_en	clk	rst	count
-	-	-	1	rval
-	0	-	0	count
1	1	posedge	0	din
0	1	posedge	0	{nmsb, count} >> 1

This table is for synchronous high reset and positive polarities.

Table 8-18. Configurable Counter Truth Table — LFSR Counter, Synchronous High Reset, Positive Polarities

load	clk_en	clk	rst	count
-	0	-	-	count
-	1	posedge	1	rval

Table 8-18. Configurable Counter Truth Table — LFSR Counter, Synchronous High Reset, Positive Polarities (cont.)

load	clk_en	clk	rst	count
1	1	posedge	0	din
0	1	posedge	0	{nmsb, count} >> 1

Table 8-19. Configurable Counter Truth Table — LFSR Counter, Synchronous High Reset, Positive Polarities

up_dn	max
1	**count** = mcount
0	**count** = 0

Truth Table for One-Hot Counter

This table is for asynchronous high reset. Here *width* is the width of port **count**.

Table 8-20. Configurable Counter Truth Table — One-Hot Counter, Asynchronous High Reset

load	clk_en	clk	rst	up_dn	count
-	-	-	1	-	rst_val
-	0	-	0	-	count
1	1	posedge	0	-	din
0	1	posedge	0	1	{count, count[width]} << 1
0	1	posedge	0	0	{count[0],count} >> 1

This table is for synchronous high reset.

Table 8-21. Configurable Counter Truth Table — One-Hot Counter, Synchronous High Reset

load	clk_en	clk	rst	up_dn	count
-	0	-	-	-	count
-	1	posedge	1	-	rst_val
1	1	posedge	0	-	din
0	1	posedge	0	1	{count, count[width]} << 1
0	1	posedge	0	0	{count[0],count} >> 1

Table 8-22. Configurable Counter Truth Table — One-Hot Counter, Synchronous High Reset

up_dn	max
1	count[width] = 1
0	count[0] = 1

Parameters

Table 8-23. Configurable Counter Parameters

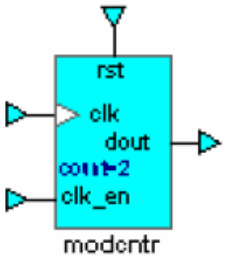
Parameter	Values	Default
din, count	Port widths (must be > 0)	Automatic
clk_en_type	ActiveHigh,ActiveLow,None	ActiveHigh
clk_type	Rising,Falling,RisingLast,FallingLast,RisingEdge,FallingEdge	Rising
count_type	Enabled,Disabled	Enabled
load_type	ActiveHigh,ActiveLow	ActiveHigh
max_type	ActiveHigh,ActiveLow,None	ActiveHigh
rst_type	SyncActiveHigh,AsyncActiveHigh,SyncActiveLow,AsyncActiveLow	AsyncActiveHigh
up_dn_type	ControlledByPort,Up,Down	ControlledByPort
max_count	Maximum count value	0
rst_val	Reset value	0
style	Binary,Johnson,LFSR,Onehot	Binary
sync_rst_priority	Enabled,Disabled	Disabled

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if ports **clk**, **clk_en**, **rst**, **load**, **up_dn** and **max** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any of ports **rst**, **din**, **load**, **clk** and **up_dn** are not connected or if at least one of ports **max** or **count** are not connected.

Modulo Counter (modcntr)

This part provides Wait-for as well as Modulo count functionality.



The counter has an edge-sensitive trigger on the port **clk** with the edge controlled by the enumerated parameter *clk_type*. (Rising, Falling). Note that RisingLast, FallingLast, RisingEdge and FallingEdge are also supported for VHDL.

The **clk** port can optionally be enabled by the **clk_en** port with the polarity (ActiveHigh or ActiveLow) controlled by the enumerated parameter *clk_en_type*. The default value for this parameter should not be changed if the clock enable feature is not going to be used.

The counter can operate in one of two modes (Wait-for and Modulo) controlled by the enumerated parameter *mode*. In Wait-for mode after a reset, the output will be asserted for the first time after as many enabled clock cycles as specified by the parameter count. The value of the integer parameter count can range from 2 to 65534 (both inclusive). Subsequently, the output will be asserted every 2n clock cycles (2n - 1 clock cycles if the style is LFSR), where n is the number of registers required to implement the counter. This results in a more compact implementation as register reloading is obviated. When operated in Modulo mode, the counter asserts the output every count number of enabled clock cycles after receiving a reset signal. Because registers need to be reloaded, this results in higher gate count.

The counter can be reset by means of the **rst** port. The **rst** port is controlled by the enumerated parameter *rst_type*. When the **rst** port is asserted (for asynchronous behavior), the internal registers in the counter are loaded with a value such that the registers are all zero after the specified number (count) of enabled clock edges have occurred, at which time the output port **dout** is asserted. The polarity of the assertion of the port **dout** is controlled by an enumerated parameter *dout_type* (ActiveHigh or ActiveLow). For synchronous behavior, the same is true except that the reset action will take place only if the **rst** port is asserted while an enabled clock edge is also present.

The counter has three styles of counting: LFSR (Linear Feedback Shift Register), Binary Increment and Binary Decrement. These are controlled by the enumerated parameter *style*.

In Binary Increment and Binary Decrement style, the internal registers are loaded with appropriate values and at each clock edge, the values are incremented or decremented by 1 respectively.

In LFSR style for counts less than 6, the counter that is actually generated will be a Johnson counter (a special case of LFSR) because this is a simpler implementation. The Johnson counter basically shifts the contents of the registers to the right by one place and fills in the most significant bit with the NOT of the previous least significant bit. For counts greater than 6, the contents of the registers are shifted to the right by one place and the most significant bit is derived using an appropriate pseudo-random number generation polynomial. LFSR style results in significantly reduced gate count compared to the binary increment or decrement styles.

To make sure that upon reset the output is asserted after the specified number (count) of clock edges, the reset state must be a unique state. Binary counters with n registers can have a

maximum of 2n states and the corresponding LFSR counters will have 2n -1 states (Johnson counters have 2n states). Because the reset state has to be unique, the maximum allowable count with n registers is 2n-1 for binary and 2n-2 for LFSR (2n-1 for Johnson). As a result when the count is equal to 2n-1, the LFSR counter will have one register more than the corresponding binary (incrementing/decrementing) counter. For all other cases however, the number of registers required will be the same.

If **clk_en** is not connected, it is driven by 1. If the clock enable feature is not used, the scalar input port **clk_en** must be unconnected and the parameter *clk_en_type* must be set to ActiveLow. Port **clk_en** can also be driven by an active driver, in which case the *clk_en_type* must be correctly adjusted. You are advised to not edit the value of *clk_en_type* if the port **clk_en** is not to be used. The HDL code for **clk_en** is optimized away if it is not used.

Parameter *sync_rst_priority* has no effect when the *rst_type* is AsyncActiveHigh or AsyncActiveLow. When *rst_type* is SyncActiveHigh or SyncActiveLow and *sync_rst_priority* is enabled, a synchronous reset will always reset synchronously regardless of the clock enable. When *sync_rst_priority* is disabled, the clock enable takes priority over the reset.

Function

creg	the internal register
newmsb	the new value of the MSB that is calculated from the current register state by using a special polynomial.
reloadval	value assigned to registers when all registers contain zero (for modulo mode only)
resetval	value assigned to registers on reset

For LFSR style

If reset is synchronous, then at every enabled **clk** trigger (rising edge if *clk_type* = Rising, falling edge if *clk_type* = Falling):

creg	= *resetval*	if **rst** is active
	= *reloadval*	if *creg* = 0 and *mode* = modulo
	= {NOT(LSB), *creg*} >> 1	otherwise when **count** < 6
	= {*newmsb*, *creg*} >> 1	otherwise when **count** ≥ 6
dout	= (*creg* = 0)	if *dout_type* = ActiveHigh
	= NOT(*creg* = 0)	if *dout_type* = ActiveLow

If reset is asynchronous:

creg	= *resetval*	if **rst** is active

else at every enabled **clk** trigger (rising edge if *clk_type* = Rising, falling edge if *clk_type* = Falling):

creg	= *reloadval*	if *creg* = 0 and *mode* = modulo
	= {NOT(LSB), *creg*} >> 1	otherwise when **count** < 6
	= {*newmsb*, *creg*} >> 1	otherwise when **count** ≥ 6
dout	= (*creg* = 0)	if *dout_type* = ActiveHigh
	= NOT(*creg* = 0)	if *dout_type* = ActiveLow

For Binary Decrement style

If reset is synchronous, then at every enabled **clk** trigger (rising edge if *clk_type* = Rising, falling edge if *clk_type* = Falling):

creg	= **count**	if **rst** is active
	= **count** -1	if *creg* = 0 and *mode* = modulo
	= *creg* -1	otherwise
dout	= (*creg* = 0)	if *dout_type* = ActiveHigh
	= NOT(*creg* = 0)	if *dout_type* = ActiveLow

If reset is asynchronous:

creg = **count** if **rst** is active

else at every enabled **clk** trigger (rising edge if *clk_type* = Rising, falling edge if *clk_type* = Falling):

creg	= **count** -1	if *creg* = 0 and *mode* = modulo
	= *creg* - 1	otherwise
dout	= (*creg* = 0)	if *dout_type* = ActiveHigh
	= NOT(*creg* = 0)	if *dout_type* = ActiveLow

For Binary Increment style

n is the minimum number of registers required.

If reset is synchronous, then at every enabled **clk** trigger (rising edge if *clk_type* = Rising, falling edge if *clk_type* = Falling):

creg	= 2^n - **count**	if **rst** is active
	= 2^n - **count** + 1	if *creg* = 0 and *mode* = modulo
	= *creg* + 1	otherwise
dout	= (*creg* = 0)	if *dout_type* = ActiveHigh
	= NOT(*creg* = 0)	if *dout_type* = ActiveLow

If reset is asynchronous:

creg = 2^n - **count** if **rst** is active

else at every enabled **clk** trigger (rising edge if *clk_type* = Rising, falling edge if *clk_type* = Falling):

creg	= 2^n - **count** + 1	if *creg* = 0 and *mode* = modulo
	= *creg* + 1	Otherwise
dout	= (*creg* = 0)	if *dout_type* = ActiveHigh
	= NOT(*creg* = 0)	if *dout_type* = ActiveLow

Truth Table

creg	The internal register
newmsb	The new value of the MSB that is calculated from the current register state (by using a LFSR polynomial for count ≥ 6; the NOT of the LSB for count < 6)
reloadval	The value assigned to registers when all registers contain zero (for modulo mode only)
resetval	The value assigned to registers on reset

The following tables are for positive polarities. For negative polarities, invert the values.

The following truth table is for the output port **dout** and applies to all counter styles.

Table 8-24. Modulo Counter Truth Table — Output Port dout, All Styles

creg (n)	dout (n)
0	1
≠0	0

For LFSR counter

This table is for synchronous high reset.

Table 8-25. Modulo Counter Truth Table — LFSR Counter, Synchronous High Reset

mode	clk_en	clk	rst	creg (n-1)	creg (n)
-	0	-	-	-	creg(n-1)
-	1	-	1	-	resetval
Modulo	1	posedge	0	0	reloadval
Wait-for	1	posedge	0	0	{newmsb, creg(n-1)} >> 1
-	1	posedge	0	≠ 0	{newmsb, creg(n-1)} >> 1

This table is for asynchronous high reset.

Table 8-26. Modulo Counter Truth Table — LFSR Counter, Asynchronous High Reset

mode	clk_en	clk	rst	creg (n-1)	creg (n)
-	0	-	0	-	creg(n-1)
-	-	-	1	-	resetval
Modulo	1	posedge	0	0	reloadval
Wait-for	1	posedge	0	0	{newmsb, creg(n-1)} >> 1
-	1	posedge	0	≠ 0	{newmsb, creg(n-1)} >> 1

For Binary Decrementing Counter

This table is for synchronous high reset.

Table 8-27. Modulo Counter Truth Table — Binary Decrementing Counter, Synchronous High Reset

mode	clk_en	clk	rst	creg (n-1)	creg (n)
-	0	-	-	-	creg(n-1)
-	1	-	1	-	count
Modulo	1	posedge	0	0	count -1
Wait-for	1	posedge	0	0	creg(n-1) - 1
-	1	posedge	0	≠ 0	creg(n-1) - 1

This table is for asynchronous high reset.

Table 8-28. Modulo Counter Truth Table — Binary Decrementing Counter, Asynchronous High Reset

mode	clk_en	clk	rst	creg (n-1)	creg (n)
-	0	-	0	-	creg(n-1)
-	-	-	1	-	count
Modulo	1	posedge	0	0	count -1
Wait-for	1	posedge	0	0	creg(n-1) - 1
-	1	posedge	0	≠ 0	creg(n-1) - 1

For Binary Incrementing Counter

This table is for synchronous high reset. (n is the minimum number of registers required.)

Table 8-29. Modulo Counter Truth Table — Binary Incrementing Counter, Synchronous High Reset

mode	clk_en	clk	rst	creg (n-1)	creg (n)
-	0	-	-	-	creg(n-1)
-	1	-	1	-	2^n - count
Modulo	1	posedge	0	0	2^n - count + 1
Wait-for	1	posedge	0	0	creg(n-1) + 1
-	1	posedge	0	≠ 0	creg(n-1) + 1

This table is for asynchronous high reset.

Table 8-30. Modulo Counter Truth Table — Binary Incrementing Counter, Asynchronous High Reset

mode	clk_en	clk	rst	creg (n-1)	creg (n)
-	0	-	0	-	creg(n-1)
-	-	-	1	-	2^n - count
Modulo	1	posedge	0	0	2^n - count + 1
Wait-for	1	posedge	0	0	creg(n-1) + 1
-	1	posedge	0	≠ 0	creg(n-1) + 1

Parameters

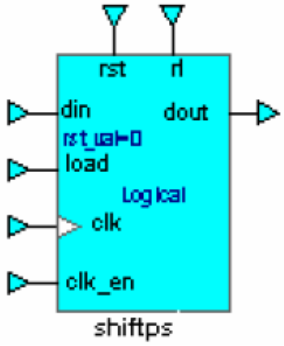
Table 8-31. Modulo Counter Parameters

Parameter	Values	Default
clk_en_type	ActiveHigh,ActiveLow,None	ActiveHigh
clk_type	Rising,Falling,RisingLast,FallingLast,RisingEdge,FallingEdge	Rising
dout_type	ActiveHigh,ActiveLow	ActiveHigh
rst_type	SyncActiveHigh,AsyncActiveHigh,SyncActiveLow,AsyncActiveLow	AsyncActiveHigh
count	Value of the count interval (1 < count < 65535)	2
mode	Modulo,Wait-for	Modulo
style	LFSR,BinaryIncrement,BinaryDecrement	LFSR
sync_rst_priority	Enabled,Disabled	Disabled

Design Rule Checks

- An error is issued if the width of any port cannot be determined.
- A warning is issued and HDL generation fails for this part if any of ports **clk**, **rst** or **dout** are not connected.

Parallel to Serial Shifter (shiftps)



This part allows you to load data into the shift register and shift out this data one bit at a time. The resultant bit is placed in the scalar output port **dout**. The number of internal flip-flops used is equal to the width of the input data port **din**. A new value can be loaded into the internal register by loading the value of the input data port **din** or by the reset operation. The reset operation has the highest priority. The load operation is controlled by the scalar input port **load**. The value of input data port **din** is loaded when the scalar input port **load** is activated. The load operation is synchronous with respect to the scalar input port **clk**.

A shift operation takes place on the internal register in the absence of a load operation or a reset operation. During the shift operation, the internal register is shifted by 1. The shift operation can be performed in one of three modes set by the enumerated parameter *mode* (Logical, Arithmetic, Circular).

The direction of the shift is decided by the scalar input port **rl**. If port **rl** is activated, the shift right operation is executed. If port **rl** is deactivated, a shift left operation is executed. The polarity of port **rl** is controlled by the enumerated parameter *rl_type* (ActiveHigh, ActiveLow). Port **rl** is considered activated if its value is 1 and the polarity is ActiveHigh, or its value is 0 and the polarity is ActiveLow. For Logical mode, 0 is shifted in (for both left and right shift operations). For Arithmetic mode, 0 is shifted into the LSB for left shift but for the right shift the MSB is retained. For Circular mode, the right shift involves shifting the LSB to the MSB while the left shift involves shifting the MSB to the LSB.

The part can be triggered on the rising edge or the falling edge of port **clk**. The polarity of port **clk** is controlled by the enumerated parameter *clk_type* (Rising, Falling). Note that RisingLast, FallingLast, RisingEdge and FallingEdge are also supported for VHDL.

The **clk** can optionally be enabled by the scalar input port **clk_en**. The polarity port **clk_en** is controlled by the enumerated parameter *clk_en_type* (ActiveHigh, ActiveLow). Do not change the value of the parameter *clk_en_type* if the **clk_en** port is not used.

The register can be reset to the value of parameter *rst_val* by activating the scalar input port **rst**. The reset mode is controlled by the enumerated parameter *rst_type* (SyncActiveHigh, AsyncActiveHigh, SyncActiveLow, AsyncActiveLow).

If the input ports **clk** and **rst** (for asynchronous behavior) are not enabled the internal register retains its value.

The scalar output port gets the value of the MSB or the LSB of the internal register. If the scalar input port **rl** is activated (RIGHT shift), the scalar output port **dout** get the value of the LSB of the internal register. If the scalar input port **rl** is not activated (LEFT shift), the scalar output port **dout** get the value of the MSB of the internal register. Note that this is true even during a reset operation or a load operation.

This part is equivalent to a shift register. This part has n Flip-Flops and shifter logic as per the shift operations (where *n* is the width of the input port **din**). Based on the parameterized state the performance of the shift register can be accurately estimated.

If **clk_en** is not connected, it is driven by 1. If the clock enable feature is not used, the scalar input port **clk_en** must be unconnected and the parameter *clk_en_type* must be set to ActiveHigh. Port **clk_en** can also be driven by an active driver, in which case the *clk_en_type* must be correctly adjusted. The HDL code for **clk_en** is optimized away if it is not used.

If the reset functionality is not used, do not change the value of the parameter *rst_type*.

Note that even though the shift register can operate with only the reset operation, a warning is given if data is not being loaded. This is because the shift register is used mostly through the loaded value and it is assumed that there are missing connections. This warning can be ignored, if only the reset operation is required.

Parameter *rst_val* can take LNBF format as described in the Constant Value (constval) model.

Function

internalreg the internal register.

If reset is synchronous, then at every enabled **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

internalreg	= *rst_val*	if **rst** is active
	= **din**	if **load** is active
	= *internalreg* >> 1	if **rl** is active (RIGHT shift as per the *mode*)
	= *internalreg* << 1	otherwise (LEFT shift as per the *mode*)

If reset is asynchronous (*rst_type* = AsyncActiveHigh, or *rst_type* = AsyncActiveLow):

internalreg = *rst_val* if **rst** is active (irrespective of the **clk** trigger)

else at every enabled **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

internalreg	= **din**	if **load** is active, else
	= *internalreg* >> 1	if **rl** is active (RIGHT shift as per the *mode*),
	= *internalreg* << 1	else (LEFT shift as per the *mode*)
dout	= LSB of *internalreg*	if the **rl** is activated (RIGHT shift), else
	= MSB of *internalreg*	(LEFT shift)

Truth Table

creg internal register.

The following tables are for positive polarities. For negative polarities, invert the values.

This table is for asynchronous high reset.

Table 8-32. Parallel to Serial Shifter Truth Table — Asynchronous High Reset

load	clk_en	Clk	rst	rl	creg
-	-	-	1	-	rst_val
-	0	-	0	-	creg
1	1	Posedge	0	-	din
0	1	Posedge	0	1	creg >> 1
0	1	Posedge	0	0	creg << 1

This table is for synchronous high reset.

Table 8-33. Parallel to Serial Shifter Truth Table — Synchronous High Reset

load	clk_en	Clk	rst	rl	creg
-	0	-	-	-	creg
-	1	Posedge	1	-	rst_val
1	1	Posedge	0	-	din
0	1	Posedge	0	1	creg >> 1
0	1	Posedge	0	0	creg << 1

Table 8-34. Parallel to Serial Shifter Truth Table — Synchronous High Reset

rl	dout
0	LSB or creg
1	MSB of creg

Parameters

Table 8-35. Parallel to Serial Shifter Parameters

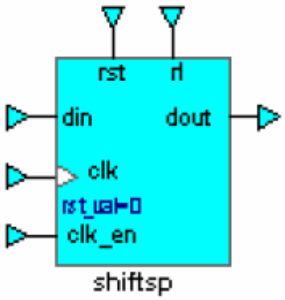
Parameter	Values	Default
din	Port width (must be > 0)	Automatic
clk_en_type	ActiveHigh,ActiveLow	ActiveHigh
clk_type	Rising,Falling,RisingLast,FallingLast,RisingEdge,FallingEdge	Rising
load_type	ActiveHigh,ActiveLow	ActiveHigh
rst_type	SyncActiveHigh,AsyncActiveHigh,SyncActiveLow,AsyncActiveLow	AsyncActiveHigh
rl_type	ActiveHigh,ActiveLow	ActiveHigh
mode	Logical,Arithmetic,Circular	Logical
rst_val	Internal register reset value	0

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if ports **clk**, **clk_en**, **load**, **rst**, **dout** and **rl** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any of ports **dout**, **clk**, **rl**, **load** or **rst** are not connected.

Serial to Parallel Shifter (shiftsp)

This part allows you to load data bits (scalar input port **din**) into the shift register and shift out this data one word at a time. The resultant word is placed in the output port **dout**. The number of internal flip-flops used is equal to the width of the output data port **dout**. A new value can be loaded into the internal register by the reset operation. The reset operation has the highest priority compared to the shift operation.



A shift operation takes place on the internal register in the absence of a load operation. During the shift operation the internal register is shifted by 1.

The direction of the shift is decided by the scalar input port **rl**. If port **rl** is activated, the shift right by 1 operation is executed. If port **rl** is deactivated, a shift left by 1 operation is executed. In both cases, port **din** is shifted into the new bit. For right shift by 1 operation, port **din** is the most MSB while for left shift by 1 operation, port **din** is the LSB. The polarity of port **rl** is controlled by the enumerated parameter *rl_type* (ActiveHigh, ActiveLow). The port **rl** is considered activated if its value is 1 and the polarity is ActiveHigh or its value is 0 and the polarity is ActiveLow.

The part can be triggered on the rising edge or the falling edge of port **clk**. The polarity of port **clk** is controlled by the enumerated parameter *clk_type* (Rising, Falling). Note that RisingLast, FallingLast, RisingEdge and FallingEdge are also supported for VHDL.

The **clk** can optionally be enabled by the scalar input port **clk_en**. The polarity of port *clk_en* is controlled by enumerated parameter *clk_en_type* (ActiveHigh, ActiveLow). Do not change the value of the parameter *clk_en_type* if the **clk_en** port is not used.

The register can be reset to the value of the parameter *rst_val* by activating the scalar input port **rst**. The mode of the **rst** is controlled by the enumerated parameter *rst_type* (SyncActiveHigh, AsyncActiveHigh, SyncActiveLow, AsyncActiveLow. If the input ports **clk** and **rst** (for asynchronous behavior) are not enabled, the internal register retains its value.

This part is equivalent to a shift register. This part has *n* flip-flops and shifter logic as per the shift operations (where *n* is the width of the output port **dout**). Based on the parameterized state, the performance of the shift register can be accurately estimated.

If **clk_en** is not connected, it is driven by 1. If the clock enable feature is not used, the scalar input port **clk_en** must be unconnected and the parameter *clk_en_type* must be set to ActiveHigh. Port **clk_en** can also be driven by an active driver, in which case the *clk_en_type* must be correctly adjusted. The HDL code for **clk_en** is optimized away if it is not used.

Parameter *rst_val* can take LNBF format as described in the Constant Value (constval) model.

Function

internalreg the internal register.

If reset is synchronous, then at every enabled **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

internalreg	= *rst_val*	if **rst** is active, else
	= {din, *internalreg*} >> 1	if **rl** is active (RIGHT shift by 1), else
	= {*internalreg*, din} << 1	(LEFT shift by 1)

If reset is asynchronous (*rst_type* = AsyncActiveHigh, or *rst_type* = AsyncActiveLow):

internalreg = *rst_val* if **rst** is active (irrespective of the **clk** trigger)

At every enabled **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling)

internalreg	= {**din**, *internalreg*} >> 1	if **rl** is active (RIGHT shift by 1), else
	= {*internalreg*, **din**} << 1	(LEFT shift by 1)

dout = *internalreg*

Truth Table

creg internal register.

The following tables are for positive polarities. For negative polarities, invert the values.

This table is for asynchronous high reset.

Table 8-36. Serial to Parallel Shifter Truth Table — Asynchronous High Reset

clk_en	clk	rst	rl	dout
-	-	1	-	rst_val
0	-	0	-	dout
1	Posedge	0	1	{**din, dout**} >> 1
1	Posedge	0	0	{**dout, din**} << 1

This table is for synchronous high reset

Table 8-37. Serial to Parallel Shifter Truth Table — Synchronous High Reset

clk_en	clk	rst	rl	dout
0	-	-	-	dout
1	Posedge	1	-	rst_val
1	Posedge	0	1	{**din, dout**} >> 1
1	Posedge	0	0	{**dout, din**} << 1

Parameters

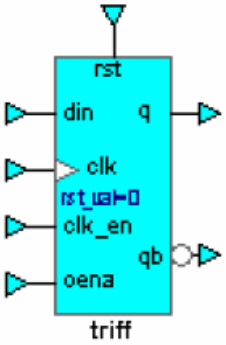
Table 8-38. Serial to Parallel Shifter Parameters

Parameter	Values	Default
dout	Port width (must be > 0)	Automatic
clk_en_type	ActiveHigh,ActiveLow	ActiveHigh
clk_type	Rising,Falling,RisingLast,FallingLast,RisingEdge,FallingEdge	Rising
rst_type	SyncActiveHigh,AsyncActiveHigh,SyncActiveLow,AsyncActiveLow	AsyncActiveHigh
rl_type	ActiveHigh,ActiveLow	ActiveHigh
rst_val	Internal register reset value	0

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if ports **clk**, **clk_en**, **rst**, **din** and **rl** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any of ports **dout**, **clk**, **rl** or **rst** are not connected.

Three-state Bank of Flip-Flops (triff)



This part provides a bank of flip-flops (register) that have tristate outputs. When the scalar input port **clk** is triggered, the data from input port **din** is shifted into the output port **q**. The output port **qb** has the bitwise inverted value of output port **q**.

The part can be triggered on the rising edge or the falling edge of the scalar input port **clk**. The polarity of the scalar input port **clk** is controlled by the enumerated parameter *clk_type* (Rising, Falling). Note that RisingLast, FallingLast, RisingEdge and FallingEdge are also supported for VHDL.

The **clk** can optionally be enabled by the scalar input port **clk_en**. The polarity of port **clk_en** is controlled by the enumerated parameter *clk_en_type* (ActiveHigh, ActiveLow). Do not change the value of the parameter *clk_en_type* if the **clk_en** port is not used.

The register can be reset to the value of the parameter *rst_val* by activating the scalar input port **rst**. The mode of the reset is controlled by the enumerated parameter *rst_type* (SyncActiveHigh, AsyncActiveHigh, SyncActiveLow, AsyncActiveLow). The output ports can be disabled by the scalar input port **oena**. The polarity of port **oena** is controlled by the enumerated parameter *oena_type* (ActiveHigh, ActiveLow).

If the port **oena** is enabled (**oena** = 1 for parameter *oena_type* ActiveHigh and **oena** = 0 for parameter *oena_type* = ActiveLow), the values stored in the flip-flops are placed in the output ports.

If the port **oena** is disabled, the value Z is placed in every bit of the output ports. (Note that in this case, both the output ports become Z.) If input ports **clk** and **rst** (for asynchronous behavior) are not enabled, the outputs of the flip-flops retain their values.

This part is equivalent to *n* flip-flops; where *n* is the width of the ports **din**, **q** and **qb**. The inferred register depends upon the type of reset behavior. The gate count greatly changes if the synthesis tool is able to pick up smaller flip-flops. Note that using **qb** does not imply extra inverters, these inverters are usually absorbed by inferred flip-flop because most have a **qb** port.

If **clk_en** is not connected, it is driven by 1. If the clock enable feature is not used, the scalar input port **clk_en** must be unconnected and the parameter *clk_en_type* must be set to ActiveHigh. Port **clk_en** can also be driven by an active driver, in which case the *clk_en_type* must be correctly adjusted.

The HDL code for unused optional ports is optimized away.

Parameter *rst_val* can take LNBF format as described in the Constant Value (constval) model with no limitation on the size of the **q** and **qb** ports.

Function

An enabled **clk** trigger is a trigger that occurs when input port **clk_en** is active.

If reset is synchronous, then at every enabled **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

q	= 'Z...' (all bits are Z)	if **oena** is disabled (irrespective of the **clk** trigger),
	= *rst_val*	else if **rst** is active
	= **din**	otherwise
qb	= 'Z...' (all bits are Z)	if **oena** is disabled (irrespective of the **clk** trigger),
	= NOT(*rst_val*)	else if **rst** is active
	= NOT(**din**)	otherwise

If reset is asynchronous:

q	= 'Z...' (all bits are Z)	if **oena** is disabled (irrespective of the **clk** trigger)
	= *rst_val*	if **rst** is active (irrespective of the **clk** trigger)

At every enabled **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

q	= **din**	otherwise

qb	= 'Z...' (all bits are Z)	if **oena** is disabled (irrespective of the **clk** trigger)
	= NOT(*rst_val*)	if **rst** is active (irrespective of the **clk** trigger)

At every enabled **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

qb	= NOT(**din**)	otherwise

Truth Table

creg the internal register

The following tables are for positive polarities. For negative polarities, invert the values. This table is for asynchronous high reset.

Table 8-39. Three-State Bank of Flip-Flops Truth Table — Asynchronous High Reset

clk_en	clk	rst	creg
-	-	1	rst_val
0	posedge	0	creg
1	posedge	0	din

This table is for synchronous high reset.

Table 8-40. Three-State Bank of Flip-Flops Truth Table — Synchronous High Reset

clk_en	clk	rst	creg
0	-	-	creg
1	posedge	1	rst_val
1	posedge	0	din

Table 8-41. Three-State Bank of Flip-Flops Truth Table — Synchronous High Reset

oena	q	qb
0	'Z...'	'Z...'
1	creg	NOT(creg)

Parameters

Table 8-42. Three-State Bank of Flip-Flops Parameters

Parameter	Values	Default
din, q, qb	Port widths (must be > 0)	Automatic
clk_en_type	ActiveHigh,ActiveLow	ActiveHigh
clk_type	Rising,Falling,RisingLast,FallingLast,RisingEdge,FallingEdge	Rising
oena_type	ActiveHigh,ActiveLow	ActiveHigh
qb_type	Enabled,Disabled	Enabled
rst_type	SyncActiveHigh,AsyncActiveHigh,SyncActiveLow,AsyncActiveLow	AsyncActiveHigh
rst_val	Register reset value (must be >= 0)	0

Design Rule Checks

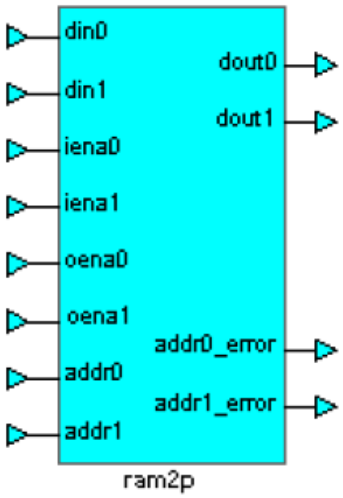
- An error is issued if the width of any port cannot be determined or if ports **clk**, **clk_en**, **oena** and **rst** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any of ports **din**, **clk**, **oena** or **rst** are not connected or if at least one of the ports **q** or **qb** are not connected.

Chapter 9
Memory Parts

This chapter describes complex memory parts. The parameterization has a big impact on the performance of these parts (gate count, delay, power…).

Take care selecting the parameterized behavior for these parts to achieve optimal performance.

Dual Port RAM (ram2p)



This part implements a dual port Random Access Memory (RAM). The depth of the RAM is given by the parameter *ram_size*. The internal memory elements have a latch behavior (that is, they are level sensitive).

The RAM provides two write and two read operations. The write operations take place if the scalar input ports **iena0** or **iena1** are enabled (**iena0** = 1 or **iena1** = 1). The read operations take place if the scalar input ports **oena0** or **oena1** are enabled (**oena0** = 1 or **oena1** = 1).

In a write operation controlled by the scalar input port **iena0**, the value of the input data bus port **din0** is written to the address location provided by the value of the input bus port **addr0**. In a write operation controlled by the scalar input port **iena1**, the value of the input data bus port **din1** is written to the address location provided by the value of the input bus port **addr1**.

In a read operation controlled by the scalar input port **oena0**, the value in the address location (provided by the value of the input bus port **addr0**) is placed in the output port **dout0**. In a read operation controlled by the scalar input port **oena1**, the value in the address location (provided by the value of the input bus port **addr1**) is placed in the output port **dout1**. If the read operation is disabled by the scalar input port **oena0**, the output port **dout0** is set to Z. If the read operation is disabled by the scalar input port **oena1**, the output port **dout1** is set to Z.

Both the read and write operations can be done simultaneously. Because at one time only two memory locations can be accessed (given by **addr0** and **addr1**), simultaneous operations can mean read and write to the same location. It is the user's responsibility not to perform both read and write operations on the same address location at the same time. An internal RAM table is maintained. When write operations are not taking place, this table maintains its value.

If the value of the address port **addr0** is greater than the value of the parameter *ram_size*, the scalar output port **addr0_error** is set to 1. In these cases, read or write operation do not take place via the address port **addr0**. If the value of the address port **addr1** is greater than the value of the parameter *ram_size*, the scalar output port **addr1_error** is set to 1. In these cases, read or write operation do not take place via the address port **addr1**.

The RAM operations are disabled if the addresses are out of range. If two write operations are done on the same address location, an 'XX...' value is inserted in that memory location.

The part does not have timing control. Users are responsible for the timing control and verifying that the values on the input ports are retained for sufficient time. You are advised to implement a good sequential memory controller for this part to work in a sequential design. Do not synthesize this part. Memory compilers perform better than synthesis on such parts.

This is a non-synthesizable part. Even though synthesis tools can accept the code generated for this part, you should not synthesize this part.

Function

ram_table is a table of internal locations. The number of locations are given by the parameter *ram_size*:

ram_table(**addr0**)	= **din0**	if **iena0** = 1. and **addr0** < *ram_size*
ram_table(**addr1**)	= **din1**	if **iena1** = 1. and **addr1** < *ram_size*
dout0	= *ram_table*(**addr0**) = ZZ...	if **oena0** = 1 and **addr0** < *ram_size* otherwise
dout1	= *ram_table*(**addr1**) = ZZ...	if **oena1** = 1 and **addr1** < *ram_size* otherwise
addr0_error	= 0 = 1	if **addr0** < *ram_size* otherwise
addr1_error	= 0 = 1	if **addr1** < *ram_size* otherwise

Parameters

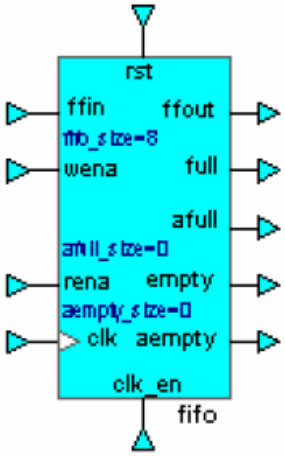
Table 9-1. Dual Port RAM Parameters

Parameter	Values	Default
din0, din1, dout0, dout1 addr0 addr1	Port widths (must be > 0) Port width (must be > 0) Port width (must be > 0)	Automatic Automatic Automatic
addr0_type addr1_type ram_size	Enabled,Disabled Enabled,Disabled Number of memory locations in the RAM table	Enabled Enabled 8

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if ports **iena0**, **iena1**, **oena0**, **oena1**, **addr0_error** and **addr1_error** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any of ports **din0**, **din1**, **iena0**, **oena0**, **dout0**, **addr1**, **iena1**, **oena1** or **dout1** are not connected.

First In First Out (fifo)



This part implements a First-In-First-Out (FIFO) queue. Data can be loaded into the internal registers and read at a later time. All the operations in this part (except for reset) are synchronous with respect to the clock. The depth is controlled by the parameter *fifo_size*.

The FIFO supports two input/output operations. A write operation takes the value of the input port **ffin** and places it in the internal registers. A read operation takes the oldest value in the internal registers and places it in the output port **ffout**. This value is retained on **ffout** until the next read operation.

The read operation is controlled by the scalar input port **rena** and the write operation is controlled by the scalar input port **wena**. The write operation has precedence over the read operation. The polarity of **rena** is controlled by the enumerated parameter *rena_type* (ActiveHigh, ActiveLow). The polarity of the scalar output port **wena** is controlled by the enumerated parameter *wena_type* (ActiveHigh, ActiveLow).

The part consists of the required number of the internal registers, an address counter and the control logic. Internally, the FIFO will be of depth *fifo_size* + 1. The extra register is used to retain the old value. It is used for driving the output port and is not a part of the FIFO memory. The address counter keeps the count of the current address. This value is used for the read and write operations. This counter decides whether the FIFO is empty (counter = 0) or full (counter = *fifo_size*). The reset operation is required before a correct operation of the FIFO (for post synthesis simulation match). Read operations cannot be done before a write operation.

Write operations are disabled if the internal registers are full (address counter = *fifo_size*). Read operations are disabled if the internal registers are empty (address counter = 0). If the internal registers are full and both write and read operations are triggered, the read operation takes place because the write operation is disabled. When the internal registers are full the read operation takes precedence over the write (because write is disabled).

The scalar output port **full** is activated if the internal registers are full. The scalar output port **empty** is activated if the internal registers are empty. The polarity of the scalar output port **full** is controlled by the parameter *full_type* (ActiveHigh, ActiveLow). The polarity of the scalar output port **empty** is controlled by the parameter *empty_type* (ActiveHigh, ActiveLow).

The part also has two ports (**afull** and **aempty**) which are used to signal an “almost-full” and “almost-empty” conditions. The polarity of these ports is controlled by the enumerated parameters *afull_type* (ActiveHigh, ActiveLow) and *aempty_type* (ActiveHigh, ActiveLow). The almost-full and almost-empty criteria are set using the integer parameters *afull_size* and *aempty_size*.

The **afull** port will then be asserted whenever the address counter equals or exceeds *afull_size*. If the parameter *afull_size* is less than or equal to zero (zero is also the default value for *afull_size*) or *afull_size* is greater than *fifo_size*, then the port **afull** will be asserted whenever the internal registers are full (address counter = *fifo_size*).

The scalar output port **aempty** is asserted whenever the address counter is equal to or less than *aempty_size*. The default value for *aempty_size* is zero. For values of *aempty_size* less than or equal to zero, assertion will take place whenever the internal registers are empty (address counter = 0).

The part can be triggered on either the rising edge or the falling edge of the scalar input port **clk**. The polarity of **clk** is controlled by the enumerated parameter *clk_type* (Rising, Falling, RisingLast, FallingLast, RisingEdge, FallingEdge).

(Note that RisingLast, FallingLast, RisingEdge and FallingEdge are supported for VHDL only.)

The **clk** can optionally be enabled by the scalar input port **clk_en**. The polarity **clk_en** is controlled by the enumerated parameter *clk_en_type* (ActiveHigh, ActiveLow).

The reset operations do not change the values stored inside the internal registers but reset the address counter to 0. After a reset operation, further read operations are disabled (until after the next write) because the FIFO is empty. The mode of reset is controlled by the enumerated parameter *rst_type* (SyncActiveHigh, AsyncActiveHigh, SyncActiveLow, AsyncActiveLow).

If the input ports **clk** and **rst** (for asynchronous behavior) are not enabled the internal register retains its value.

The parameter *simultaneous_rw* (Allowed, Disallowed) controls whether simultaneous read and write operations are enabled.

This part is equivalent to a FIFO with *n* * (*fifo_size*+1) flip-flops, an address counter and control logic (where *n* is the width of the output port **ffout**).

Not using certain operations (not connecting port **full** or port **empty**) affects the final gate count and the delay. Leave the clock enable port (**clk_en**) unconnected if it is not required. The FIFO performance can be accurately estimated from the parameterized state.

The HDL code for the unused optional port is optimized away.

Function

cnt internal address counter
creg internal register

If reset is synchronous, then at every enabled **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

cnt = 0 if **rst** is activated, else
= *cnt* + 1 if scalar input port **wena** is activated and count < *fifo_size*, else
= *cnt* - 1 if scalar input port **rena** is activated and count > 0
= *cnt* otherwise

If reset is asynchronous:

cnt = 0 if input port **rst** is active (irrespective of the **clk** trigger), else

at every enabled **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

cnt = *cnt* + 1 if scalar input port **wena** is activated and **count** < *fifo_size*, else
= *cnt* - 1 if scalar input port **rena** is activated and **count** > 0, else
= *cnt* otherwise

For a write operation (if **cnt** < *fifo_size*; and **wena** is enabled):

creg(*cnt*) = **ffin**

ffout = *creg*(0)

Truth Table

The following tables are for positive polarities. For negative polarities, invert the values. There are *fifo_size* + 1 internal registers (*creg*) (each has a width of input **ffin**). The address count is stored in the register *cnt*.

This table is for asynchronous high reset.

Table 9-2. First In First Out Truth Table — Asynchronous High Reset

wena	rena	clk_en	clk	rst	cnt	creg
-	-	-	-	1	0	creg
-	-	0	-	0	cnt	creg
1	-	1	posedge	0	if (cnt < *fifo_size*) cnt+1	if (cnt < *fifo_size*) creg[cnt+1] = ffin
0	1	1	posedge	0	if (cnt not equal to 0) cnt -1	if (cnt not equal to 0) creg[cnt] = creg[cnt-1]

This table is for synchronous high reset.

Table 9-3. First In First Out Truth Table — Synchronous High Reset

wena	rena	clk_en	clk	rst	cnt	creg
-	-	0	-	-	cnt	creg
-	-	1	posedge	1	0	creg-
1	-	1	posedge	0	if (cnt < fifo_size) cnt+1	if (cnt < fifo_size) creg[cnt+1] = ffin
0	1	1	posedge	0	if (cnt not equal to 0) cnt -1	if (cnt not equal to 0) creg[cnt] = creg[cnt-1]

ffout = *creg*[0]
aempty = (*cnt* ≤ *aempty_size*)
afull = (*cnt* ≥ *afull_size*)
empty = (*cnt* = 0)
full = (*cnt* = *fifo_size*)

Parameters

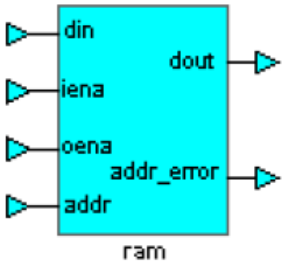
Table 9-4. First In First Out Parameters

Parameter	Values	Default
ffin, ffout	Port widths (must be > 0)	Automatic
aempty_type	ActiveHigh,ActiveLow	ActiveHigh
afull_type	ActiveHigh,ActiveLow	ActiveHigh
clk_type	Rising,Falling,RisingLast,FallingLast,RisingEdge,FallingEdge	Rising
clk_en_type	ActiveHigh,ActiveLow	ActiveHigh
empty_type	ActiveHigh,ActiveLow	ActiveHigh
full_type	ActiveHigh,ActiveLow	ActiveHigh
rena_type	ActiveHigh,ActiveLow	ActiveHigh
rst_type	SyncActiveHigh,AsyncActiveHigh,SyncActiveLow,AsyncActiveLow	AsyncActiveHigh
wena_type	ActiveHigh,ActiveLow	ActiveHigh
aempty_size	Almost-empty condition evaluation	0
afull_size	Almost-full condition evaluation	0
fifo_size	FIFO size (maximum number of data elements)	0
simultaneous_rw	Allowed,Disallowed (simultaneous read and write)	Disallowed

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if ports **clk**, **clk_en**, **rst**, **wena**, **rena**, **full** and **empty** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any of ports **ffin**, **ffout**, **wena**, **rena**, **clk** or **rst** are not connected.

Single Port RAM (ram)



This part implements a Random Access Memory (RAM). The depth of the RAM is given by the parameter *ram_size*. The internal memory elements have a latch behavior (that is, they are level sensitive).

The RAM has write and read operations. The write operation takes place if the scalar input port **oena** is enabled (**oena** = 1). The read operation takes place if the scalar input port **iena** is enabled (**iena** = 1).

In a write operation, the value of the input bus **din** is written to the address location specified by the value of the input bus **addr**.

In a read operation, the value in the address location specified by **addr** is placed on the output port **dout**. If the write operation is disabled, the output port **dout** goes to tristate mode.

Even though read and write operations can be done simultaneously, it is not recommended. Because only one memory address can be accessed at any given time, a simultaneous operation means read and write to the same location.

An internal RAM table is maintained. When write is not taking place, this table maintains its value. If the value of the address port is greater than the value of *ram_size*, the scalar output port **addr_error** is set to 1. In such cases, read or write operations do not take place. The RAM operations are disabled if the address is out of range.

The part has no timing control. Users are responsible for the timing control and to verify that the values on the input ports are retained for sufficient time. You need a good sequential memory controller for this part to work in a sequential design. We recommend that you do not synthesize this part, because memory compilers perform much better than logic synthesis on memory parts.

This is a non-synthesizable part. Even though synthesis tools can accept the code generated for this part, you are advised not to synthesize this part.

Function

ram_table is a table of internal locations. The number of locations are given by the parameter *ram_size*:

ram_table(**addr**)	= **din**	if **iena** = 1. and **addr** < *ram_size*
dout	= *ram_table*(**addr**) = ZZ...	if **oena** = 1 and **addr** < *ram_size* otherwise
addr_error	= 0 = 1	if **addr** < *ram_size* otherwise

Parameters

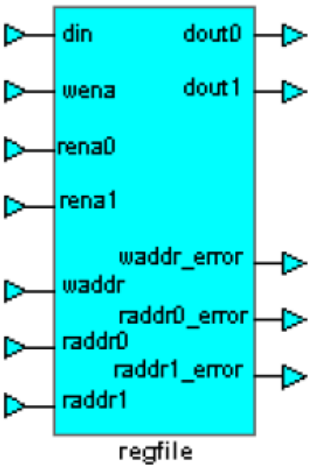
Table 9-5. Single Port RAM Parameters

Parameter	Values	Default
din, dout, addr	Port widths (must be > 0) Port widths (must be > 0)	Automatic Automatic
ram_size	Number of memory locations in the RAM table	8

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if ports **iena**, **oena** and **addr_error** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any of ports **din**, **iena**, **oena**, **addr** or **dout** are not connected.

Register File (regfile)



This part implements a Register File (RAM based). This part is useful in processor designs. The depth of the RAM is given by the parameter *regfile_size*. The internal memory elements have a latch behavior (that is, they are level sensitive).

The Register File provides two read and one write operations. These operations can be done simultaneously. The write operations takes place if the scalar input port **wena** is enabled (**wena** = 1). The read operations takes place if the scalar input ports **rena0** or **rena1** are enabled (**rena0** = 1 or **rena1** = 1).

In a write operation, the value of the input bus **din** is written to the address location specified by the value of the input bus **waddr**. In a read operation controlled by the scalar input port **rena0**, the value in the address location (specified by the value of the input bus **raddr0**) is placed in the output port **dout0**.

In a read operation controlled by the scalar input port **rena1**, the value in the address location (provided by the value of the input bus port **raddr1**) is placed in the output port **dout1**. If the read operation is disabled by the scalar input port **rena0**, the output port **dout0** goes to tristate. If the read operation is disabled by the scalar input port **rena1**, the output port **dout1** goes to tristate. While the read and write operations can be done simultaneously, you should not enable both read and write operations on the same address location at the same time. An internal register file table is maintained. When write operations are not taking place, this table maintains its value.

If the value of the address port **raddr0** is greater than the value of the parameter *regfile_size*, the scalar output port **raddr0_error** is set to 1. In these cases, read operations do not take place via the address port **raddr0**. If the value of the address port **raddr1** is greater than the value of the parameter *regfile_size*, the scalar output port **raddr1_error** is set to 1. In these cases, read operations do not take place via the address port **raddr1**. If the value of the address port **waddr** is greater than the value of the parameter *regfile_size* (the size of the RAM table), the scalar output port **waddr_error** is set to 1. In these cases, write operations do not take place via the address port **waddr**. The Register File operations are disabled if the addresses are out of range.

The part does not have timing control. Users are responsible for the timing control and for verifying that the values on the input ports are retained for sufficient time. You are advised to implement a good sequential memory controller for this part to work in a sequential design. Do not synthesize this part. Memory compilers perform better than synthesis on such parts.

This is a non-synthesizable part. Even though synthesis tools can accept the code generated for this part, do not synthesize this part.

Function

regfile is a table of internal locations. The number of locations is given by the parameter *regfile_size*:

regfile(**waddr**)	= **din**	if **wena** = 1 & **waddr** < *regfile_size*
dout0	= *regfile*(**raddr0**) = ZZ…	if **rena0** = 1 & **waddr0** < *regfile_size* otherwise
dout1	= *regfile*(**raddr1**) = ZZ...	if **rena1** = 1 & **waddr1** < *regfile_size* otherwise
raddr0_error	= 0 = 1	if **raddr0** < *regfile_size* otherwise
raddr1_error	= 0 = 1	if **raddr1** < *regfile_size* otherwise
waddr_error	= 0 = 1	if **waddr** < *regfile_size* otherwise

Parameters

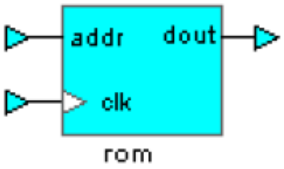
Table 9-6. Register File Parameters

Parameter	Values	Default
din, dout0, dout1 raddr0 raddr1 waddr	Port widths (must be > 0) Port width (must be > 0) Port width (must be > 0) Port width (must be > 0)	Automatic Automatic Automatic Automatic
raddr0_error_type raddr1_error_type waddr_error_type regfile_size	Enabled,Disabled Enabled,Disabled Enabled,Disabled Number of memory locations in the register file	Enabled Enabled Enabled 8

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if ports **wena**, **rena**, **rena1**, **raddr0_error**, **raddr1_error** and **waddr_error** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any of ports **rena0**, **rena1**, **raddr0**, **raddr1**, **waddr**, **rena**, **dout0** or **dout1** are not connected.

ROM (rom)



This part implements a synthesizable Read Only Memory (ROM) generator which reads an initialization file during HDL generation. The generator supports Intel HEX file format.

The part can be triggered on either the rising edge or the falling edge of the scalar input port **clk**. The polarity of clk is controlled by the enumerated parameter *clk_type* (Rising, Falling, RisingLast, FallingLast, RisingEdge, FallingEdge). Note that RisingLast, FallingLast, RisingEdge and FallingEdge are supported for VHDL only.

The ROM is inferred by creating a ROM table. The generated HDL uses a CASE statement to initialize the contents of the ROM table. This implementation is efficient when there are multiple extended linear addressing records with empty locations. Empty locations need not be explicitly addressed but are handled by the default branch of the CASE statement. This part can be synthesized using Altera or Xilinx technologies.

The ROM table is initialized based on the values in a file. The pathname of this file is given by the parameter *rom_file*. The file must be in Intel HEX format.

Each Intel Hex record comprises five fields (data length, address, type, data, and checksum) as described in the Hexadecimal File Format (Intel Hex) section.

The Intel HEX file format only supports byte-addressing, so the width of the **dout** port must be exactly 8 bits. The width of the **addr** port should be 16 bits (if the HEX file contains no extended linear addressing) or 31 bits (if the HEX file contains extended linear address records). If the address port width is less than these values, the ROM table may be reduced compared to the number of records in the HEX file. In this case, a warning is issued but does not stop HDL generation.

Although the HEX file can support a maximum of 32 bits of address (using extended linear addressing) the ROM generator only supports address widths of a maximum 31 bits because an address width of 32 bits would make the ROM table array exceeds the maximum level which the array can represent.

Note that extended segment addressing (record type 02) is not supported.

The size of the ROM table is determined by the number of address locations inside the Intel HEX file. For example, if the address port width is 16 bits, the ROM table size will be 65535.

Only the ROM table locations with corresponding addresses and data in the HEX file are addressed.

Note

If the HEX file addresses a large ROM, the ROM table constructed in the HDL code will be very large. You are advised to unset the **Perform Checks** option in the **Checks** tab of the Main Settings dialog box before generating HDL for a ROM component. If this option is set, the parser checks every line in the code (which may reach 500,000 lines) and HDL generation may be very slow.

The actual record is a string of characters. For example:

```
:100000002e15a0e3000081e502d7a0e300a0a0e345
```

For clarity, the following line shows the fields separated by spaces with alternate fields in bold:

```
:10 0000 00 2e15a0e3000081e502d7a0e300a0a0e3 45
```

- In the example above, the "10" gives the length of data in the record. The data length field indicates that it is #h10 bytes which is equivalent to 128 bits of data. Each byte is transferred to an entry in the ROM table referenced with a CASE statement. The HEX file may contain records of different data lengths.
- The "0000" is the starting address of the data in the record. The enumerated parameter *endian_mode* (LittleEndian, BigEndian) effects the way data within the ROM table is saved. In BigEndian mode, the most significant bit of the data is written first then the least significant bit of the data whereas in LittleEndian mode, the least significant bit is written first.
- The “00” is the type of record (“00” = data).
- The "2e15a0e3000081e502d7a0e300a0a0e3" field contains the content of the record. For this example, if LittleEndian is used, the first data byte “e3” will be addressed in location 0. The last byte “2e” will be addressed in location 15. If BigEndian is used, the first data byte “2e” will be addressed in location 0 and the last byte “e3” will be addressed in location 15.
- The "45" represents the checksum field.

Example

The data field 2e15a0e3000081e502d7a0e300a0a0e3 is converted to the following VHDL CASE statement when LittleEndian mode is set:

```
CASE mw_I0addr_i IS
   WHEN 0 => mw_I0rom_table(0) <= "11100011"; -- e3
   WHEN 1 => mw_I0rom_table(1) <= "10100000"; -- a0
   WHEN 2 => mw_I0rom_table(2) <= "10100000"; -- a0
...
...
   WHEN 15 => mw_I0rom_table(15) <= "00101110"; -- 2e
   WHEN OTHERS => mw_I0rom_table(n) <= (OTHERS => ‘X’);
END CASE;
```

The corresponding code for Verilog is:

```
case(addr)
   0:mem[0] = 8'b11100011; -- e3
   1:mem[1] = 8'b10100000; -- a0
   2:mem[2] = 8'b10100000; -- a0
...
...
   15:mem[15] = 8'b00101110; -- 2e
   default:mem[n] = 8'bx;
endcase
```

Function

romtable is the table of values created from the ROM file.

dout = *romtable*(**addr**) if **addr** < *rom_size*

Parameters

Table 9-7. ROM Parameters

Parameter	Values	Default
addr dout	Port width (must be > 0) Port width (must be > 0)	Automatic Automatic
clk_type endian_mode rom_file	Rising,Falling,RisingLast,FallingLast,RisingEdge,FallingEdge LiitleEndian, BigEndian Pathname of file used to initialize the ROM	Rising LiitleEndian null

Design Rule Checks

- An error is issued if the width of any port cannot be determined, if port **clk** does not have a fixed width of 1, if the *rom_file* parameter is empty, if the width of **dout** is not 8 bits, if the Intel HEX file does not end with an end-of-file (EOF) record or if any record in the file violates the syntax for Intel Hex files.
- An error is also issued if the **addr** port width decodes to a number of locations less than the Intel HEX records. In this case, the generation is halted and a message suggests the minimum required address width. For example, if a 3 bit address has been used (decodes to 8 locations) but the HEX file contains 16 address locations, the message suggests having an address width of at least 4 bits.
- A warning is issued and HDL generation fails for this part if any of ports **addr**, **clk** or **dout** are not connected.
- A warning is issued if the **addr** port width decodes to a number of locations greater than the Intel HEX records but the HDL generation continues.

Hexadecimal File Format (Intel Hex)

An Intel HEX file is an ASCII description of a binary file. Each line in the file contains a HEX record.

The file must end with an EOF record of the form:

:00000001FF

The file can contain any number of HEX records where each record consists of five fields arranged as follows:

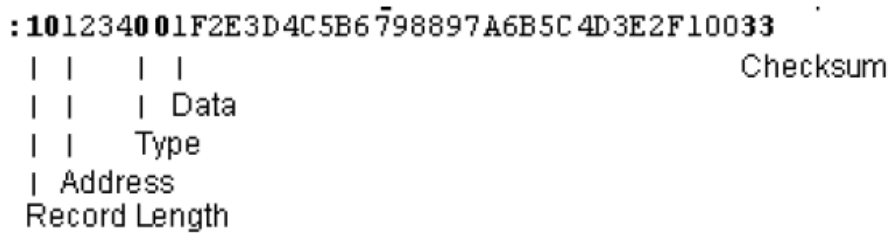
:LLAAAATT[DD...]CC

Each field consists of two or more hexadecimal digits:.

:	All records start with a colon.
LL	Length specifies the number of data bytes (DD) in the record.
AAAA	Starting address for the data in the record.
TT	Type of record: 00 - data 01 - end-of-file 02 - extended segment address (not supported) 04 - extended linear (32-bit) address
DD	Represents one byte of data. A record can contain the number of data bytes specified in the record length (LL) field.
CC	Checksum calculated by summing the values of all hex digit pairs in the record modulo 256 and taking the two's complement.

Data records are terminated with a carriage return and line feed.

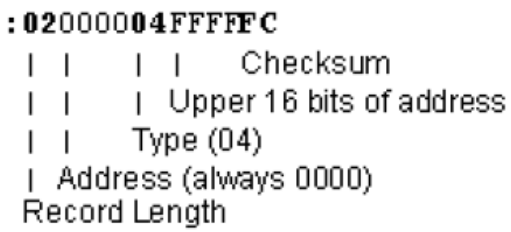
An example data record would be:



Extended linear address (32-bit) records contain the upper 16 bits (bits 16 to 31) of the data address and act as an offset for all subsequent data records.

The value remains in effect until changed by another extended linear address record.

The record always has 2 data bytes and is of the form:



Extended segment address records contain bits 4 to 9 of the data address and act as an offset for all subsequent data records. The value remains in effect until changed by another extended segment address record.

The extended segment address record always has 2 data bytes and is of the form:

```
:02000002 1200CD
 | |    | |    Checksum
 | |    | Bits 4-9 of address
 | |    Type (02)
 | Address (always 0000)
Record Length
```

Note

Extended segment address records are not supported by the ROM generator.

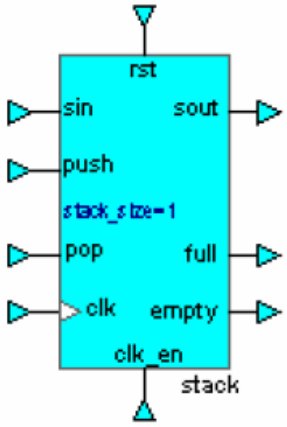
An Intel HEX file must end with an end-of-file (EOF) record. This record must have the value 01 in the record type field. An EOF record always appears as follows:

```
:00000001FF
 | |    | Checksum
 | |    Type (01)
 | Address (always 0000)
Record Length
```

Example HEX file:

```
:10001300AC12AD13AE10AF1112002F8E0E8F0F2244
:10000300E50B250DF509E50A350CF5081200132259
:03000000020023D8
:0C002300787FE4F6D8FD7581130200031D
:10002F00EFF88DF0A4FFEDC5F0CEA42EFEEC88F016
:04003F00A42EFE22CB
:00000001FF
```

Stack (stack)



This part implements a stack. Data can be loaded (pushed) into the internal registers and read (popped) at a later time. All operations in this part (except for reset) are synchronous with respect to the clock.

The depth is controlled by the parameter *stack_size*. The stack supports two input/output operations.

A push operation takes the value of the input port **sin** and places it in the internal registers.

A pop operation takes the newest value in the internal registers and places it in the output port **sout**. This value is retained on **sout**, until the next pop operation.

The pop operation is controlled by the scalar input port **pop**. The push operation is controlled by the scalar input port **push** and has precedence over the pop operation.

The polarity of the scalar input port **pop** is controlled by the enumerated parameter *pop_type* (ActiveHigh, ActiveLow). The polarity of the scalar output port **push** is controlled by the enumerated parameter *push_type* (ActiveHigh, ActiveLow).

The part consists of the required number of the internal registers, an address counter and the control logic. The address counter keeps count of the current address. This value is used to disable the pop and push operations. This counter decides whether the stack is empty (counter = 0) or full (counter = *stack_size*). The reset operation is required before a correct operation of the stack (for post synthesis simulation match). Pop operations cannot be done before a push operation.

Push operations are disabled if the internal registers are full (address counter = *stack_size*). Pop operations are disabled in the internal registers are empty (address counter = 0). If the internal registers are full and both push and pop operations are triggered, the pop operation takes place. Only when the internal registers are full does the pop operation takes precedence over the push (because write is disabled). Scalar output port **full** is activated if the internal registers are full. Scalar output port **empty** is activated if the internal registers are empty. The polarity of the scalar output port **full** is controlled by the enumerated parameter *full_type* (ActiveHigh, ActiveLow). The polarity of the scalar output port **empty** is controlled by the enumerated parameter *empty_type* (ActiveHigh, ActiveLow).

The part can be triggered on either the rising or falling edge of the scalar input port **clk**. The polarity of **clk** is controlled by the enumerated parameter *clk_type* (Rising, Falling, RisingLast, FallingLast, RisingEdge, FallingEdge).

(Note that RisingLast, FallingLast, RisingEdge and FallingEdge are supported for VHDL only.)

The **clk** can optionally be enabled by the scalar input port **clk_en**. The polarity of **clk_en** is controlled by the enumerated parameter *clk_en_type* (ActiveHigh, ActiveLow). Do not change the value of the parameter *clk_en_type*, if the **clk_en** port is not used.

The reset operations do not change the values stored inside the internal registers but reset the address counter to 0. After a reset operation, further read operations are disabled (until after the next write) because the LIFO is empty.

The mode of the reset is controlled by the enumerated parameter *rst_type* (SyncActiveHigh, AsyncActiveHigh, SyncActiveLow, AsyncActiveLow). Do not change the value of the parameter *rst_type* if the **rst** port is not used. If input ports **clk** and **rst** (for asynchronous behavior) are not enabled, the internal register retains its value.

This part is equivalent to a stack. This part has *n* * (*stack_size*+1) flip-flops, the address counter and the control logic (where *n* is the width of the output port **sout**). Not using certain operations (not connecting scalar output port **full** or not connecting scalar output port **empty**) affects the final gate count and the delay. Leave the clock enable port **clk_en** unconnected if it is not required. If **clk_en** is not connected, the HDL code for the unused port is optimized away.

The performance of the stack can be accurately estimated from the parameterized state.

Function

cnt is the internal address counter, *creg* is the internal register.

If reset is synchronous:

At every enabled **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

cnt	= 0	if **rst** is activated, else
	= *cnt* + 1	if scalar input port **push** is activated and count < *stack_size*, else
	= *cnt* - 1	if scalar input port **pop** is activated and count > 0, else
	= *cnt*	otherwise

If reset is asynchronous:

cnt = 0 if input port **rst** is active (irrespective of the **clk** trigger), else

at every enabled **clk** trigger (rising edge if *clk_type* = Rising; falling edge if *clk_type* = Falling):

cnt	= *cnt* + 1	if scalar input port **push** is activated and count < *stack_size*, else
	= *cnt* - 1	if scalar input port **pop** is activated and count > 0, else
	= *cnt*	otherwise

For a push operation (if *cnt* < *stack_size*; and push is enabled)

creg(1) = **sin**

sout = *creg*(0)

Truth Table

There are *stack_size* + 1 internal registers *creg* (each has a width of input **sin**). The address count is stored in the register *cnt*. This table is for asynchronous high reset and positive polarities. For negative polarities, invert the values.

Table 9-8. Stack Truth Table — Asynchronous High Reset, Positive Polarities

clk_en	clk	rst	push	pop	cnt	creg
-	-	1	-	-	0	creg
0	-	0	-	-	cnt	creg
1	Posedge	0	1	-	if (cnt < stack_size) cnt+1	if (cnt < stack_size) creg[1] = sin
1	Posedge	0	0	1	if (cnt not equal to 0) cnt -1	if (cnt not equal to 0) creg[cnt] = creg[cnt-1]

This table is for synchronous high reset and positive polarities. For negative polarities, invert the values.

Table 9-9. Stack Truth Table — Synchronous High Reset, Positive Polarities

clk_en	clk	rst	push	pop	cnt	creg
0	-	-	-	-	cnt	creg
1	Posedge	1	-	-	0	creg-
1	Posedge	0	1	-	if (cnt < stack_size) cnt+1	if (cnt < stack_size) creg[1] = sin
1	Posedge	0	0	1	if (cnt not equal to 0) cnt -1	if (cnt not equal to 0) creg[cnt] = reg[cnt-1]

sout = *creg*[0]

empty = (*cnt* = 0)

full = (*cnt* = *stack_size*)

Parameters

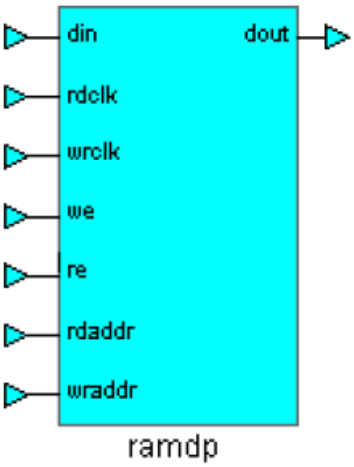
Table 9-10. Stack Parameters

Parameter	Values	Default
sin, sout	Port widths (must be > 0)	Automatic
clk_en_type	ActiveHigh,ActiveLow	ActiveHigh
clk_type	Rising,Falling,RisingLast,FallingLast,RisingEdge,FallingEdge	Rising
empty_type	ActiveHigh,ActiveLow	ActiveHigh
full_type	ActiveHigh,ActiveLow	ActiveHigh
pop_type	ActiveHigh,ActiveLow	ActiveHigh
push_type	ActiveHigh,ActiveLow	ActiveHigh
rst_type	SyncActiveHigh,AsyncActiveHigh,SyncActiveLow,AsyncActiveLow	AsyncActiveHigh
stack_size	Size of the FIFO (maximum number of data elements)	1

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if any of ports **clk**, **clk_en**, **rst**, **pop**, **push**, **full** and **empty** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any of ports **rst**, **push**, **pop**, **sin**, **clk** or **sout** are not connected.

Synthesizable Dual-Port RAM (ramdp)



This part implements a synthesizable dual-port Random Access Memory (RAM) which will infer a FIFO style dual port RAM with separate read **din** and write **dout** data ports synchronized to separate clocks

This part has independent input and output synchronous ports.

The size of the RAM is specified by the *address_width* and *data_width* parameters.

The width of address ports **rdaddr** and **wraddr** must be equal to the *address_width* parameter.

However, if the *address_width* parameter is empty, then the size of the RAM is determined from the width of the **rdaddr** or **wraddr** ports.

The width of input port **din** must be equal to or greater than the *data_width* parameter. If the *data_width* parameter is empty, then the size of the RAM is determined from the width of the **din** port.

The RAM can be initialized using the *initial_value* parameter which must be a valid LNBF string. All locations in the RAM will have the same initial value defined by the *initial_value* parameter.

The RAM has write and read operations. A write operation takes place if **wrclk** goes high and the optional scalar input port **we** is enabled (**we** = 1). A read operation takes place if **rdclk** goes high and the optional scalar input port **re** is enabled (**re** = 1). If the optional re and we ports are not used the write and read operations are controlled by solely the **wrclk** and **rdclk** ports.

In a write operation, the value of the input bus **din** is written to the address location specified by the value of the input bus **wraddr**. In a read operation, the value in the address location specified by **rdaddr** is placed on the output port **dout**.

The enumerated parameters *wrclk_type* and *rdclk_type* (Rising, Falling, RisingLast, FallingLast, RisingEdge, FallingEdge) and *we_type* (ActiveHigh, ActiveLow) control which clock edge is used for the read and write operations.

(Note that RisingLast, FallingLast, RisingEdge and FallingEdge are supported for VHDL only.)

This part can be synthesized using Altera or Xilinx technologies.

Parameters

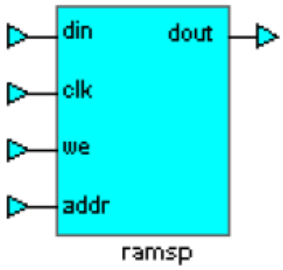
Table 9-11. Synthesizable Dual-Port RAM Parameters

Parameter	Values	Default
din, dout	Port widths (must be > 0)	Automatic
rdaddr, wraddr	Port widths (must be > 0)	Automatic
rdclk_type	Rising,Falling,RisingLast,FallingLast,RisingEdge,FallingEdge	Rising
re_type	ActiveHigh, ActiveLow	ActiveHigh
we_type	ActiveHigh, ActiveLow	ActiveHigh
wrclk_type	Rising,Falling,RisingLast,FallingLast,RisingEdge,FallingEdge	Rising
address_width	Width of the addressable memory	3
data_width	Width of the RAM data	8
initial_value	Initial value of RAM	0

Design Rule Checks

- An error is issued if the width of any port cannot be determined, if any of ports **rdclk**, **wrclk**, **re** and **we** do not have a fixed width of 1, if ports **din** and **dout** do not have the same width, if the width of ports **rdaddr** or **wraddr** do not equal the value of the *address_width* parameter or if the width of input port **din** is less than the *data_width* parameter.
- A warning is issued and HDL generation fails for this part if any of ports **din**, **dout**, **rdaddr**, **wraddr**, **rdclk** or **wrclk** are not connected.
- A warning is issued and generation continues if the width of input port **din** is greater than the *data_width* parameter.

Synthesizable Single-Port RAM (ramsp)



This part implements a synthesizable single-port Random Access Memory (RAM) with separate read **din** and write **dout** data ports synchronized to the same clock

The size of the RAM is specified by the *address_width* and *data_width* parameters.

The width of address port **addr** must be equal to the *address_width* parameter. If the *address_width* parameter is empty, then the size of the RAM is determined from the width of **addr** port.

The width of input port **din** must be equal to or greater than the *data_width* parameter. If the *data_width* parameter is empty, then the size of the RAM is determined from the width of the **din** port.

The RAM can be initialized using the *initial_value* parameter which must be a valid LNBF string. All locations of the RAM will have the same initial value defined in the *initial_value* parameter.

The RAM has write and read operations. The write operation takes place if the scalar input port **we** is enabled (**we** = 1). The read operation takes place if the scalar input port **we** is disabled (**we** = 0). In a write operation, the value of the input bus **din** is written to the address location specified by the value of the input bus **addr**. In a read operation, the value in the address location specified by **addr** is placed on the output port **dout**.

The enumerated parameters *clk_type* (Rising, Falling, RisingLast, FallingLast, RisingEdge, FallingEdge) and *we_type* (ActiveHigh, ActiveLow) control which clock edge is used for the read and write operations.

(Note that RisingLast, FallingLast, RisingEdge and FallingEdge are supported for VHDL only.)

This part can be synthesized using Altera or Xilinx technologies.

Parameters

Table 9-12. Synthesizable Single-Port RAM Parameters

Parameter	Values	Default
din, dout	Port widths (must be > 0)	Automatic
addr	Port widths (must be > 0)	Automatic
clk_type	Rising,Falling,RisingLast,FallingLast,RisingEdge,FallingEdge	Rising
we_type	ActiveHigh, ActiveLow	ActiveHigh
address_width	Width of the addressable memory	3
data_width	Width of the RAM data	
initial_value	Initial value of RAM	0

Design Rule Checks

- An error is issued if the width of any port cannot be determined, if any of ports **clk** and **we** do not have a fixed width of 1, if ports **din** and **dout** do not have the same width, if the width

of the port **addr** is not equal to the *address_width* parameter or if the width of port **din** is less than the *data_width* parameter.

- A warning is issued and HDL generation fails for this part if any of ports **din**, **dout**, **addr**, **clk** or **we** are not connected.
- A warning is issued and generation continues if the width of input port **din** is greater than the *data_width* parameter.

Chapter 10
Primitive Parts

This chapter describes Verilog primitive parts which are not frequently used by designers. These parts should be used only when Verilog code generation is required. Although they can also be implemented in VHDL, these parts are less efficient than other standard parts.

Introduction

Designs at the logic level of abstraction describe a digital circuit in terms of primitive logic functions such as AND, NAND and OR etc. They allow for the nets interconnecting the logic functions to carry 0, 1, X and Z values.

The switch level of modelling provides a level of abstraction between the logic and analog-transistor level of abstraction which describes the interconnection of transmission gates (which are abstractions of individual MOS and CMOS transistors). The switch level transistors are modelled as being either on or off, conducting or not conducting, respectively.

The values carried by the interconnections are abstracted from the whole range of analog voltages or currents to a small number of discrete values. These values are referred to as signal *strengths*.

Switch level modelling allows for the strength of a driving gate and the size of the capacitor storing charge on a tri-stated register net to be modelled. This capability provides for more accurate simulation of the electrical properties of the transistors than would a logic simulation.

The primitives are supported in both Verilog and VHDL. In VHDL, the primitives are simulated in order to provide all the same functionality which they provide in Verilog. Hence, the code generation becomes slightly lengthy and complex. The following description is mainly based on the behavior and modelling done in Verilog for the primitives and explains their simulated behavior in VHDL to some extent.

A gate or switch declaration names a gate or switch type as well as specifying its output signal strengths and delays. It contains one or more gate instances. Gate instances include an instance name and a required terminal connection list. The terminal connection list specifies how the gate or switch connects to other components in the part. All the instances contained in a gate or switch declaration have the same output strengths and delays.

The drive strength specifications specify the strengths of the values on the output terminals of the instances in the gate declaration. Strengths are supported for the logic levels, 0 and 1. The strength specification for the logic 0 is controlled by an enumerated parameter called *strength0_type* and similarly by *strength1_type* for logic 1.

It is possible to specify the strength of the output signals for the following gate primitives:

Gate types that support driving strengths

and	buf	xor	pullup
nand	not	xnor	pulldown
or	notif0	bufif0	
nor	notif1	bufif1	

As stated earlier, the drive strength specification has two parts. A gate declaration must contain both the parts, or should contain no parts, with the exception of *pullup* and *pulldown* sources. One of the parts specifies the strength of a signal with a value of 1 and the other specifies the strength of a signal with a value of 0.

The strength1 or strength0 specification, which specifies the strength of an output signal with a value of 1 and 0 respectively uses one of the following keywords:

Table 10-1. Keywords of Drive Strength Specifications

Strength name	Strength level
None	
Supply0	4
Strong0	3
Pull0	2
Weak0	1
Highz0	0
Highz1	0
Weak1	1
Pull1	2
Strong1	3
Supply1	4
None	

The strength specification follows the gate type keyword and precedes any delay specification. The strength0 specification follows the strength1 specification. You should not enter the following strength combinations (HighZ0, HighZ1) and (HighZ1, HighZ0). An error would be generated if either of these combination is assigned for a particular gate instance.

In VHDL, these strength specifications are mapped to the IEEE-1164 standard, which consists of nine values (0, 1, L, H, W, U, X, - and Z).

This mapping is shown in the following table:

Table 10-2. Strength Specifications Mapped to IEEE-1164 Standard

Strength name	Logic level	Mapped level
None		
Supply0	0 L	0 0
Strong0	0 L	0 0
Pull0	0 L	0 0
Weak0	0 L	L L
Highz0	0 L	Z Z
Highz1	1 H	Z Z

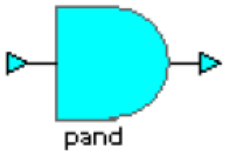
Table 10-2. Strength Specifications Mapped to IEEE-1164 Standard (cont.)

Strength name	Logic level	Mapped level
Weak1	1 H	H H
Pull1	1 H	1 1
Strong1	1 H	1 1
Supply1	1 H	1 1

The delay specifies the propagation delay through the gates and switches in a declaration. Three delay specifications are provided depending on the gate type. There are three enumerated parameters *rise_delay, fall_delay* and *turnoff_delay* which control the rise time, fall time and turnoff time respectively of a particular gate instance. All the delay values are user specified and should be entered as integers. The default value for all delays is assumed to be 0. *pullup* and *pulldown* source declarations do not include the delay specifications.

All the gate instances include an identifier, which is the name assigned depending upon the instance of a particular gate. The name is useful in tracing the operation of the overall system. All primitives support the nine-value standard and do not have an option for setting the coding style to four-value as is provided for many of the other ModuleWare parts.

AND Primitive (pand)



This is a highly parameterized AND primitive which has a user-specified number of inputs and a scalar output.

The rise time is controlled by an enumerated parameter *rise_delay* and the fall time is controlled by an enumerated parameter *fall_delay*. The smaller of the two delays applies on output transitions to X. If there is no delay specification, there is no propagation delay through the gate. The width of the port **din** determines the number of inputs for this primitive.

The strength level can be controlled by the enumerated parameters *strength0_type* and *strength1_type*. If *strength0_type* is not specified where *strength1_type* is specified, *strength0_type* is taken as Strong0. Whereas if *strength1_type* is not specified where *strength0_type* is specified, *strength1_type* is taken as Strong1.

This part is equivalent to *n* AND gates, where *n* is the width of the port.

Function

dout = **din0** AND **din1** AND **din2** AND…

Truth Table

Table 10-3. AND Primitive Truth Table

pand	**0**	**1**	**X**	**Z**
0	0	0	0	0
1	0	1	X	X
X	0	X	X	X
Z	0	X	X	X

Parameters

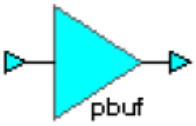
Table 10-4. AND Primitive Parameters

Parameter	**Values**	**Default**
din	Number of inputs (must be >= 1)	Automatic
fall_delay rise_delay strength0_type strength1_type	Integer value of the fall time for the gate Integer value of the rise time for the gate None, Supply0, Strong0, Pull0, Weak0, HighZ0 None, Supply1, Strong1, Pull1, Weak1, HighZ1	0 0 None None

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if both strength types are set to HighZ.
- A warning is issued and HDL generation fails for this part if any of ports are not connected.

Buffer Primitive (pbuf)



This is a highly parameterized buffer primitive which has a user-specified number of outputs and a scalar input.

The rise time is controlled by an enumerated parameter *rise_delay* and the fall time is controlled by an enumerated parameter *fall_delay*. The smaller of the two delays applies on output transitions to X. If there is no delay specification, there is no propagation delay through the gate. The width of the port **dout** determines the number of outputs for this primitive.

The strength level can be controlled by the enumerated parameters *strength0_type* and *strength1_type*. If *strength0_type* is not specified where *strength1_type* is specified, *strength0_type* is taken as Strong0. Whereas if *strength1_type* is not specified where *strength0_type* is specified, *strength1_type* is taken as Strong1.

This part is equivalent to *n* buffers, where *n* is the width of the port.

Function

dout0, **dout1**, **dout2**, … = buffered **din**

Truth Table

Table 10-5. Buffer Primitive Truth Table

pbuf	
Input	Output(s)
0	0
1	1
X	X
Z	X

Parameters

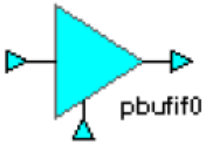
Table 10-6. Buffer Primitive Parameters

Parameter	**Values**	**Default**
din	Number of inputs (must be >= 1)	Automatic
fall_delay rise_delay strength0_type strength1_type	Integer value of the fall time for the gate Integer value of the rise time for the gate None, Supply0, Strong0, Pull0, Weak0, HighZ0 None, Supply1, Strong1, Pull1, Weak1, HighZ1	0 0 None None

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if both strength types are set to HighZ.
- A warning is issued and HDL generation fails for this part if any of ports are not connected.

Bufif0 Primitive (pbufif0)



This is a highly parameterized *bufif0* primitive which has a scalar input and a scalar output. The functionality is controlled by a scalar input **ena**.

The rise, fall and turnoff delays can be specified by separate enumerated parameters *rise_delay*, *fall_delay* and *turnoff_delay* respectively.

If there is no delay specification, there is no propagation delay through the gate.

If *turnoff_delay* is not specified, the smallest of the rise and fall delays applies on output transitions to X and Z. If *turnoff_delay* is specified, it determines the delay of transitions to Z, and the smallest of the three delays determines the delay of transitions to X.

The strength level can be controlled by the enumerated parameters *strength0_type* and *strength1_type*. If *strength0_type* is not specified where *strength1_type* is specified, *strength0_type* is taken as Strong0. Whereas if *strength1_type* is not specified where *strength0_type* is specified, *strength1_type* is taken as Strong1.

Some combinations of data input values and control input values cause these gates to output one of the two values without a preference for either value. The logic table for this gate includes two values representing such unknown results. The value L is a result representative of a value of 0 or Z. The value H is a result representative of a value of 1 or Z.

Function

dout = buffered **din** if **ena** is logic 0

Truth Table

Table 10-7. Bufif0 Primitive Truth Table

pbufif0		**Control**			
		0	**1**	**X**	**Z**
D	**0**	0	Z	L	L
A	1	1	Z	H	H
T	X	X	Z	X	X
A	Z	X	Z	X	X

Parameters

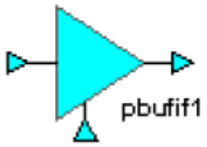
Table 10-8. Bufif0 Primitive Parameters

Parameter	Values	Default
fall_delay	Integer value of the fall time for the gate	0
rise_delay	Integer value of the rise time for the gate	0
strength0_type	None, Supply0, Strong0, Pull0, Weak0, HighZ0	None
strength1_type	None, Supply1, Strong1, Pull1, Weak1, HighZ1	None
turnoff_delay	Integer value of the turnoff time for the gate	0

Design Rule Checks

- An error is issued if the width of any port cannot be determined, if ports **din**, **dout** and **ena** do not have a fixed width of 1 or if both strength types are set to HighZ.
- A warning is issued and HDL generation fails for this part if any of ports are not connected.

Bufif1 Primitive (pbufif1)



This is a highly parameterized *bufif1* primitive which has a scalar input and a scalar output. The functionality is controlled by a scalar input **ena**.

The rise, fall and turnoff delays can be specified by separate enumerated parameters *rise_delay*, *fall_delay* and *turnoff_delay* respectively.

If there is no delay specification, there is no propagation delay through the gate.

If *turnoff_delay* is not specified, the smallest of the rise and fall delays applies on output transitions to X and Z. If *turnoff_delay* is specified, it determines the delay of transitions to Z, and the smallest of the three delays determines the delay of transitions to X.

The strength level can be controlled by the enumerated parameters *strength0_type* and *strength1_type*. If *strength0_type* is not specified where *strength1_type* is specified, *strength0_type* is taken as Strong0. Whereas if *strength1_type* is not specified where *strength0_type* is specified, *strength1_type* is taken as Strong1.

Some combinations of data input values and control input values cause these gates to output one of the two values without a preference for either value. The logic table for this gate includes two values representing such unknown results. The value L is a result representative of a value of 0 or Z. The value H is a result representative of a value of 1 or Z.

Function

dout = buffered **din** if **ena** is logic 1

Truth Table

Table 10-9. Bufif1 Primitive Truth Table

pbufif1		**Control**			
		0	**1**	**X**	**Z**
D	**0**	Z	0	L	L
A	1	Z	1	H	H
T	X	Z	X	X	X
A	Z	Z	X	X	X

Parameters

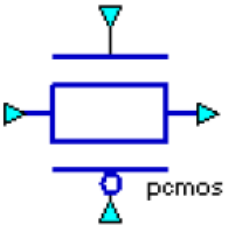
Table 10-10. Bufif1 Primitive Parameters

Parameter	Values	Default
fall_delay	Integer value of the fall time for the gate	0
rise_delay	Integer value of the rise time for the gate	0
strength0_type	None, Supply0, Strong0, Pull0, Weak0, HighZ0	None
strength1_type	None, Supply1, Strong1, Pull1, Weak1, HighZ1	None
turnoff_delay	Integer value of the turnoff time for the gate	0

Design Rule Checks

- An error is issued if the width of any port cannot be determined, if ports **din**, **dout** and **ena** do not have a fixed width of 1 or if both strength types are set to HighZ.
- A warning is issued and HDL generation fails for this part if any of ports are not connected.

CMOS Primitive (pcmos)



This is a highly parameterized *cmos* primitive which has a scalar input, a scalar output and two control inputs.

The rise, fall and turnoff delays can be specified by separate enumerated parameters *rise_delay*, *fall_delay* and *turnoff_delay* respectively.

If there is no delay specification, there is no propagation delay through the gate.

If *turnoff_delay* is not specified, the smallest of the rise and fall delays applies on output transitions to X and Z. If *turnoff_delay* is specified, it determines the delay of transitions to Z, and the smallest of the three delays determines the delay of transitions to X.

Some combinations of data input values and control input values cause these gates to output one of the two values without a preference for either value. The logic table for this gate includes two values representing such unknown results. The value L is a result representative of a value of 0 or Z. The value H is a result representative of a value of 1 or Z.

Function

dout = **din** generated according to the following truth tables

Truth Table

Table 10-11. CMOS Primitive Truth Table

pcmos		**Control (pena)**			
		0	**1**	**X**	**Z**
D	**0**	0	Z	L	L
A	1	1	Z	H	H
T	X	X	Z	X	X
A	Z	Z	Z	Z	Z

Table 10-12. CMOS Primitive Truth Table

pcmos		**Control (nena)**			
		0	**1**	**X**	**Z**
D	**0**	Z	0	L	L
A	1	Z	1	H	H
T	X	Z	X	X	X
A	Z	Z	Z	Z	Z

Parameters

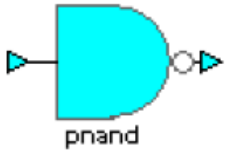
Table 10-13. CMOS Primitive Parameters

Parameter	Values	Default
fall_delay	Integer value of the fall time for the gate	0
rise_delay	Integer value of the rise time for the gate	0
turnoff_delay	Integer value of the turnoff time for the gate	0

Design Rule Checks

- An error is issued if the width of any port cannot be determined, if ports **din**, **dout**, **pena** and **nena** do not have a fixed width of 1 or if both strength types are set to HighZ.
- A warning is issued and HDL generation fails for this part if any of ports are not connected.

NAND Primitive (pnand)



This is a highly parameterized NAND primitive which has a user-specified number of inputs and a scalar output.

The rise time is controlled by an enumerated parameter *rise_delay* and the fall time is controlled by an enumerated parameter *fall_delay*. The smaller of the two delays applies on output transitions to X. If there is no delay specification, there is no propagation delay through the gate. The width of the port **din** determines the number of inputs for this primitive.

The strength level can be controlled by the enumerated parameters *strength0_type* and *strength1_type*. If *strength0_type* is not specified where *strength1_type* is specified, *strength0_type* is taken as Strong0. Whereas if *strength1_type* is not specified where *strength0_type* is specified, *strength1_type* is taken as Strong1.

This part is equivalent to *n* NAND gates, where *n* is the width of the port.

Function

dout = NOT (**din0** AND **din1** AND **din2** AND …

Truth Table

Table 10-14. NAND Primitive Truth Table

pnand	0	1	X	Z
0	1	1	1	1
1	1	0	X	X
X	1	X	X	X
Z	1	X	X	X

Parameters

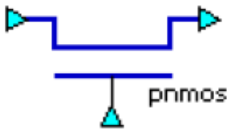
Table 10-15. NAND Primitive Parameters

Parameter	Values	Default
din	Number of inputs (must be >= 1)	Automatic
fall_delay rise_delay strength0_type strength1_type	Integer value of the fall time for the gate Integer value of the rise time for the gate None, Supply0, Strong0, Pull0, Weak0, HighZ0 None, Supply1, Strong1, Pull1, Weak1, HighZ1	0 0 None None

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if both strength types are set to HighZ.
- A warning is issued and HDL generation fails for this part if any of ports are not connected.

NMOS Primitive (pnmos)



This is a highly parameterized *nmos* primitive which has a scalar input, a scalar output and a control input.

The rise, fall and turnoff delays can be specified by separate enumerated parameters *rise_delay*, *fall_delay* and *turnoff_delay* respectively. If there is no delay specification, there is no propagation delay through the gate.

If *turnoff_delay* is not specified, the smallest of the rise and fall delays applies on output transitions to X and Z. If *turnoff_delay* is specified, it determines the delay of transitions to Z, and the smallest of the three delays determines the delay of transitions to X.

Some combinations of data input values and control input values cause these gates to output one of the two values without a preference for either value. The logic table for this gate includes two values representing such unknown results. The value L is a result representative of a value of 0 or Z. The value H is a result representative of a value of 1 or Z.

Function

dout = **din** generated according to the following truth table

Truth Table

Table 10-16. NMOS Primitive Truth Table

pnmos		**Control**			
		0	**1**	**X**	**Z**
D	**0**	Z	0	L	L
A	1	Z	1	H	H
T	X	Z	X	X	X
A	Z	Z	Z	Z	Z

Parameters

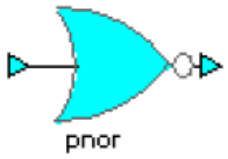
Table 10-17. NMOS Primitive Parameters

Parameter	**Values**	**Default**
fall_delay	Integer value of the fall time for the gate	0
rise_delay	Integer value of the rise time for the gate	0
turnoff_delay	Integer value of the turnoff time for the gate	0

Design Rule Checks

- An error is issued if the width of any port cannot be determined, if ports **din**, **dout** and **ena** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any of ports are not connected.

NOR Primitive (pnor)



This is a highly parameterized NOR primitive which has a user-specified number of inputs and a scalar output.

The rise time is controlled by an enumerated parameter *rise_delay* and the fall time is controlled by an enumerated parameter *fall_delay*. The smaller of the two delays applies on output transitions to X. If there is no delay specification, there is no propagation delay through the gate. The width of the port **din** determines the number of inputs for this primitive.

The strength level can be controlled by the enumerated parameters *strength0_type* and *strength1_type*. If *strength0_type* is not specified where *strength1_type* is specified, *strength0_type* is taken as Strong0. Whereas if *strength1_type* is not specified where *strength0_type* is specified, *strength1_type* is taken as Strong1.

This part is equivalent to *n* NOR gates, where *n* is the width of the port.

Function

dout = NOT (**din0** OR **din1** OR **din2** OR …

Truth Table

Table 10-18. NOR Primitive Truth Table

pnor	0	1	X	Z
0	1	0	X	X
1	0	0	0	0
X	X	0	X	X
Z	X	0	X	X

Parameters

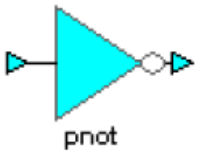
Table 10-19. NOR Primitive Parameters

Parameter	Values	Default
din	Number of inputs (must be >= 1)	Automatic
fall_delay rise_delay strength0_type strength1_type	Integer value of the fall time for the gate Integer value of the rise time for the gate None, Supply0, Strong0, Pull0, Weak0, HighZ0 None, Supply1, Strong1, Pull1, Weak1, HighZ1	0 0 None None

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if both strength types are set to HighZ.
- A warning is issued and HDL generation fails for this part if any of ports are not connected.

NOT Primitive (pnot)



This is a highly parameterized NOT primitive which has a user-specified number of outputs and a scalar input.

The rise time is controlled by an enumerated parameter *rise_delay* and the fall time is controlled by an enumerated parameter *fall_delay*. The smaller of the two delays applies on output transitions to X. If there is no delay specification, there is no propagation delay through the gate. The width of the port **dout** determines the number of outputs for this primitive.

The strength level can be controlled by the enumerated parameters *strength0_type* and *strength1_type*. If *strength0_type* is not specified where *strength1_type* is specified, *strength0_type* is taken as Strong0. Whereas if *strength1_type* is not specified where *strength0_type* is specified, *strength1_type* is taken as Strong1.

This part is equivalent to *n* NOT gates, where *n* is the width of the port.

Function

dout0, **dout1**, **dout2**, … = NOT of **din**

Truth Table

Table 10-20. NOT Primitive Truth Table

pnot	
Input	Output(s)
0	1
1	0
X	X
Z	X

Parameters

Table 10-21. NOT Primitive Parameters

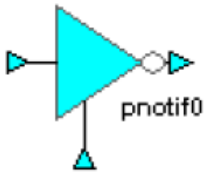
Parameter	**Values**	**Default**
dout	Number of inputs (must be >= 1)	Automatic
fall_delay rise_delay strength0_type strength1_type	Integer value of the fall time for the gate Integer value of the rise time for the gate None, Supply0, Strong0, Pull0, Weak0, HighZ0 None, Supply1, Strong1, Pull1, Weak1, HighZ1	0 0 None None

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if both strength types are set to HighZ.
- A warning is issued and HDL generation fails for this part if any of ports are not connected.

Notif0 Primitive (pnotif0)

This is a highly parameterized *notif0* primitive which has a scalar input and a scalar output. The functionality is controlled by a scalar input *ena*.



The rise, fall and turnoff delays can be specified by separate enumerated parameters *rise_delay*, *fall_delay* and *turnoff_delay* respectively.

If there is no delay specification, there is no propagation delay through the gate.

If *turnoff_delay* is not specified, the smallest of the rise and fall delays applies on output transitions to X and Z. If *turnoff_delay* is specified, it determines the delay of transitions to Z, and the smallest of the three delays determines the delay of transitions to X.

The strength level can be controlled by the enumerated parameters *strength0_type* and *strength1_type*. If *strength0_type* is not specified where *strength1_type* is specified, *strength0_type* is taken as Strong0. Whereas if *strength1_type* is not specified where *strength0_type* is specified, *strength1_type* is taken as Strong1.

Some combinations of data input values and control input values cause these gates to output one of the two values without a preference for either value. The logic table for this gate includes two values representing such unknown results. The value L is a result representative of a value of 0 or Z. The value H is a result representative of a value of 1 or Z.

Function

dout = NOT of **din** if **ena** is logic 0

Truth Table

Table 10-22. Notif0 Primitive Truth Table

pnotif0		**Control**			
		0	**1**	**X**	**Z**
D	**0**	1	Z	H	H
A	1	0	Z	L	L
T	X	X	Z	X	X
A	Z	X	Z	X	X

Parameters

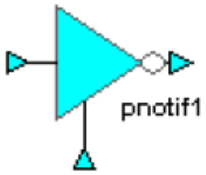
Table 10-23. Notif0 Primitive Parameters

Parameter	Values	Default
fall_delay	Integer value of the fall time for the gate	0
rise_delay	Integer value of the rise time for the gate	0
strength0_type	None, Supply0, Strong0, Pull0, Weak0, HighZ0	None
strength1_type	None, Supply1, Strong1, Pull1, Weak1, HighZ1	None
turnoff_delay	Integer value of the turnoff time for the gate	0

Design Rule Checks

- An error is issued if the width of any port cannot be determined, if ports **din**, **dout** and **ena** do not have a fixed width of 1 or if both strength types are set to HighZ.
- A warning is issued and HDL generation fails for this part if any of ports are not connected.

Notif1 Primitive (pnotif1)



This is a highly parameterized *notif1* primitive which has a scalar input and a scalar output. The functionality is controlled by a scalar input *ena*.

The rise, fall and turnoff delays can be specified by separate enumerated parameters *rise_delay*, *fall_delay* and *turnoff_delay* respectively.

If there is no delay specification, there is no propagation delay through the gate.

If *turnoff_delay* is not specified, the smallest of the rise and fall delays applies on output transitions to X and Z. If *turnoff_delay* is specified, it determines the delay of transitions to Z, and the smallest of the three delays determines the delay of transitions to X.

The strength level can be controlled by the enumerated parameters *strength0_type* and *strength1_type*. If *strength0_type* is not specified where *strength1_type* is specified, *strength0_type* is taken as Strong0. Whereas if *strength1_type* is not specified where *strength0_type* is specified, *strength1_type* is taken as Strong1.

Some combinations of data input values and control input values cause these gates to output one of the two values without a preference for either value. The logic table for this gate includes two values representing such unknown results. The value L is a result representative of a value of 0 or Z. The value H is a result representative of a value of 1 or Z.

Function

dout = NOT of **din** if **ena** is logic 1

Truth Table

Table 10-24. Notif1 Primitive Truth Table

pnotif1		**Control**			
		0	**1**	**X**	**Z**
D	**0**	Z	1	H	H
A	1	Z	0	L	L
T	X	Z	X	X	X
A	Z	Z	X	X	X

Parameters

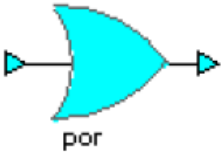
Table 10-25. Notif1 Primitive Parameters

Parameter	Values	Default
fall_delay	Integer value of the fall time for the gate	0
rise_delay	Integer value of the rise time for the gate	0
strength0_type	None, Supply0, Strong0, Pull0, Weak0, HighZ0	None
strength1_type	None, Supply1, Strong1, Pull1, Weak1, HighZ1	None
turnoff_delay	Integer value of the turnoff time for the gate	0

Design Rule Checks

- An error is issued if the width of any port cannot be determined, if ports **din**, **dout** and **ena** do not have a fixed width of 1 or if both strength types are set to HighZ.
- A warning is issued and HDL generation fails for this part if any of ports are not connected.

OR Primitive (por)



This is a highly parameterized OR primitive which has a user-specified number of inputs and a scalar output.

The rise time is controlled by an enumerated parameter *rise_delay* and the fall time is controlled by an enumerated parameter *fall_delay*. The smaller of the two delays applies on output transitions to X. If there is no delay specification, there is no propagation delay through the gate. The width of the port **din** determines the number of inputs for this primitive.

The strength level can be controlled by the enumerated parameters *strength0_type* and *strength1_type*. If *strength0_type* is not specified where *strength1_type* is specified, *strength0_type* is taken as Strong0. Whereas if *strength1_type* is not specified where *strength0_type* is specified, *strength1_type* is taken as Strong1.

This part is equivalent to *n* OR gates, where *n* is the width of the port.

Function

dout = **din0** OR **din1** OR **din2** OR …

Truth Table

Table 10-26. OR Primitive Truth Table

por	**0**	**1**	**X**	**Z**
0	0	1	X	X
1	1	1	1	1
X	X	1	X	X
Z	X	1	X	X

Parameters

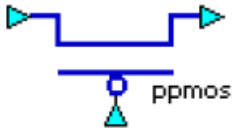
Table 10-27. OR Primitive Parameters

Parameter	**Values**	**Default**
din	Number of inputs (must be >= 1)	Automatic
fall_delay rise_delay strength0_type strength1_type	Integer value of the fall time for the gate Integer value of the rise time for the gate None, Supply0, Strong0, Pull0, Weak0, HighZ0 None, Supply1, Strong1, Pull1, Weak1, HighZ1	0 0 None None

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if both strength types are set to HighZ.
- A warning is issued and HDL generation fails for this part if any of ports are not connected.

PMOS Primitive (ppmos)



This is a highly parameterized *pmos* primitive which has a scalar input, a scalar output and a control input.

The rise, fall and turnoff delays can be specified by separate enumerated parameters *rise_delay*, *fall_delay* and *turnoff_delay* respectively. If there is no delay specification, there is no propagation delay through the gate.

If *turnoff_delay* is not specified, the smallest of the rise and fall delays applies on output transitions to X and Z. If *turnoff_delay* is specified, it determines the delay of transitions to Z, and the smallest of the three delays determines the delay of transitions to X.

Some combinations of data input values and control input values cause these gates to output one of the two values without a preference for either value. The logic table for this gate includes two values representing such unknown results. The value L is a result representative of a value of 0 or Z. The value H is a result representative of a value of 1 or Z.

Function

dout = **din** generated according to the following truth table.

Truth Table

Table 10-28. PMOS Primitive Truth Table

ppmos		**Control**			
		0	**1**	**X**	**Z**
D	**0**	0	Z	L	L
A	1	1	Z	H	H
T	X	X	Z	X	X
A	Z	Z	Z	Z	Z

Parameters

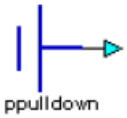
Table 10-29. PMOS Primitive Parameters

Parameter	**Values**	**Default**
fall_delay	Integer value of the fall time for the gate	0
rise_delay	Integer value of the rise time for the gate	0
turnoff_delay	Integer value of the turnoff time for the gate	0

Design Rule Checks

- An error is issued if the width of any port cannot be determined, if ports **din**, **dout** and **ena** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any of ports are not connected.

Pulldown Primitive (ppulldown)



This is a highly parameterized *pulldown* primitive which has a user-specified number of scalar inputs. The width of the input port **din** determines the number of inputs.

This part places a logic value of logic 0 on the inputs connected to its input port.

There are no delay specifications for this part because the signals this part places on the connected nets continue throughout simulation without any variation.

The strength level can be controlled by the enumerated parameter *strength0_type*.

Function

dout = 00000… depending upon the number of inputs

Parameters

Table 10-30. Pulldown Primitive Parameters

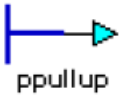
Parameter	Values	Default
din	Number of inputs (must be >= 1)	Automatic
strength0_type	None, Supply0, Strong0, Pull0, Weak0	None

Design Rule Checks

- An error is issued if the width of port **dout** cannot be determined or if both strength types are set to HighZ.
- A warning is issued and HDL generation fails for this part if port **dout** is not connected.

Pullup Primitive (ppullup)

This is a highly parameterized *pullup* primitive which has a user-specified number of scalar inputs. The width of the input port **din** determines the number of inputs.



This part places a logic value of logic 1 on the inputs connected to its input port.

There are no delay specifications for this part because the signals this part places on the connected nets continue throughout simulation without any variation.

The strength level can be controlled by the enumerated parameter *strength1_type*.

Function

dout = 11111… depending upon the number of inputs

Parameters

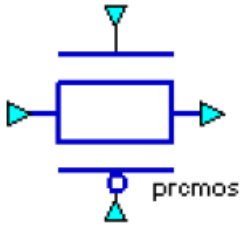
Table 10-31. Pullup Primitive Parameters

Parameter	Values	Default
din	Number of inputs (must be >= 1)	Automatic
strength1_type	None, Supply1, Strong1, Pull1, Weak1	None

Design Rule Checks

- An error is issued if the width of port **dout** cannot be determined or if both strength types are set to HighZ.
- A warning is issued and HDL generation fails for this part if port **dout** is not connected.

RCMOS Primitive (prcmos)



This is a highly parameterized *rcmos* primitive which has a scalar input, a scalar output and two control inputs. This has a significantly higher source to drain impedance during conduction as compared to normal *cmos*.

The rise, fall and turnoff delays can be specified by separate enumerated parameters *rise_delay*, *fall_delay* and *turnoff_delay* respectively.

If there is no delay specification, there is no propagation delay through the gate.

If *turnoff_delay* is not specified, the smallest of the rise and fall delays applies on output transitions to X and Z. If *turnoff_delay* is specified, it determines the delay of transitions to Z, and the smallest of the three delays determines the delay of transitions to X.

Some combinations of data input values and control input values cause these gates to output one of the two values without a preference for either value. The logic table for this gate includes two values representing such unknown results. The value L is a result representative of a value of 0 or Z. The value H is a result representative of a value of 1 or Z.

The generated VHDL for this part is exactly the same as for the CMOS Primitive (pcmos). However, the Verilog code is different.

Function

dout = **din** generated according to the following truth tables.

Truth Table

Table 10-32. RCMOS Primitive Truth Table

prcmos		Control (pena)			
		0	**1**	**X**	**Z**
D	**0**	0	Z	L	L
A	1	1	Z	H	H
T	X	X	Z	X	X
A	Z	Z	Z	Z	Z

Table 10-33. RCMOS Primitive Truth Table

prcmos		Control (nena)			
		0	**1**	**X**	**Z**
D	**0**	Z	0	L	L
A	1	Z	1	H	H
T	X	Z	X	X	X
A	Z	Z	Z	Z	Z

Parameters

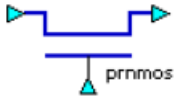
Table 10-34. RCMOS Primitive Parameters

Parameter	Values	Default
fall_delay	Integer value of the fall time for the gate	0
rise_delay	Integer value of the rise time for the gate	0
turnoff_delay	Integer value of the turnoff time for the gate	0

Design Rule Checks

- An error is issued if the width of any port cannot be determined, if ports **din**, **dout**, **pena** or **nena** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any of ports are not connected.

RNMOS Primitive (prnmos)



This is a highly parameterized *rnmos* primitive which has a scalar input, a scalar output and a control input. This has a significantly higher source to drain impedance during conduction as compared to normal *nmos*.

The rise, fall and turnoff delays can be specified by separate enumerated parameters *rise_delay*, *fall_delay* and *turnoff_delay* respectively. If there is no delay specification, there is no propagation delay through the gate.

If *turnoff_delay* is not specified, the smallest of the rise and fall delays applies on output transitions to X and Z. If *turnoff_delay* is specified, it determines the delay of transitions to Z, and the smallest of the three delays determines the delay of transitions to X.

Some combinations of data input values and control input values cause these gates to output one of the two values without a preference for either value. The logic table for this gate includes two values representing such unknown results. The value L is a result representative of a value of 0 or Z. The value H is a result representative of a value of 1 or Z.

Function

dout = **din** generated according to the following truth table

Truth Table

Table 10-35. RNMOS Primitive Truth Table

prnmos		**Control**			
		0	**1**	**X**	**Z**
D	**0**	Z	0	L	L
A	1	Z	1	H	H
T	X	Z	X	X	X
A	Z	Z	Z	Z	Z

Parameters

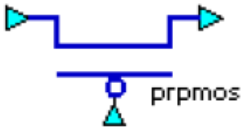
Table 10-36. RNMOS Primitive Parameters

Parameter	**Values**	**Default**
fall_delay	Integer value of the fall time for the gate	0
rise_delay	Integer value of the rise time for the gate	0
turnoff_delay	Integer value of the turnoff time for the gate	0

Design Rule Checks

- An error is issued if the width of any port cannot be determined, if ports **din**, **dout** and **ena** do not have a fixed width of 1.
- A warning is issued and HDL generation fails for this part if any of ports are not connected.

RPMOS Primitive (prpmos)



This is a highly parameterized *rpmos* primitive which has a scalar input, a scalar output and a control input. This has a significantly higher source to drain impedance during conduction as compared to normal *pmos*.

The rise, fall and turnoff delays can be specified by separate enumerated parameters *rise_delay*, *fall_delay* and *turnoff_delay* respectively. If there is no delay specification, there is no propagation delay through the gate.

If *turnoff_delay* is not specified, the smallest of the rise and fall delays applies on output transitions to X and Z. If *turnoff_delay* is specified, it determines the delay of transitions to Z, and the smallest of the three delays determines the delay of transitions to X.

Some combinations of data input values and control input values cause these gates to output one of the two values without a preference for either value. The logic table for this gate includes two values representing such unknown results. The value L is a result representative of a value of 0 or Z. The value H is a result representative of a value of 1 or Z.

Function

dout = **din** generated according to the following truth table.

Truth Table

Table 10-37. RPMOS Primitive Truth Table

prpmos		**Control**			
		0	**1**	**X**	**Z**
D	**0**	0	Z	L	L
A	1	1	Z	H	H
T	X	X	Z	X	X
A	Z	Z	Z	Z	Z

Parameters

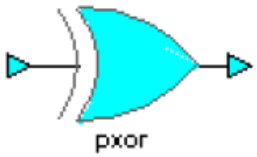
Table 10-38. RPMOS Primitive Parameters

Parameter	**Values**	**Default**
fall_delay	Integer value of the fall time for the gate	0
rise_delay	Integer value of the rise time for the gate	0
turnoff_delay	Integer value of the turnoff time for the gate	0

Design Rule Checks

- An error is issued if the width of any port cannot be determined, if ports **din**, **dout** and **ena** do not have a fixed width of 1 or if both strength types are set to HighZ.
- A warning is issued and HDL generation fails for this part if any of ports are not connected.

XOR Primitive (pxor)



This is a highly parameterized XOR primitive which has a user-specified number of inputs and a scalar output.

The rise time is controlled by an enumerated parameter *rise_delay* and the fall time is controlled by an enumerated parameter *fall_delay*. The smaller of the two delays applies on output transitions to X. If there is no delay specification, there is no propagation delay through the gate. The width of the port **din** determines the number of inputs for this primitive.

The strength level can be controlled by the enumerated parameters *strength0_type* and *strength1_type*. If *strength0_type* is not specified where *strength1_type* is specified, *strength0_type* is taken as Strong0. Whereas if *strength1_type* is not specified where *strength0_type* is specified, *strength1_type* is taken as Strong1.

This part is equivalent to *n* XOR gates, where *n* is the width of the port.

Function

dout = **din0** XOR **din1** XOR **din2** XOR …

Truth Table

Table 10-39. XOR Primitive Truth Table

pxor	0	1	X	Z
0	0	1	X	X
1	1	0	X	X
X	X	X	X	X
Z	X	X	X	X

Parameters

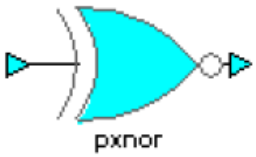
Table 10-40. XOR Primitive Parameters

Parameter	Values	Default
din	Number of inputs (must be >= 1)	Automatic
fall_delay rise_delay strength0_type strength1_type	Integer value of the fall time for the gate Integer value of the rise time for the gate None, Supply0, Strong0, Pull0, Weak0, HighZ0 None, Supply1, Strong1, Pull1, Weak1, HighZ1	0 0 None None

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if both strength types are set to HighZ.
- A warning is issued and HDL generation fails for this part if any of ports are not connected.

XNOR Primitive (pxnor)



This is a highly parameterized *xnor* primitive which has a user specified number of inputs and a scalar output.

The rise time is controlled by an enumerated parameter *rise_delay* and the fall time is controlled by an enumerated parameter *fall_delay*. The smaller of the two delays applies on output transitions to X. If there is no delay specification, there is no propagation delay through the gate. The width of the port **din** determines the number of inputs for this primitive.

The strength level can be controlled by the enumerated parameters *strength0_type* and *strength1_type*. If *strength0_type* is not specified where *strength1_type* is specified, *strength0_type* is taken as Strong0. Whereas if *strength1_type* is not specified where *strength0_type* is specified, *strength1_type* is taken as Strong1.

This part is equivalent to *n* XNOR gates, where *n* is the width of the port.

Function

dout = NOT (**din0** XOR **din1** XOR **din2** XOR …

Truth Table

Table 10-41. XNOR Primitive Truth Table

pxnor	0	1	X	Z
0	1	0	X	X
1	0	1	X	X
X	X	X	X	X
Z	X	X	X	X

Parameters

Table 10-42. XNOR Primitive Parameters

Parameter	Values	Default
din	Number of inputs (must be >= 1)	Automatic
fall_delay rise_delay strength0_type strength1_type	Integer value of the fall time for the gate Integer value of the rise time for the gate None, Supply0, Strong0, Pull0, Weak0, HighZ0 None, Supply1, Strong1, Pull1, Weak1, HighZ1	0 0 None None

Design Rule Checks

- An error is issued if the width of any port cannot be determined or if both strength types are set to HighZ.
- A warning is issued and HDL generation fails for this part if any of ports are not connected.

Chapter 11
Stimulus Parts

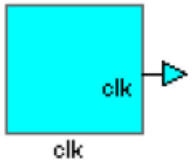
This chapter describes non-synthesizable stimulus generator parts for use in test benches used for design verification:

Note

Note that parameters for the Stimulus parts are entered using custom dialog boxes which are illustrated on the following pages. An error is reported if you attempt to enter an invalid value in the dialog box so that incorrect data is never passed to the HDL generators.

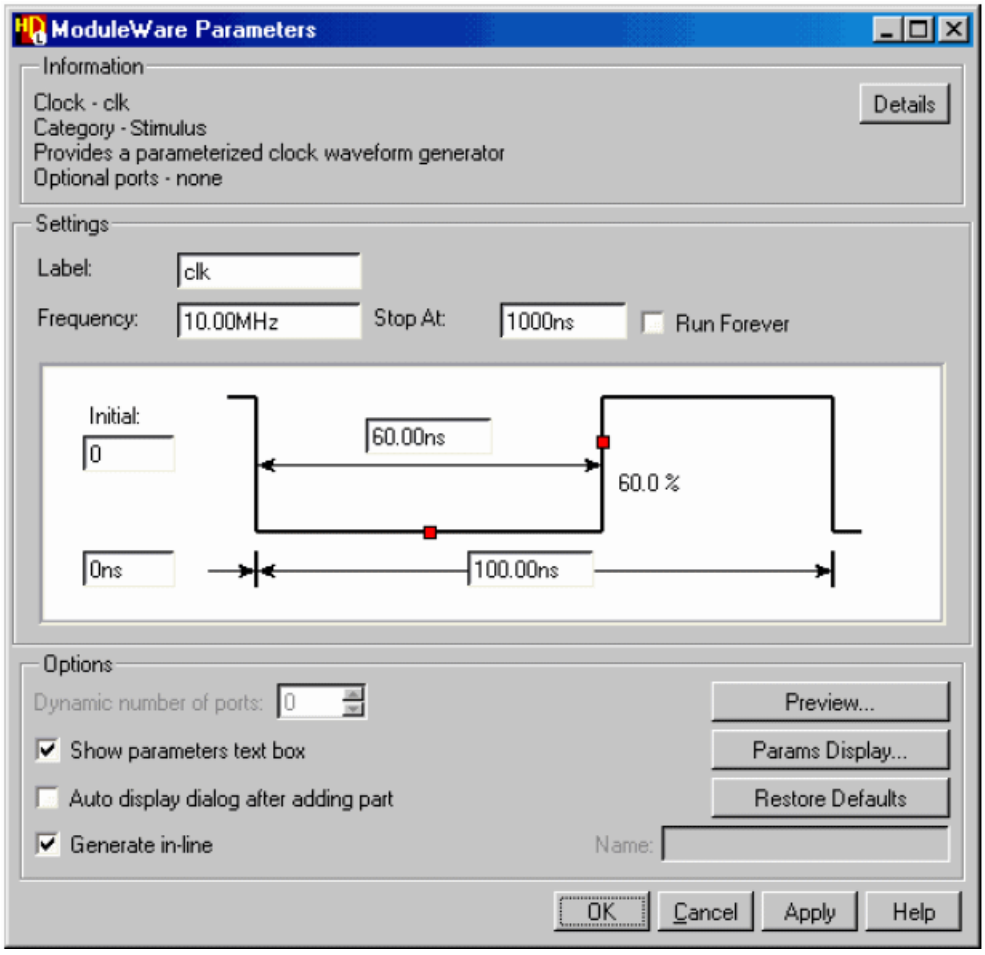
Simple Clock (clk)

This part generates a simple repeating scalar output clock waveform.



You can specify a label for the generated VHDL process (or Verilog *initial* block). If not specified, a unique label is generated from the part name (for example, *clk1*, *clk2*..*clkN*).

The clock frequency can be entered directly or by setting the clock period on the waveform display. (Whichever parameter is specified, the other is calculated and displayed.)



You can choose whether the clock stops at a specified time or runs forever. If the run forever option is unselected, a time must be specified.

You can specify the initial value of the clock (0, 1, X or Z) before the timing offset and the offset time (which must be less than 1 second) before the clock generator starts.

You can specify the cycle time by entering the required time on the first part of the clock waveform or by dynamically sliding the red “handle” on the vertical clock edge. The corresponding duty cycle (which cannot exceed 100%) is automatically calculated and shown on the waveform display.

The default clock has a rising edge. You can invert the waveform by double-clicking on the red "handle" on the horizontal waveform.

Time intervals can optionally be specified in milliseconds (ms), microseconds (us), nanoseconds (ns) or picoseconds (ps). Seconds are assumed if no unit is specified. Spaces are automatically used between the time value and unit in the generated HDL.

Frequencies can be specified in kilohertz (kHz), megahertz (MHz), gigahertz (GHz) or terahertz (THz). If no units are specified, hertz are assumed.

Parameters

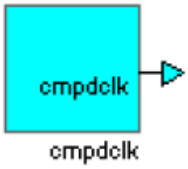
Table 11-1. Simple Clock Parameters

Parameter	Values	Default
Label	A unique HDL identifier	clk
Frequency	Real number (optionally in kHz, MHz, GHz or THz)	10.00MHz
Clock Period	Real number (optionally in ms, us, ns or ps)	100ns
Stop At	Real number (optionally in ms, us, ns or ps)	1000ns
Run Forever	On or Off	Off
Initial Value	0, 1, X or Z	0
Timing Offset	Real number (optionally in ms, us, ns or ps)	0ns
Cycle Time	Real number (optionally in ms, us, ns or ps)	50ns
Duty Cycle	Calculated from the cycle time	50%

Checks

- An entry error is issued if invalid time units are specified in the dialog box or if a parameter is outside its permitted range.

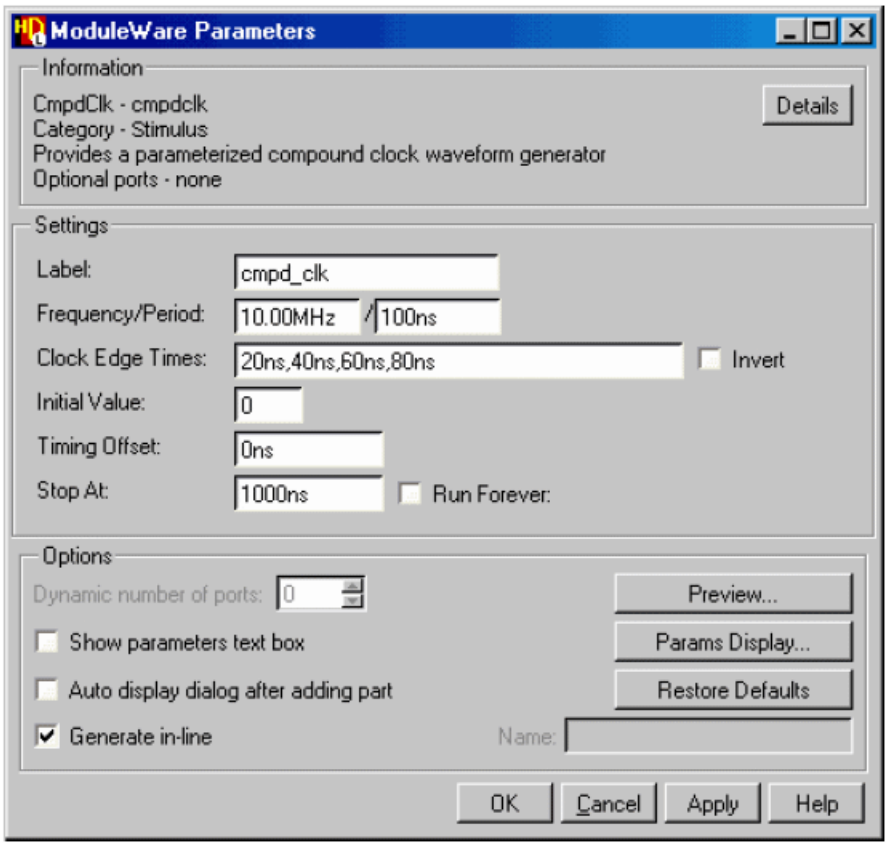
Compound Clock (cmpdclk)



This part generates a repeating waveform of multiple scalar output clock pulses

You can specify a label for the generated VHDL process (or Verilog *initial* block). If not specified, a unique label is generated from the part name (for example, *cmpd_clk1*, *cmpd_clk2*..*cmpd_clkN*).

You can specify the clock frequency or period. (Whichever parameter is specified, the other is calculated and displayed.)



You can specify any number of clock edge times by entering a comma separated list in ascending order. The specified edge times must be less than the clock period. The default clock has a rising edge but you can set the invert option to generate a falling edge clock.

You can specify the initial value of the clock (0, 1, X, or Z) before the timing offset and the offset time (which must be less than 1 second) before the clock generator starts.

You can choose whether the clock stops at a specified time or runs forever. If the run forever option is unselected, a time must be specified.

Time intervals can optionally be specified in milliseconds (ms), microseconds (us), nanoseconds (ns) or picoseconds (ps). Seconds are assumed if no unit is specified. Spaces are automatically used between the time value and unit in the generated HDL.

Frequencies can be specified in kilohertz (kHz), megahertz (MHz), gigahertz (GHz) or terahertz (THz). If no units are specified, hertz are assumed.

Parameters

Table 11-2. Compound Clock Parameters

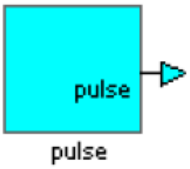
Parameter	Values	Default
Label	A unique HDL identifier	cmpd_clk
Frequency	Real number (optionally in kHz, MHz, GHz or THz)	10.00MHz
Clock Period	Real number (optionally in ms, us, ns or ps)	100ns
Clock Edge Times	List of real numbers (optionally in ms, us, ns or ps)	20ns,40ns,60ns,80ns
Invert	On or Off	Off
Initial	0, 1, X, Z	0
Timing Offset	Real number (optionally in ms, us, ns or ps)	0ns
Stop At	On or Off	On
Run Forever	Real number (optionally in ms, us, ns or ps)	1000ns

Checks

- An entry error is issued if invalid time units are specified or a parameter is outside its permitted range.

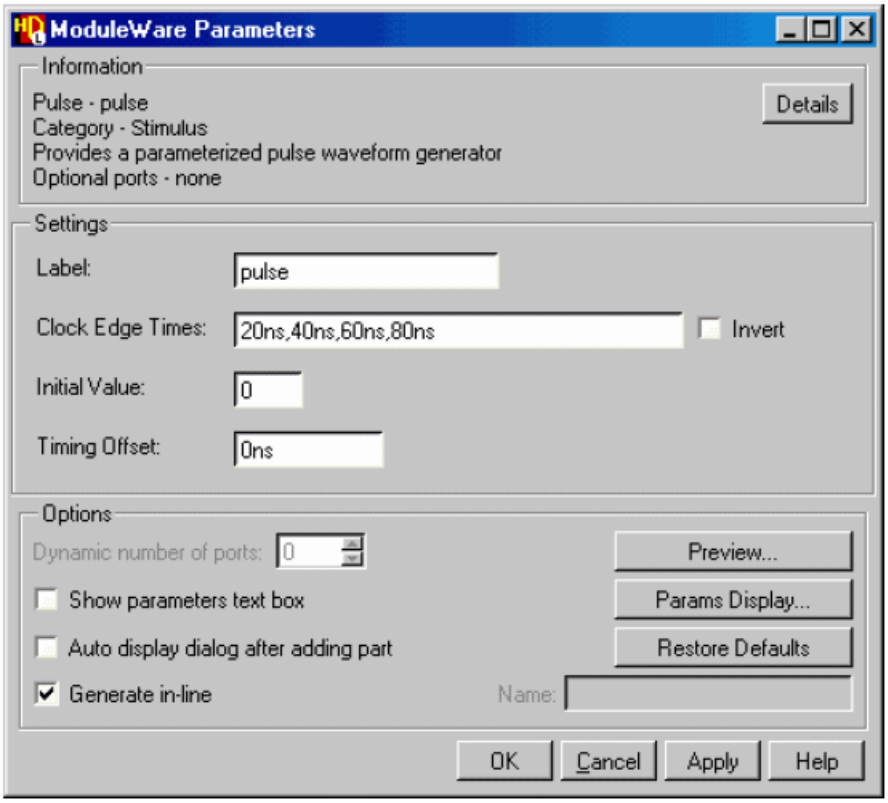
Pulse (pulse)

This part generates a simple scalar output pulse waveform or a pulse waveform with multiple clock edges.



You can specify a label for the generated VHDL process (or Verilog *initial* block). If not specified, a unique label is generated from the part name (for example, *pulse1*, *pulse2*..*pulseN*).

You can specify any number of clock edge times by entering a comma separated list in ascending order. The default pulses have a rising edge but you can set the invert option to generate falling edge pulses.



You can specify the initial value of the pulse (0, 1, X or Z) before the timing offset and the offset time (which must be less than 1 second) before the pulse generator starts.

Time intervals can optionally be specified in milliseconds (ms), microseconds (us), nanoseconds (ns) or picoseconds (ps). Seconds are assumed if no unit is specified. Spaces are automatically used between the time value and unit in the generated HDL.

Frequencies can be specified in kilohertz (kHz), megahertz (MHz), gigahertz (GHz) or terahertz (THz). If no units are specified, hertz are assumed.

Parameters

Table 11-3. Pulse Parameters

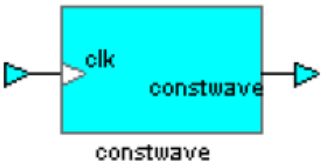
Parameter	Values	Default
Label	A unique HDL identifier	pulse
Clock Edge Times	List of real numbers (optionally in ms, us, ns or ps)	20ns,40ns,60ns,80ns
Invert	On or Off	Off
Initial	0, 1, X, Z	0
Timing Offset	Real number (optionally in ms, us, ns or ps)	0ns

Checks

- An entry error is issued if invalid time units are specified or a parameter is outside its permitted range.

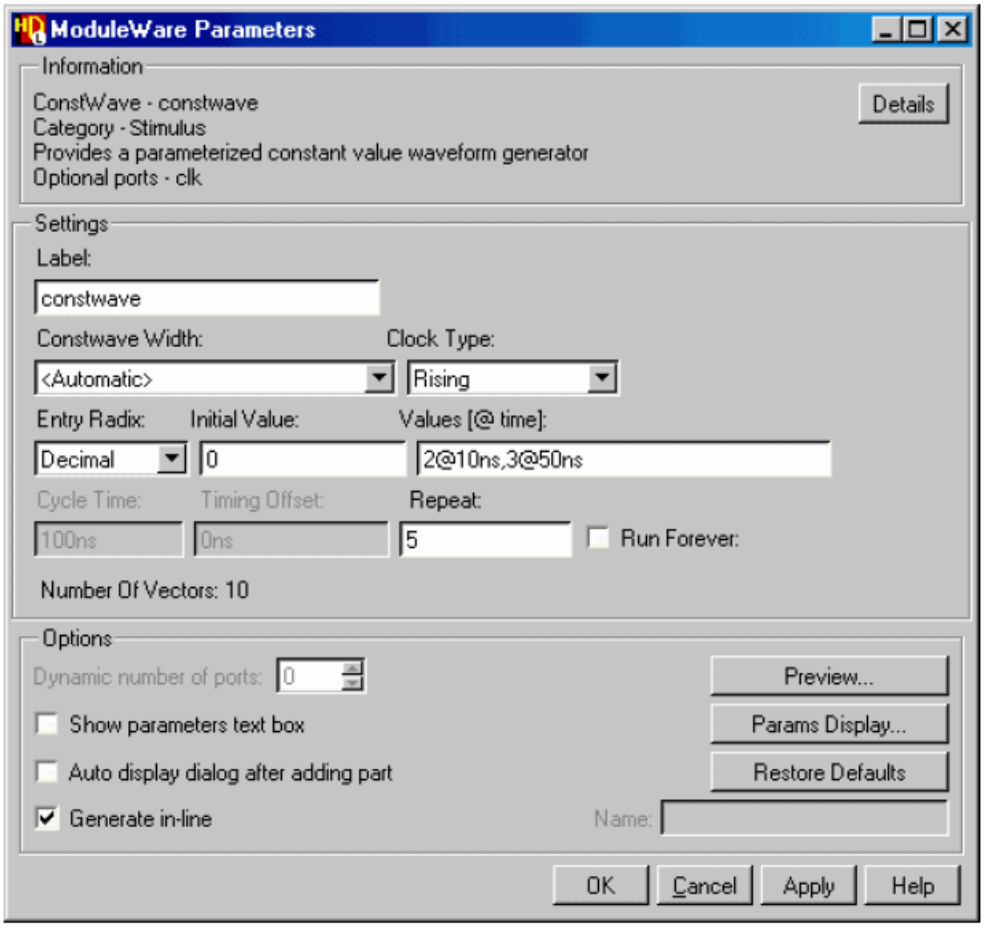
Constant Wave (constwave)

This part generates a constant value based waveform.



You can specify a label for the generated VHDL process (or Verilog *initial* block). If not specified, a unique label is generated from the part name (for example, *constwave1*, *constwave2*..*constwaveN*).

The width of the **constwave** output signal is set to *<Automatic>* by default but you can optionally set 2, 4, 8, 16, 32 or 64 bits by choosing from a dropdown list.



The clock type can be *Rising*, *Falling* or *None*. If you are using VHDL, you can also choose *RisingLast*, *FallingLast*, *RisingEdge* or *FallingEdge* transitions. If you choose *None*, the cycle time value must be specified.

You can choose whether a decimal, binary, octal or hexadecimal entry radix is used as the base for the initial value and *values@time* fields.

You can specify the initial value by entering a number based on the selected entry radix parameter that can fit in the width of the **constwave** signal. An error is issued if the number specified exceeds the specified bit width.

The constant values applied to the waveform can be specified as a comma separated list of *value@time* entries where each *value* is a constant number based on the entry radix. The timing sequence must be in ascending order. If no time is entered and there is only one value, the value applies to the entire sequence.

If the clock type is not *None*, the *value@time* entries must be within the clock period at run time.

If the clock type is *None*, the cycle time must be specified and the *value@time* entries must all be within this cycle time. You can also specify a timing offset (which must be less than 1 second) before the waveform generator starts.

You can specify the number of times to repeat the waveform or allow it to run forever. When a number of repeats is set, the number of vectors generated is shown in the dialog box.

Time intervals can optionally be specified in milliseconds (ms), microseconds (us), nanoseconds (ns) or picoseconds (ps). Seconds are assumed if no unit is specified. Spaces are automatically used between the time value and unit in the generated HDL.

Parameters

Table 11-4. Constant Wave Parameters

Parameter	Values	Default
Label	A unique HDL identifier	constwave
Width	**constwave** port width (must be > 0)	<Automatic>
Clock Type[1]	Rising,Falling,RisingLast,FallingLast,RisingEdge,FallingEdge,None	Rising
Entry Radix	Decimal,Binary,Octal,Hex	Decimal
Initial Value	A number based on the selected entry radix parameter	0
Values	List of value@ time entries (optionally in ms, us, ns or ps)	2@10ns,3@50ns
Cycle Time	Real number (optionally in ms, us, ns or ps)	100ns
Timing Offset	Real number (optionally in ms, us, ns or ps)	0ns
Repeat	Integer number of times to repeat	5
Run Forever	On or Off	Off

1. *RisingLast, FallingLast, RisingEdge* and *FallingEdge* clock types are only available in VHDL.

Checks

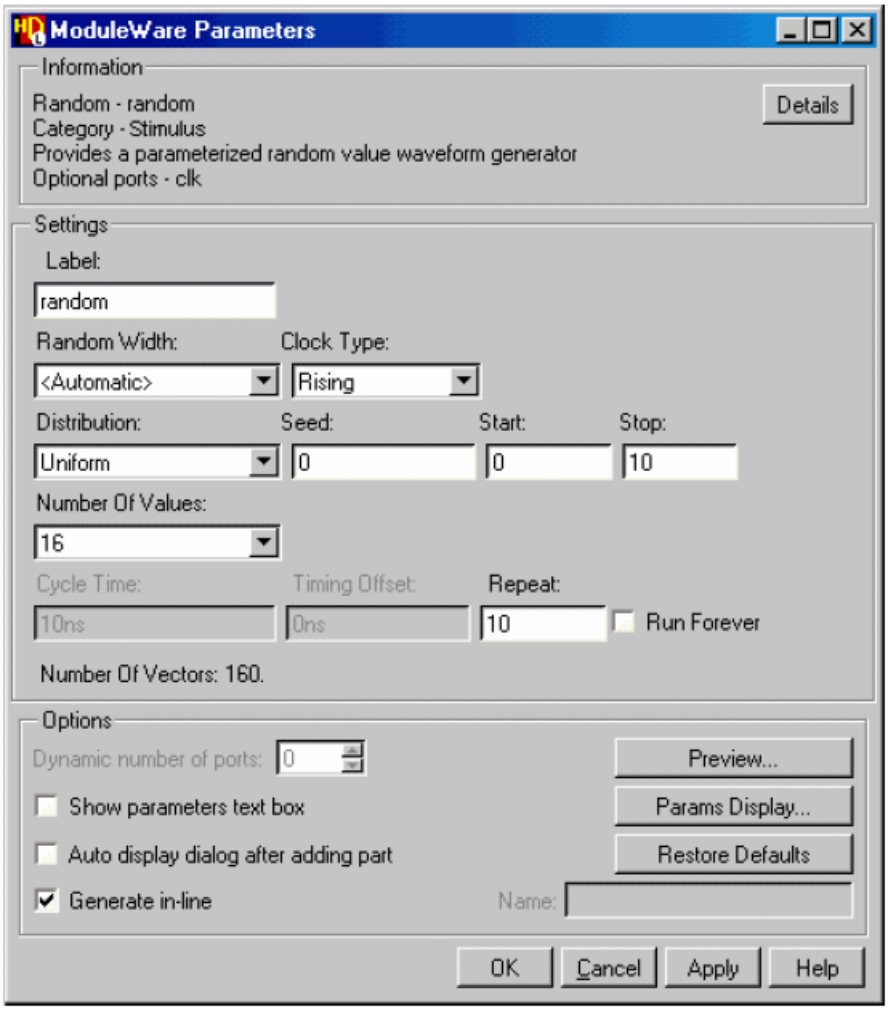
- An entry error is issued if invalid time units are specified or a parameter is outside its permitted range.
- A design rule checking error is issued if the radixed number does not fit into the **constwave** width.

Random Value (random)

This part generates a random value based waveform comprising a sequence of pseudo-random data values

The VHDL implementation of this part requires references to random generators defined in the *hds_package_library*. When you instantiate this part you are prompted to add package references for this library and the standard IEEE *std_logic_arith* or *numeric_std* libraries.

You can specify a label for the generated VHDL process (or Verilog *always* block). If not specified, a unique label is generated from the part name (for example, *random1*, *random2*..*randomN*).



The width of the **random** output signal is set to <*Automatic*> by default but you can optionally set 2, 4, 8, 16, 32 or 64 bits by choosing from a dropdown list.

The clock type can be *Rising*, *Falling* or *None*. If you are using VHDL, you can also choose *RisingLast*, *FallingLast*, *RisingEdge* or *FallingEdge* transitions.

You can choose from a list of supported probalistic distribution functions and specify a seed value to ensure that the same random sequence is applied each time.

For a Uniform distribution, you can specify seed, start and stop values.

For a Normal distribution, you can specify seed, mean and standard deviation values.

For an Exponential or Poisson distribution, you can specify seed and mean values.

For an Erlang distribution, you can specify seed, K_stage and mean value.

For a Chi_square or T distribution, you can specify seed and degree of freedom values.

You can also choose from a number of values list.

These distribution functions are defined in the Verilog Language Reference Manual. Equivalent functions for VHDL are defined in the *hds_package_library*.

If the clock type is *None*, the cycle time must be specified. You can also specify a timing offset (which must be less than 1 second) before the waveform generator starts.

You can specify the number of times to repeat the waveform or allow it to run forever. When a number of repeats is set, the number of vectors generated is shown in the dialog box.

Time intervals can optionally be specified in milliseconds (ms), microseconds (us), nanoseconds (ns) or picoseconds (ps). Seconds are assumed if no unit is specified. Spaces are automatically used between the time value and unit in the generated HDL.

Parameters

Table 11-5. Random Value Parameters

Parameter	Values	Default
Label	A unique HDL identifier	random
Width	**random** port width (must be > 0)	<Automatic>
Clock Type[1]	Rising,Falling,RisingLast,FallingLast,RisingEdge,FallingEdge,None	Rising
Distribution	Uniform,Normal,Exponential,Poisson,Erlang,Chi_square,T	Uniform
Seed	Integer seed value for all distributions	0
Start	Integer start value Uniform	0
Stop	Integer stop value for Uniform	10
Mean	Integer mean value for Normal, Exponential, Poisson or Erlang	0
Mean	Integer mean value for Erlang	10
Std Deviation	Integer standard deviation value for Normal	10
K_stage	Integer K_stage value for Erlang	0
Degree of Freedom	Integer degree of freedom value for Chi_Square or T	0
No of Values	1,2,4,8,16,32,64,128,256,512,1024	16
Cycle Time	Real number (optionally in ms, us, ns or ps)	10ns
Timing Offset	Real number (optionally in ms, us, ns or ps)	0ns
Repeat	Integer number of times to repeat	10
Run Forever	On or Off	Off

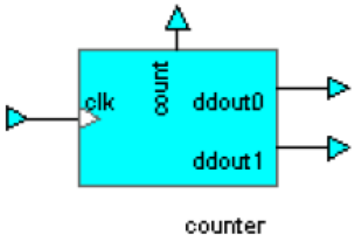
1. *RisingLast*, *FallingLast*, *RisingEdge* and *FallingEdge* clock types are only available in VHDL.

Checks

- An entry error is issued if invalid time units are specified or a parameter is outside its permitted range.

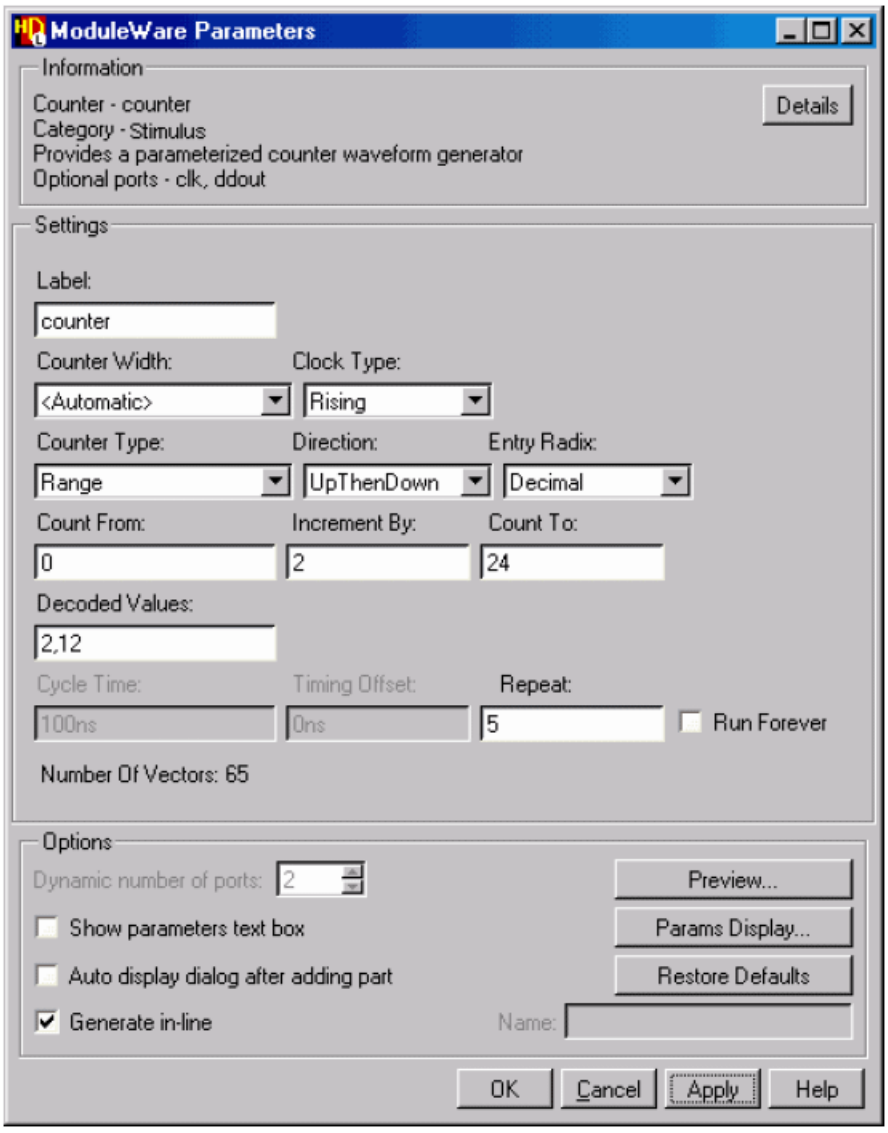
Counter Value (counter)

This part provides a counter based waveform generator.



You can specify a label for the generated VHDL process (or Verilog *always* block). If not specified, a unique label is generated from the part name (for example, *counter1*, *counter2*..*counterN*).

The width of the **count** output signal is set to <*Automatic*> by default but you can optionally set 2, 4, 8, 16, 32 or 64 bits by choosing from a dropdown list.



The clock type can be *Rising*, *Falling* or *None*. If you are using VHDL, you can also choose *RisingLast*, *FallingLast*, *RisingEdge* or *FallingEdge* transitions. If you choose *None*, the cycle time value must be specified.

You can choose from a list of supported counter types.

A Range counter generates a sequence of data values which count up or down by a fixed amount between specified Count From and Count To values.

A Binary counter generates a binary data sequence starting from 0 to 2^{N-1} where N is the width in bits of the **count** port. You can use the Increment By field to specify the counter step.

A Gray counter generates a counting data stream in which only a single bit transition per simulation cycle is allowed. It also starts from 0 to the maximum Gray number that the width of the count port can support. You can use the Increment By field to specify the counter step.

A OneHot counter generates a stream of data values which move a logic 1 through a field of logic 0's, from left to right.

A ZeroHot counter generates a stream of data values which move a logic 0 through a field of logic 1's, from left to right.

A Johnson counter generates a bit stream which starts with all 0's. On the next cycle, the most significant bit becomes a 1. Then on the next cycle, the two most significant bits are 1's. This continues until all bits are 1's. Then the process starts all over again replacing 1's with 0's.

For each type of counter, you can specify the count direction which can be up, down, up and then down or down and then up. Note that when a number of repeats is specified, *UpThenDown* followed by *DownThenUp* is considered as one repetition.

You can specify a decimal, binary, octal or hexadecimal radix which is used as the base for the From, To and Increment By fields.

You can also specify a comma-separated list of values to be decoded. A new output port is created for each decode flag. For example, output ports **ddout0** and **ddout1** are required for the two decoded values 2, 12 shown in the example dialog box. By default the output goes to 1 when the counter reaches a specified value.

If the clock type is *None*, the cycle time must be specified. You can also specify a timing offset (which must be less than 1 second) before the waveform generator starts.

You can specify the number of times to repeat the waveform or allow it to run forever. When a number of repeats is set, the number of vectors generated is shown in the dialog box.

Time intervals can optionally be specified in milliseconds (ms), microseconds (us), nanoseconds (ns) or picoseconds (ps). Seconds are assumed if no unit is specified. Spaces are automatically used between the time value and unit in the generated HDL.

Parameters

Table 11-6. Counter Value Parameters

Parameter	Values	Default
Label	A unique HDL identifier	counter
Width	**count** port width (must be > 0)	<Automatic>
Clock Type[1]	Rising,Falling,RisingLast,FallingLast,RisingEdge,FallingEdge,None	Rising
Counter Type	Range,Binary,Gray,OneHot,ZeroHot,Johnson	Range
Direction	Up,Down,UpThenDown,DownThenUp	Up
Entry Radix	Decimal,Binary,Octal,Hex	Decimal
Count From	Integer value for Range counter	0
Increment By	Integer value for Range, Binary and Gray counters	2
Count To	Integer value for Range counter	24
Decoded Values	Comma separated list of Integer values	2,12,16
Cycle Time	Real number (optionally in ms, us, ns or ps)	100ns
Timing Offset	Real number (optionally in ms, us, ns or ps)	0ns
Repeat	Integer number of times to repeat	5
Run Forever	On or Off	Off

1. *RisingLast*, *FallingLast*, *RisingEdge* and *FallingEdge* clock types are only available in VHDL.

Checks

- An entry error is issued if invalid time units are specified.
- A design rule checking error is issued if a parameter is outside its permitted range.
- A warning is issued if no vectors would result from the specified parameters or if a very high number (> 10,000) would result.

Appendix A
Name List

This appendix indexes the ModuleWare parts by name.

Appendix B
Function List

This appendix indexes the ModuleWare parts by function.

Function	Description	Page
Primitive	CMOS Primitive (pcmos)	240
Primitive	NAND Primitive (pnand)	242
Primitive	NMOS Primitive (pnmos)	243
Primitive	NOR Primitive (pnor)	244
Primitive	NOT Primitive (pnot)	245
Primitive	Notif0 Primitive (pnotif0)	247
Primitive	Notif1 Primitive (pnotif1)	249
Primitive	OR Primitive (por)	251
Primitive	PMOS Primitive (ppmos)	252
Primitive	Pulldown Primitive (ppulldown)	253
Primitive	Pullup Primitive (ppullup)	254
Primitive	RCMOS Primitive (prcmos)	255
Primitive	RNMOS Primitive (prnmos)	257
Primitive	RPMOS Primitive (prpmos)	258
Primitive	XNOR Primitive (pxnor)	260
Primitive	XOR Primitive (pxor)	259
Pulldown	Pulldown Primitive (ppulldown)	253
Pullup	Pullup Primitive (ppullup)	254
RAM	Dual Port RAM (ram2p)	206
RAM	Dual Port RAM (ram2p)	206
RAM	Synthesizable Dual-Port RAM (ramdp)	225
RAM	Synthesizable Single-Port RAM (ramsp)	227
RCMOS	RCMOS Primitive (prcmos)	255
Register File	Register File (regfile)	214
RNMOS	RNMOS Primitive (prnmos)	257
ROM	ROM (rom)	216
RPMOS	RPMOS Primitive (prpmos)	258
Selector	Fixed Bit Selector (fbitsel)	92
Shifter	Fixed Shifter (fixshift)	94
Shifter	Left Shifter (lshift)	123

Index

— U —

— V —

— W —

— X —

End-User License Agreement

The latest version of the End-User License Agreement is available on-line at:
www.mentor.com/eula

IMPORTANT INFORMATION

USE OF ALL SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE PRODUCTS. USE OF SOFTWARE INDICATES CUSTOMER'S COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.

END-USER LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Software (as defined in Section 2) and hardware (collectively "Products") between the company acquiring the Products ("Customer"), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity ("Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If Customer does not agree to these terms and conditions, promptly return or, in the case of Software received electronically, certify destruction of Software and all accompanying items within five days after receipt of Software and receive a full refund of any license fee paid.

1. **ORDERS, FEES AND PAYMENT.**

   1.1. To the extent Customer (or if agreed by Mentor Graphics, Customer's appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement ("Order(s)"), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement, any applicable addenda and the applicable quotation, whether or not these documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order will not be effective unless agreed in writing by an authorized representative of Customer and Mentor Graphics.

   1.2. Amounts invoiced will be paid, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will state separately in the applicable invoice(s). Unless timely provided with a valid certificate of exemption or other evidence that items are not taxable, Mentor Graphics will invoice Customer for all applicable taxes including, but not limited to, VAT, GST, sales tax and service tax. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer's sole responsibility. If Customer appoints a third party to place purchase orders and/or make payments on Customer's behalf, Customer shall be liable for payment under Orders placed by such third party in the event of default.

   1.3. All Products are delivered FCA factory (Incoterms 2000), freight prepaid and invoiced to Customer, except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics retains a security interest in all Products delivered under this Agreement, to secure payment of the purchase price of such Products, and Customer agrees to sign any documents that Mentor Graphics determines to be necessary or convenient for use in filing or perfecting such security interest. Mentor Graphics' delivery of Software by electronic means is subject to Customer's provision of both a primary and an alternate e-mail address.

2. **GRANT OF LICENSE.** The software installed, downloaded, or otherwise acquired by Customer under this Agreement, including any updates, modifications, revisions, copies, documentation and design data ("Software") are copyrighted, trade secret and confidential information of Mentor Graphics or its licensors, who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics grants to Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software solely: (a) in machine-readable, object-code form (except as provided in Subsection 5.2); (b) for Customer's internal business purposes; (c) for the term of the license; and (d) on the computer hardware and at the site authorized by Mentor Graphics. A site is restricted to a one-half mile (800 meter) radius. Customer may have Software temporarily used by an employee for telecommuting purposes from locations other than a Customer office, such as the employee's residence, an airport or hotel, provided that such employee's primary place of employment is the site where the Software is authorized for use. Mentor Graphics' standard policies and programs, which vary depending on Software, license fees paid or services purchased, apply to the following: (a) relocation of Software; (b) use of Software, which may be limited, for example, to execution of a single session by a single user on the authorized hardware or for a restricted period of time (such limitations may be technically implemented through the use of authorization codes or similar devices); and (c) support services provided, including eligibility to receive telephone support, updates, modifications, and revisions. For the avoidance of doubt, if Customer requests any change or enhancement to Software, whether in the course of

receiving support or consulting services, evaluating Software, performing beta testing or otherwise, any inventions, product improvements, modifications or developments made by Mentor Graphics (at Mentor Graphics' sole discretion) will be the exclusive property of Mentor Graphics.

3. **ESC SOFTWARE.** If Customer purchases a license to use development or prototyping tools of Mentor Graphics' Embedded Software Channel ("ESC"), Mentor Graphics grants to Customer a nontransferable, nonexclusive license to reproduce and distribute executable files created using ESC compilers, including the ESC run-time libraries distributed with ESC C and C++ compiler Software that are linked into a composite program as an integral part of Customer's compiled computer program, provided that Customer distributes these files only in conjunction with Customer's compiled computer program. Mentor Graphics does NOT grant Customer any right to duplicate, incorporate or embed copies of Mentor Graphics' real-time operating systems or other embedded software products into Customer's products or applications without first signing or otherwise agreeing to a separate agreement with Mentor Graphics for such purpose.

4. **BETA CODE.**

4.1. Portions or all of certain Software may contain code for experimental testing and evaluation ("Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to Customer a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. This grant and Customer's use of the Beta Code shall not be construed as marketing or offering to sell a license to the Beta Code, which Mentor Graphics may choose not to release commercially in any form.

4.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer's use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer's evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.

4.3. Customer agrees to maintain Beta Code in confidence and shall restrict access to the Beta Code, including the methods and concepts utilized therein, solely to those employees and Customer location(s) authorized by Mentor Graphics to perform beta testing. Customer agrees that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on Customer's feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this Subsection 4.3 shall survive termination of this Agreement.

5. **RESTRICTIONS ON USE.**

5.1. Customer may copy Software only as reasonably necessary to support the authorized use. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. Customer shall maintain a record of the number and primary location of all copies of Software, including copies merged with other software, and shall make those records available to Mentor Graphics upon request. Customer shall not make Products available in any form to any person other than Customer's employees and on-site contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Products and ensure that any person permitted access does not disclose or use it except as permitted by this Agreement. Customer shall give Mentor Graphics written notice of any unauthorized disclosure or use of the Products as soon as Customer learns or becomes aware of such unauthorized disclosure or use. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive any source code from Software. Log files, data files, rule files and script files generated by or for the Software (collectively "Files"), including without limitation files containing Standard Verification Rule Format ("SVRF") and Tcl Verification Format ("TVF") which are Mentor Graphics' proprietary syntaxes for expressing process rules, constitute or include confidential information of Mentor Graphics. Customer may share Files with third parties, excluding Mentor Graphics competitors, provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Customer may use Files containing SVRF or TVF only with Mentor Graphics products. Under no circumstances shall Customer use Software or Files or allow their use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Software, or disclose to any third party the results of, or information pertaining to, any benchmark.

5.2. If any Software or portions thereof are provided in source code form, Customer will use the source code only to correct software errors and enhance or modify the Software for the authorized use. Customer shall not disclose or permit disclosure of source code, in whole or in part, including any of its methods or concepts, to anyone except Customer's employees or contractors, excluding Mentor Graphics competitors, with a need to know. Customer shall not copy or compile source code in any manner except to support this authorized use.

5.3. Customer may not assign this Agreement or the rights and duties under it, or relocate, sublicense or otherwise transfer the Products, whether by operation of law or otherwise ("Attempted Transfer"), without Mentor Graphics' prior written consent and payment of Mentor Graphics' then-current applicable relocation and/or transfer fees. Any Attempted Transfer without Mentor Graphics' prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics' option, result in the immediate termination of the Agreement and/or the licenses granted under this Agreement. The terms

of this Agreement, including without limitation the licensing and assignment provisions, shall be binding upon Customer's permitted successors in interest and assigns.

5.4. The provisions of this Section 5 shall survive the termination of this Agreement.

6. **SUPPORT SERVICES.** To the extent Customer purchases support services, Mentor Graphics will provide Customer updates and technical support for the Products, at the Customer site(s) for which support is purchased, in accordance with Mentor Graphics' then current End-User Support Terms located at http://supportnet.mentor.com/about/legal/.

7. **AUTOMATIC CHECK FOR UPDATES; PRIVACY.** Technological measures in Software may communicate with servers of Mentor Graphics or its contractors for the purpose of checking for and notifying the user of updates and to ensure that the Software in use is licensed in compliance with this Agreement. Mentor Graphics will not collect any personally identifiable data in this process and will not disclose any data collected to any third party without the prior written consent of Customer, except to Mentor Graphics' outside attorneys or as may be required by a court of competent jurisdiction.

8. **LIMITED WARRANTY.**

8.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Products, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Products will meet Customer's requirements or that operation of Products will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. Customer must notify Mentor Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Software under an Order and does not renew or reset, for example, with the delivery of (a) Software updates or (b) authorization codes or alternate Software under a transaction involving Software re-mix. This warranty shall not be valid if Products have been subject to misuse, unauthorized modification or improper installation. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF THE PRODUCTS TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF THE PRODUCTS THAT DO NOT MEET THIS LIMITED WARRANTY, PROVIDED CUSTOMER HAS OTHERWISE COMPLIED WITH THIS AGREEMENT. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) PRODUCTS PROVIDED AT NO CHARGE; OR (C) BETA CODE; ALL OF WHICH ARE PROVIDED "AS IS."

8.2. THE WARRANTIES SET FORTH IN THIS SECTION 8 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO PRODUCTS PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

9. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF MENTOR GRAPHICS OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL MENTOR GRAPHICS' OR ITS LICENSORS' LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT RECEIVED FROM CUSTOMER FOR THE HARDWARE, SOFTWARE LICENSE OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 9 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

10. **HAZARDOUS APPLICATIONS.** CUSTOMER ACKNOWLEDGES IT IS SOLELY RESPONSIBLE FOR TESTING ITS PRODUCTS USED IN APPLICATIONS WHERE THE FAILURE OR INACCURACY OF ITS PRODUCTS MIGHT RESULT IN DEATH OR PERSONAL INJURY ("HAZARDOUS APPLICATIONS"). NEITHER MENTOR GRAPHICS NOR ITS LICENSORS SHALL BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF MENTOR GRAPHICS PRODUCTS IN OR FOR HAZARDOUS APPLICATIONS. THE PROVISIONS OF THIS SECTION 10 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

11. **INDEMNIFICATION.** CUSTOMER AGREES TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE OR LIABILITY, INCLUDING ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH THE USE OF PRODUCTS AS DESCRIBED IN SECTION 10. THE PROVISIONS OF THIS SECTION 11 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

12. **INFRINGEMENT.**

12.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Product acquired by Customer hereunder infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay costs and damages finally awarded against Customer that are attributable to the action. Customer understands and agrees that as conditions to Mentor Graphics' obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance

to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.

12.2. If a claim is made under Subsection 12.1 Mentor Graphics may, at its option and expense, (a) replace or modify the Product so that it becomes noninfringing; (b) procure for Customer the right to continue using the Product; or (c) require the return of the Product and refund to Customer any purchase price or license fee paid, less a reasonable allowance for use.

12.3. Mentor Graphics has no liability to Customer if the action is based upon: (a) the combination of Software or hardware with any product not furnished by Mentor Graphics; (b) the modification of the Product other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of the Product as part of an infringing process; (e) a product that Customer makes, uses, or sells; (f) any Beta Code or Product provided at no charge; (g) any software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; or (h) infringement by Customer that is deemed willful. In the case of (h), Customer shall reimburse Mentor Graphics for its reasonable attorney fees and other costs related to the action.

12.4. THIS SECTION 12 IS SUBJECT TO SECTION 9 ABOVE AND STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS FOR DEFENSE, SETTLEMENT AND DAMAGES, AND CUSTOMER'S SOLE AND EXCLUSIVE REMEDY, WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY PRODUCT PROVIDED UNDER THIS AGREEMENT.

13. **TERMINATION AND EFFECT OF TERMINATION.** If a Software license was provided for limited term use, such license will automatically terminate at the end of the authorized term.

13.1. Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement immediately upon written notice if Customer: (a) exceeds the scope of the license or otherwise fails to comply with the licensing or confidentiality provisions of this Agreement, or (b) becomes insolvent, files a bankruptcy petition, institutes proceedings for liquidation or winding up or enters into an agreement to assign its assets for the benefit of creditors. For any other material breach of any provision of this Agreement, Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement upon 30 days written notice if Customer fails to cure the breach within the 30 day notice period. Termination of this Agreement or any license granted hereunder will not affect Customer's obligation to pay for Products shipped or licenses granted prior to the termination, which amounts shall be payable immediately upon the date of termination.

13.2. Upon termination of this Agreement, the rights and obligations of the parties shall cease except as expressly set forth in this Agreement. Upon termination, Customer shall ensure that all use of the affected Products ceases, and shall return hardware and either return to Mentor Graphics or destroy Software in Customer's possession, including all copies and documentation, and certify in writing to Mentor Graphics within ten business days of the termination date that Customer no longer possesses any of the affected Products or copies of Software in any form.

14. **EXPORT.** The Products provided hereunder are subject to regulation by local laws and United States government agencies, which prohibit export or diversion of certain products and information about the products to certain countries and certain persons. Customer agrees that it will not export Products in any manner without first obtaining all necessary approval from appropriate local and United States government agencies.

15. **U.S. GOVERNMENT LICENSE RIGHTS.** Software was developed entirely at private expense. All Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to US FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in this Agreement, except for provisions which are contrary to applicable mandatory federal laws.

16. **THIRD PARTY BENEFICIARY.** Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, Microsoft Corporation and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.

17. **REVIEW OF LICENSE USAGE.** Customer will monitor the access to and use of Software. With prior written notice and during Customer's normal business hours, Mentor Graphics may engage an internationally recognized accounting firm to review Customer's software monitoring system and records deemed relevant by the internationally recognized accounting firm to confirm Customer's compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FLEXlm or FLEXnet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics' request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement. The provisions of this Section 17 shall survive the termination of this Agreement.

18. **CONTROLLING LAW, JURISDICTION AND DISPUTE RESOLUTION.** The owners of certain Mentor Graphics intellectual property licensed under this Agreement are located in Ireland and the United States. To promote consistency around the world, disputes shall be resolved as follows: excluding conflict of laws rules, this Agreement shall be governed by and construed under the laws of the State of Oregon, USA, if Customer is located in North or South America, and the laws of Ireland if Customer is located outside of North or South America. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of the courts of Portland, Oregon when the laws of Oregon apply, or Dublin, Ireland when the laws of Ireland apply. Notwithstanding the foregoing, all disputes in Asia arising out of or in relation to this Agreement shall be resolved by arbitration in Singapore before a single arbitrator to be appointed by the chairman of the Singapore International

Arbitration Centre ("SIAC") to be conducted in the English language, in accordance with the Arbitration Rules of the SIAC in effect at the time of the dispute, which rules are deemed to be incorporated by reference in this section. This section shall not restrict Mentor Graphics' right to bring an action against Customer in the jurisdiction where Customer's place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.

19. **SEVERABILITY.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.

20. **MISCELLANEOUS.** This Agreement contains the parties' entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements, including but not limited to any purchase order terms and conditions. Some Software may contain code distributed under a third party license agreement that may provide additional rights to Customer. Please see the applicable Software documentation for details. This Agreement may only be modified in writing by authorized representatives of the parties. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.

Rev. 100615, Part No. 246066