ORCA®
VHDL Simulation Manual

For Use With
Synopsys® FPGA Express™ version 3.5 or lower, Model Technology® Modelsim/PLUS Workstation® 5.2 or higher Modelsim/VHDL Windows® Version 4.7 or higher Synopsys VSS™ Version 99.05 or higher, ORCA 4.1, and ispLEVER 2.0 and higher

Technical Support Line: 1-800-LATTICE or 408-826-6002 (international)
VHDL Simulation Manual

Modelsim/PLUS Workstation, Modelsim/VHDL Windows, V-System, VCOM and VSIM are trademarks of Model Technology Incorporated.
ModelSim is a registered trademark of Model Technology, Inc.
Synopsys and the Synopsys logo, SmartModel and Smartmodels are registered trademarks of Synopsys, Inc.; FPGA Express, VHDL System Simulator, VSS, VSS Expert, VSS Family are trademarks of Synopsys, Inc.
EPIC and Timing Wizard are trademarks of Xilinx, Inc.
ORCA and Scuba are registered trademarks of Lattice Semiconductor Corporation.
Synplicity is a registered trademark of Synplicity, Inc.
Unix is a registered trademark of X/Open Company, Inc.
All other brands or product names are the trademarks or registered trademarks of their respective owners.
# ORCA/VHDL Simulation Interface

## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVERVIEW</td>
<td>1</td>
</tr>
<tr>
<td>SOFTWARE REQUIREMENTS</td>
<td>2</td>
</tr>
<tr>
<td>SETTING THE DESIGN ENVIRONMENT</td>
<td>2</td>
</tr>
<tr>
<td>Environment Variables</td>
<td>2</td>
</tr>
<tr>
<td>Workstation Setup Instructions</td>
<td>3</td>
</tr>
<tr>
<td>MTI ModelSim Installation</td>
<td>4</td>
</tr>
<tr>
<td>SIMULATION USING VHDL ORCA MODELS</td>
<td>6</td>
</tr>
<tr>
<td>Simulation with MTI ModelSim</td>
<td>8</td>
</tr>
<tr>
<td>SIMULATION USING VHDL NEOPRIMS FOR ORCA 2CA/2TA/2TB</td>
<td>9</td>
</tr>
<tr>
<td>SIMULATION USING VHDL NEOPRIMS FOR ORCA SERIES 3</td>
<td>12</td>
</tr>
<tr>
<td>USING ORCA 3 SPECIAL CELLS IN VHDL SIMULATIONS</td>
<td>15</td>
</tr>
<tr>
<td>Using the Series 3 PCM Cells</td>
<td>15</td>
</tr>
<tr>
<td>Module Configuration</td>
<td>15</td>
</tr>
<tr>
<td>PCM[B,T] Port Declarations</td>
<td>16</td>
</tr>
<tr>
<td>Synthesis Support</td>
<td>16</td>
</tr>
<tr>
<td>FREQUENCY Attribute</td>
<td>17</td>
</tr>
<tr>
<td>Using the Series 3 Clock Controller Cells</td>
<td>21</td>
</tr>
<tr>
<td>USING ORCA 4 SPECIAL CELLS IN VHDL SIMULATIONS</td>
<td>23</td>
</tr>
<tr>
<td>Series 4 PLL Cells</td>
<td>23</td>
</tr>
<tr>
<td>Module Configuration</td>
<td>23</td>
</tr>
<tr>
<td>[H]PPLL Port Declarations</td>
<td>24</td>
</tr>
<tr>
<td>PLL[1,2] Port Declarations</td>
<td>24</td>
</tr>
<tr>
<td>Synthesis Support</td>
<td>24</td>
</tr>
<tr>
<td>Using the Series 4 PLL</td>
<td>25</td>
</tr>
<tr>
<td>SIMULATION WITH MTI ModelSim</td>
<td>28</td>
</tr>
<tr>
<td>System Bus SmartModel Simulation (Series 4)</td>
<td>29</td>
</tr>
<tr>
<td>Front End VHDL Simulation With MTI ModelSim</td>
<td>29</td>
</tr>
<tr>
<td>Back End VHDL Simulation With MTI ModelSim</td>
<td>30</td>
</tr>
<tr>
<td>SIMULATION WITH SYNOPTSYS VSS</td>
<td>31</td>
</tr>
<tr>
<td>APPENDIX</td>
<td>33</td>
</tr>
<tr>
<td>MEMORY INITIALIZATION</td>
<td>33</td>
</tr>
<tr>
<td>A front-end netlist from a SCUBA Module Generated in Module/IP Manager</td>
<td>33</td>
</tr>
<tr>
<td>A section of the back-end netlist from ngd2vhd</td>
<td>35</td>
</tr>
<tr>
<td>USING GSR/PUR IN VHDL</td>
<td>37</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Programmable GSR Support (Series 3)</td>
<td>37</td>
</tr>
<tr>
<td>USING TSALL IN VHDL</td>
<td>40</td>
</tr>
<tr>
<td>ILLEGAL NAMES/CHARACTERS IN ORCA DESIGN FLOWS</td>
<td>42</td>
</tr>
</tbody>
</table>
OVERVIEW

This manual describes the Optimized Reconfigurable Cell Array (ORCA) interface between the ORCA® place and route tools in the ispLEVER software and your preferred VHDL compiler/simulator (currently ModelSim® or VSS™). Together, the tools provide a powerful and integrated high-level simulation environment for ORCA Field-Programmable Gate Arrays (FPGAs), using VITAL 3.0 (VITAL’95) compliant libraries. Library support is subject to change.

Note

Note that version numbers for the interface software are continually being updated with each release. Check with the vendor on version number and how it affects the file names or syntax that make it possible to perform tasks as they relate to simulation of designs with ORCA. Check with technical support with compatibility and support issues.
SOFTWARE REQUIREMENTS

The ORCA/VHDL Interface is compatible with the following software:

- Latest version of ispLEVER software with ORCA devices installed.
- Model Technology ModelSim/PLUS Workstation™ version 5.2 or later
- Model Technology ModelSim/VHDL Windows™ version 4.7 or later

SETTING THE DESIGN ENVIRONMENT

This section helps you customize your VHDL environment for designing an ORCA FPGA.

Environment Variables

Make sure you have installed the latest version of ispLEVER with ORCA devices installed and that the FOUNDARY environment variable is set. The FOUNDARY variable points to the ispFPGA (PC) or ispfpga (UNIX) directory.

The variable should be set as follows:

```
$ setenv FOUNDARY <ispfpga_directory>
```

Note

Note that these variables are automatically set by the software and you should not have to manually set them. If for some reason, they are changed, this provides instruction for resetting them.
**Workstation Setup Instructions**

Make sure you have installed the current version of ispLEVER and have satisfied the hardware and software requirements described in the ORCA documentation.

Correct installation of ORCA will create the following directories (assumes $FOUNDORY as the top level directory):

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vhdl</td>
<td>ORCA/VHDL Interface Directory</td>
</tr>
<tr>
<td>vhdl/data</td>
<td>ORCA/VHDL Interface Data Directory</td>
</tr>
<tr>
<td>vhdl/data/neoprims/src</td>
<td>VITAL 3.0-compliant NeoPrim library source code</td>
</tr>
<tr>
<td>vhdl/data/orca2/src</td>
<td>VHDL ORCA2 library source code</td>
</tr>
<tr>
<td>vhdl/data/orca3/src</td>
<td>VHDL ORCA3 library source code</td>
</tr>
<tr>
<td>vhdl/data/orca4/src</td>
<td>VHDL ORCA4 library source code</td>
</tr>
<tr>
<td>vhdl/data/neoprims/mti</td>
<td>MTI-specific setup and compile scripts</td>
</tr>
<tr>
<td>vhdl/data/orca2/mti</td>
<td>MTI-specific setup and compile scripts</td>
</tr>
<tr>
<td>vhdl/data/orca3/mti</td>
<td>MTI-specific setup and compile scripts</td>
</tr>
<tr>
<td>vhdl/data/orca4/mti</td>
<td>MTI-specific setup and compile scripts</td>
</tr>
</tbody>
</table>
MTI ModelSim Installation

ModelSim’s MTI simulation program is integrated with the ispLEVER software suite and is installed with most installation implementation options for the ispLEVER software unless you specifically deselect these options during installaton. See current installation notices that came with your ispLEVER software for details.

You can use this version of the simulator or another version for which you are already licensed. You must obtain a license for the simulator to enable it in the flow in ispLEVER. See the *ModelSim User’s Manual* in the ispLEVER documentation library for information on installation and use of this tool. Use the Help > ispLEVER Help > Third Party Manuals contents link to access this information.

The remainder of this manual describes how to simulate using the ORCA and NeoPrim VITAL library. The design flow on the following page shows where the simulation library fits in with respect to ORCA.
Simulation Using VHDL ORCA Models and NeoPrim Models with ORCA

Front End Pre-ORCA Simulation

1. Structural VHDL Netlist (.vhd)
2. VITAL 3.0 Compliant ORCA Library
3. VHDL Compiler / Simulator

Back End Post-ORCA Simulation

1. EDIF (.edn) for ORCA
2. ORCA Map, Place and Route
3. VHDLNeoPrim/ORCA Netlist (.vhd) and SDF file (.sdf)
4. VITAL 3.0 Compliant NeoPrim/ORCA Library
5. VHDL Compiler / Simulator
SIMULATION USING VHDL ORCA MODELS

Generate a structural VHDL netlist in terms of ORCA components either by synthesizing behavioral code or by using Module/IP Manager to generate ORCA SCUBA modules, as in the following:

```vhdl
-- VHDL netlist generated by SCUBA 4.2
-- Tue Nov 26 16:27:33 2002

library IEEE, ORCA;
use IEEE.std_logic_1164.all;
use ORCA.orcacompl.all;

entity acsreg is
  port (waddr: in std_logic_vector(3 downto 0);
        datain: in std_logic_vector(2 downto 0);
        clk: in std_logic; wren: in std_logic;
        raddr: in std_logic_vector(3 downto 0);
        dataout: out std_logic_vector(2 downto 0));
end acsreg;

architecture Structure of acsreg is

  -- internal signal declarations
  signal mem_rdout_0: std_logic;
  signal scuba_vhi: std_logic;

  -- local component declarations
  component DCE16X2
    port (AD0: in std_logic; AD1: in std_logic;
          AD2: in std_logic; AD3: in std_logic;
          DI0: in std_logic; DI1: in std_logic;
          CK: in std_logic; WREN: in std_logic;
          WPE: in std_logic; RAD0: in std_logic;
          RAD1: in std_logic; RAD2: in std_logic;
          RAD3: in std_logic; DO0: out std_logic;
          DO1: out std_logic; RDO0: out std_logic;
          RDO1: out std_logic);
  end component;
```
component VHI
    port (Z: out std_logic);
end component;

begin
    -- component instantiation statements
    mem_0_0_1: DCE16X2
        port map (AD0=>waddr(0), AD1=>waddr(1),
                  AD2=>waddr(2), AD3=>waddr(3), DI0=>datain(2),
                  DI1=>datain(2), CK=>clk, WREN=>wren, WPE=>scuba_vhi,
                  RAD0=>raddr(0), RAD1=>raddr(1), RAD2=>raddr(2),
                  RAD3=>raddr(3), DO0=>open, DO1=>open,
                  RDO0=>dataout(2), RDO1=>mem_rdout_0);

    scuba_vhi_inst: VHI
        port map (Z=>scuba_vhi);

    mem_0_1_0: DCE16X2
        port map (AD0=>waddr(0), AD1=>waddr(1),
                  AD2=>waddr(2), AD3=>waddr(3), DI0=>datain(0),
                  DI1=>datain(1), CK=>clk, WREN=>wren, WPE=>scuba_vhi,
                  RAD0=>raddr(0), RAD1=>raddr(1), RAD2=>raddr(2),
                  RAD3=>raddr(3), DO0=>open, DO1=>open,
                  RDO0=>dataout(0), RDO1=>dataout(1));

end Structure;

Note

If your synthesized netlist is missing the library declaration and use clauses for ORCA, add them as shown below:

library IEEE,ORCA;
use IEEE.std_logic_1164.all;
use ORCA.orcacomps.all;
CAUTION

If there is a GSR component in the netlist, make sure that the GSR port is of type input.

Simulation with MTI ModelSim

The following shows command line instructions for simulation with ModelSim. Refer the the ModelSim User’s Manual in the ispLEVER documentation library for more information on graphical user interface instructions.

To verify the functionality of the design using MTI’s ModelSim:

1. Create an MTI work directory in the directory in which you generated the VHDL structural netlist, using the MTI command:
   
   vlib work

   (Make sure that you have a copy of modelsim.ini from the $FOUNDRY/vhdl/data/mti/misc directory in your current directory.)

2. Compile the structural VHDL netlist into this work directory, using MTI’s vcom:
   
   vcom <file_name>.vhd

3. Simulate it using the ORCA VHDL library and MTI’s vsim:
   
   vsim -t ps <entity_name> [arch_name]

   You must simulate in picosecond resolution in order to notice the default unit delay (0.1 ns) in the ORCA models. Otherwise 0.0 ns is used by the simulator.

   Note

   For a complete description of how to use MTI’s ModelSim tools, refer to ModelSim/Workstation User’s Manual. For ModelSim PC simulation, all these commands should be executed from the ModelSim window on the PC.
SIMULATION USING VHDL NEOPRIMS FOR ORCA 2CA/2TA/2TB

Note that these provide command line instruction for all simulation model generation. Refer to the ispLEVER Help system for instruction on use of the graphical user interface tools in Project Navigator.

1. Generate a back-annotated VHDL netlist from ORCA by executing the following commands:

   ```bash
   ngdanno <file_name>.ncd -o <file_name>.nga <file_name>.ngm
   ngd2vhd -w <infile>
   ```

   **Command summary:**

   ```bash
   ngdanno [-o <ngafile>.nga] [-p <prffile>.prf] <ncdfile>.ncd>
   [<ngmfile>.ngm]>
   ```

   *-o:* Output file name(s)
   *-p:* Preference file
   *<file_name>.ncd:* Input file from mapper containing the placement routing and timing information
   *<file_name>.nga:* (generic annotated file) Contains the logical design combined with the post-layout delays from the physical design
   *<file_name>.ngm:* (generic mapping file) Contains mapping information

   ```bash
   ```

   *-n:* Output flattened netlist
   *-p:* Connect pullup/down device to unsourced net
   *-t:* Target this VHDL file to Mentor Graphics Time Explorer
   *-w:* Overwrite the output file
   *-x:* Do not generate the cross reference file
   *<infile>:* Input file: .ngd or .nga
   *<outfile>:* Output file name (default is ‘<infile>.vhd’)
The netlist generated by `ngd2vhd` is described in terms of VHDL NeoPRIM cells, as in the following example:

```vhdl
-- entity TNEOLATCH

library IEEE, VITAL, ORCA;
use IEEE.STD_LOGIC_1164.all;
use NEOPRIMS.neocomp.all;
entity TNEOLATCH is
  port (VOUT: out Std_logic; VIN: in Std_logic; RST: in Std_logic;
        CLK: in Std_logic);
end TNEOLATCH;

architecture Structure of TNEOLATCH is
  signal I1_GSR_OR: Std_logic;
  signal I1_GSR_OR_1_INV: Std_logic;
  signal GND: Std_logic;
  signal GSR: Std_logic := '1';
  component NEOZERO
    port (VOUT: out Std_logic);
  end component;
  component NEOINV
    port (VIN: in Std_logic; VOUT: out Std_logic);
  end component;
  component NEOOR2
    port (VIN0: in Std_logic; VIN1: in Std_logic;
         VOUT: out Std_logic);
  end component;
  component NEOLATCH
    port (VIN: in Std_logic; CLK: in Std_logic;
         SET: in Std_logic;
         RST: in Std_logic; VOUT: out Std_logic);
  end component;
begin
  I1: NEOLATCH
    port map (VIN=>VIN, CLK=>CLK, SET=>GND,
              RST=>I1_GSR_OR, VOUT=>VOUT);
  I1_GSR_OR_3: NEOOR2
```

ORCA/VHDL Simulation
port map (VIN0=>RST, VIN1=>I1_GSR_OR_1_INV, VOUT=>I1_GSR_OR);
I1_GSR_OR_1_INV_4: NEOINV
port map (VIN=>GSR, VOUT=>I1_GSR_OR_1_INV);
GND_5: NEOZERO
port map (VOUT=>GND);
end Structure;

library NEOPRIMS;
configuration Structure_CON of TNEOLATCH is
for Structure
for all:NEOZERO use entity NEOPRIMS.NEOZERO(V);
end for;
for all:NEOINV use entity NEOPRIMS.NEOINV(V); end for;
for all:NEOOR2 use entity NEOPRIMS.NEOOR2(V); end for;
for all:NEOLATCH use entity NEOPRIMS.NEOLATCH(V);
end for;
end for;
end Structure_CON;
SIMULATION USING VHDL NEOPRIMS FOR ORCA SERIES 3

Note that this section provides command line instruction for all simulation model generation. Refer to the ispLEVER Help system for instruction on use of the graphical user interface tools in Project Navigator.

Specifically, this section describes how to run simulations using VHDL Neoprims/ORCA3. Note that ORCA supports ORCA3 backannotation to Neoprime as well as ORCA3 libraries.

To use VHDL Neoprims for ORCA Series 3 simulation:

1. Generate a back-annotated VHDL netlist from ORCA by executing the following commands:

   `ldbanno [-w] [-z] [-a] [-n <type>] [-l <libtype>] [ncdfile]`

   Command summary:
   - `-w` Overwrite the output file(s)
   - `-z` Zero delay (do not calculate or write delays; `-z` takes precedence over `-a`)
   - `-a` Write all delays (Even if they are zero)
   - `<type>` Netlist type to write out, e.g., VHDL etc
   - `<libtype>` Library element type to use, i.e., or3c00 (default) or neoprim
   - `<ncdfile>` Input file ‘.ncd’

   Use `ldbanno -h <type>` for detailed netlist and a notation options. For example, use the following syntax:

   `ldbanno -w -n vhdl -l filename.ncd`

   The netlist generated by the back-end netlister tool is described in terms of VHDL NeoPRIM cells, as in the following example:

   ```
   -- entity TNEOLATCH

   library IEEE, VITAL, ORCA3;
   use IEEE.STD_LOGIC_1164.all;
   ```
use NEOPRIMS.neocomp.all;
entity TNEOLATCH is
  port (VOUT: out Std_logic; VIN: in Std_logic;
       RST: in Std_logic;
       CLK: in Std_logic);
end TNEOLATCH;

architecture Structure of TNEOLATCH is
  signal I1_GSR_OR: Std_logic;
  signal I1_GSR_OR_1_INV: Std_logic;
  signal GND: Std_logic;
  signal GSR: Std_logic := '1';
  component NEOZERO
    port (VOUT: out Std_logic);
  end component;
  component NEOINV
    port (VIN: in Std_logic; VOUT: out Std_logic);
  end component;
  component NEOOR2
    port (VIN0: in Std_logic; VIN1: in Std_logic;
          VOUT: out Std_logic);
  end component;
  component NEOLATCH
    port (VIN: in Std_logic; CLK: in Std_logic;
         SET: in Std_logic;
         RST: in Std_logic; VOUT: out Std_logic);
  end component;
begin
  I1: NEOLATCH
     port map (VIN=>VIN, CLK=>CLK, SET=>GND,
              RST=>I1_GSR_OR, VOUT=>VOUT);

  I1_GSR_OR_3: NEOOR2
     port map (VIN0=>RST, VIN1=>I1_GSR_OR_1_INV,
              VOUT=>I1_GSR_OR);
  I1_GSR_OR_1_INV_4: NEOINV
     port map (VIN=>GSR, VOUT=>I1_GSR_OR_1_INV);
  GND_5: NEOZERO
     port map (VOUT=>GND);
end Structure;
library NEOPRIMS;
configuration Structure_CON of TNEOLATCH is
  for Structure
    for all:NEOZERO use entity NEOPRIMS.NEOZERO(V);
    end for;
    for all:NEOINV use entity NEOPRIMS.NEOINV(V); end for;
    for all:NEOOR2 use entity NEOPRIMS.NEOOR2(V); end for;
    for all:NEOLATCH use entity NEOPRIMS.NEOLATCH(V);
    end for;
  end for;
end Structure_CON;
USING ORCA 3 SPECIAL CELLS IN VHDL SIMULATIONS

Using the Series 3 PCM Cells

The PCM is a special function block that is used to modify or condition clock signals for optimum system performance. Some of the functions that can be performed with the PCM are clock skew reduction, duty cycle adjustment, clock multiplication, clock delay reduction, and clock phase adjustment. By using PLC logic resources in conjunction with PCM, many other functions, such as frequency control, are possible. PCMBUFs are the PCMs in the bypass mode. PCMBUFs should be used to generate clocks without any delays.

PCM has two modes of operation, delay-locked loop (DLL) and phase-locked loop (PLL). Some operations can be performed by either mode and some are specific to a particular mode. In general, DLL mode is preferable to PLL mode for the same function because it is less sensitive to input clock noise.

In a design, it is recommended to use the individual models for faster simulation if the PCM is operated in a particular mode throughout the operation. For example, if the PCM is being used as DLL1X mode throughout the operation, then it is better to use DLL1XT or DLL1XB models instead of the full PCM models PCMB or PCMT.

Module Configuration

The models can be programmed in the following two ways:

1. Program the models by passing the following parameters (generics) on the HDL model instantiations.

<table>
<thead>
<tr>
<th>Generics</th>
<th>VHDL Types</th>
<th>Possible values</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCM</td>
<td>: PCM_MODE</td>
<td>:= DLL1X, DLLPD, PLL, PCMBUF</td>
</tr>
<tr>
<td>DUTY</td>
<td>: REAL</td>
<td>:= 3.125 to 96.875</td>
</tr>
<tr>
<td>PDELAY</td>
<td>: INTEGER</td>
<td>:= 1 to 31</td>
</tr>
<tr>
<td>DIV0</td>
<td>: INTEGER</td>
<td>:= 0 to 8</td>
</tr>
<tr>
<td>DIV1</td>
<td>: INTEGER</td>
<td>:= 0 to 8</td>
</tr>
</tbody>
</table>
2. By writing/reading the appropriate bits to the configuration registers during simulation.

The full PCM model is initialized to the mode specified by the parameters (default is PCMBUF), and the registers are initialized based on the values of the parameters passed. See the FPGA Data book, July 1999, for more detailed information about each of the attributes.

### Note

For PLL and PCM in PLL mode, the simulator resolution is in ps instead of ns.

**PCM[BT] Port Declarations**

**INPUTS:** CLKin, FB, WE, RE, A[2:0], DI[7:0]

**OUTPUTS:** ECLK, SCLK, LOCK, DO[7:0]

**Synthesis Support**

- For all synthesis tools, PCMB, PCMT are *don’t touch* elements.
- Since there is one input pin added to PLLB and PLLT elements, they are don’t touch elements until the synthesis library is updated.
- The following are examples of the syntax for the attributes that are passed by the synthesis tools as attributes to the output EDIF netlist.

<table>
<thead>
<tr>
<th>Generics</th>
<th>VHDL Types</th>
<th>Possible values</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIV2</td>
<td>INTEGER</td>
<td>:= 0 to 8</td>
</tr>
<tr>
<td>DISABLED_GSR</td>
<td>Std_logic</td>
<td>:= '0' or '1'</td>
</tr>
<tr>
<td>DISABLED_DONE</td>
<td>Std_logic</td>
<td>:= '0' or '1'</td>
</tr>
<tr>
<td>PWron</td>
<td>Std_logic</td>
<td>:= '0' or '1'</td>
</tr>
<tr>
<td>FBDELAY</td>
<td>TIME</td>
<td>:= in ps</td>
</tr>
</tbody>
</table>
VHDL:

\begin{verbatim}
attribute PCM       : string;
attribute DUTY      : string;
attribute PDELAY    : string;
attribute DIV0      : string;
attribute PCM of U_1    : label is "PLL";
attribute DUTY of U_1   : label is "50.0";
attribute PDELAY of U_1 : label is "1";
attribute DIV0 of U_1   : label is "1";
\end{verbatim}

- For VHDL: The backannotation tool will write out the attributes as proper vhdl types. It should be targeted to ORCA3 library to get the actual behavior of PCM.
- For more detailed information about the PCM functionality refer to the FPGA Data book, January 1998.

**FREQUENCY Attribute**

The PCM needs a FREQUENCY attribute when the PLL[BT] or PCM[BT] is used. The value of the FREQUENCY attribute is the SCLK frequency (the CLKIN frequency was used previously). In most cases, the SCLK frequency must be between 5 and 100 MHz.

See the Data sheet for valid FREQUENCY value for each of the Series 3 families of devices.

\begin{verbatim}
library IEEE,ORCA3;
use IEEE.Std_logic_1164.all;
use ORCA3.ORCACOMP.all;

entity TEST is
port (  
        D, SP, CLKIN1, CLKIN2, CD, PD, PRESETG, PRESETP: in std_ulogic;
        OFE1P3BX_Q, DLL1XB_LOCK, OFS1P3BX_Q, OFE1P3DX_Q, DLL1XT_LOCK,
        OFS1P3DX_Q: out std_ulogic);
end TEST;

architecture TEST_STR of TEST is
component DLL1XB
port (  
        \end{verbatim}
CLKIN: in std_ulogic;
    ECLK, SCLK, LOCK: out std_ulogic);
end component;
component DLL1XT
    port (    
        CLKN: in std_ulogic;
        ECLK, SCLK, LOCK: out std_ulogic);
end component;
component GSR
    port (    
        GSR: in std_ulogic);
end component;
component IBM
    port (    
        I: in std_ulogic;
        O: out std_ulogic);
end component;
component OB6
    port (    
        I: in std_ulogic;
        O: out std_ulogic);
end component;
component OFE1P3BX
    port (    
        D, SP, ECLK, PD: in std_ulogic;
        Q: out std_ulogic);
end component;
component OFE1P3DX
    port (    
        D, SP, ECLK, CD: in std_ulogic;
        Q: out std_ulogic);
end component;
component OFS1P3BX
    port (    
        D, SP, SCLK, PD: in std_ulogic;
        Q: out std_ulogic);
end component;
component OFS1P3DX
    port (    
        D, SP, SCLK, CD: in std_ulogic;
        Q: out std_ulogic);
end component;
component PUR
    port (PUR: in std_ulogic);
end component;
signal DB, SPB, CLKIN1B, CLKIN2B, CDB, PDB, GSR_1, OFE1P3BX_QB, DLL1XB_LOCKB, OFS1P3BX_QB, OFE1P3DX_QB, DLL1XT_LOCKB, OFS1P3DX_QB, ECLK1B, ECLK1T, SCLK1B, SCLK1T: std_ulogic;
begin
    IN1: IBM port map (I=>D, O=>DB);
    IN2: IBM port map (I=>SP, O=>SPB);
    IN3: IBM port map (I=>CLKIN1, O=>CLKIN1B);
    IN4: IBM port map (I=>CLKIN2, O=>CLKIN2B);
    IN5: IBM port map (I=>CD, O=>CDB);
    IN6: IBM port map (I=>PD, O=>PDB);
    IN7: IBM port map (I=>PRESETG, O=>GSR_1);
    OUT1: OB6 port map (I=>OFE1P3BX_QB, O=>OFE1P3BX_Q);
    OUT2: OB6 port map (I=>DLL1XB_LOCKB, O=>DLL1XB_LOCK);
    OUT3: OB6 port map (I=>OFS1P3BX_QB, O=>OFS1P3BX_Q);
    OUT4: OB6 port map (I=>OFE1P3DX_QB, O=>OFE1P3DX_Q);
    OUT5: OB6 port map (I=>DLL1XT_LOCKB, O=>DLL1XT_LOCK);
    OUT6: OB6 port map (I=>OFS1P3DX_QB, O=>OFS1P3DX_Q);
    G1: GSR port map (GSR=>GSR_1);
    P1: PUR port map (PUR=>PRESETP);
    OFE1P3BX1: OFE1P3BX port map (D=>DB, SP=>SPB, ECLK=>ECLK1B, PD=>PDB, Q=>OFE1P3BX_Q);
    OFE1P3DX1: OFE1P3DX port map (D=>DB, SP=>SPB, ECLK=>ECLK1T, CD=>CDB, Q=>OFE1P3DX_Q);
    OFS1P3BX1: OFS1P3BX port map (D=>DB, SP=>SPB, SCLK=>SCLK1B, PD=>PDB, Q=>OFS1P3BX_Q);
    OFS1P3DX1: OFS1P3DX port map (D=>DB, SP=>SPB, SCLK=>SCLK1T, CD=>CDB, Q=>OFS1P3DX_Q);
    DLL1XB_1: DLL1XB
        generic map (DIV0 => 1, DISABLED_GSR => ‘1’, DUTY => 50.0)
        port map (CLKIN=>CLKIN1B, ECLK=>ECLK1B, SCLK=>
SCLK1B, LOCK=>DLL1XB_LOCKB);
DLL1XT_1: DLL1XT
  generic map (DIV0 => 2, DISABLED_GSR => '0', DUTY => 40.0)
  port map (CLKIN=>CLKIN2B, ECLK=>ECLK1T, SCLK=>
  SCLK1T, LOCK=>DLL1XT_LOCKB);
end TEST_STR;

library ORCA3;
configuration TEST_STR_CON of TEST is
  for TEST_STR
    for all: DLL1XB use entity ORCA3.DLL1XB; end for;
    for all: DLL1XT use entity ORCA3.DLL1XT; end for;
    for all: GSR use entity ORCA3.GSR; end for;
    for all: IBM use entity ORCA3.IBM; end for;
    for all: OB6 use entity ORCA3.OB6; end for;
    for all: OFE1P3BX use entity ORCA3.OFE1P3BX; end for;
    for all: OFE1P3DX use entity ORCA3.OFE1P3DX; end for;
    for all: OFS1P3BX use entity ORCA3.OFS1P3BX; end for;
    for all: OFS1P3DX use entity ORCA3.OFS1P3DX; end for;
    for all: PUR use entity ORCA3.PUR; end for;
  end for;
end TEST_STR_CON;
Using the Series 3 Clock Controller Cells

The clock controller cells (CLKCNTL[BLRT]) can come from a dedicated pad or from PCMBUF. They have two inputs, CLKIN and SHUTOFF. When SHUTOFF is asserted it stops the clock after two clock cycles.

```vhdl
library IEEE, ORCA3;
use ORCA3.orcacomponent.all;
use IEEE.Std_logic_1164.all;
entity CLKCNTL is
  port (I1, I3, D, PRESETG, PRESETP: in std_ulogic;
        CLKCNTLL_O: out std_ulogic);
end CLKCNTL;
architecture CLKCNTL_STR of CLKCNTL is
  component CLKCNTLL
    port (CLKIN, SHUTOFF: in std_ulogic;
          CLKOUT: out std_ulogic);
  end component;
  component FD1S3AX
    port (D, CK: in std_ulogic;
          Q, QN: out std_ulogic);
  end component;
  component GSR
    port (GSR: in std_ulogic);
  end component;
  component IBM
    port (I: in std_ulogic;
          O: out std_ulogic);
  end component;
  component OB6
    port (I: in std_ulogic;
          O: out std_ulogic);
  end component;
  component PCMBUFB
```
port (  
    CLKIN: in std_ulogic;  
    ECLK, SCLK: out std_ulogic); 
end component;
component PUR  
port (  
    PUR: in std_ulogic); 
end component;
signal I1B, SHUTOFF_I, D4, GSR_1, PUR_1, CLKCNTLLB_I,  
CLKCNTLB_I, 
    CLKCNTLL_IX, Q1, NC1: std_ulogic; 
begin  
    IN1: IBM port map (I=>I1, O=>I1B); 
    IN3: IBM port map (I=>I3, O=>SHUTOFF_I); 
    IN4: IBM port map (I=>D, O=>D4); 
    IN5: IBM port map (I=>PRESETG, O=>GSR_1); 
    IN6: IBM port map (I=>PRESETP, O=>PUR_1); 
    PCMB: PCMBUFB port map (CLKIN=>I1B, ECLK=>CLKCNTLLB_I, SCLK=>  
        CLKCNTLB_I); 
    G1: GSR port map (GSR=>GSR_1); 
    P1: PUR port map (PUR=>PUR_1); 
    C1: CLKCNTLL port map (CLKIN=>CLKCNTLLB_I, SHUTOFF=>SHUTOFF_I,  
        CLKOUT=>  
            CLKCNTLL_IX); 
    F1: FD1S3AX port map (D=>D4, CK=>CLKCNTLL_IX, Q=>Q1, QN=>NC1);  
    OUT1: OB6 port map (I=>Q1, O=>CLKCNTLL_0); 
end CLKCNTL_STR;

library ORCA3;  
configuration CLKCNTL_STR_CON of CLKCNTL is  
    for CLKCNTL_STR  
        for all: CLKCNTLL use entity ORCA3.CLKCNTLL; end for;  
        for all: FD1S3AX use entity ORCA3.FD1S3AX; end for;  
        for all: GSR use entity ORCA3.GSR; end for;  
        for all: IBM use entity ORCA3.IBM; end for;  
        for all: OB6 use entity ORCA3.OB6; end for;  
        for all: PCMBUFB use entity ORCA3.PCMBUFB; end for;  
        for all: PUR use entity ORCA3.PUR; end for;  
    end for; 
end CLKCNTL_STR_CON;
USING ORCA 4 SPECIAL CELLS IN VHDL SIMULATIONS

Series 4 PLL Cells

The PLL is a special function block that is used to modify or condition clock signals for optimum system performance. The Series 4 PLL is composed of eight parts: four dedicated PLLs (PLL1 and PLL2) designed to meet Broadband Network applications and clocking needs, two high speed programmable PLLs, (HPPLLS) and two low speed programmable PLLs, (PPLLs) for general purpose.

Programmable PLLs have four modes of operation, BYPASS, DUTYCYCLE, PHSHIFT, and DELAY.

Module Configuration

The [H]PPLL models can be programmed in the following two ways:

1. Program the models by passing the following parameters (generics) on the HDL model instantiations.

   For example for the [H]PPLL models:

<table>
<thead>
<tr>
<th>Generics</th>
<th>Possible values</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCLKMODE</td>
<td>“BYPASS”, “DUTYCYCLE”, “PHSHIFT”, “DELAY”</td>
</tr>
<tr>
<td>NCLKMODE</td>
<td>“BYPASS”, “DUTYCYCLE”, “PHSHIFT”, “DELAY”</td>
</tr>
<tr>
<td>VCOTAP</td>
<td>0 to 7</td>
</tr>
<tr>
<td>PDELAY</td>
<td>1 to 31 (32nds of a full cycle)</td>
</tr>
<tr>
<td>DIV0</td>
<td>0 to 8 (0 bypasses the divider)</td>
</tr>
<tr>
<td>DIV1</td>
<td>0 to 8 (0 bypasses the divider)</td>
</tr>
<tr>
<td>DIV2</td>
<td>0 to 8 (0 bypasses the divider)</td>
</tr>
<tr>
<td>DIV3</td>
<td>0 to 8 (0 bypasses the divider)</td>
</tr>
</tbody>
</table>
2. You can configure the PPLL through the EPIC™ user interface from a dialog box after double clicking on a placed element. See the EPIC “Editing Series 4 Designs” chapter in the EPIC User’s Guide for more detailed instructions.

[H]PPLL Port Declarations

INPUTS: CLKN, FB

OUTPUTS : MCLK, NCLK, LOCK, INTFB

PLL[1,2] Port Declarations

INPUTS: CLKN

OUTPUTS : CLKOUT, LOCK

Synthesis Support

- For all synthesis tools, HPPLL, PPLL, and PLL are don’t touch elements.
- For VHDL, it is necessary to define each parameter and also to specify that same value as an attribute to be passed through the synthesis tool to the EDIF. Alternatively, the attribute can be added to the EDIF file by using an editor.
- Refer to the ORCA synthesis vendor manuals for examples of the syntax for the attributes that are passed by the synthesis tools as attributes to the output EDIF netlist. These manuals are the ORCA Mentor Graphics® Interface Manual and the ORCA Synplicity® Interface Manual.

For more detailed information about the PLL functionality refer to the latest FPGA Data book.

<table>
<thead>
<tr>
<th>Generics</th>
<th>Possible values</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISABLED_GSR</td>
<td>:= '0' or '1' (defaults to 1)</td>
</tr>
</tbody>
</table>
**Using the Series 4 PLL**

The following example is a netlist that illustrates the way to incorporate PLL elements in your design. *You should always use the Module/IP Manager to generate SCUBA® modules for Series 4 PLL modules.*

```vhdl
library IEEE, ORCA4;
use IEEE.Std_logic_1164.all;
use ORCA4.ORCACOMP.all;

entity TEST is
  port
  (D, SP, CLKIN1, CLKIN2, CD, PD, PRESETN: in std_ulogic;
   OFE1P3BX_Q, DLLPDB_LOCK, OFS1P3BX_Q, OFE1P3DX_Q, PLLT_LOCK,
   OFS1P3DX_Q: out std_ulogic);
end TEST;

architecture TEST_STR of TEST is
  component GSR
    port
    (GSR: in std_ulogic);
  end component;
  component IBM
    port
    (I: in std_ulogic;
     O: out std_ulogic);
  end component;
  component OB6
    port
    (I: in std_ulogic;
     O: out std_ulogic);
  end component;
  component OFE1P3BX
    port
    (D, SP, ECLK, PD: in std_ulogic;
     Q: out std_ulogic);
  end component;
  component OFE1P3DX
    port
    (D, SP, ECLK, CD: in std_ulogic;
```


Q: out std_ulogic);
end component;
component OFS1P3BX
port (  
  D, SP, SCLK, PD: in std_ulogic;  
  Q: out std_ulogic);  
end component;
component OFS1P3DX
port (  
  D, SP, SCLK, CD: in std_ulogic;  
  Q: out std_ulogic);  
end component;
signal DB, SPB, CLKIN1B, CLKIN2B, CDB, PDB, GSR_1, OFE1P3BX_QB, 
  DLLPDB_LOCKB, OFS1P3BX_QB, OFE1P3DX_QB, PLLT_LOCKB, OFS1P3DX_QB, 
  ECLK1B, ECLK2B, SCLK1B, SCLK2B, INTB1, INTB2: std_ulogic;
begin
IN1: IBM port map (I=>D, O=>DB);
IN2: IBM port map (I=>SP, O=>SPB);
IN3: IBM port map (I=>CLKIN1, O=>CLKIN1B);
IN4: IBM port map (I=>CLKIN2, O=>CLKIN2B);
IN5: IBM port map (I=>CD, O=>CDB);
IN6: IBM port map (I=>PD, O=>PDB);
IN7: IBM port map (I=>PRESETN, O=>GSR_1);
OUT1: OB6 port map (I=>OFE1P3BX_QB, O=>OFE1P3BX_Q);
OUT2: OB6 port map (I=>DLLPDB_LOCKB, O=>DLLPDB_LOCK);
OUT3: OB6 port map (I=>OFS1P3BX_QB, O=>OFS1P3BX_Q);
OUT4: OB6 port map (I=>OFE1P3DX_QB, O=>OFE1P3DX_Q);
OUT5: OB6 port map (I=>PLLT_LOCKB, O=>PLLT_LOCK);
OUT6: OB6 port map (I=>OFS1P3DX_QB, O=>OFS1P3DX_Q);
G1: GSR port map (GSR=>GSR_1);
OFE1P3BX_1: OFE1P3BX port map (D=>DB, SP=>SPB, ECLK=>ECLK1B, 
  PD=>PDB,  
  Q=>OFE1P3BX_QB);
OFE1P3DX_1: OFE1P3DX port map (D=>DB, SP=>SPB, ECLK=>ECLK2B, 
  CD=>CDB,  
  Q=>OFE1P3DX_QB);
OFS1P3BX_1: OFS1P3BX port map (D=>DB, SP=>SPB, SCLK=>SCLK1B, 
  PD=>PDB,  
  Q=>OFS1P3BX_QB);
OFSP1P3DX_1: OFSP1P3DX port map (D=>DB, SP=>SPB, SCLK=>SCLK2B, CD=>CDB,  
Q=>OFSP1P3DX_QB);
PLL1: PPLL  
generic map (DIV0 => 1, DIV1 => 3, DIV2 => 2, DIV3 => 2,  
MCLKMODE => "DELAY", NCLKMODE => "DELAY", VCOTAP => 2)  
port map (CLKIN=>CLKIN1B, FB=>INTB1, MCLK=>ECLK1B, NCLK=>  
SCLK1B, LOCK=>DLPDB_LOCKB, INTFB=>INTB1);
PLL2: PPLL  
generic map (DIV0 => 2, DIV1 => 2, DIV2 => 1, DIV3 => 3,  
MCLKMODE => "PHSHIFT", NCLKMODE => "PHSHIFT", VCOTAP => 4)  
port map (CLKIN=>CLKIN2B, FB=>SCLK2B, MCLK=>ECLK2B, NCLK=>  
SCLK2B, LOCK=>PLL_T_LOCKB, INTFB=>INTB2);
end PCM4_STR;

library ORCA4;
configuration TEST_STR_CON of TEST is  
for TEST_STR  
for all: GSR use entity ORCA4.GSR; end for;
for all: IBM use entity ORCA4.IBM; end for;
for all: OB6 use entity ORCA4.OB6; end for;
for all: OFE1P3BX use entity ORCA4.OFE1P3BX; end for;
for all: OFE1P3DX use entity ORCA4.OFE1P3DX; end for;
for all: OFS1P3BX use entity ORCA4.OFS1P3BX; end for;
for all: OFS1P3DX use entity ORCA4.OFS1P3DX; end for;
for all: PLL use entity ORCA4.PLL; end for;
end for;
end TEST_STR_CON;
SIMULATION WITH MTI ModelSim

Verify that the design functions correctly using MTI’s ModelSim:

1. Make sure that you have a copy of modelsim.ini from the $FOUNDRY/vhdl/data/orca4/mti directory in your current directory.

2. Create an MTI work directory under your current directory (in which you generated the back-annotated VHDL from ORCA), using the MTI command:
   
   \texttt{vlib work}

3. Compile the back-annotated VHDL into the work directory, using MTI’s \texttt{vcom}:
   
   \texttt{vcom <file_name>.vhd}

4. Simulate it with the SDF file using the NeoPrim VITAL library and MTI’s \texttt{vsim}:
   
   \texttt{vsim -sdfmax <file_name>.sdf -t ps <entity_name> [arch_name]}

   Note that you must simulate in picosecond resolution as the delays in the SDF are in picoseconds.
   The default delay value without SDF input is 0.1 ns.

   \begin{center}
   \textbf{CAUTION}
   \end{center}

   When simulating a testbench, make sure the SDF file is applied to the instance name of the back-annotated netlist or component.

   \texttt{vsim -sdfmax <instancename_of_design>=<design_name>.sdf -t ps}
   \texttt{<testbench_entity_name> [testbench_arch_name]}

   For a complete description of how to use the ORCA design verification tools, see the appropriate topic in the online help system. For MTI’s ModelSim refer to \textit{ModelSim/Workstation User’s Manual}.

   \begin{center}
   \textbf{Note}
   \end{center}

   For ModelSim PC simulation, all these commands should be executed from the ModelSim window on the PC.
System Bus SmartModel Simulation (Series 4)

This section describes Series 4 system bus SmartModel® simulation using VHDL.

Front End VHDL Simulation With MTI ModelSim

To perform front end VHDL simulation with MTI ModelSim, do the following:

1. Set the environment variable LMC_HOME to point to the directory where SmartModels® are installed. These are installed when ORCA is installed.

2. Create an MTI work directory in the directory in which you generated the VHDL structural netlist using the MTI command:

   `vlib work`

3. Copy the modelsim.unx (for UNIX®) or modelsim.pc (for PC) from the $FOUNDRY/vhdl/data/orca4/mti directory into your current directory as modelsim.ini.

4. Copy the SYSBUS.vhd file and sysbus_top.vhd (for UNIX) or sysbus_top.vhd.pc (for PC) from $FOUNDRY/vhdl/data/orca4/mti directory.

5. Compile the structural VHDL netlist into this work directory, using MTI's vcom:

   `vcom sysbus_top.vhd SYSBUS.vhd <file_name>.vhd`

6. Simulate it using the ORCA VHDL library and MTI's vsim:

   `vsim <entity_name> [arch_name]`

   **Note**

   Alternative the swift interface wrapper netlist `sysbus_top.vhd` can be generated by following command:

   `sm_entity -v sysbus_top > sysbus_top.vhd`

   For ModelSim PC simulation, all these commands should be executed from the ModelSim window on the PC.
Back End VHDL Simulation With MTI ModelSim

To perform back end VHDL simulation with MTI ModelSim, do the following:

1. Set the environment variable `LMC_HOME` to point to the directory where SmartModels are installed. These are installed when ORCA is installed.

2. Generate the swift interface wrapper netlist with following command:
   
   ```
   sm_entity -v sysbus_top_ts > sysbus_top_ts.vhd
   ```


4. Compile the structural VHDL netlist into this work directory, using MTI's vcom:
   
   ```
   vcom sysbus_top_ts.vhd SYSBUS_TS.vhd <file_name>.vhd
   ```

   The SmartModel `sysbus_top_ts.vhd` already has the timing numbers for speed grade -2. If you wish to simulate with a different speed grade then you must do the following:

   a. Copy the `system_top_ts_1.tf` (for -1 speed grade) or `system_top_ts.m.tf` (for minimum speed grade) from `$FOUNDRY/vhdl/data/orca4/mti` directory to your working directory as `system_top_ts_.tf` file.

   b. Set the environment variable `LMC_HOME` to point to the working directory.

   c. ModelSim will automatically pick up the new timing file for simulation.

4. Simulate it using the ORCA VHDL library and MTI's vsim:
   
   ```
   vsim <entity_name> [arch_name]
   ```
SIMULATION WITH SYNOPSYS VSS

You must have Synopsys VSS version 99.05 or later to be able to use the NeoPrim VITAL libraries. Please contact your Synopsys representative for an upgrade.

Verify functionality with timing using Synopsys VSS:

1. Copy dotsynopsys_vss.setup from the $FOUNDROY/vhdl/data/orca3/vss directory to .synopsys_vss.setup in the current directory.

2. Create a work directory in the same directory in which you generated the back-annotated VHDL from ORCA Foundry.

3. Compile the back-annotated VHDL into this work directory, using Synopsys vhdlan:

   vhdlan <file_name>.vhd

4. Simulate it with the SDF file using the NeoPrim VITAL library and Synopsys’ vhdlsim:

   vhdlsim -t ns -tres 0.1 -sdf_max -sdf <file_name>.sdf <config_name>

   For other options of vhdlsim and back-annotation with SDF, refer to the VSS Reference Manual. You can also use an include command file and -e (or -i) <filename> option to execute your simulations from a command file.

   The default delay value without SDF input is 0.0 ns.

   Currently the VHDL writer does not output initial values on RAM components. To initialize a RAM component to a value, you must write this value to memory using input vectors during simulation.
For a complete description of how to use the ORCA design verification tools, see the appropriate topic in the online help system. For a complete description of how to use Synopsys’ VSS refer to *VSS User’s Guide and VSS Reference Manual (version 99.05).*
MEMORY INITIALIZATION

Memory initialization is now possible during front-end (using ORCA components) and back-end (using NEOPRIM components) simulations. It is accomplished by using a *generic* hexadecimal string called INITVAL attached to the memory component (default state of all the memory components is zero). The INITVAL string will be loaded into memory sequentially from the most significant to the least significant bit from left to right. Following are sample netlists:

A front-end netlist from a SCUBA Module Generated in Module/IP Manager

```vhdl
-- VHDL netlist generated by SCUBA 4.2
-- Tue Nov 26 11:16:35 2001

library IEEE, ORCA;
use IEEE.std_logic_1164.all;
use ORCA.orcacomp.all;

entity meminit is
  port (waddr: in std_logic_vector(3 downto 0);
       datain: in std_logic_vector(3 downto 0);
       clk: in std_logic; wren: in std_logic;
       raddr: in std_logic_vector(3 downto 0);
       dataout: out std_logic_vector(3 downto 0));
end meminit;

architecture Structure of meminit is

-- internal signal declarations
signal scuba_vhi: std_logic;

-- local component declarations
component DCF16X2
  generic (initval : in String);
end component;
```
port (AD0: in std_logic; AD1: in std_logic;
    AD2: in std_logic; AD3: in std_logic;
    DI0: in std_logic; DI1: in std_logic;
    CK: in std_logic; WREN: in std_logic;
    WPE: in std_logic; RAD0: instd_logic;
    RAD1: in std_logic; RAD2: in std_logic;
    RAD3: in std_logic; RDO0: out std_logic;
    RDO1: out std_logic; DO0: out std_logic;
    DO1: out std_logic);
end component;
component VHI
    port (Z: out std_logic);
end component;

begin
    -- component instantiation statements
    mem_0_0_1: DCF16X2
        generic map (initval=>"0x0000130100122231")
        port map (AD0=>waddr(0), AD1=>waddr(1), AD2=>waddr(2),
            AD3=>waddr(3), DI0=>datain(2), DI1=>datain(3), CK=>clk,
            WREN=>wren, WPE=>scuba_vhi, RAD0=>raddr(0),
            RAD1=>raddr(1), RAD2=>raddr(2), RAD3=>raddr(3),
            RDO0=>dataout(2), RDO1=>dataout(3), DO0=>open,
            DO1=>open);
    scuba_vhi_inst: VHI
        port map (Z=>scuba_vhi);
    mem_0_1_0: DCF16X2
        generic map (initval=>"0x1213112020021002")
        port map (AD0=>waddr(0), AD1=>waddr(1), AD2=>waddr(2),
            AD3=>waddr(3), DI0=>datain(0), DI1=>datain(1), CK=>clk,
            WREN=>wren, WPE=>scuba_vhi, RAD0=>raddr(0),
            RAD1=>raddr(1), RAD2=>raddr(2), RAD3=>raddr(3),
            RDO0=>dataout(0), RDO1=>dataout(1), DO0=>open,
            DO1=>open);
end Structure;
A section of the back-end netlist from ngd2vhdl

(for DCF16x2 instance mem_0_0_1 from previous page)

-- VHDL netlist produced by 'ngd2vhdl' version 9_1.10
-- Created Tue Nov 26 11:51:33 1996
-- Design mapped to part or2c15as2084

-- entity mem_0_0_1_DCF16X2_SDPM

library IEEE, NEOPRIMS;
use IEEE.STD_LOGIC_1164.all;
use NEOPRIMS.neocomp.all;

entity mem_0_0_1_DCF16X2_SDPM is
    port (AD0: in Std_logic; AD1: in Std_logic;
        AD2: in Std_logic; AD3: in Std_logic;
        DI0: in Std_logic; DI1: in Std_logic;
        CK: in Std_logic; WREN: in Std_logic;
        WPE: in Std_logic; RAD0: in Std_logic;
        RAD1: in Std_logic; RAD2: in Std_logic;
        RAD3: in Std_logic; DO0: out Std_logic;
        DO1: out Std_logic; RDO0: out Std_logic;
        RDO1: out Std_logic);
end mem_0_0_1_DCF16X2_SDPM;

architecture Structure of mem_0_0_1_DCF16X2_SDPM is
    signal LOW: Std_logic;
    signal HIGH: Std_logic;
    component NEOSR16
        generic (initval: String);
        port (DIN: in Std_logic; RAD0: in Std_logic;
            RAD1: in Std_logic; RAD2: in Std_logic;
            RAD3: in Std_logic; RCLK: in Std_logic;
            WAD0: in Std_logic; WAD1: in Std_logic;
            WAD2: in Std_logic; WAD3: in Std_logic;
            WCLK: in Std_logic; WREN: in Std_logic;
            WPE: in Std_logic; SET: in Std_logic;
            RST: in Std_logic; DOUT: out Std_logic);
    end component;
    component NEOONE
ORCA/VHDL Simulation

port (VOUT: out Std_logic);
end component;

component NEOZERO
  port (VOUT: out Std_logic);
end component;

begin
  ZERO: NEOZERO
    port map (VOUT=>LOW);
  ONE: NEOONE
    port map (VOUT=>HIGH);
  RAM16X1_C: NEOSR16
    generic map (initval=>"0x0000")
    port map (DIN=>DI0, RAD0=>AD0, RAD1=>AD1, RAD2=>AD2,
               RAD3=>AD3, RCLK=>HIGH, WADO=>AD0, WAD1=>AD1,
               WAD2=>AD2, WAD3=>AD3, WCLK=>CK, WREN=>WREN,
               WPE=>WPE, SET=>LOW, RST=>LOW, DOUT=>DO0);
  RAM16X1_D: NEOSR16
    generic map (initval=>"0x0000")
    port map (DIN=>DI1, RAD0=>AD0, RAD1=>AD1, RAD2=>AD2,
               RAD3=>AD3, RCLK=>HIGH, WADO=>AD0, WAD1=>AD1,
               WAD2=>AD2, WAD3=>AD3, WCLK=>CK, WREN=>WREN,
               WPE=>WPE, SET=>LOW, RST=>LOW, DOUT=>DO1);
  RAM16X1_A: NEOSR16
    generic map (initval=>"0x0d23")
    port map (DIN=>DI0, RAD0=>RAD0, RAD1=>RAD1, RAD2=>RAD2,
               RAD3=>RAD3, RCLK=>HIGH, WADO=>AD0, WAD1=>AD1,
               WAD2=>AD2, WAD3=>AD3, WCLK=>CK, WREN=>WREN, WPE=>WPE,
               SET=>LOW, RST=>LOW, DOUT=>RDO0);
  RAM16X1_B: NEOSR16
    generic map (initval=>"0x041e")
    port map (DIN=>DI1, RAD0=>RAD0, RAD1=>RAD1, RAD2=>RAD2,
               RAD3=>RAD3, RCLK=>HIGH, WADO=>AD0, WAD1=>AD1,
               WAD2=>AD2, WAD3=>AD3, WCLK=>CK, WREN=>WREN, WPE=>WPE,
               SET=>LOW, RST=>LOW, DOUT=>RDO1);
end Structure;
USING GSR/PUR IN VHDL

Programmable GSR Support (Series 3)

Programmable GSR support allows the user to disable the GSR functionality on selected register elements. This feature gives the user the ability to retain design information on a PFU basis when using GSR to set/reset the rest of design.

The PUR component will now be used to set/reset register elements at power-up. GSR is used for global set/reset at all other times of operation. The PUR signal will need to be instantiated in your HDL design since it is currently not inferred.

In the current version of ispLEVER, programmable GSR is supported via an attribute attached to instantiated library cell instances in the HDL design netlist. The attribute is "DISABLED_GSR" with value "1". The resulting synthesized output EDIF netlist will contain the attribute "DISABLED_GSR" with value "1" for each component instance containing the attribute. The ORCA tools will recognize this attribute and disable the GSR functionality accordingly.

Example VHDL to implement programmable GSR:

```vhdl
library IEEE, ORCA3;
use IEEE.Std_logic_1164.all;
use ORCA3.ORCACOMP.all;

entity TEST is
  port ( 
    D0, D1, SP, CK, SD, PD, CD, PRESETG, PRESETP: in std_ulogic;
    FL1S1B_Q, FL1S1D_Q, FL1P3BX_Q, FL1P3DX_Q: out std_ulogic);
end TEST;

architecture TEST_STR of TEST is
  signal D0B, D1B, SPB, CKB, SDB, PDB, CDB, GSR_1, FL1S1B_QB,
  FL1S1D_QB, FL1P3BX_QB, FL1P3DX_QB, FL1S1B_QNB, FL1S1D_QNB, FL1P3BX_QNB,
  FL1P3DX_QNB: std_ulogic;
begin
  IN1: IBM port map (I=>D0, O=>D0B);
  IN2: IBM port map (I=>D1, O=>D1B);
  IN3: IBM port map (I=>SP, O=>SPB);
  IN4: IBM port map (I=>CK, O=>CKB);
end TEST;
```
IN5: IBM port map (I=>SD, O=>SDB);
IN6: IBM port map (I=>PD, O=>PDB);
IN7: IBM port map (I=>CD, O=>CDB);
IN8: IBM port map (I=>PRESETG, O=>GSR_1);
OUT1: OB6 port map (I=>FL1S1B_QB, O=>FL1S1B_Q);
OUT2: OB6 port map (I=>FL1S1D_QB, O=>FL1S1D_Q);
OUT3: OB6 port map (I=>FL1P3BX_QB, O=>FL1P3BX_Q);
OUT4: OB6 port map (I=>FL1P3DX_QB, O=>FL1P3DX_Q);
G1: GSR port map (GSR=>GSR_1);
P1: PUR port map (PUR=>PRESETP);
FL1S1B1: FL1S1B
    generic map (DISABLED_GSR => "1")
    port map (D0=>D0B, D1=>D1B, CK=>CKB, SD=>SDB, PD=>
PDB, Q=>FL1S1B_QB, QN=>FL1S1B_QNB);
FL1S1D1: FL1S1D port map (D0=>D0B, D1=>D1B, CK=>CKB, SD=>SDB,
CD=>
    CDB, Q=>FL1S1D_QB, QN=>FL1S1D_QNB);
FL1P3BX1: FL1P3BX port map (D0=>D0B, D1=>D1B, SP=>SPB,
CK=>CKB, SD=>
    SDB, PD=>PDB, Q=>FL1P3BX_QB, QN=>FL1P3BX_QNB);
FL1P3DX1: FL1P3DX
    generic map (DISABLED_GSR => "1")
    port map (D0=>D0B, D1=>D1B, SP=>SPB, CK=>CKB, SD=>
    SDB, CD=>CDB, Q=>FL1P3DX_QB, QN=>FL1P3DX_QNB);
end TEST_STR;

library ORCA3;
configuration TEST_STR_CON of TEST is
    for TEST_STR
        for all: FL1P3BX use entity ORCA3.FL1P3BX; end for;
        for all: FL1P3DX use entity ORCA3.FL1P3DX; end for;
        for all: FL1S1B use entity ORCA3.FL1S1B; end for;
        for all: FL1S1D use entity ORCA3.FL1S1D; end for;
        for all: GSR use entity ORCA3.GSR; end for;
        for all: IBM use entity ORCA3.IBM; end for;
        for all: OB6 use entity ORCA3.OB6; end for;
        for all: PUR use entity ORCA3.PUR; end for;
    end for;
end TEST_STR_CON;
Alternatively, GSR can be used to initialize all the sequential elements in your design to a known state without having to instantiate the GSR component. This can be attained from an MTI command file during front-end simulations by the following

```force global/gsrnet 0 @0 ns, 1 @10 ns```

and back-end simulations by

```force gsrnet 0 @0 ns, 1 @10 ns```

Note

GSRNET has to be held LOW long enough for the entire design to initialize accommodating delay through individual gates.
USING TSALL IN VHDL

A TSALL component can be instantiated in the front-end VHDL netlist and driven using an external pin. This will tristate all the output buffers. For example in the netlist below, the TS input drives the TSALL component which is by default active LOW. Therefore, the TS input will tristate all the output buffers when it is LOW.

Note

The TSALL connection to all the output buffers is implicit in the front-end netlist, but explicit in the back-end netlist.

library IEEE, ORCA;
use IEEE.Std_logic_1164.all;
use ORCA.orcacomp.all;
entity TRI is
  port (TS, I15, I22, I30: in std_ulogic;
       OB6_Z, OBZ6_Z : out std_ulogic);
end TRI;

architecture TRI_STR of TRI is
  component AND2
    port (A, B: in std_ulogic;
          Z: out std_ulogic);
  end component;
  component IBM
    port (I: in std_ulogic;
          O: out std_ulogic);
  end component;
  component OB6
    port (I: in std_ulogic;
          O: out std_ulogic);
  end component;
  component OBZ6
    port (O: out std_ulogic);
  end component;
I, T: in std_ulogic;
O: out std_ulogic);
end component;
component TSALL
port (
    TSALL: in std_ulogic);
end component;
signal TSALL_1, OB6_I, OB6_IX, OBZ6_I, OBZ6_IX, OB_T,
    OB_TX : std_ulogic;
beg
IN99: IBM port map (I=>TS, O=>TSALL_1);
T1: TSALL port map (TSALL=>TSALL_1);
IN3: IBM port map (I=>I15, O=>OB6_I);
G3: AND2 port map (A=>OB6_I, B=>OB6_I, Z=>OB6_IX);
OB6_1: OB6 port map (I=>OB6_IX, O=>OB6_Z);
IN7: IBM port map (I=>I22, O=>OB_T);
IN12: IBM port map (I=>I30, O=>OBZ6_I);
G4: AND2 port map (A=>OB_T, B=>OB_T, Z=>OB_TX);
G7: AND2 port map (A=>OBZ6_I, B=>OBZ6_I, Z=>OBZ6_IX);
OBZ6_1: OBZ6 port map (I=>OBZ6_IX, T=>OB_TX, O=>OBZ6_Z);
end TRI_STR;
library ORCA;
configuration TRI_STR_CON of TRI is
for TRI_STR
    for all: AND2 use entity ORCA.AND2; end for;
    for all: IBM use entity ORCA.IBM; end for;
    for all: OB6 use entity ORCA.OB6; end for;
    for all: OBZ6 use entity ORCA.OBZ6; end for;
    for all: TSALL use entity ORCA.TSALL; end for;
end for;
end TRI_STR_CON;

Alternatively, TSALL can be used to tristate all the output buffers in your design to a
known state without having to instantiate the TSALL component. This can be
accomplished by driving either global/tsall_net (active LOW) in the front-end or tsallnet
(active HIGH) at the back-end.
ILLEGAL NAMES/CHARACTERS IN ORCA DESIGN FLOWS

Certain names and characters used in a design may cause problems at some point in your ORCA design flow. See *Illegal Names and Characters* in the *ORCA Macro Library* section of the *ORCA Libraries Manual* for a list of names and characters to avoid.
B
Back end VHDL
Simulation with MTI ModelSim, 30

C
Commands
ldban, 12
ngd2vhd, 9
ngdanno, 9

D
Design
environment setting, 2
simulation
VHDL Neoprim model (2CA/2TA), 9
VHDL Neoprim model (Series 3), 12
VHDL ORCA model, 6
with MTI V-System, 28
with Synopsys VSS, 31
Directory structure, 3

E
Environment variables, 2

F
Figures
Simulation flow, 5

G
GSR
instantiating in VHDL, 37
used to initialize all sequential elements, 39

I
Installation instructions
PC, 4
workstation, 3
MTI V-System, 4
Instantiating
GSR in VHDL, 37
PUR in VHDL, 42
TSALL in VHDL, 40

L
ldban (command), 12

M
Memory initialization, 33
ModelSim
system bus simulation (Series 4) using VHDL, 29
Module configuration
PPLL (Series 4), 23
MTI ModelSim
Back End VHDL Simulation with, 30
Front End VHDL Simulation with, 29
MTI V-System
installation
PC, 4
workstation, 4
simulation, 28

N
ngd2vhd (command), 9
ngdanno (command), 9

P
PLL
port declarations (Series 4), 24
synthesis support for, 24
PLL (Series 4)
using in Verilog simulation, 23
PPLL
module configuration (Series 4), 23
port declarations (Series 4), 24
synthesis support for, 24
PUR
instantiating in VHDL, 42

R
Requirements
software, 2

S
Setting
design environment, 2
SIMULATION
Front End VHDL with MTI ModelSim, 29
Simulation
   flow
   (figure), 5
   Front End VHDL with MTI ModelSim, 30
   VHDL Neoprim model (2CA/2TA), 9
   VHDL Neoprim model (Series 3), 12
   VHDL ORCA model, 6
   with MTI V-System, 28
   with Synopsys VSS, 31
Software requirements, 2
Synopsys
   VSS
   simulation, 31
System bus (Series 4)
   simulation with MTI ModelSim, 29

T
TSALL
   instantiating in VHDL, 40
   used to tristate output buffers to a known state, 41

V
VHDL
   Back end simulation with MTI ModelSim
      (Series 4), 30
   Front end simulation with MTI ModelSim
      (Series 4), 29
   Neoprim model simulation (2CA/2TA), 9
   Neoprim model simulation (Series 3), 12
   ORCA model simulation, 6
   simulation
      using Neoprim models (2CA/2TA), 9
      using Neoprim models (Series 3), 12
      using ORCA models, 6
   system bus simulation (Series 4), 29