ORCA®
Mentor Graphics™ Interface Manual

For Use With
Leonardo Spectrum™ Version 2002a or higher, ORCA 4.0, and ispLEVER 2.0 or higher

Technical Support Line: 1-800-LATTICE or 408-826-6002 (international)
Mentor Graphics Interface Manual

LeonardoSpectrum is a trademark of Mentor Graphics Company. ORCA is a registered trademark of Lattice Semiconductor Corporation. Verilog is a registered trademark of the Cadence Design Systems, Inc. All other brands or product names are the trademarks or registered trademarks of their respective owners.
# ORCA/Mentor Graphics 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>ORCA/LEONARDO SPECTRUM LIBRARY FILES</td>
<td>3</td>
</tr>
<tr>
<td>DESIGN FLOW</td>
<td>3</td>
</tr>
<tr>
<td>Series 4 Synthesis Flow</td>
<td>5</td>
</tr>
<tr>
<td>DESIGN ENTRY</td>
<td>7</td>
</tr>
<tr>
<td>SYNTHESIS AND OPTIMIZATION FEATURES</td>
<td>9</td>
</tr>
<tr>
<td>Instantiating Components in VHDL</td>
<td>10</td>
</tr>
<tr>
<td>Instantiating Components in Verilog</td>
<td>11</td>
</tr>
<tr>
<td>Using Macros</td>
<td>11</td>
</tr>
<tr>
<td>Adding I/O Buffers</td>
<td>13</td>
</tr>
<tr>
<td>1. Letting Leonardo Spectrum Add I/O</td>
<td>13</td>
</tr>
<tr>
<td>2. Instantiating Specific I/O Buffers</td>
<td>13</td>
</tr>
<tr>
<td>3. Adding I/O via the Control File</td>
<td>15</td>
</tr>
<tr>
<td>Locking I/O Pins</td>
<td>15</td>
</tr>
<tr>
<td>1. Locking I/O in the Code</td>
<td>15</td>
</tr>
<tr>
<td>2. Locking I/O in the Preference File</td>
<td>16</td>
</tr>
<tr>
<td>3. Setting the pin location in the Leonardo Spectrum GUI</td>
<td>16</td>
</tr>
<tr>
<td>Handling Multiple Verilog Files</td>
<td>16</td>
</tr>
<tr>
<td>Handling Multiple VHDL Files</td>
<td>18</td>
</tr>
<tr>
<td>Inferring GSR</td>
<td>19</td>
</tr>
<tr>
<td>List of Modgen Operators</td>
<td>19</td>
</tr>
<tr>
<td>PASSING PREFERENCES</td>
<td>20</td>
</tr>
<tr>
<td>APPENDIX</td>
<td>22</td>
</tr>
<tr>
<td>Using DIN/DOUT Properties</td>
<td>22</td>
</tr>
<tr>
<td>Using DIN/DOUT with Mentor Graphics Leonardo Spectrum</td>
<td>23</td>
</tr>
<tr>
<td>Memory Initialization</td>
<td>25</td>
</tr>
</tbody>
</table>
Illegal Names and Characters ............................................................. 27
Programmable GSR Support (Series 3) .............................................. 27
Passing FREQUENCY Preferences (Series 3 and 4) ......................... 30
OVERVIEW

This manual describes the Optimized Reconfigurable Cell Array (ORCA) Library and the interface between the Mentor Graphic’s design tools and the ORCA place and route tools. Together, the tools provide a powerful and integrated high-level design environment for ORCA Field-Programmable Gate Arrays (FPGA).

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 creating designs with ORCA. Check with technical support with compatibility and support issues.
SOFTWARE REQUIREMENTS

The ORCA/Mentor Graphics™ interface is compatible with the following software:

- Latest version of ispLEVER software with ORCA devices installed.
- Synplify version 7.3 or later

SETTING THE DESIGN ENVIRONMENT

This section helps you customize your ORCA 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 FOUNDRY environment variable is set. The FOUNDRY variable points to the ispFPGA (PC) or ispfpga (UNIX) directory.

The variable should be set as follows:

$ setenv FOUNDRY <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.
ORCA/LEONARDO SPECTRUM LIBRARY FILES

Library files in Leonardo Spectrum are updated in their software synchronously with ORCA updates and should not pose any issues for synthesis.

For updated library files for Verilog in the ispLEVER environment, you can retrieve the files in the userware directory after ispLEVER installation. Refer to the complete paths below for PC and UNIX.

Verilog header files:

$FOUNDORY\userware\NT\4E_VERILOG_HEADERS\orca4_leonardo.v
$FOUNDORY/userware/unix/4E_Verilog_Headers/orca4_leonardo.v

Note

We do not provide updated VHDL header files as these updates are encoded in their software in a synchronous fashion. There may be some cases where Verilog header files are needed.

DESIGN FLOW

This section describes the interface between Synplicity’s Synplify and ORCA. The interface allows you to:

• Synthesize an ORCA design using Leonardo Spectrum. You can use the integrated version of Leonardo Spectrum in ispLEVER’s Project Navigator. Use the Tools > LeonardoSpectrum Synthesis command to open the program.

• In Project Navigator, set up an ORCA FPGA project and import your source file.

• In Project Navigator, run your ORCA project’s imported source file through the Place & Route Design process in the Process window. Right click on the process and click Start.

• In Project Navigator, run the Generate Bitstream process in the Process window. Right click on the process and click Start.

Note

You can also run the design flow from the command line.
The design flow for generating logic designs with Mentor Graphics and ORCA is shown in Figure 1 on page 4.

Mentor Graphics Design Flow

1. Module/IP Manager
2. Create HDL Design
3. ORCA SCUBA Library
4. ORCA Cell Library
5. Select and Load Library
6. Read Design into Leonardo Spectrum
7. Set Constraints
8. Optimize Logic & Technology Map
9. Establish Area/Timing Goals
10. Generate EDIF Netlist
11. ispLEVER ORCA DESIGN FLOW
12. MAP DESIGN (map)
13. MAP TRACE REPORT (trce)
14. PLACE & ROUTE TRACE REPORT (par)
15. Netlist Writer (ngd2vhd/ ngd2ver)
16. Write VHDL, Verilog SDF Netlists
17. GENERATE BITSTREAM (bitgen)
18. Write .v, .vhd, .sdf Files
19. ORCA Device
Series 4 Synthesis Flow

This section contains any information that is necessary for performing design synthesis using VHDL and Verilog HDL for Series 4 designs.

Series 4 library elements, for example, Block RAMs (BR512X18), Programmable PLLs (PPLL), and LVDS buffers, should be fully supported by most synthesis vendors; however, there is some user intervention required for instantiation and assigning any properties to these elements. For updated library files for Verilog and VHDL, retrieve the orca4_leonardo.v from the $FOUNDRY/userware/nt | sol/4E_VERILOG_HEADERS directory.

Actions necessary to ensure proper instantiation and for assigning properties for Series 4 elements are described in the following subsections.

Using VHDL Through Synthesis

If some Series 4 elements are not yet supported by Mentor Graphics, VHDL designs may require these elements to be defined as black boxes or macros during synthesis.

To enter properties for Series 4 elements using VHDL, the syntax structures are the same for each of the vendors. An example of assigning properties for Mentor Graphics is provided below.

VHDL syntax for use with Mentor Graphics:

\[
\text{attribute initval: string;}
\]

\[
\text{attribute initval of mem\_0\_0\_15 : label is “0x00000000000000000000000000000000”};
\]

Using Verilog Through Synthesis

When you instantiate Series 4 elements in Verilog HDL, synthesis tools may issue warning or error messages. To avoid this, we have provided Verilog header files, which contain port definitions for new Series 4 elements. These files are provided for all the synthesis tools. For Mentor Graphics/Leonardo Spectrum it is located in the file path shown as:

\[
$FOUNDRY/userware/nt | sol/4E_VERILOG_HEADERS/orca4_leonardo.v
\]
Depending on what tool is being used, you should add proper header file along with other Verilog files to the project.

To enter attributes for Series 4 elements using Verilog, follow the Verilog syntax for use with Mentor Graphics shown below:

```verilog
// exemplar begin

// exemplar attribute mem_0_0_15 initval 0x00000000000000000000000000000000
// exemplar attribute mem_0_1_14 initval 0x0123456789ABCDEFFEDCBA9876543210
// exemplar attribute mem_0_1_14 initval 0x0123456789ABCDEFFEDCBA9876543210

// exemplar end
```
DESIGN ENTRY

The following steps outline how to synthesize ORCA designs with Mentor Graphics design tools and ORCA:

1. Create a design in Verilog HDL or in VHDL. The design may be technology-independent or contain ORCA-specific functions and components; however, it cannot contain instances of functions from other technology libraries. If it contains ORCA cells, then you must specify the appropriate source library. See the Synplify online help system for a list of names and characters that should not be used for components, nets, or sites (instances).

2. (Optional) Verify that the design description is correct by simulating the HDL description.

3. Use ispLEVER’s Module/IP Manager to create a SCUBA® (Synthesis Compiler for User programmaBle Arrays) module, which is a parameterized module generator optimized for ORCA FPGAs. The output from Module/IP Manager can be in EDIF, VHDL, or Verilog. Since the modules generated are optimized for the ORCA architecture, they provide speed and area benefits over synthesis. The output from Module/IP Manager (VHDL or Verilog only) can be directly read into Leonardo Spectrum or can be integrated with the user design as a sub-module. For a complete description of Module/IP Manager, refer to the Module/IP Manager online help system in ispLEVER.

   **Note**

   You can run SCUBA from the command line; however, we recommend using the Module/IP Manager as options can contain lengthy arguments for more complex modules.

4. Read the design into Leonardo Spectrum. Select the appropriate ORCA target library depending on the target FPGA (ORCA 2CA/2TA, ORCA 3C/3T). Use of Module/IP Manager ORCA modules provides faster and better optimization for most datapath designs.

To see a full usage list for Leonardo Spectrum, type

```
   gc -help
```

or

```
   gc -batchhelp (For running in Batch Mode)
```
5. Insert I/O buffers on all the external ports in the design using CHIP mode (necessary before taking the design into an ispLEVER ORCA flow).

Note

If the buffers are instantiated in the source, choose MACRO mode.

6. Use Spectrum to synthesize the design to meet the desired area target and/or desired timing constraints. The synthesized EDIF design can be viewed under Leonardo Spectrum level 3 as an RTL schematic, a Technology Schematic, or a schematic of only the Critical Path.
SYNTHESIS AND OPTIMIZATION FEATURES

Mentor Graphics uses an ORCA-specific, “fan-in limited” synthesis and optimization algorithm to take advantage of the look-up-table logic of ORCA PFUs. ORCA-specific options are entered in the GUI under specific tabs. ORCA-specific options are as follows:

• **Under the “Technology” tab**
  
  In the “Technology Settings” window, select the appropriate family (2CA/2TA or 3C/CT). Select the part and speed grade. The “Max Fanout” can be left blank or set to the desired value. The method for assigning GSR is also specified on this page.

  In the “Advanced Settings” window, the user specifies whether to write ORCALUT symbols (recommend choosing this option). The user can also specify that certain FF’s and synchronous modules be excluded. For the 2CA/2TA, the user can specify the use of b-input LUTS.

• **Under the “Input” tab**
  
  In the window under the “Input Files” tab, the user specifies the directory and all of the HDL files [VHDL is order dependent]. For most applications, the “Onehot” encoding style for FSMs will give the best results.

• **Under the “Constraints’ tab**
  
  Under the “Module” tab the user can specify “Don’t Touch” modules.

• **Under the “Optimize tab**
  
  The user can specify several “Advanced Optimization Settings”.

• **Under the “output” tab**
  
  The user can specify the output format. The user should select “EDIF” as the input to ORCA.
Instantiating Components in VHDL

The following code provides a way to instantiate ORCA library components into a design. Note that the pin order information is specified in the ORCA library manual. This code instantiates a CU4P3DX (4-bit up-counter).

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity reg is
  port(   CLOCK:          in std_logic;
          CLK_EN:         in std_logic;
          CARRY_IN:       in std_logic;
          ASYNC_CLEAR:    in std_logic;
          COUNT:          out std_logic_vector(3 downto 0);
          CARRY_OUT:      out std_logic);
end reg;
architecture behave of reg is
component cu4p3dx port (  
  CI:                     in std_logic;
  SP:                     in std_logic;
  CK:                     in std_logic;
  CD:                     in std_logic;
  CO:                     out std_logic;
  Q0:                     out std_logic;
  Q1:                     out std_logic;
  Q2:                     out std_logic;
  Q3:                     out std_logic);
end component;
begin
  U1:     cu4p3dx port map (  
    CI      => CARRY_IN,
    SP      => CLK_EN,
    CK      => CLOCK,
    CD      => ASYNC_CLEAR,
    CO      => CARRY_OUT,
    Q0      => COUNT(0),
    Q1      => COUNT(1),
    Q2      => COUNT(2),
    Q3      => COUNT(3));
end behave;
```
Instantiating Components in Verilog

The following code provides a way to instantiate ORCA library components into a design. This code instantiates a CU4P3DX (4-bit up-counter).

```verilog
module reg1 (CLOCK, CLK_EN, CARRY_IN, ASYNC_CLEAR, COUNT, CARRY_OUT);
  input CLOCK;
  input CLK_EN;
  input CARRY_IN;
  input ASYNC_CLEAR;
  output [3:0] COUNT;
  output CARRY_OUT;
  cu4p3dx  U1  (.CI(CARRY_IN), .SP(CLK_EN), .CK(CLOCK),
                   .CD(ASYNC_CLEAR), .CO(CARRY_OUT), .Q0(COUNT[0]), .Q1(COUNT[1]),
                   .Q2(COUNT[2]), .Q3(COUNT[3]));
endmodule

module cu4p3dx  (CI,SP,CK,CD,CO,Q0,Q1,Q2,Q3);
  input CI,SP,CK,CD;
  output CO,Q0,Q1,Q2,Q3;
endmodule
```

Using Macros

In many cases it is desirable to add a custom macro to the design in either VHDL or Verilog. The actual instantiation of the macro is the same as for one the ORCA library components, but there are additional steps to take.

1. Create the macro in the EPIC™ Device Editor.

2. Run the following command at the DOS prompt from your project directory:

   ```sh
cmpread <macroname>.nmc > <macroname>.txt
```

   where `<macroname>` is the name of the macro. This will convert the NMC file to ASCII to obtain the pin order of the macro. The `ncdread` program utility is located in the `$FOUNDRY/bin/nt | sol` path.

3. Look at the text file. It will look something like:

```text
Loading macro from file "TEST.NMC".
```
Loading device for application ncdread from file
'att2c04.nph' in environment
C:/ATTORCA.

NC_MACRODEF <test>
  1 comp(s) -
    comp 0 - $COMP_0
  10 external pin(s) :
    pin <  INTADDRD>
    pin <  INTADDRC>
    pin <  INTADDRB>
    pin <  INTADDRA>
    pin <  ENABLE>
    pin <    CLK>
    pin < ADDR_REGA>
    pin < ADDR_REGB>
    pin < ADDR_REGC>
    pin < ADDR_REGD>
reference comp <$COMP_0>
relplaceinfo -
    comp <$COMP_0> at <2,84> is placed.
relrouteinfo -
  relative routing information exists.

This is the pin order for the macro which is required in the instantiation.

4. Instantiate the macro in your code as you would any other component.

5. Run through Leonardo Spectrum as usual.

6. Run ORCA mapper. If the macro is located anywhere other than in your current
working directory (such as a macro library directory), then add that path to the search
path located under the SEARCH button.

7. Process the design as usual.
Adding I/O Buffers

There are three ways to add I/O buffers to your design. Each method is described below.

1. **Letting Leonardo Spectrum Add I/O**

Leonardo Spectrum will automatically add I/O to a design when the “Add I/O Pads” option is selected under the “Optimize-Optimize” tab. The following buffers will be used:

- IBM: CMOS input buffer
- OB6: 6ma sink output buffer
- OBZ6: 6ma sink output buffer - tristate
- BMZ6: CMOS input, 6ma sink tristate, bidirectional buffer

2. **Instantiating Specific I/O Buffers**

I/O may be instantiated in the VHDL or Verilog code directly. The following two sections provide examples for VHDL and Verilog.

**VHDL**

```vhdl
entity reg is
    port(CLOCK:  in std_logic;
         CLK_EN: in std_logic;
         IN0:    in std_logic;
         IN1:    in std_logic;
         OUT0:   out std_logic;
         OUT1:   out std_logic);
end reg;

architecture behave of reg is
  component ob12f port (  
    I:           in std_logic;
    O:           out std_logic);
end component;
  component ibt port (  
    I:           in std_logic;
    O:           out std_logic);
end component;
  signal ADDR0: std_logic;
  signal ADDR1: std_logic;
```
signal CLK_1: std_logic;
begin
  U1: ob12f port map (I => ADDR0, O => OUT0);
  U2: ob12f port map (I => ADDR1, O => OUT1);
  U3: ibt port map (I => CLOCK, O => CLK_1);
end behave;

Verilog

module reg1 (CLOCK, CLK_EN, CARRY_IN, ASYNC_CLEAR, COUNT,
             CARRY_OUT);
  input CLOCK
  input CLK_EN;
  input CARRY_IN;
  input ASYNC_CLEAR;
  output [3:0] COUNT;
  output CARRY_OUT;
  wire CLOCK_IN;

  cu4p3dx U1 (.CI(CARRY_IN), .SP(CLK_EN), .CK(CLOCK_IN),
               .CD(ASYNC_CLEAR), .CO(CARRY_OUT),
               .Q0(COUNT[0]), .Q1(COUNT[1]), .Q2(COUNT[2]),
               .Q3(COUNT[3]));
  ibt     U2  (.I(CLOCK),.O(CLOCK_IN));
endmodule

module cu4p3dx (CI,SP,CK,CD,CO,Q0,Q1,Q2,Q3);
  input CI,SP,CK,CD;
  output CO,Q0,Q1,Q2,Q3;
endmodule

module ibt (I,O);
  input I;
  output O;
endmodule
3. Adding I/O via the Control File

I/O may be added through the use of the Mentor Graphics Control file. The control file is an ASCII file that contains the GATE command. Here is an example:

    GATE IBT (I=CLOCK_PAD, O=CLOCK)

This command will add a TTL input buffer (IBT) for the signal CLOCK. The control file will override the CHIP mode defaults.

Locking I/O Pins

There are three ways to lock I/O using the ORCA/Mentor Graphics design flow. Each method is discussed below with advantages/disadvantages.

1. Locking I/O in the Code

The first method to lock I/O using Leonardo Spectrum is through the attribute statement. Here is a section of code in VHDL that describes the locking of the signal CLOCK:

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity reg is
    port(CLOCK: in std_logic;
         CLK_EN: in std_logic;
         IN0:    in std_logic;
         IN1:    in std_logic;
         IN2:    in std_logic;
         IN3:    in std_logic;
         OUT0:   out std_logic;
         OUT1:   out std_logic;
         OUT2:   out std_logic;
         OUT3:   out std_logic);
    attribute loc:string;
    attribute loc of CLOCK:signal is "91";
end reg;
```

**Advantage**

The signal name is the actual port name in your code.
Disadvantage

When locking I/O using the attribute statement in VHDL, you cannot lock the I/O associated with STD_LOGIC_VECTORs. These busses must be locked in the ORCA preference file.

2. Locking I/O in the Preference File

The second method is to lock the I/O in the ORCA preference file. The syntax for this is:

LOCATE COMP "X1" SITE "91";

Disadvantage

After running through synthesis, the signal names usually change. Therefore, the designer would need to check either the EDIF file or the mapped NCD file for the names. To check the NCD file, the file needs to be converted to ASCII. The command to do this is:

ncdread mapped.ncd > mapped.txt

where mapped.ncd is the NCD file to convert and mapped.txt is the text output of the NCD. The ncdread program utility is located in the $FOUNDRACT/bin/nt | sol path.

3. Setting the pin location in the Leonardo Spectrum GUI

The pin location can be set by entering a pin location under the Constraints - Input tab. Under the “Input Constraint” section there is a place for the user to enter a “Pin Location.”

Advantages

• Can be saved with the project
• Easy, hierarchical entry method
• Can lock bus elements individually

Disadvantage

Not associated with the HDL

Handling Multiple Verilog Files
To enter multiple Verilog files into Leonardo Spectrum, you can also add an ‘include’ statement to the Verilog code at the top level, or enter multiple files in the “Open Files” window under the “Input” tab. Note that this statement must be AFTER the ‘end module’ statement. Add a line for each Verilog file. Provide either the full path name, or set the Mentor Graphics properties to the current working directory.

```verilog
include "filename.v"
```
Handling Multiple VHDL Files

When you are trying to synthesize multiple VHDL files there are a few options to do this.

- Option 1 (not ideal for many files): Within the Leonardo Spectrum main window, under Input tab enter the top level design and path. Within this window, enter each of the VHDL files in the “Open Files” window.
- Option 2 (better suited for synthesizing a large number of VHDL files):
  1. Create a text file (for example named files.cmd) and specify all the files except the top level with the following syntax.

    -VHDL_FILE=<pathname>filename.VHD

    where the full path name should be specified

  2. Within the main Explorer window, choose the top level file in the input section (i.e., Directory and filename).

  3. Select Control Files and in the Command Files field enter the path and filename created in the step above.
Inferring GSR

- The tool will automatically try to infer a GSR only if EVERY flip-flop in the module being compiled has a defined asynchronous set or reset signal attached. Otherwise, it is best to explicitly name the net that is the GSR signal by selecting “Auto” under the “Technology” tab as the value for “Assign GSR”.

- The GSR net must have a port in the top level entity and everywhere in the design.

- Sample VHDL code where GSR can be inferred automatically. Note with $GL=preset$
  the following would infer FD1S3AY and GSR components. Without the options (or by turning off GSR inferring) a FD1S3BX is inferred.

```vhdl
if (preset='1') then
    q <= '1';
elsif (phi'event and phi='1') then
    q <= d;
end if;
```

List of Modgen Operators

Leonardo Spectrum supports the following VHDL operators for Module Generation. Modgen provides faster and better optimization of structured logic than the algorithms used for random logic. Implementations of ORCA 2CA/2TA or ORCA 3C/3T specific structured logic are inferred from the device independent code.

<table>
<thead>
<tr>
<th>Logical Operators</th>
<th>and</th>
<th>or</th>
<th>nand</th>
<th>nor</th>
<th>xor</th>
<th>xnor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational Operators</td>
<td>=</td>
<td>/=</td>
<td>&lt;</td>
<td>&lt;=</td>
<td>&gt;</td>
<td>&gt;=</td>
</tr>
<tr>
<td>Shift Operators</td>
<td>sll</td>
<td>srl</td>
<td>sla</td>
<td>sra</td>
<td>rol</td>
<td>ror</td>
</tr>
<tr>
<td>Adding Operators</td>
<td>+</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiply Operators</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Misc. Operators</td>
<td>+1</td>
<td>-1</td>
<td>(x)</td>
<td>abs()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Elements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PASSING PREFERENCES

ORCA offers an extensive list of useful preferences and ORCA attributes. Refer to the ORCA Attributes Desk Reference, the ORCA Attributes help topic in the ispLEVER online help system and to the Leonardo Spectrum Reference Manual for details on how to use them.

Timing, placement, and other preferences and properties can be passed to ORCA using one of two techniques.

• VHDL attributes
• Leonardo Spectrum’s control to preference file translator

The following is a simple example of how preferences and attributes can be passed to ORCA using VHDL attributes:

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
library exemplar;
use exemplar.exemplar_1164.all;

entity gsrex is
  port (phi, preset, d : in std_logic;
        q : out std_logic);
  attribute loc : string;
  attribute loc of phi: signal is "56";
  attribute frequency : string;
  attribute frequency of phi: signal is "33MHz";
end gsrex;

architecture orca of gsrex is
begin
  process (phi, preset)
  begin
    if (preset='1') then
      q <= '1';
    elsif (phi'event and phi='1') then
      q <= d;
    end if;
  end process;
end orca;
```
end process;
end orca;

Leonardo Spectrum also has the ability to automatically translate some Mentor Graphics control file information to the equivalent ORCA preference file syntax using the -PREF[erence] option. An example control file for the VHDL netlist on the previous page would be:

```
CLOCK_CYCLE 20 PHI
PULSE_WIDTH 10 PHI
ARRIVAL_TIME 35 D
REQUIRED_TIME 20 Q
```

The equivalent preference file output from Leonardo Spectrum is:

```
SCHEMATIC START ;

SCHEMATIC END ;
# USER Defined Preferences...
period net PHI_int 20.000000 ns high 10.000000 ns;
offset IN comp D 35.000000 ns AFTER comp PHI;
offset OUT comp Q 20.000000 ns AFTER comp PHI;
```

<table>
<thead>
<tr>
<th>*.ctr Syntax</th>
<th>*.prf Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOCK_CYCLE 20 CLK</td>
<td>PERIOD NET CLK 20 NS HIGH 10 NS</td>
</tr>
<tr>
<td>PULSE_WIDTH 10 CLK</td>
<td></td>
</tr>
<tr>
<td>ARRIVAL_TIME 35 READ</td>
<td>OFFSET IN COMP READ 35 NS AFTER COMP CLK</td>
</tr>
<tr>
<td>REQUIRED_TIME 20 RXRDY</td>
<td>OFFSET OUT COMP RXRDY 20 NS AFTER COMP CLK</td>
</tr>
</tbody>
</table>
**Using DIN/DOUT Properties**

ORCA’s DIN and DOUT properties enable the ORCA toolset to place registers within PFUs so that the direct in/out routing resources will be used. Placing the DIN property on an input buffer forces the software to use a direct-in routing resource. Placing the DOUT property on an output buffer forces the software to use a direct-out routing resource.

Properties differ from preferences in that they must be appear in the EDIF netlist. Mentor Graphics' synthesis tool allows the user to add DIN/DOUT as attributes in the Verilog or VHDL source code. These properties are written into the ORCA preference file after EDIF2NGD is run. For more information on ORCA properties, refer to the appropriate topic in the online help system.

The following rules apply when using the DIN/DOUT properties:

1. DIN/DOUT must be used in conjunction with the LOC preference (for the input/output component only).

2. DIN cannot be used when the flip-flop is fed by combinational logic.

3. DOUT cannot be used when there is combinational logic between the flip-flop and the output buffer.

4. The control logic (i.e., the clock enable and local set reset) must be the same for each flip-flop that is placed within a given PFU.

5. The DIN/DOUT properties must be in the EDIF netlist.

6. DIN/DOUT can both be used on a bidirectional buffer.
Using DIN/DOUT with Mentor Graphics Leonardo Spectrum

The following three steps are required to add the DIN/DOUT properties to the EDIF netlist:

1. Instantiate the buffers which will require DIN/DOUT in your source file.

2. Add the following attributes to your source file:

   **VHDL**

   ```
   attribute LOC:string;
   attribute DIN:boolean;
   attribute DOUT:boolean;
   attribute DIN of inbuf0: label is true ;
   attribute LOC of inbuf0: label is "22" ;
   attribute DOUT of outbuf0: label is true ;
   attribute LOC of outbuf0: label is "24" ;
   ```

   **Verilog**

   ```
   //Exemplar attribute inbuf0 LOC 22
   //Exemplar attribute inbuf0 DIN true
   //Exemplar attribute outbuf0 LOC 24
   //Exemplar attribute outbuf0 DOUT true
   ```

   In the above example, outbuf0 is the name of the instantiated OB12F output buffer component on the net "data_out". The location of the buffer will be pin 24 after being mapped in ORCA. Inbuf0 is the name of the IBM input buffer component on the net "data_in". The location of the buffer will be pin 22 after mapping in Orca.

3. Write out the EDIF file and map the design using ORCA. The preference file after mapping will contain the following preferences:

   ```
   LOCATE COMP "data_out" SITE 22 ;
   USE DOUT COMP "data_out" ;
   LOCATE COMP "data_in" SITE 24 ;
   USE DOUT COMP "data_in" ;
   ```

   Below is an example of a VHDL file and the output preference file from ORCA.

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

entity example is
  port (
    clk, reset, carryin: in std_logic;
    data_in   : in std_logic_vector (1 downto 0);
    data_out  : out std_logic_vector (1 downto 0));
end example;
architecture example_arch of example is

signal din_buf, din_reg, qout: std_logic_vector (1 downto 0);

attribute LOC:string;
attribute DIN:boolean;
attribute DOUT:boolean;

attribute DIN of inbuf0: label is true;
attribute LOC of inbuf0: label is "22";
attribute DIN of inbuf1: label is true;
attribute LOC of inbuf1: label is "23";
attribute DOUT of outbuf0: label is true;
attribute LOC of outbuf0: label is "13";
attribute DOUT of outbuf1: label is true;
attribute LOC of outbuf1: label is "14";

component IBM
  port ( I: in std_logic;
         O: out std_logic);
end component;

component OB12F
  port ( I: in std_logic;
         O: out std_logic);
end component;

begin

  inbuf0: IBM port map ( data_in(0), din_buf(0) );
  inbuf1: IBM port map ( data_in(1), din_buf(1) );
outbuf0: OB12F port map ( qout(0), data_out(0) );
outbuf1: OB12F port map ( qout(1), data_out(1) );

process (clk, reset, carryin, din_reg)
begin

    if (reset = '1') then
        din_reg <= "00";
        qout <= "00";
    elsif (clk'event and clk = '1') then
        din_reg <= din_buf;
        if carryin = '1' then
            qout <= din_reg + 1;
        else
            qout <= din_reg;
        end if;
    end if;

end process;
end example_arch;

Preferences generated by the ORCA MAP tool (from EDIF file):

SCHEMATIC START;

LOCATE COMP "DATA_IN_1" SITE "23";
USE DIN COMP "DATA_IN_1";
LOCATE COMP "DATA_IN_0" SITE "22";
USE DIN COMP "DATA_IN_0";
LOCATE COMP "DATA_OUT_0" SITE "13";
USE DOUT COMP "DATA_OUT_0";
LOCATE COMP "DATA_OUT_1" SITE "14";
USE DOUT COMP "DATA_OUT_1";
SCHEMATIC END;

Memory Initialization
Leonardo Spectrum’s ability to recognize and pass attributes from VHDL to EDIF enables a user to initialize all the ORCA memory components in their structural VHDL netlist as listed below.

```vhdl
library IEEE, ORCA;
use IEEE.std_logic_1164.all;
use ORCA.orcacompon.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
    port (AD0: in std_logic; AD1: in std_logic;
          AD2: in std_logic; AD3: in std_logic;
          D0: in std_logic; D1: 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; D00: out std_logic;
          D01: out std_logic);
end component;

component VHI
    port (Z: out std_logic);
end component;

attribute initval: string;
attribute initval of mem_0_0_1 : label is
    "0x0000130100122231";
```
attribute initval of mem_0_1_0 : label is 
"0x1213112020021002";

begin 
-- component instantiation statements 
mem_0_0_1: DCF16X2 
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 
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;

The netlist above must be modified appropriately for simulation. Please refer to the appropriate topic in the online help system.

Illegal Names and Characters

Certain names and characters used in a design may cause problems at some point in your ORCA design flow. See the appropriate topic in the online help system for a list of names and characters to avoid.

For Verilog, the memory initialization string is placed as a hex property on the module.

DCF15x2 mem_0_0_1 (....)
property meminit=0x0000130100122231;

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 the 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 ORCA 2002 programmable GSR will be 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 property "DISABLED_GSR" with value "1" for each component instance containing the attribute. The ORCA tools will recognize this property and disable the GSR functionality accordingly.

Example VHDL to implement programmable GSR:

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
entity attr_chk is
  port(d, clk, preset : in std_logic;
       qout : out std_logic);
architecture flop of attr_chk is
  component FD1S3BX
      generic (disabled_gsr: String);
      port(D, CK, PD : in std_logic;
           Q, QN : out std_logic);
  end component;
  attribute disabled_gsr: String;
  attribute disabled_gsr of u1:label is "1");
begin
  u1 : FD1S3BX generic map (disabled_gsr => "1")
     port map(
       D => d, CK =>clk, PD => preset, Q => qout, QN => open);
end flop;
```

Example Verilog to implement programmable GSR:

```verilog
module attr_chk (d, clk, preset, qout);
  input d, clk, preset;
  output qout;
```
FD1S3BX u1(.D(d),.CK(clk),.PD(preset),.Q(qout));
defparam u1.DISABLED_GSR=1
   // exemplar attribute DISABLED_GSR "1"
endmodule

* Note that boldfaced code is necessary for simulation.

The HDL code examples include "generic map" entries (VHDL) and a "defparam" entry (Verilog). These must be manually added to your post-synthesis HDL netlist for simulation after synthesis. Synthesis tools currently do not produce these entries in their synthesized netlists.
Passing FREQUENCY Preferences (Series 3 and 4)

FREQUENCY attributes will be converted into properties in the EDIF during synthesis which the mapper will then translate into FREQUENCY preferences on the output of the PLL. You must ensure that the FREQUENCY values are consistent with the input clock frequency, DIV0, DIV1, DIV2, DIV3 values and external divider, if any.

The following example shows how attributes are passed in the HDL code using Mentor Graphics:

```vhdl
// exemplar begin

// exemplar attribute TOPLEFTTOP MCLKMODE DELAY
// exemplar attribute TOPLEFTTOP NCLKMODE DELAY
// exemplar attribute TOPLEFTTOP VCOTAP 2
// exemplar attribute TOPLEFTTOP DIV0 5
// exemplar attribute TOPLEFTTOP DIV1 4
// exemplar attribute TOPLEFTTOP DIV2 3
// exemplar attribute TOPLEFTTOP DIV3 2
// exemplar attribute TOPLEFTTOP LOCATE ULPPLL

// exemplar attribute ck0 FREQUENCY 30
// exemplar attribute ck1 FREQUENCY 20
```

```
INDEX

A
Adding
  I/O buffers, 13
Attributes
  passing FREQUENCY preference in HDL, 30

B
Buffers
  instantiating, 13

D
Design
  entry, 7
  environment
    setting, 2
    flow, 3
  DIN (property), 22
  DOUT (property), 22

E
Environment variables, 2
Exemplar
  adding multiple Verilog files, 16
  adding multiple VHDL files, 18
Control File
  adding I/O, 15
  passing FREQUENCY preferences as attributes in HDL, 30
  synthesizing Series 4 designs in, 5

F
FREQUENCY preference, 30

G
GSR
  inferring, 19

I
I/O buffers
  adding to a design, 13
  adding via Exemplar Control File, 15
  instantiating, 13
Inferring
  GSR, 19
Instantiating
  components in Verilog, 11
  components in VHDL, 10
  I/O buffers, 13
  macros, 11

L
Libraries
  Synplify, 3
  Locking I/O pins, 15

M
Macros
  instantiating, 11
Memory initialization, 25
Modgen
  operators, 19

P
passing using Exemplar, 30
Preferences
  passing in Exemplar, 20
Properties
  DIN, 22
  DOUT, 22

R
Requirements
  software, 2

S
Series 4
  synthesis using VHDL and Verilog, 5
Series 4 architecture
  synthesis with Exemplar, 5
Setting
  design environment, 2
Software requirements, 2
Synplify
library files, 3
Synthesis
    and Exemplar, 9
    passing FREQUENCY preferences during, 30

V
Verilog
    adding multiple files in Exemplar, 16
VHDL
    adding multiple files in Exemplar, 18
    using to synthesize Series 4 design, 5