Copyright

Copyright © 1991-1996 Exemplar Logic, Inc.

Trademarks

Exemplar Logic® is a registered trademark of Exemplar Logic, Inc.
Galileo™, Leonardo™, Exemplar VHDL Discovery Kit™ and MODGen™ are trademarks of Exemplar Logic, Inc.
V-System/VHDL™ and V-System/Verilog™ are trademarks of Model Technology.
Verilog® and Verilog-XL® are registered trademarks of Cadence Design Systems, Inc.
All other trademarks remain the property of their respective owners.

Disclaimer

Although Exemplar Logic, Inc. has tested the software and reviewed the documentation, Exemplar Logic, Inc. makes no warranty or representation, either express or implied, with respect to this software and documentation, its quality, performance, merchantability, or fitness for a particular purpose.
## Contents

1. **Introduction** ................................................................. 1  
   Overview ............................................................... 1  
   Synthesis Libraries .................................................. 2  
   In This Manual ......................................................... 3  

2. **Executing lGen** ............................................................ 5  
   Options ................................................................. 5  
   Check File .............................................................. 6  
   Integration With Exemplar .......................................... 6  

3. **Gate Properties** .......................................................... 9  
   Template for Describing a Gate ................................... 9  
   Input Library Considerations ..................................... 12  
   Gate Functionality ................................................... 12  
   Combinational Gates ............................................... 13  
   Sequential Gates, Tri-State Gates ............................... 13  
   Designating a Gate as Non-Mappable ......................... 13
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structural Description</td>
<td>14</td>
</tr>
<tr>
<td>Specifying Electrical Information</td>
<td>16</td>
</tr>
<tr>
<td>Path Delays</td>
<td>16</td>
</tr>
<tr>
<td>I/O Pads</td>
<td>24</td>
</tr>
<tr>
<td>SAME_TECH_NOOPT Attribute</td>
<td>24</td>
</tr>
<tr>
<td>Load Handling</td>
<td>25</td>
</tr>
<tr>
<td>Gate Selection Rules for Exemplar</td>
<td>27</td>
</tr>
<tr>
<td>Gate Area Cost</td>
<td>28</td>
</tr>
<tr>
<td>An Example for IGen Syntax</td>
<td>29</td>
</tr>
<tr>
<td>Linear Timing Model</td>
<td>31</td>
</tr>
<tr>
<td>Piecewise Linear Timing Model</td>
<td>32</td>
</tr>
<tr>
<td>Nonlinear Timing Model</td>
<td>33</td>
</tr>
<tr>
<td>Introduction</td>
<td>33</td>
</tr>
<tr>
<td>Delay Calculation in Nonlinear Model</td>
<td>34</td>
</tr>
<tr>
<td>Lookup Table Templates (lut Template)</td>
<td>38</td>
</tr>
<tr>
<td>Example of Using Templates in the Nonlinear Delay Model</td>
<td>41</td>
</tr>
<tr>
<td>4. Global Library Properties</td>
<td>43</td>
</tr>
<tr>
<td>Weights for Area/Delay Trade-off</td>
<td>43</td>
</tr>
<tr>
<td>Derating Factors for Temp/Voltage/Process Parameters</td>
<td>44</td>
</tr>
<tr>
<td>Default Delay Parameters</td>
<td>45</td>
</tr>
<tr>
<td>Network Optimization Parameters</td>
<td>46</td>
</tr>
<tr>
<td>Conversion Factor from Unit Loads to Internal Load Units</td>
<td>47</td>
</tr>
<tr>
<td>Reporting Information</td>
<td>48</td>
</tr>
<tr>
<td>Library Version</td>
<td>48</td>
</tr>
</tbody>
</table>
Introduction

Overview

This document describes how to use lGen to create synthesis libraries for use with the Exemplar synthesis tools. Exemplar can use the library in one of two ways:

• To allow mapping out of a technology (TTL, FPGA, CMOS gate array). Libraries created for this purpose are referred to as Input Synthesis Libraries.

• To allow mapping into a CMOS gate array technology. Libraries created for this purpose are referred to as Output Synthesis Libraries.

The flow used for creating an output library and using it with Exemplar is shown in Figure 1-1. This shows VHDL as the design source, but any of the inputs accepted by Exemplar could be used.
Synthesis Libraries

In general, a synthesis library consists of two main parts. The first is the actual gate descriptions: an enumeration of all gates in the library with their input/output pins, gate functionality, and electrical characteristics. The second part contains general information about the library, and is used to describe default electrical parameters, device information, and other general information.

The primary source file into the program is the <library_name>.lgn file, which contains this information. The information is in textual format, designed to be robust enough to encompass all details needed by the synthesis algorithms as well as being readable and self-documenting. The file can either be entered manually using an editor, or automatically translated by a technology vendor from its internal data base.

The primary output file IGen produces is the binary library file. This is the file used by Exemplar.
In This Manual

This manual covers the building of synthesis libraries using the lGen tool from Exemplar’s Exemplar. Tool usage, gate level properties, global library properties, and library verification are discussed in the following chapters. An appendix provides the descriptions of the predefined primitives available when assembling the library. Another appendix provides an example source file. The final appendix provides installation and licensing information.

If you require assistance using this product, please call the Exemplar Customer Support Hot Line at 510-337-fpga (3742), or send e-mail to support@exemplar.com.
Executing lGen

lGen is a command line activated program, and takes several options to direct its operation. Following is the command syntax:

```
1Gen [-i format] [-o formats] [-p password] [-r revision] 
[-n] [-v] input_file.lgn output_file.syn
```

lGen will create an output file output_file.syn for use with Exemplar.

Options

- **-i format** Specify single input format (l = lgen, a=ascii). 1gen is Exemplar’s text format described in this manual.
- **-o format** Specify one or more output formats (a = ascii, l=lgn). If the output format is lgen, a password is required if one was used when the library was compiled.
- **-p password** Establishes a password to protect the library from unauthorized reverse compilation. This option is also required to reverse compile (go from ascii to lgen format) a library.
- **-r revision** Directs lGen to read older versions of the library. Used only for reverse compilation. The revision is the last two digits in the library file name.
Check File

To generate a check file, the following command syntax is required:

\[ \text{lGen} \ [-i \text{format}] \ -x \ \text{input\_file1} \ \text{input\_file2} \ \text{output\_file} \]

The -x option directs \text{lGen} to check the logical equivalence of primitives in \text{input\_file1} against primitives in \text{input\_file2}. The default for the input format is \text{lgn}, so that if \text{input\_file1}.\text{lgn} and \text{input\_file2}.\text{lgn} were used as inputs, the file extensions could be dropped when invoking \text{lGen}.

Integration With Exemplar

Once you have created a new technology library, you will want to use it with Exemplar.

To add the library to the Technology menus on the user interface, the User Profile must be edited. Edit the \$\text{EXEMPLAR/data/leonardo.ini} file, adding the following line:

\[
\text{\{Tech name, Lib name, modgen Lib name, Lib type, # passes (std. exh), Vendor name and Contact (if a vendor supplied library).\}} \nsample:\{"Adel Act2 act2 act2 "FPGA Enhanced" || ||"" ""\}
\]

The field <lib\_name> should be the actual library file name, minus the “##.syn” suffix. (This suffix contains the revision number of Exemplar’s data structure.) The <tech\_name> field is what will appear in the Technology menu on the main window for Exemplar, and may be of any length. Any kind of white space may exist between the fields, and the line may occur anywhere in the \text{master.rc} file.
For examples of the library lines, see $EXEMPLAR/data/leonardo.ini.

If you are using Exemplar from the command line, use the -or, -
target=<lib_name> source=<lib> switch as you normally would, with
<lib> equal to the actual library file name, minus the “##.syn” suffix.
This chapter discusses the gate-level properties that can be included in synthesis libraries. These properties include both gate functionality and electrical characteristics, as discussed below.

Template for Describing a Gate

- GATE <name>
- I/O section
- Functional description
- Structural description (optional)
- Special properties (area, reporting)
- Electrical information

Example:

GATE “mx3_1” ( 
I/O section: 
    INPUTS D0, D1, D2, S0, S1; 
    OUTPUTS Y; 

Functional description:
FUNCTION (  
    Y = D0*S0*S1 + D1*S0*S1 + D2*S1;  
);  

Structural description:  
NETLIST (  
    instance O1 ( MX4 (D0, So, S1, D1, D2, D2, Y));  
);  
SET EXMPLR_FLATTEN_ALWAYS = 1;  

Special properties:  
area = 1.000; SET comb_module = 1.000;  
SET COMBINABLE;  

Electrical information:  
INPUT D0 (  
    LIN = 1.143;  
    DELAY (  
        DRIVE = ( 0.350, 0.350);  
        PROP = ( 4.000, 4.000)  
        PHASE = UNKNOWN;  
    );  
);  

INPUT D1 (  
    LIN = 2.286;  
    DELAY (  
        DRIVE = ( 0.350, 0.350);  
        PROP = ( 4.000, 4.000)  
    );  
);
PHASE = UNKNOWN;
);
);
INPUT D2 ( 
LIN = 1.143;
DELAY ( 
DRIVE = (0.350, 0.350);
PROP = (4.000, 4.000)
PHASE = UNKNOWN;
);
);
INPUT S1 ( 
LIN = 1.143;
DELAY ( 
DRIVE = (0.350, 0.350);
PROP = (4.000, 4.000)
PHASE = UNKNOWN;
);
);
INPUT S0 ( 
LIN = 1.143;
DELAY ( 
DRIVE = (0.350, 0.350);
PROP = (4.000, 4.000)
PHASE = UNKNOWN;
Input Library Considerations

If you are building a library for input synthesis only, then only gate functionality descriptions are required. The area and electrical properties of the gates are not needed, and the global library properties are also not needed.

Gate Functionality

Gate functionality can be described as a set of Boolean equations, as a set of predefined primitives, or a combination of both. This allows specification of complex combinational gates (e.g. AND, OR, AND-OR), non-combinational gates (e.g. LATCH, FLIP-FLOP), or combinations of both (JK-FF implemented using a DFF and some AND-OR logic).
Combinational Gates

Combinational gates can be described as simple Boolean expressions. The supported boolean operations are the binary AND, OR, XOR (*, +, ^ respectively), and the unary NOT (!). Parentheses can be used freely, and the form in which a Boolean function is described is not important. Consider these two examples of a 2-input nand gate:

```
GATE nand2a (  
  INPUTS A, B;  
  OUTPUTS Z;  
  FUNCTION (! (A * B));  
);  
GATE nand2b (  
  INPUTS A, B;  
  OUTPUTS Z;  
  FUNCTION (!A + !B);  
);
```

Both descriptions are valid forms for 2-input nand gate functionality in IGen.

Sequential Gates, Tri-State Gates

Sequential and tri-state gates can be described using predefined primitives. The primitives available include DLATCH, DFF, RSLATCH, TRST, IBUF, OBUF, TBUF, BDBUF, CBUF, PULLUP, PULLDN, TRSTMEM, CONST1, and CONST0. These are defined in detail in Appendix A. These primitives take parameters much like a procedure call in a programming language. Signals in the gate are passed as parameters, and can be either input or output signals as described in the gate interface, internal signals, which need not be defined, or the character '-' which means that no signal is connected to the primitive for a given parameter. The order in which gate functionality is specified is not important.

Designating a Gate as Non-Mappable

It may be necessary or desirable to designate specific gates for input only. This is accomplished by inserting a line, with the statement NOMAP. This statement directs Exemplar to ignore this gate when synthesizing to this output library.
Structural Description

For gates that are described hierarchically a structural description is needed. This describes the gate by using netlist instantiation of other library gates. The library gates that are instantiated have to be defined first.

Example:

GATE “mx3_1” (I/O section:
    INPUTS D0, D1, D2, S0, S1;
    OUTPUTS Y;
Functional description:
    FUNCTION (Y = D0*S0*S1 + D1*S0*S1 + D2*S1);
Structural description:
    NETLIST (instance O1 (MX4 (D0, So, S1, D1, D2, Y));)
    SET EXMPLR_FLATTEN_ALWAYS = 1;
Special properties:
    area = 1.000;SET comb_module = 1.000;
    SET COMBINABLE;
Electrical information:
    INPUT D0 (LIN = 1.143;
DELAY (  
  DRIVE = ( 0.350, 0.350);  
  PROP = ( 4.000, 4.000)  
  PHASE = UNKNOWN;  
);
);
INPUT D1 (  
  LIN = 2.286;  
  DELAY (  
    DRIVE = ( 0.350, 0.350);  
    PROP = ( 4.000, 4.000)  
    PHASE = UNKNOWN;  
  );  
);
INPUT D2 (  
  LIN = 1.143;  
  DELAY (  
    DRIVE = ( 0.350, 0.350);  
    PROP = ( 4.000, 4.000)  
    PHASE = UNKNOWN;  
  );  
);
INPUT S1 (  
  LIN = 1.143;  
  DELAY (
Specifying Electrical Information

Path Delays

The Path specifications identify the different paths in the gate from input pins to output pins, etc. These paths identify how delays are propagated through the gate during static timing analysis.

A path can be setup between any two pins. There could be a path from:

- input pin to input pin. example: a timing constraint between d and the clock of a dff.
• input pin to output pin — example: input of a nand gate to its output.
• output pin to output — example: path from Q to QN of a gate.

A pin can be involved in any number of paths. If the “to” pin of a path is ignored, IGen assumes paths to all the outputs. Alternately, multiple pin names can be specified separated by commas to create more than one path simultaneously. IGen tries to infer the clock pin if the gate has a clock automatically. In most cases path types are inferred automatically. In some special cases where this is not possible, path types need to be specified explicitly. This is discussed in detail later in this chapter.

The PATH_TYPE = DISCONNECT disconnects propagation paths and this should be used for asynchronous paths. These paths can be connected again at run time by having a CONNECT statement in the control file Exemplar or by using the CONNECT interact command in Leonardo’s shell.

Some of these paths are inferred from the pins they are connecting and the attributes that are being set on them. (For example, A path from CLK to Q is inferred if the input pin happens to be a clock pin.) A constraint path is inferred if you are setting a constraint like setup or hold on the path to a clock. Special paths like PAD_TO_Y and asynchronous path (disconnected) need to be specified explicitly by setting a PATH_TYPE attribute in IGen on that path.

Sometimes it is not possible to infer the clock pin, and IGen will give a warning. In such cases, the clock pin will need to be explicitly specified with an attribute on the clock pin of the gate. SET CLOCK_PIN.

This is an example of a simple path from input to output.

```
INPUT A, B {
  LIN = 1.000;
  DELAY {
    SETUP = (4.2, 4.2);
    HOLD = (3.0, 3.0);
  };
};
```
This is an example of a constraint between D and CLK of a flip-flop.

```
INPUT D {
  LIN = 1.000;
  DELAY CLK {
    SETUP = (4.2, 4.2);
    HOLD = (3.0, 3.0);
  };
}
```

- Dependent Outputs and Independent Outputs:

  Since paths can be specified in any number of ways, this gives complete flexibility in modeling a gate. There is more than one way to model the same gate. A dependent output is one where the inverted output is defined through one of the other outputs. An independent output is one which is not dependent on any other output.

  In this example, the output QBAR is dependent on the output Q. There is a path from Q to QBAR, and the load at Q affects the delay through this path.
Alternatively, the path to QBAR can be specified directly from the CLK. This means the load at Q does not affect the delay to pin QBAR.

```plaintext
INPUT CLK (  
  LIN = 2.000;  
  DELAY Q (  
    DRIVE = (0.5, 0.5),  
    PROP = (7.300, 7.300);  
  );  
);  
OUTPUT Q (  
  LMAX = 999.000;  
  DELAY QBAR (  
    DRIVE = (0.5, 0.5);  
    PROP = (8.300, 8.300);  
  );  
);  
DELAY QBAR (  
  DRIVE = (0.5, 0.5);  
  PROP = (8.300, 8.300);  
);  
); 
```
• Asynchronous Paths:

The timing analyzer needs to ignore asynchronous paths when analyzing in synchronous mode. Therefore, asynchronous paths should be disconnected. This can be done with PATH_TYPE = DISCONNECT specified in the path. This disconnects the path by default and it can be connected selectively from the control file, or from interactive still in Leonardo.

```
INPUT CLR (
  LIN = 1.000;
  DELAY CLK (
    SETUP = (5.200, 5.200);
  );

  DELAY Q (
    DRIVE = (0.5, 0.5);
    PROP = (7.300, 7.300);
    PATH_TYPE = DISCONNECT;
  );
);
```
Path Type Specification for Bidirectional Buffers

For Exemplar to calculate the correct timing for a design with bidirectional buses, the PATH_TYPE should be set to PAD_TO_Y. An example of this is shown below.

```
GATE bdbuf (  
    INPUTS en, a;  
    OUTPUTS in, pad;  
    FUNCTION (  
        en_bar = !en;  
        BDBUF(a, en_bar, in, pad);  
    );  
    AREA = 0.000;  
    INPUT en, a (  
        DELAY pad (  
            ----  
        );  
    );  
    OUTPUT pad (  
        SET PAD;  
        DELAY in (  
            PATH_TYPE = PAD_TO_Y;  
        );  
    );  
);  
)
```

Syntax:

```
PATH_TYPE = PAD_TO_Y;  
PATH_TYPE = DISCONNECT;
```

[Sometimes it is not possible to infer the clock pin from the functionality. In such cases you must put a clock attribute on the clock pin of the gate.SET CLOCK_PIN]

Examples for timing paths:
• The example below specifies a path from D to CLK, which is inferred to be of type
CONSTRAINT since it sets a SETUP between these 2 pins.

```plaintext
INPUT D {
  LIN = 1.000;
  DELAY CLK {
    SETUP = ( 4.200, 4.200);
  };
};
```

• The next example specifies a path from Q to QBAR, which defaults to a
propagation path. It is an example of an output->output path.

```plaintext
OUTPUT Q {
  LIN = 2.000;
  DELAY QBAR {
    DRIVE = ( 0.000, 0.000);
    PROP = ( 7.300, 7.300);
  };
};
```

• The next example specifies a path from PAD to Y. There is no way to infer this
path to be of a BDBUF and to be from PAD to Y. Therefore, you need to explicitly
specify the path type by setting PATH_TYPE = PAD_TO_Y.

```plaintext
OUTPUT PAD {
  SET PAD;
  DELAY Y {
    DRIVE = (0.000, 0.000);
    PROP = ( 5.000, 5.000);
    PHASE = UNKNOWN;
    PATH_TYPE = PAD_TO_Y;
  };
};
```
The example below specifies 2 paths from CLR. One to the CLK, which is inferred to be of type CONSTR since the timing information is SETUP. The second path is from CLR to Q. This defaults to PROPAGATION_PATH and needs to be specified explicitly with a PATH_TYPE = DISCONNECT since this is an asynchronous path.

```
INPUT CLR (  
LIN = 1.000;  
DELAY CLK (  
SETUP = ( 5.200, 5.200);  
);  
DELAY Q (  
DRIVE = ( 0.000, 0.000);  
PROP = ( 7.300, 7.300);  
PATH_TYPE = DISCONNECT;  
);  
);
```

A path is specified between 2 pins as follows:

```
INPUT A ( /* from input pin A */  
DELAY O ( /* to pin O */

```

If in the path specification, the to_pin is left out, this defaults to all output pins. For example:

```
INPUT A ( /* from input pin A */  
DELAY ( /* to all output pins */
```

There is a path from input pin A to all outputs of this cell. For a combinational gate with just one output, the to_pin can be ignored.
I/O Pads

PAD Attribute

I/O buffers should have the pin corresponding to the I/O pad annotated with the SET PAD attribute. This is required if the pad is to be connected using the GATE construct in the control file. This construct can also be used to promote internal buffers and tristates to I/O buffers.

The following descriptions build the same tristate buffer:

```plaintext
GATE tbuf1 (  
  INPUTS en, d;  
  OUTPUTS pad;  
  FUNCTION (  
    TRST(d, en, pad);  
  );  
  AREA = 1.0;  
  OUTPUT pad (SET PAD);  
);  

GATE tbuf2 (  
  INPUTS en, d;  
  OUTPUTS pad;  
  FUNCTION (  
    TBUF(d, en, pad);  
  );  
  AREA = 1.0;  
  OUTPUT pad (SET PAD);  
);  
```

SAME_TECH_NOOPT Attribute

There are some cells that may be needed, but that are inappropriate for Exemplar to use automatically. Two cases where this commonly occurs are in I/O buffers and macros. For I/O buffers, due to Exemplar run-time considerations, it is often best to use only the basic buffers as the automatically mapped buffers. Other buffers should be designated as NOMAP (see Section 3.2.3 above), and the SAME_TECH_NOOPT attribute should be set so that the gate can be used if designated specifically, either
from the control file or from an instantiation in VHDL or Verilog. This would also hold when using Exemplar strictly for optimization (same technology for input and output).

**Load Handling**

The following properties can be used to specify loads in IGen.

- **cap_load (used to be cin):** Can be specified on input pins of a gate to indicate the input capacitance of this pin. Load should be usually expressed in terms of pF.
- **default_cap_load:** Default value for cap_load. This value will be assigned to all input pins that do not have a cap_load value.
- **max_cap_load (used to be LMAX):** Can be specified on output pins of a gate to indicate the maximum capacitance that can be driven by this pin.
- **default_max_cap_load:** Default value for max_cap_load. This value will be assigned to all output pins that do not have a cap_load value.
- **fanout_load:** Gives the fanout load value for an input pin. The sum of all fanout_load values for input pins connected to a driving output pin must not exceed the max_fanout value for that output pin. There are no fixed units for fanout_load; typical units are standard loads or pin count.
- **default_fanout_load:** Default value for fanout load for all input pins in a design which do not have a fanout_load specified.
- **max_fanout_load:** Defines the maximum fanout load that an output pin can drive. Exemplar, and, Leonardo model fanout restrictions by associating a fanout_load value with each input pin and a max_fanout_load value with each output pin in a cell.

The max_fanout attribute is an implied design-rule constraint. The optimizer attempts to resolve max_fanout violations, possibly at the expense of other design constraints. Although you can use capacitance as the unit for your max_fanout and fanout_load specifications, it should be used to constrain routability requirements. In case a fanout violation will occur the optimizer will resolve it by buffering, or, replicating logic.

- **default_max_fanout_load:** Default value for max_fanout load for all output pins in a design which do not have a max_fanout_load specified.
- **max_transition:** The max_transition value defines a design-rule constraint for the maximum acceptable transition time of an input or output pin. If max_transition is used with an output pin, that pin can be used only to drive a net for which the cell
can provide a transition time at least as fast as the defined limit. A \texttt{max\_transition}
value used with an input pin indicates that the pin cannot be connected to a net that
has a transition time greater than the defined limit.

The \texttt{max\_transition} value you define is checked against \texttt{Dt}, the transition delay
calculated by the timing analyzer, which is the rise and fall resistance multiplied by the
sum of the pin and wire capacitances: If the calculated Transition time is greater than
the value you specify with \texttt{max\_transition}, a design rule violation is detected.
Transition time violations will be resolved during optimization by buffering the net
connected to the input, or, output pin.

- \texttt{default\_max\_transition}: Sets a default value for \texttt{max\_transition} for all output pins in
  the library that do not have a \texttt{max\_transition} value.

Example:

\begin{verbatim}
LIBRARY test (

... 

GATE AOI1C ( 
    inputs A, B, C; 
    outputs Y; 

    function ( 
        Y = B*!C + A*!C; 
    ); 

    area = 1.000000; 

    input A ( 
        fanout_load = 1.0; 
        cap_load = 1.904676; 
        max_transition = 3.0; 
    ); 

    input B ( 
        fanout_load = 1.0; 
        cap_load = 1.904676; 
        max_transition = 3.0; 
    ); 

    input C ( 
        fanout_load = 1.0; 
        cap_load = 1.904676; 
        max_transition = 3.0; 
    ); 

) 
\end{verbatim}
output Y (  
    max_fanout_load = 1.0;  
    max_cap_load = 30.0;  
    max_transition = 5.0;  
);  
...  
);  
...  
SET default_max_transition = 5.0;  
SET default_fanout_load = 1.0;  
SET default_cap_load = 1.904676;  
SET default_max_fanout_load = 16.0;  
SET default_max_cap_load = 32.0;  
)

Using the tristate buffer above as an example, the SAME_TECH_NOOPT attribute is set as follows.

```plaintext
GATE tbuf2 (  
    INPUTS en, d;  
    OUTPUTS pad;  
    FUNCTION (  
        TBUF(d, en, pad);  
    );  
    AREA = 1.0;  
    NOMAP;  
    SET SAME_TECH_NOOPT;  
    OUTPUT pad (SET PAD);  
);  
```

The implementation is exactly the same for other gates.

**Gate Selection Rules for Exemplar**

Exemplar performs automatic selection of gates in a class to get the best performance out of the circuit. Gates are members in the same class if they have the same functionality, and the same pin names. Therefore, it is important to make sure that pin names are the same, if automatic selection is desired. This is especially important for buffers and inverters, which are used by the buffering algorithm, and it is highly recommended to have all inverters in one class, and all buffers in one class, so that accurate buffering is done.
When determining the optimal design solution, Exemplar uses a weighted calculation that takes into account the total area of the design, the critical path delay, the number of nets formed, and the number of pins. These weighting factors are set in the global library properties. The values required for the delay and area estimates are described in the following sections.

**Gate Area Cost**

IGen uses the following structure to establish the area cost of a gate:

\[
\text{AREA} = \text{area\_value}
\]

area\_value is the area of the gate, in the units defined by the Set area\_units global library property (see Chapter 4).
An Example for IGen Syntax

The gate description in IGen consists of the I/O definition, the gate functionality, and the other gate characteristics. For example, a 2-input nand gate could be described as follows.

```
GATE nand2 {
    INPUTS a, b;
    OUTPUTS z;
    FUNCTION {
        z = !b + !a;
    };
    AREA = 1.000;

    INPUT a {
        LIN = 1.000;
        DELAY ( 
            DRIVE = (0.058, 0.041);
            PROP = (0.140, 0.180);
            PHASE = INV;
        );
    }

    INPUT b {
        LIN = 1.000;
        DELAY ( 
            DRIVE = (0.058, 0.041);
            PROP = (0.140, 0.180);
            PHASE = INV;
        );
    }

    OUTPUT z {
        LMAX = 36.000;
    }
}
```
Example of a DFF:

```
GATE dff (  
    INPUTS d, clk, clr;  
    OUTPUTS q, qn;  
    FUNCTION (  
        DFF(-, clr, d, clk, q, qn);  
    );  
    AREA = 1.0;  
    INPUT d (  
        LIN = 1.000;  
        DELAY CLK (  
            SETUP = (4.2, 4.2);  
            HOLD = (3.0, 3.0);  
        );  
    );  
    INPUT CLR (  
        LIN = 1.000;  
        DELAY CLK (  
            SETUP = (5.200, 5.200);  
        );  
        DELAY Q (  
            DRIVE = (0.5, 0.5);  
            PROP = (7.300, 7.300);  
            PATH_TYPE = DISCONNECT;  
        );  
    );  
    INPUT CLK (  
        LIN = 2.000;  
        DELAY Q (  
            DRIVE = (0.5, 0.5),  
            PROP = (7.300, 7.300);  
        );  
    );  
    OUTPUT Q (  
        LMAX = 999.000;  
        DELAY QN (  
            DRIVE = (0.5, 0.5);  
            PROP = (8.300, 8.300);  
        );  
    );  
    OUTPUT QN (  
        LMAX = 999.000;  
    );
```
Linear Timing Model

The Exemplar CMOS gate array synthesis algorithms support the simple lumped RC model. The model for total delay through a gate is

\[ D_{\text{total}} = D_{\text{intrinsic}} + D_{\text{slope}} + D_{\text{transition}} + D_{\text{connect}} \]

**D\text{intrinsic}** is the delay through the gate from an input pin to the output pin or along a path.

**D\text{slope}** is the additional delay incurred due to the input ramp.

\[ D_{\text{slope}} = D_{\text{transition of prev stage}} \times S_s \]

\( S_s \) = slope sensitivity factor.

**D\text{transition}** is the delay due to the loading of the gate.

\[ D_{\text{transition}} = R_{\text{driver}} \times (C_{\text{fanout}} + C_{\text{connect}}) \]

The \( R_{\text{driver}} \) is the drive resistance of the gate.

\( C_{\text{fanout}} \) is the load due to the fanouts of the gate.

\( C_{\text{connect}} \) is the interconnect load.

**D\text{connect}** = \( R_{\text{wire}} \times (C_{\text{fanout}} + C_{\text{connect}}) \)

The \( R_{\text{wire}} \) is the resistance of the wire.

The \( C_{\text{fanout}} \) and \( C_{\text{connect}} \) are the loads on the gate due to the input pin capacitance and the interconnect capacitance.

The syntax for specifying delay parameters are:

**PROP** = \( (\text{rise}_{\text{delay}}, \text{fall}_{\text{delay}}) \) The inherent propagation delay through the gate (per input pin).

**DRIVE** = \( (\text{rise}_{\text{drive}}, \text{fall}_{\text{drive}}) \) The driving capability of the gate (per input pin).

**SETUP** = \( (\text{setup}_{\text{rise}}, \text{setup}_{\text{fall}}) \) Setup requirement of an input pin, with respect to the clock signal.

**HOLD** = \( (\text{hold}_{\text{rise}}, \text{hold}_{\text{fall}}) \) Hold time requirement of an input pin, with respect to the clock signal.

**LIN** = load_value The input load of an input pin.

**LMAX** = lmax_value The maximum allowed load on the output pin.

**PHASE** = INV, NONINV, or UNKNOWN The inversion nature of an input pin with respect to the output pin.
Units for these parameters should be consistent, based on the delay equation above.

Distinction is made between rise and fall values of PROP, DRIVE, and SETUP parameters. Those parameters are specified as a pair of numbers \((x, y)\), where the first number in the pair is the rise parameter, and the second one is the fall parameter. If both numbers are the same, a single number can replace the pair.

*Piecewise Linear Timing Model*

This section discusses IGen support for Piecewise Linear libraries (PWL libraries). A piecewise linear library allows delay computations to match the nonlinear delay curve more closely than a linear library. In a PWL library, drive resistance is specified for different pieces, the pieces being a function of either the total capacitance or wire length, etc. Also, the intercept of the timing curve is specified for each piece. In a PWL the transition delay is computed as

\[
D_t = R_i \times (C_{\text{pins}} + C_{\text{wire}}) + Y_i,
\]

\(i\) is the piece number.

The total delay is computed as:

\[
D_{\text{total}} = D_{\text{intrinsic}} + D_{\text{slope}} + D_{\text{transition}} + D_{\text{interconnect}}
\]

The slope delay is the delay due to the input slope or ramp. Intrinsic delay, transition delay and interconnect delay are delay through the gate, delay due to output load and delay due to the interconnect respectively. This is the same as in the linear delay model.

To define a PWL library, the definition of pieces has to be specified to begin with. Each piece defines a range of load driven by the gate or the wire length due to its fanouts. This association is specified by a `PIECE_DEFINE` statement.

```
PIECE_DEFINE WIRE_LENGTH (10.0, 20.0, 30.0, 40.0);
```

This statement specifies that for wire_lengths of up through 10.0 use piece 0. For wire lengths ranging from greater than 10.0 to 20.0 use piece 1, etc. The keyword `WIRE_LENGTH` specifies that we are associating the pieces with wire_length. This
association can be set up for WIRE_LENGTH or TOTAL_CAP or PIN_CAP or WIRE_CAP. TOTAL_CAP computes the value of the total capacitive load to choose the piece, PIN_CAP uses only the input pin capacitances driven by this gate and WIRE_CAP uses only the interconnect capacitance to compute the piece number. In the piece define statement, the indices are implicit. The index starts from 0 and proceeds upwards in steps of 1. Any number of pieces can be specified. (The WIRE_CAP is computed by using the wire load model discussed in the global parameters section).

The drive resistance and intercept are expressed as triplets of numbers in a PWL library.

\[
\text{DRIVE} = (\langle 0, 1.0, 1.0 \rangle, \langle 1, 2.0, 2.0 \rangle, \langle 2, 3.0, 3.0 \rangle); \\
\text{INTERCEPT} = (\langle 0, 1.0, 1.0 \rangle, \langle 1, 2.0, 2.0 \rangle, \langle 2, 3.0, 3.0 \rangle); 
\]

The first number in the triplet is the index or piece number. The second and third numbers are the rise and fall values of drive resistance or intercept. This completely represents a piecewise linear curve with its various points. If the drive resistance or intercept happens to be a linear function of pieces, not all pieces need to be specified. If some piece specifications are missing IGen extrapolates to find the value of drive resistance for the missing piece. For pieces beyond the ones specified, it extrapolates using the last two points in the curve. Basically any 2 points are necessary to completely specify drive resistance in a PWL library. (IGen extrapolates downwards also. example: if piece 2 and 3 are specified, it extrapolates downwards to find the value of drive for piece 0 and piece 1.)

**Nonlinear Timing Model**

**Introduction**

The nonlinear delay model uses sophisticated table lookup and an a polynomial curve interpolation technique to compute delay paths. This is very important for ASIC and sub-micron technologies where delay computation needs to be extremely accurate. This is normally used by the timing analyzer to report a path delay. User can switch on the option of using this delay computation during synthesis flow (mapping to target technology cells) also. This model has been proven to be a flexible model to provide close approximation for a wide variety of submicron delay modeling schemes.
Delay Calculation in Nonlinear Model

Total Delay Equation

The delay computation of logic stage can involves computing the time taken from input pin of a gate to the input pin of it fanout gate(s). The total delay is comprised of two major components, Dcd & Dic.

\[
D_{total} = D_{cd} + D_{ic}
\]

\[
D_{total} = D_{td} + D_{pd} + D_{ic}
\]

\[
D_{cd} = D_{td} + D_{pd}
\]

Cell Delay (Dcd)

The delay contributed by the gate itself, typically measured from the 50% input pin voltage to the 50% output pin voltage. Dcd is computed in one of two ways, depending on the timing data provided.

Two groups are used to define cell delay tables:

- cell_rise
- cell_fall

If cell delay tables are defined for a delay path, Propagation delay tables must not be specified for that arc.

Interconnect Delay (Dic)

The interConnect delay is calculated by the same method used in other delay model (linear and piecewise linear).

A third component, Dtd, corresponds to the time required for the output pin to change state. This is sometimes known as the output ramp time.

Transition Delay (Dtd)

The time between two reference voltage levels on the output pin. These levels may be 20% to 80% or 10% to 50% for example. Dtd is computed by performing table lookup and interpolation. The CMOS nonlinear timing model supports two methods of
computing Dcd. Two groups are used to define transition delay tables: rise_transition
fall_transition Transition tables must be specified for all delay paths. Dtd is a function
of capacitance at the output pin, and may also be a function of input transition time in
submicron technology.

Dcd may be computed directly by performing table lookup and interpolation in a cell
delay table provided in the library. Or Dcd may be computed using the propagation
and transition tables since, \( Dcd = Dtd + Dpd \).

**Propagation Delay (Dpd)**

A typical measurement for Dpd is the time from the 50% input pin voltage until the
gate output just begins to switch, for example the 10% output voltage is reached. Thus,
when a Dtd value defined from the 10% to 50% output voltage is added to Dpd, the
result is a 50% input to 50% output cell delay. If cell delay tables are provided for a
timing arc, the total delay equation used is: \( D_{total} = Dcd + Dic \)

If propagation delay tables are provided instead, the total delay equation becomes:

\[ D_{total} = Dpd + Dtd + Dic \]

Propagation delay, Dpd is often a function of output loading and input transition time.
Two groups are used to define propagation delay tables: rise_propagation
fall_propagation If propagation delay tables are defined for a delay path, cell delay
tables must not be specified. The presence of propagation delay tables indicates that
cell delays will be computed by adding the propagation and transition delays.

**Connect Delay (Dic)**

Dic is the time it takes the voltage at an input pin to charge after the driving output pin
has made a transition. This delay is also called the time-of-flight delay; the time it
takes for a waveform to travel along a wire. The CMOS nonlinear timing model
computes connect delay using the same equations as the linear timing model.

**Nonlinear Delay Model Calculation**

Let’s say the following schematic represents a part of user’s netlist and we want to
compute the Cell delay through Gate Gb. Let’s say that the user specifies the nonlinear
model with Propagation delay tables for Gate Gb. with a lut template that uses Input
net transition and total output capacitance. In order to compute the delay through the
gate Gb, at first the transition delay on net n1 and n2 needs to be computed. To
compute Rise propagation time of gate Gb for path a to z the Fall net transition time on n1 is used. To compute Rise propagation time of Gate Gb for b to z, the Fall net transition time on n2 is used. Since the n1(transition time) and n2(transition time) have been computed while processing gate Ga and Gc, we can directly use the maximum of these two to determine the input net.

Figure 3-1  Delay Computation for Combinational Gates

Equation 1:  \( Ax + By + Cxy + D = z \)
One can derive the coefficients A, B, C, and D by using common mathematical methods such as Gaussian elimination.

Figure 3-2
Example: Determining Coefficients and Solving the propagation delay

\[
\begin{align*}
A \times 2.064 + B \times 0.843 + C \times 2.064 \times 0.843 + D &= 2.028 \\
A \times 2.064 + B \times 1.106 + C \times 2.064 \times 1.106 + D &= 1.066 \\
A \times 2.748 + B \times 0.843 + C \times 2.748 \times 0.843 + D &= 2.394 \\
A \times 2.748 + B \times 1.106 + C \times 2.748 \times 1.106 + D &= 1.482
\end{align*}
\]

Coefficient Values:
\[
A = 0.301, \quad B = -4.231, \quad C = 0.278, \quad D = 4.491
\]
Now we can use Eqn. 1 to interpolate and obtain the Propagation delay value at this particular co-ordinate (2.55, 0.95) -

\[ Z = 0.299 \times 2.55 + (-4.225) \times 0.95 + 0.278 \times 2.55 \times 0.95 + 4.486 \]

\[ = 1.911 \]

**Lookup Table Templates (lut Template)**

*What are Nonlinear Timing Model lut Templates?*

A delay path in a cell can use some pre-defined information stored at the library level. This pre-defined information can be represented as a look-up table template. Normally, a lut template has two-axis, X axis and Y axis. A lut template defines the interval on the axes used in the table. Each lut template has an unique name by which it is referenced while specifying a delay path. The lut templates can only be defined at the library level (top level) at which the Gate(s), Wire load(s) etc. are defined.

There are two variables that can be associated with the lut template. A table variable indicates which parameter is used to index into the table along a particular axis. The statements, variable_x or variable_y indicate that the given variable will be used respectively for the first or second table axis. Use one of the following attributes to define variable_x and variable_y:

- input_net_transition
- total_output_net_capacitance
- output_net_length
- output_net_wire_cap
- output_net_pin_cap

Use only variable_x for a one-dimensional table. The table breakpoint values for a given axis are defined using the variable_x and variable_y statements. The variable_x breakpoint values correspond to the possible parameter values indicated by variable_x and the variable_y breakpoint values correspond to variable_y. A list of n floating point numbers is assigned to variable_x or variable_y. Only values greater than or equal to 0.0 are valid. The floating point values in the list must be in monotonically increasing order. The size of each dimension is determined by the number of floating point numbers in variable_x or variable_y.
Where to Define the Nonlinear Timing Model Template?

Users can define the nonlinear timing model anywhere in the library and use it anywhere. There is no restriction on pre-defining the lut template.

Syntax for the Lookup Table Template

Syntax for the lut_template Statement:

\[
\text{lut_template <namestring> (}
\begin{align*}
\text{variable_x} & \quad \text{input_net_transition |} \\
& \quad \text{total_output_net_capacitance |} \\
& \quad \text{output_net_length |} \\
& \quad \text{output_net_wire_capacitance |} \\
& \quad \text{output_net_pin_capacitance;}
\text{variable_y} & \quad \text{input_net_transition |} \\
& \quad \text{total_output_net_capacitance |} \\
& \quad \text{output_net_length |} \\
& \quad \text{output_net_wire_capacitance |} \\
& \quad \text{output_net_pin_capacitance;}
\text{variable_x} = \text{(float, ..., float)}; \\
\text{variable_y} = \text{(float, ..., float)};
\end{align*}
\)]

Delay Paths

There are two ways of specifying a cells’ delay.

1. Defining the cell delay directly.
2. Defining the propagation delay and transition delay.

Using 1 specify:

\[
\text{transition_rise,} \\
\text{transition_fall,} \\
\text{cell_delay_rise,} \\
\text{cell_delay_fall}
\]
Using 2 specify:

transition_rise
transition_fall
propagation_rise
propagation_fall

Redefining the Variable Indices of lut Template

Users can modify the breakpoints on an axis in a lookup table while 'instantiating' a lookup table template. The modification applies to this particular delay path. All other places where this 'lut template' is instantiated remains unchanged. The modification of the axes must happen before the actual definition of delay path 'parameter'. The number of breakpoints(intervals) defined for either variable (x or y) while re-defining must match the original template description. The delay values of the table is assigned to a keyword named 'parameter'. Parameter has (Nx * Ny) floating point numbers for a two-dimensional table or is either a list of Nx floating point numbers for a one-dimensional table. Nx and Ny are the size of variable_x and variable_y of the lut template. Or in other words, the number of breakpoints defined on variable_x or variable_y in the lut template definition. Each list represents a row in the table. The number of floating point numbers in a list must equal Nx, and the number of lists in 'parameter' must equal Ny. Ny is "1" for an one-dimensional table.

Syntax for Defining Delay Paths

The syntax for defining the delay values for a particular path is:

```plaintext
<delay_attribute> = <lut_template_name> (  
    variable_x = ("float, ..., float");  
    variable_y = ("float, ..., float");  
    parameter = (  
        (float, ..., float),  
        ...,  
        (float, ..., float)  
    );  
)  
```

delay_attribute is one of the following six groups:

transition_rise
transition_fall
cell_delay_rise
cell_delay_fall
propagation_rise
propagation_fall

For defining delay path with clear, only fall delay values are valid. For defining delay path with preset, only rise delay values are valid.

Using the Predefined lut Template 'scalar'

If a table contains only one value, you can use a predefined scalar table template as the template for that timing path. To use the scalar table template, put the string scalar in the place where you define the table. This nonlinear look up table delay specification reduces to linear library specifications.

Example of Using Templates in the Nonlinear Delay Model

```
# Setting the attribute for library compiler
SET delay_library_model = non_linear;
...
# template of size 5x5
lut_template prop_5x5 ( 
    variable_x input_net_transition;
    variable_y total_output_net_capacitance;

    variable_x = ("0.0, 0.5, 1.5, 2.0");
    variable_y = ("0.0, 2.0, 4.0, 6.0");
); 

# template of size 5x1
lut_template trans_5x1 ( 
    variable_x input_net_transition;
    variable_x = ("0.0, 0.5, 1.5, 2.0"); ) ; ...

# a Gate and it's nonlinear timing specification
GATE an2 ( 
    ...

    INPUT A ( 
        DELAY Q ( 
            cell_delay_rise = "prop_5x5" ( 
```
 variable_x = (0.0072, 0.3590, 0.7108, 1.0626, 1.4144);
 variable_y = (0.0520, 0.1999, 0.3478, 0.4958, 0.6437);
 parameter = (  
    (0.1396, 0.1387, 0.1385, 0.1391, 0.1404),
    (0.2588, 0.2574, 0.2566, 0.2566, 0.2574),
    (0.3781, 0.3762, 0.3750, 0.3745, 0.3748),
    (0.4974, 0.4951, 0.4935, 0.4927, 0.4926),
    (0.6167, 0.6141, 0.6123, 0.6112, 0.6109)  
  );
 cell_delay_fall = "prop_5x5" (  
    variable_x = (0.0114, 0.6957, 1.3800, 2.0643, 2.7486);
    variable_y = (0.0520, 0.1999, 0.3478, 0.4958, 0.6437);
    parameter = (  
      (0.1763, 0.5455, 0.8614, 1.1241, 1.3334),
      (0.3254, 0.7787, 1.1787, 1.5254, 1.8188),
      (0.4741, 0.9897, 1.4520, 1.8610, 2.2167),
      (0.6225, 1.1786, 1.6813, 2.1308, 2.5269),
      (0.7705, 1.3452, 1.8666, 2.3348, 2.7496)  
    );
 transition_rise = "trans_5x1" (  
    variable_x = (0.0520, 0.1999, 0.3478, 0.4958, 0.6437);
    parameter = (  
      (0.1071, 0.4117, 0.7163, 1.0209, 1.3254)  
    );
 transition_fall = "trans_5x1" (  
    variable_x = (0.0520, 0.1999, 0.3478, 0.4958, 0.6437);
    parameter = (  
      (0.0590, 0.2269, 0.3948, 0.5627, 0.7306)  
    );
 ...  
);
In IGen, general information for a technology is specified using the SET construct. These global attributes are assigned values which can be strings or numbers, to be used by Exemplar. These variables can contain information about various default values, derating factors, and other technology properties.

Comments are preceded by the '#' character.

Weights for Area/Delay Trade-off

When determining the optimal design solution, Exemplar uses a weighted calculation that takes into account the total area of the design, the critical path delay, the number of nets formed, and the number of pins. Area or delay optimization can be emphasized using switches from either the user interface or the command line. The following SET constructs determine the relative weighting when either of these switches is invoked.

```plaintext
SET opt_for_area_weights =
  <size_weight>  <delay_weight>  <pin_count_weight>
  <net_count_weight>
SET opt_for_speed_weights =
  <size_weight>  <delay_weight>  <pin_count_weight>
  <net_count_weight>
```
From experience, using

```
SET opt_for_area_weights = 1.0 0.2 0.01 0.00
```

for area weighting and

```
SET opt_for_speed_weights = 0.2 1.0 0.01 0.00
```

for delay weighting produces good results.

**Derating Factors for Temp/Voltage/Process Parameters**

The following attributes are used to determine the delay derating to be used by Exemplar when estimating critical path delay.

```
SET nominal_temp = <temp>
SET max_temp = <temp>
SET min_temp = <temp>
SET temp_slope = <slope>
SET nominal_voltage = <voltage>
SET max_voltage = <voltage>
SET min_voltage = <voltage>
SET voltage_slope = <slope>
SET <type>_process = <factor>
SET nominal_process = <type>
```

An attribute nominal_<attribute> is the condition in which the library parameter <attribute> is currently defined. The attributes max_<attribute> and min_<attribute> are extreme values for a given parameter <attribute>. Between these values <attribute>_slope is used to determine derating factor. <type> can be any keyword (such as typical, worst, fast etc.) which is used to determine the process factor.
**Default Delay Parameters**

The following attributes set the default delay parameters for the gates in the library. If no delay information is given in the gate description, these parameters are used to estimate path delays.

- **SET default_input_drive = <up_drive> <down_drive>**
  
  This specifies the default value for drive resistance. If drive resistance is missing in a particular output pin, this value is used.

- **SET default_output_load = <load>**
  
  This specifies the default output load at the primary outputs. If this is not specified, the load is assumed to be 0. This value can be selectively specified for primary output nodes from the control file.

- **SET default_input_cap = <load>**
  
  This specifies the default input capacitance of input pins of a gate. If the value for Lin of a pin is not specified, this value is used. If this value is also not specified, Lin is assumed to be 0.

- **SET default_slope_sensitivity = <rise> <fall>**
  
  This specifies the default value of slope sensitivity for a path. This is used if the slope sensitivity for a path is ignored.

- **SET default_prop_delay = <rise> <fall>**
  
  This specifies the default propagation delay for a path. This value is used if the prop delay for a node is ignored.

- **SET default_intercept = <rise> <fall>**

**Global Library Properties**
This specifies the default y intercept of the timing curve for PWL (Piecewise Linear) libraries.

```
SET default_setup = <rise> <fall>
```

This specifies the default values for setup. This value is used if setup is not specified for a constraint path.

```
SET default_hold = <rise> <fall>
```

This specifies the default values for hold. This value is used if hold is not specified for a constraint path.

```
SET delay_connect_model = best | balanced | worst;
```

This specifies the default interconnect delay model. This can also be controlled from the command line with a -wire_tree option or wire_tree variable when using Leonardo. In the best case, the wire resistance is 0. In the worst case, all the interconnect load has to be charged through the interconnect resistance. In the balanced case, the resistance and load are balanced over different branches of the net. If this value is ignored, best case is assumed.

```
SET delay_library_model = linear | piecewise_linear | nonlinear
```

This attribute specifies the type of library. If this value is ignored, a linear model is assumed.

**Network Optimization Parameters**

The following attributes affect the manner in which Exemplar constructs the final circuit.

```
SET max_load = <load>
```
This specifies the default maximum load that an output pin can drive. If Lmax for an output pin is not specified, then this value is used. If this value also is not specified, then a very high value is assumed for Lmax. This means the output pin can drive a very high load.

```
SET rep_logic = TRUE
```

To trigger the logic replication algorithm (for handling fanout limitations by replicating logic, instead of inserting buffers).

```
SET fanin_limit = <number_of_fanin>
```

This attribute is used by the lookup table optimization algorithms. Specifies the number of inputs for the lookup table.

```
SET use_wire_max_load
```

Use Wire Load When Checking For LMAX Violations. If this attribute is set interconnect capacitance is also included in checking for max fanout violations.

**Conversion Factor from Unit Loads to Internal Load Units**

```
SET  delay_scale_factor = <factor>
SET  resistance_scale_factor = <factor>
SET  load_scale_factor = <factor>
```

Where factor is a positive real number. This parameter is useful in setting the weight of wiring delay vs. intrinsic (no load) cell delay or in converting the load from standard load to capacitance (real) load.

For example: if delay info in the library is specified in ps a SET delay_scale_factor = 0.001 should be used.
Reporting Information

The following attributes store information for Exemplar’s log and design summary files.

Library Version

The library version is reported in the run window and log file, and allows the user to confirm that the most recent library is being using when running Exemplar.

```
SET lib_version = <version_number>
```

Device Information

When running Exemplar with the reporting switch set to either full or extended (from the command line: -report=[full, extended]), In the design summary file, Exemplar reports on the implementation of the design in the various devices in the target technology. Devices in the library are `DEVICE[i]` where i = 1, 2... n

```
SET device[i] = “<device_name> <#cells> <#registers> <#io_pins>”
```

Area Units

To define the area units used in reporting area estimates for the design, use the following attribute.

```
SET area_units = <area_units>
```

<area_units> is a keyword used to define the area cost of the gates, for example, blocks, gates, square mils, etc.
Delay Units

To define the delay units used in reporting delay estimates for the design, use the following attribute.

```
SET delay_units = <delay_units>
```

<delay_units> is a keyword used to define the delay units, for example, ns, levels, etc.

Wire Load Table

The wire load table or interconnect load table gives an estimate of the wire load (load due to the interconnect between two gates).

One of the terms is the interconnect delay (i.e., delay due to the wire connecting the driver of a net to its fanouts) and is only an estimate. The wire load table, or interconnect load table, associates a wire length with the number of fanouts, i.e., given a fanout it gives an estimate of the wire length. (Since in any network we know the fanout at each driver, we can estimate the wire length). The wire load table also contains parameters for

- resistance (wire_res)
- capacitance (wire_cap)
- area (wire_area)
- slope (wire_slope)

per unit of wire length.
If the fanout of a node is 3, the third item in this table gives an estimate of the wirelength for fanout of 3. If the piece table has a piece type of wire length, this value is used to get to the right piece. If the piece type is wire cap, the wire length is multiplied by the wire_cap per unit length parameter, which is part of the wire table as shown below:

```plaintext
WIRE_TABLE <name> ( 
    WIRE_RES = 1.0;
    WIRE_CAP = 2.0;
    WIRE_AREA = 3.0;
    WIRE_SLOPE = 1.0;
    WIRE_LENGTH = ((0, 0.0), (1, 1.0), (2, 2.8), (10, 17.0))
);
```

The example above defines a wire table of name <name> with wire resistance per unit length = 1.0, etc. The wire_length table is made up of a set of a pair of numbers. The pair of numbers in brackets represents the index of the entry and the value. If fanout = 1 wire_length = 1.0, fanout = 2, wire_length = 2.8, fanout = 10 wire_length = 17.0. This table can have any number of entries. The indices of the fanout table can have gaps (need not be continuous). The values of wirelength for the missing indices are calculated by interpolation. For values of fanout greater than the highest index in the table, the value of wirelength is calculated by extrapolating the highest wirelength using the wire_slope parameter given as part of the wire_table definition. If the piece_type is wire_cap, then the piece which defines the value of drive has to be picked up by computing the wire_cap and finding the right piece number. For a fanout of 2 (from the above definitions of wire table), the wire length is 2.8. The wire_cap is wire_length * wire_cap per unit length. 2.0 * 2.8 = 5.6 pF. The piece table is looked up to find the piece that represents this value of cap. In the above definition of the piece_table, if the piece_type was WIRE_CAP, then the piece number for a capacitance of 5.8 would be piece 3 (the last piece). For values beyond the last piece, the last piece itself is used.
To get the drive of a gate for this value of fanout, the value in the 3rd index is picked up from the drive table. An example of specifying the drive, \( y_{\text{intr}} \) in piecewise linear model is given below. For the piecewise linear model, only the drive and the \( y \) intercept of the delay curve are represented as piecewise linear.

```plaintext
INPUT CLR (
    LIN = 1.000;
    DELAY CLK (
        SETUP = (5.2, 5.2);
    );
    DELAY Q (
        DRIVE = ((1, 0.0, 0.0), (2, 0.0, 0.0), (4, 0.0, 0.0), (7, 0.0, 0.0));
        INTERCEPT = ((1, 0.0, 0.0), (3, 0.0, 0.0), (6, 0.0, 0.0), (8, 0.0, 0.0));
        PROP = (7.3, 7.3);
        PATH_TYPE = DISCONNECT;
    );
    );
```

There could be any number of wire_tables specified in the library. They all should have a unique name.

The Delay Analysis package allows for different (Piece Wise Linear) wire tables. Available wire tables should be documented with the library. FPGA libraries do not have multiple wire load tables. If you have developed your own library, you can select the wire load table on the command line in Exemplar.

```plaintext
-wire_table=<string>
```

and by setting the wire_table variable in Leonardo.

or on the GUI under the Synthesis option in the Logic Explorer menu or Time Analysis option in Time Explorer menu. If you do not want to use any wire load model, you can specify so on the command line

```plaintext
-no_wire_table
```

or by setting the no wire_table variable in Leonardo.
or by selecting the checkbox on the GUI under the Synthesis option in the Logic Explorer menu or Time Analysis option in Time Explorer menu. The delay package allows you to choose the interconnect model used to compute wire delays with the option

\[-\text{wire\_tree}=\text{best} | \text{balanced} | \text{worst} \text{ (balanced is default)}\]

Interconnect delay is computed with the following equation:

\[D_{\text{connect}} = R_{\text{wire}} \times (C_{\text{wire}} + C_{\text{pins}})\]

The wire_tree option determines how the value of \(R_{\text{wire}}\), which is a library parameter, is used to compute \(D_{\text{connect}}\):

- **Best**: \(R_{\text{wire}}\) is 0, i.e., the resistance of the wire does not play a part in the interconnect delay.
- **Balanced**: Each segment of the wire resistance is equally distributed on each of the branches of the net.
- **Worst**: The full wire resistance is used on each branch of the net.
Guidelines and Key Words

Library Building Rules and Guidelines

- **lGen** requires at least a 2-input nand gate and an inverter to be included in the library.
- VCC and GND will be added, with those names, when you build the synthesis library, unless you specify those functions with other names.
- Gate and pin names are case insensitive.
- No loops in combinational logic are allowed. Most loop situations can be specified by using one of the predefined primitives.
- Complex combinational cells with more than one output are allowed for input only. These cells must be designated as NOMAP in a library that will also be used for output.
- For mappable gates, lGen supports at most one predefined primitive per gate. However, there is no limitation to the combinational logic that can accompany that primitive.
- Gate, pin, and node names which are also defined as keywords, or contain non-alphanumeric characters should be quoted. Keywords include the names of the predefined primitives from sections 3.2.1 and 3.2.2, as well as the following:

<table>
<thead>
<tr>
<th>AREA</th>
<th>DELAY</th>
<th>DRIVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNCTION</td>
<td>GATE</td>
<td>INPUT</td>
</tr>
<tr>
<td>INPUTS</td>
<td>LIN</td>
<td>LMAX</td>
</tr>
<tr>
<td>NOMAP</td>
<td>OUTPUT</td>
<td>OUTPUTS</td>
</tr>
</tbody>
</table>
### Checklist of New Library Features

Following are guidelines for using libraries with the new Library Features in Exemplar IGEN:

1. At least one path needs to exist from input pin to output pin in a gate.

2. When specifying constraints (setup/hold) for flipflops and latches, make sure both the from and to pins are specified. If the to pin is missing, it defaults to all the outputs.

3. The path from CLEAR or PRESET to Q and QN of a flipflop should have path_type set to disconnect. (PATH_TYPE = DISCONNECT). This option makes these paths disconnected by default. However, these can be reconnected from the control file on a per run basis, for specific gates, instances, etc.

4. The path from PAD to Y of BDUBFs should have a path_type of PAD_TO_Y. (PATH_TYPE = PAD_TO_Y), which is a special type of path.

5. The to pin in a path for combinational gates can be ignored. This action causes a default to all the outputs (in case of combinational gates only one output)

6. The to pin in sequential gates for CLK to Q or QN can be ignored as well for the same reason as above.

Check if the wire_table (wire_load_model) is in the new format.

---

<table>
<thead>
<tr>
<th>PAD</th>
<th>PHASE</th>
<th>PROP</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET</td>
<td>SETUP</td>
<td>SAME_TECH_NOOPT</td>
</tr>
<tr>
<td>PATH_TYPE</td>
<td>PAD_TO_Y</td>
<td>DISCONNECT</td>
</tr>
<tr>
<td>CONS_PIN_TRANS</td>
<td>RELATED_PIN_TRANS</td>
<td>LUT_TEMPLATE</td>
</tr>
<tr>
<td>VARIABLE_X</td>
<td>VARIABLE_Y</td>
<td>PARAMETER</td>
</tr>
<tr>
<td>CELL_DELAY_RISE</td>
<td>CELL_DELAY_FULL</td>
<td>TRANSITION_RISE</td>
</tr>
<tr>
<td>TRANSITION_FALL</td>
<td>PROPAGATION_FALL</td>
<td>SETUP_RISE</td>
</tr>
<tr>
<td>SETUP_FULL</td>
<td>HOLD_RISE</td>
<td>HOLD_FALL</td>
</tr>
<tr>
<td>INPUT_NET_TRANS</td>
<td>OUTPUT_NET_LENGTH</td>
<td>OUTPUT_PIN_CAPS</td>
</tr>
<tr>
<td>OUTPUT_WIRE_CAPS</td>
<td>OUTPUT_TOTAL_CAPS</td>
<td>INTERCEPT</td>
</tr>
<tr>
<td>SLOPE</td>
<td>WIRE_TABLE</td>
<td>HOLD</td>
</tr>
<tr>
<td>CIN</td>
<td>WIRE_LENGTH</td>
<td>WIRE_CAP</td>
</tr>
<tr>
<td>PIN_CAP</td>
<td>TOTAL_CAP</td>
<td></td>
</tr>
</tbody>
</table>

For example, a gate named DELAY would be specified as GATE “DELAY”
This appendix describes the predefined primitives used to describe gate functionality when creating synthesis libraries using lBuild.

The function call is given first, followed by the truth table for that primitive. Usage examples are given both in this appendix and the following appendix, which contains a library example.

**Sequential and Tristate Logic**

<table>
<thead>
<tr>
<th>D</th>
<th>CK</th>
<th>SD</th>
<th>RD</th>
<th>Q</th>
<th>QN</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>^</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>^</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>x</td>
<td>^</td>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>QN</td>
</tr>
</tbody>
</table>
DFFE(SD, RD, D, CK, E, Q, QN)

<table>
<thead>
<tr>
<th>D</th>
<th>CK</th>
<th>E</th>
<th>SD</th>
<th>RD</th>
<th>Q</th>
<th>QN</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>^</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>^</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>x</td>
<td>^</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>QN</td>
</tr>
</tbody>
</table>

DLATCH(SD, RD, D, G, Q, QN)

<table>
<thead>
<tr>
<th>D</th>
<th>G</th>
<th>SD</th>
<th>RD</th>
<th>Q</th>
<th>QN</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>x</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>QN</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

If a DLATCH is modeled as

\[
DLATCH(-,-,D,G,Q,QBAR)
\]

then the algorithm which attempts to map a DLATCH out of combinational logic will not find a DLATCH. (And Leonardo will not find them from either VHDL or Verilog).

The better way to model a DLATCH is

\[
QBAR = !Q;
DLATCH(-,-,D,G,Q,-);
\]
Predefined Primitives

RSLATCH(S, R, Q, QN)

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>QN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>QN</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

TRST(A, E, Z)

Internal tristate device.

<table>
<thead>
<tr>
<th>A</th>
<th>E</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
<td>Z</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Buffers

IBUF(PAD, Y)

Input buffer connected to pad.

<table>
<thead>
<tr>
<th>PAD</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

OBUF(D, PAD)

Output buffer connected to pad.

<table>
<thead>
<tr>
<th>D</th>
<th>PAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

TBUF(D, E, PAD)
Tristate buffer connected to pad.

<table>
<thead>
<tr>
<th>D</th>
<th>E</th>
<th>PAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
<td>z</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

BDBUF(D, E, Y, PAD)

Bidirectional buffer connected to pad.

<table>
<thead>
<tr>
<th>D</th>
<th>E</th>
<th>Y</th>
<th>PAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>x</td>
<td>0</td>
<td>PAD</td>
<td>z</td>
</tr>
</tbody>
</table>

CBUF(X, Y)

Internal clock buffer.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Constants**

PULLUP(O)

Weak device continuously hooked up to VCC.

PULLDN(O)

Weak device continuously hooked up to Ground.

TRSTMEM(Z)

Weak device continuously sustaining existing value on node.

CONST1
VCC function.

CONST0

GND function.
Library Source Example

Linear Delay Model Example

LIBRARY test;
GATE GND (INPUTS; OUTPUTS Y; FUNCTION (Y = CONST0);
    AREA = 0.000);
GATE VCC (INPUTS; OUTPUTS Y; FUNCTION (Y = CONST1);
    AREA = 0.000);
GATE BUF (INPUTS A; OUTPUTS Y; FUNCTION (Y = A);
    AREA = 1.000);
    INPUT A (LIN = 1.000; DELAY (}
DRIVE = (0.1, 0.1);
SLOPE = (0.0, 0.0);
PROP = (7.300, 7.300);
PHASE = UNKNOWN;
);
);
OUTPUT Y (  
LMAX = 999.000;
);
);
GATE INV (  
INPUTS A;
OUTPUTS Y;
FUNCTION (  
Y = !A;
);
AREA = 1.000;
INPUT A (  
LIN = 1.000;
DELAY (  
DRIVE = (0.1, 0.1);
PROP = (7.300, 7.300);
SLOPE = (0.0, 0.0);
PHASE = UNKNOWN;
  );
);
OUTPUT Y (  
LMAX = 999.000;
  );
);
GATE INBUF (  
INPUTS PAD;
OUTPUTS Y;
FUNCTION (  
IBUF (PAD, Y);
);
AREA = 0.000;SET io_module = 1.000;
INPUT PAD (  
LIN = 0.000;
DELAY (  
DRIVE = (0.1, 0.1);
PROP = (10.000, 10.000);
SLOPE = (0.0, 0.0);
PHASE = UNKNOWN;
  );
) SET PAD;
```
C

);   OUTPUT Y (      LMAX = 999.000;   );
GATE CLKBUF (      INPUTS PAD;      OUTPUTS Y;      FUNCTION (          CBUF (PAD, Y);   );
            AREA =  0.000;SET io_module =  1.000;
            INPUT PAD (      LIN =  0.000;
                DELAY (                      DRIVE = (0.5, 0.5);
                                    PROP = ( 0.000, 0.000);
                                    SLOPE = (0.0, 0.0);
                                    PHASE = UNKNOWN;
                );
                SET PAD;
            );
            OUTPUT Y (      LMAX =  999.0;
            );
GATE XOR (      INPUTS A, B;
      OUTPUTS Y;
      FUNCTION (          Y = A*!B + !A*B;
      );
            AREA =  1.000;
            INPUT A (      LIN =  1.000;
                DELAY (                      DRIVE = (0.5, 0.5);
                                    PROP = ( 7.300, 7.300);
                                    SLOPE = (0.0, 0.0);
                                    PHASE = UNKNOWN;
                );
            );
            INPUT B (      LIN =  1.000;
                DELAY (                      DRIVE = (0.5, 0.5);
                );

Library Source Example

C

65
PROP = ( 7.300, 7.300);  
SLOPE = (0.0, 0.0);  
PHASE = UNKNOWN;

OUTPUT Y (  
LMAX = 999.000;  
);

GATE XNOR (  
  INPUTS A, B;  
  OUTPUS Y;  
  FUNCTION (  
    Y = !A*B + A*B;  
  );  
  AREA = 1.000;  
  INPUT A (  
    LIN = 1.000;  
    DELAY (  
      DRIVE = (0.5, 0.5);  
      PROP = ( 7.300, 7.300);  
      SLOPE = (0.0, 0.0);  
      PHASE = UNKNOWN;  
    );  
  );  
  INPUT B (  
    LIN = 1.000;  
    DELAY (  
      DRIVE = (0.5, 0.5);  
      PROP = ( 7.300, 7.300);  
      SLOPE = (0.0, 0.0);  
      PHASE = UNKNOWN;  
    );  
  );  
  OUTPUT Y (  
    LMAX = 999.000;  
  );  
);

GATE NAND2 (  
  INPUTS A, B;  
  OUTPUS Y;  
  FUNCTION (  
    Y = !B + !A;  
  );  
  AREA = 1.000;  
  INPUT A (
LIN = 1.000;
DELAY (  
    DRIVE = (0.5, 0.5);
    PROP = (7.300, 7.300);
    SLOPE = (0.0, 0.0);
    PHASE = UNKNOWN;
);
);  
INPUT B (  
    LIN = 1.000;
    DELAY (  
        DRIVE = (0.5, 0.5);
        PROP = (7.300, 7.300);
        SLOPE = (0.0, 0.0);
        PHASE = UNKNOWN;
    );
    );
);  
OUTPUT Y (  
    LMAX = 999.000;
    );
);  
GATE NOR2 (  
    INPUTS A, B;
    OUTPUTS Y;
    FUNCTION (  
        Y = !A*!B;
    );
    AREA = 1.000;
    INPUT A (  
        LIN = 1.000;
        DELAY (  
            DRIVE = (0.5, 0.5);
            PROP = (7.300, 7.300);
            SLOPE = (0.0, 0.0);
            PHASE = UNKNOWN;
        );
    );
    );  
INPUT B (  
    LIN = 1.000;
    DELAY (  
        DRIVE = (0.5, 0.5);
        PROP = (7.300, 7.300);
        SLOPE = (0.0, 0.0);
        PHASE = UNKNOWN;
    );
    );
OUTPUT Y ( 
    LMAX = 999.000;
 );
);
GATE NAND3 ( 
    INPUTS A, B, C;
    OUTPUTS Y;
    FUNCTION ( 
        Y = !C + !B + !A;
    );
    AREA = 2.000;
    INPUT A ( 
        LIN = 2.000;
        DELAY ( 
            DRIVE = (0.5, 0.5);
            PROP = (7.300, 7.300);
            SLOPE = (0.0, 0.0);
            PHASE = UNKNOWN;
        );
    );
    INPUT B ( 
        LIN = 1.000;
        DELAY ( 
            DRIVE = (0.5, 0.5);
            PROP = (7.300, 7.300);
            SLOPE = (0.0, 0.0);
            PHASE = UNKNOWN;
        );
    );
    INPUT C ( 
        LIN = 1.000;
        DELAY ( 
            PROP = (7.300, 7.300);
            SLOPE = (0.0, 0.0);
            DRIVE = (0.5, 0.5);
            PHASE = UNKNOWN;
        );
    );
    OUTPUT Y ( 
        LMAX = 999.000;
    );
);)
GATE AND3 ( 
    INPUTS A, B, C;
    OUTPUTS Y;
    FUNCTION (
Y = A*B*C;
);
AREA = 1.000;
INPUT A ( 
   LIN = 1.000;
   DELAY ( 
      PROP = ( 7.300,  7.300);
      SLOPE = (0.0, 0.0);
      DRIVE = (0.5, 0.5);
      PHASE = UNKNOWN;
   );
);
INPUT B ( 
   LIN = 1.000;
   DELAY ( 
      DRIVE = (0.5, 0.5);
      PROP = ( 7.300,  7.300);
      SLOPE = (0.0, 0.0);
      PHASE = UNKNOWN;
   );
);
INPUT C ( 
   LIN = 1.000;
   DELAY ( 
      DRIVE = (0.5, 0.5);
      PROP = ( 7.300,  7.300);
      SLOPE = (0.0, 0.0);
      PHASE = UNKNOWN;
   );
);
OUTPUT Y ( 
   LMAX = 999.000;
);
);
GATE OUTBUF ( 
   INPUTS D;
   OUTPUTS PAD; 
   FUNCTION ( 
      OBUF (D, PAD);
   );
)
AREA = 0.000;SET io_module = 1.000;
INPUT D ( 
   LIN = 1.000;
   DELAY ( 
      PROP = ( 0.000, 0.000);
      SLOPE = (0.0, 0.0);
DRIVE = (0.5, 0.5);
PHASE = UNKNOWN;
);
)
OUTPUT PAD ( 
  SET PAD;
);
)
GATE TRIBUFF ( 
  INPUTS D, E;
  OUTPUTS PAD;
  FUNCTION ( 
    TBUF (D, E, PAD);
  );
  AREA = 0.000;SET io_module = 1.000;
INPUT D ( 
  LIN = 1.000;
  DELAY ( 
    DRIVE = (0.5, 0.5);
    PROP = (0.000, 0.000);
    SLOPE = (0.0, 0.0);
    PHASE = UNKNOWN;
  );
 );
INPUT E ( 
  LIN = 1.000;
  DELAY ( 
    DRIVE = (0.5, 0.5);
    PROP = (0.000, 0.000);
    SLOPE = (0.0, 0.0);
    PHASE = UNKNOWN;
  );
 );
OUTPUT PAD ( 
  SET PAD;
);
)
GATE BIBUF ( 
  INPUTS D, E;
  OUTPUTS PAD, Y;
  FUNCTION ( 
    BDBUF (D, E, Y, PAD);
  );
  AREA = 0.000;SET io_module = 1.000;
INPUT D ( 
  LIN = 1.000;
DELAY PAD {
    DRIVE = (0.5, 0.5);
    PROP = (0.000, 0.000);
    SLOPE = (0.0, 0.0);
    PHASE = UNKNOWN;
};

INPUT E {
    LIN = 1.000;
    DELAY PAD {
        PROP = (0.000, 0.000);
        SLOPE = (0.0, 0.0);
        DRIVE = (0.5, 0.5);
        PHASE = UNKNOWN;
    }
};

OUTPUT PAD {
    SET PAD;
    DELAY Y {
        PROP = (7.300, 7.300);
        SLOPE = (0.0, 0.0);
        DRIVE = (0.5, 0.5);
        PHASE = UNKNOWN;
        PATH_TYPE = PAD_TO_Y;
    }
};

OUTPUT Y {
    LMAX = 999.000;
};

GATE OR3 {
    INPUTS A, B, C;
    OUTPUTS Y;
    FUNCTION {
        Y = C + B + A;
    };
    AREA = 1.000;

    INPUT A {
        LIN = 1.000;
        DELAY {
            PROP = (7.300, 7.300);
            SLOPE = (0.0, 0.0);
            DRIVE = (0.5, 0.5);
            PHASE = UNKNOWN;
        }
    };
}
INPUT B {
    LIN = 1.000;
    DELAY {
        PROP = (7.300, 7.300);
        SLOPE = (0.0, 0.0);
        DRIVE = (0.5, 0.5);
        PHASE = UNKNOWN;
    };
};

INPUT C {
    LIN = 1.000;
    DELAY {
        PROP = (7.300, 7.300);
        SLOPE = (0.0, 0.0);
        DRIVE = (0.5, 0.5);
        PHASE = UNKNOWN;
    };
};

OUTPUT Y {
    LMAX = 999.000;
};

GATE DL1 {
    INPUTS D, G;
    OUTPUTS Q;
    FUNCTION {
        DLATCH (-, -, D, G, Q, -);
    };
    AREA = 1.000;
    #
    #SET COMPLEMENT = DL1A;
    INPUT D {
        LIN = 1.000;
        DELAY G {
            SETUP = (4.200, 4.200);
        };
    };

    INPUT G {
        LIN = 1.000;
        DELAY Q {
            PROP = (7.300, 7.300);
            SLOPE = (0.0, 0.0);
            DRIVE = (0.5, 0.5);
        };
    };

    OUTPUT Q {
        LMAX = 999.000;
    };
}
GATE DLMA {
    INPUTS A, B, G, S;
    OUTPUTS Q;
    FUNCTION {
        [3272] = !G;
        [3271] = A*S + B*S;
        DLATCH (-, -, [3271], [3272], Q, -);
    }
    AREA = 1.000;
    INPUT A {
        LIN = 1.000;
        DELAY G {
            SETUP = (4.200, 4.200);
        }
    }
    INPUT B {
        LIN = 1.000;
        DELAY G {
            SETUP = (4.200, 4.200);
        }
    }
    INPUT S {
        LIN = 1.000;
        DELAY G {
            SETUP = (4.200, 4.200);
        }
    }
    INPUT G {
        LIN = 1.000;
        DELAY Q {
            PROP = (7.300, 7.300);
            SLOPE = (0.0, 0.0);
            DRIVE = (0.5, 0.5);
        }
    }
    OUTPUT Q {
        LMAX = 999.000;
    }
}

GATE DF1 {
    INPUTS CLK, D;
    OUTPUTS Q, QN;
    FUNCTION {
        QN = !Q;
    }
}
DFF (−, −, D, CLK, Q, −);
);
AREA = 2.000;
#SET COMPLEMENT = DF1A;
INPUT D (       
    LIN = 1.000;
    DELAY CLK (      
        SETUP = ( 4.200, 4.200);   
    );
);
INPUT CLK (      
    LIN = 2.000;
    DELAY Q (        
        PROP = ( 7.300, 7.300);   
        SLOPE = (0.0, 0.0);       
        DRIVE = (0.5, 0.5);      
    );
);
OUTPUT Q (       
    LMAX = 999.000;
    DELAY QN (       
        PROP = ( 7.300, 7.300);   
        SLOPE = (0.0, 0.0);       
        DRIVE = (0.5, 0.5);      
    );
);
OUTPUT QN (      
    LMAX = 999.000;
    );
);
GATE DFC1 (     
    INPUTS CLK, CLR, D;  
    OUTPUTS Q, QN;        
    FUNCTION (           
        QN = !Q;               
        DFF (−, CLR, D, CLK, Q, −);       
    );
);
AREA = 2.000;
#SET COMPLEMENT = DFC1C;
INPUT D (       
    LIN = 1.000;
    DELAY CLK (      
        SETUP = ( 4.200, 4.200);   
    );
);
INPUT CLR (     

LIN = 1.000;
DELAY CLK {
    SETUP = ( 5.200, 5.200);
};
DELAY Q {
    PROP = ( 7.300, 7.300);
    SLOPE = (0.0, 0.0);
    DRIVE = (0.5, 0.5);
    PATH_TYPE = DISCONNECT;
}
INPUT CLK {
    LIN = 2.000;
    DELAY Q {
        DRIVE = (0.5, 0.5);
        PROP = ( 7.300, 7.300);
        SLOPE = (0.0, 0.0);
    }
};
OUTPUT Q {
    LMAX = 999.000;
    DELAY QN {
        DRIVE = (0.5, 0.5);
        PROP = ( 8.300, 8.300);
        SLOPE = (0.0, 0.0);
    }
};
OUTPUT QN {
    LMAX = 999.000;
};
SET default_input_drive = 0.000000 0.000000;
SET default_output_load = 1.000000;
SET max_load = 10;
SET opt_for_area_weights = 1.0 0.0 0.2 0.0;
SET opt_for_speed_weights = 0.01 1.0 0.2 0.0;
SET nominal_process = typical;
SET fast_process = 0.60;
SET typical_process = 1.00;
SET worst_process = 1.03;
SET nominal_temp = 25.0;
SET min_temp = -55.0;
SET max_temp = 125.0;
SET temp_slope = 0.0025;
SET nominal_voltage = 5.0;
SET min_voltage = 4.5;
SET max_voltage = 5.5;
SET voltage_slope = -0.14;
SET lib_version = "1.14";
SET library_security = actel;
SET rep_logic = true;
SET use_wire_max_load;
SET delay_connect_model = worst;
#defines a wire_load model

WIRE_TABLE "minimum" {
  wire_res = 1.0;
  wire_cap = 1.0;
  wire_area = 3.0;
  wire_slope = 1.0;
  wire_length = ((0, 0.0), (1, 1.0), (2, 2.0), (8, 8.0))
};

WIRE_TABLE "nominal" {
  wire_res = 1.0;
  wire_cap = 1.0;
  wire_area = 3.0;
  wire_slope = 1.0;
  wire_length = ((0, 0.0), (1, 2.0), (2, 4.0), (3, 6.0))
};

Piecewise Linear Delay Model Example

LIBRARY test {
  GATE GND ( 
    INPUTS;
    OUTPUTS Y;
    FUNCTION ( 
      Y = CONST0;
      area = 0.000;
    );
  );
  GATE VCC ( 
    INPUTS;
    OUTPUTS Y;
    FUNCTION ( 
      Y = CONST1;
      area = 0.000;
    );
  );
  GATE BUF ( 
    INPUTS A;
  );
OUTPUTS Y;
FUNCTION ( 
    Y = A;
); 
AREA = 1.000;
INPUT A ( 
    LIN = 1.000;
    DELAY ( 
        DRIVE =((0, 0.1, 0.1), (1, 0.2, 0.2), (2, 0.3, 0.3), (4, 0.4, 0.4));
        INTERCEPT= ((0, 0.1, 0.1), (1, 0.2, 0.2), (2, 0.3, 0.3), (4, 0.4, 0.4));
        PROP = ( 7.300, 7.300);
        SLOPE = ( 0.0, 0.0);
        PHASE = UNKNOWN;
    );
    OUTPUT Y ( 
        LMAX = 999.000;
    );
); 
GATE INV ( 
    INPUTS A;
    OUTPUTS Y;
    FUNCTION ( 
        Y = !A;
    ); 
    AREA = 1.000;
    INPUT A ( 
        LIN = 1.000;
        DELAY ( 
            DRIVE =((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));
            INTERCEPT= ((0, 0.1, 0.1), (1, 0.2, 0.2), (2, 0.3, 0.3), (3, 0.4, 0.4));
            PROP = ( 7.300, 7.300);
            SLOPE = ( 0.0, 0.0);
            PHASE = UNKNOWN;
        );
        OUTPUT Y ( 
            LMAX = 999.000;
        );
    );
); 
GATE INBUF ( 
    INPUTS PAD;
    OUTPUTS Y;
    FUNCTION ( 
        IBUF (PAD, Y);
    );
\begin{verbatim}
AREA = 0.000; SET io_module = 1.000;
INPUT PAD {
  LIN = 0.000;
  DELAY {
    DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));
    INTERCEPT = ((0, 0.1, 0.1), (1, 0.2, 0.2), (2, 0.3, 0.3), (3, 0.4, 0.4));
    PROP = (10.000, 10.000);
    SLOPE = (0.0, 0.0);
    PHASE = UNKNOWN;
  };
  SET PAD;
};
OUTPUT Y {
  LMAX = 999.000;
};
};
GATE CLKBUF {
  INPUTS PAD;
  OUTPUTS Y;
  FUNCTION {
    CBUF (PAD, Y);
  };
  AREA = 0.000; SET io_module = 1.000;
};
INPUT PAD {
  LIN = 0.000;
  DELAY {
    DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));
    INTERCEPT = ((0, 0.1, 0.1), (1, 0.2, 0.2), (2, 0.3, 0.3), (3, 0.4, 0.4));
    PROP = (0.500, 0.500);
    SLOPE = (0.0, 0.0);
    PHASE = UNKNOWN;
  };
  SET PAD;
};
OUTPUT Y {
  LMAX = 999.0;
};
};
GATE XOR {
  INPUTS A, B;
  OUTPUTS Y;
  FUNCTION {
    Y = A*!B + !A*B;
  };
  AREA = 1.000;
}INPUT A (}
\end{verbatim}
LIN = 1.000;
DELAY {
  DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));
  INTERCEPT =((0, 0.1, 0.1),(1, 0.2, 0.2),(2, 0.3, 0.3),(3, 0.4, 0.4));
  PROP = ( 7.300, 7.300);
  SLOPE = ( 0.0, 0.0);
  PHASE = UNKNOWN;
};

INPUT B {
  LIN = 1.000;
  DELAY {
    DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));
    INTERCEPT =((0, 0.1, 0.1),(1, 0.2, 0.2),(2, 0.3, 0.3),(3, 0.4, 0.4));
    PROP = ( 7.300, 7.300);
    SLOPE = ( 0.0, 0.0);
    PHASE = UNKNOWN;
  };
}

OUTPUT Y {
  LMAX = 999.000;
};

GATE XNOR {
  INPUTS A, B;
  OUTPUTS Y;
  FUNCTION (Y = !A*!B + A*B);
};

AREA = 1.000;

INPUT A {
  LIN = 1.000;
  DELAY {
    DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));
    INTERCEPT =((0, 0.1, 0.1),(1, 0.2, 0.2),(2, 0.3, 0.3),(3, 0.4, 0.4));
    PROP = ( 7.300, 7.300);
    SLOPE = ( 0.0, 0.0);
    PHASE = UNKNOWN;
  };
}

INPUT B {
  LIN = 1.000;
  DELAY {
    DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));
    INTERCEPT =((0, 0.1, 0.1),(1, 0.2, 0.2),(2, 0.3, 0.3),(3, 0.4, 0.4));
    PROP = ( 7.300, 7.300);
}
SLOPE = ( 0.0, 0.0);
PHASE = UNKNOWN;
)

OUTPUT Y (LMAX = 999.000;
)

GATE NAND2 (INPUTS A, B;
OUTPUTS Y;
FUNCTION (Y = !B + !A;
)
AREA = 1.000;
INPUT A (LIN = 1.000;
DELAY (DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));
INTERCEPT =((0, 0.1, 0.1),(1, 0.2, 0.2),(2, 0.3, 0.3),(3, 0.4, 0.4));
PROP = (7.300, 7.300);
SLOPE = (0.0, 0.0);
PHASE = UNKNOWN;
)
);
INPUT B (LIN = 1.000;
DELAY (DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));
INTERCEPT =((0, 0.1, 0.1),(1, 0.2, 0.2),(2, 0.3, 0.3),(3, 0.4, 0.4));
PROP = (7.300, 7.300);
SLOPE = (0.0, 0.0);
PHASE = UNKNOWN;
)
);
OUTPUT Y (LMAX = 999.000;
)

GATE NOR2 (INPUTS A, B;
OUTPUTS Y;
FUNCTION (Y = !A*!B;
)
AREA = 1.000;
INPUT A (  
    LIN = 1.000;  
    DELAY (  
        DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));  
        INTERCEPT =((0, 0.1, 0.1),(1, 0.2, 0.2),(2, 0.3, 0.3),(3, 0.4, 0.4));  
        PROP = ( 7.300, 7.300);  
        SLOPE = ( 0.0, 0.0);  
        PHASE = UNKNOWN;  
    );  
);  
INPUT B (  
    LIN = 1.000;  
    DELAY (  
        DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));  
        INTERCEPT =((0, 0.1, 0.1),(1, 0.2, 0.2),(2, 0.3, 0.3),(3, 0.4, 0.4));  
        PROP = ( 7.300, 7.300);  
        SLOPE = ( 0.0, 0.0);  
        PHASE = UNKNOWN;  
    );  
);  
OUTPUT Y (  
    LMAX = 999.000;  
);  
);  
GATE NAND3 (  
    INPUTS A, B, C;  
    OUTPUTS Y;  
    FUNCTION (  
        Y = !C + !B + !A;  
    );  
    AREA = 2.000;  
    INPUT A (  
        LIN = 2.000;  
        DELAY (  
            DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));  
            INTERCEPT =((0, 0.1, 0.1),(1, 0.2, 0.2),(2, 0.3, 0.3),(3, 0.4, 0.4));  
            PROP = ( 7.300, 7.300);  
            SLOPE = ( 0.0, 0.0);  
            PHASE = UNKNOWN;  
        );  
    );  
    INPUT B (  
        LIN = 1.000;  
        DELAY (  
            DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));  
            INTERCEPT =((0, 0.1, 0.1),(1, 0.2, 0.2),(2, 0.3, 0.3),(3, 0.4, 0.4));  
        );  
    );
PROP = (7.300, 7.300);  
SLOPE = (0.0, 0.0);  
PHASE = UNKNOWN;

INPUT C (  
    LIN = 1.000;  
    DELAY (  
        PROP = (7.300, 7.300);  
        SLOPE = (0.0, 0.0);  
        DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));  
        INTERCEPT = (0, 0.1, 0.1), (1, 0.2, 0.2), (2, 0.3, 0.3), (3, 0.4, 0.4));  
        PHASE = UNKNOWN;
    )
);

OUTPUT Y (  
    LMAX = 999.000;
);

GATE AND3 (  
    INPUTS A, B, C;  
    OUTPUTS Y;  
    FUNCTION (  
        Y = A*B*C;
    );  
    AREA = 1.000;
    INPUT A (  
        LIN = 1.000;  
        DELAY (  
            PROP = (7.300, 7.300);  
            SLOPE = (0.0, 0.0);  
            DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));  
            INTERCEPT = (0, 0.1, 0.1), (1, 0.2, 0.2), (2, 0.3, 0.3), (3, 0.4, 0.4));  
            PHASE = UNKNOWN;
        )
    );
    INPUT B (  
        LIN = 1.000;  
        DELAY (  
            DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));  
            INTERCEPT = (0, 0.1, 0.1), (1, 0.2, 0.2), (2, 0.3, 0.3), (3, 0.4, 0.4));  
            PROP = (7.300, 7.300);  
            SLOPE = (0.0, 0.0);  
            PHASE = UNKNOWN;
        )
    );
INPUT C (
  LIN = 1.000;
  DELAY (
    DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));
    INTERCEPT = ((0, 0.1, 0.1), (1, 0.2, 0.2), (2, 0.3, 0.3), (3, 0.4, 0.4));
    PROP = (7.300, 7.300);
    SLOPE = (0.0, 0.0);
    PHASE = UNKNOWN;
  );
);
OUTPUT Y (
  LMAX = 999.000;
);
);
GATE OUTBUF (
  INPUTS D;
  OUTPUTS PAD;
  FUNCTION ( OBUF (D, PAD);
  );
  AREA = 0.000; SET io_module = 1.000;
INPUT D (
  LIN = 1.000;
  DELAY ( 
    PROP = (0.000, 0.000);
    SLOPE = (0.0, 0.0);
    DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));
    INTERCEPT = ((0, 0.1, 0.1), (1, 0.2, 0.2), (2, 0.3, 0.3), (3, 0.4, 0.4));
    PHASE = UNKNOWN;
  );
);
OUTPUT PAD ( 
  SET PAD;
);
);
GATE TRIBUFF ( 
  INPUTS D, E;
  OUTPUTS PAD;
  FUNCTION ( TBUF (D, E, PAD);
  );
  AREA = 0.000; SET io_module = 1.000;
INPUT D ( 
  LIN = 1.000;
  DELAY ( 
    DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));
  );
)
\begin{verbatim}
INTERCEPT =((0, 0.1, 0.1),(1, 0.2, 0.2),(2, 0.3, 0.3),(3, 0.4, 0.4));
PROP = (0.000, 0.000);
SLOPE = (0.0, 0.0);
PHASE = UNKNOWN;

INPUT E
    LIN = 1.000;
    DELAY
        DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));
        INTERCEPT =((0, 0.1, 0.1),(1, 0.2, 0.2),(2, 0.3, 0.3),(3, 0.4, 0.4));
        PROP = (0.000, 0.000);
        SLOPE = (0.0, 0.0);
        PHASE = UNKNOWN;
    );
OUTPUT PAD
    SET PAD;
); GATE BIBUF
    INPUTS D, E;
    OUTPUTS PAD, Y;
    FUNCTION
        BDBUF (D, E, Y, PAD);
    );
    AREA = 0.000; SET io_module = 1.000;
INPUT D
    LIN = 1.000;
    DELAY PAD
        DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));
        INTERCEPT =((0, 0.1, 0.1),(1, 0.2, 0.2),(2, 0.3, 0.3),(3, 0.4, 0.4));
        PROP = (0.000, 0.000);
        SLOPE = (0.0, 0.0);
        PHASE = UNKNOWN;
    );
); INPUT E
    LIN = 1.000;
    DELAY PAD
        PROP = (0.000, 0.000);
        SLOPE = (0.0, 0.0);
        DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));
        INTERCEPT =((0, 0.1, 0.1),(1, 0.2, 0.2),(2, 0.3, 0.3),(3, 0.4, 0.4));
        PHASE = UNKNOWN;
    );
\end{verbatim}
C

OUTPUT PAD {
  SET PAD;
  DELAY Y {
    PROP = ( 7.300, 7.300);
    SLOPE = ( 0.0, 0.0);
    DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));
    INTERCEPT = ((0, 0.1, 0.1), (1, 0.2, 0.2), (2, 0.3, 0.3), (3, 0.4, 0.4));
    PHASE = UNKNOWN;
    PATH_TYPE = PAD_TO_Y;
  };
}
OUTPUT Y {
  LMAX = 999.000;
};
GATE OR3 {
  INPUTS A, B, C;
  OUTPUTS Y;
  FUNCTION {
    Y = C + B + A;
  };
  AREA = 1.000;
  INPUT A {
    LIN = 1.000;
    DELAY {
      PROP = ( 7.300, 7.300);
      SLOPE = ( 0.0, 0.0);
      DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));
      INTERCEPT = ((0, 0.1, 0.1), (1, 0.2, 0.2), (2, 0.3, 0.3), (3, 0.4, 0.4));
      PHASE = UNKNOWN;
    };
  };
  INPUT B {
    LIN = 1.000;
    DELAY {
      PROP = ( 7.300, 7.300);
      SLOPE = ( 0.0, 0.0);
      DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));
      INTERCEPT = ((0, 0.1, 0.1), (1, 0.2, 0.2), (2, 0.3, 0.3), (3, 0.4, 0.4));
      PHASE = UNKNOWN;
    };
  };
  INPUT C {
    LIN = 1.000;
    DELAY {

PROP = ( 7.300, 7.300);
SLOPE = ( 0.0, 0.0);
DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));
INTERCEPT=((0, 0.1, 0.1),(1, 0.2, 0.2),(2, 0.3, 0.3),(3, 0.4, 0.4));
PHASE = UNKNOWN;

OUTPUT Y (LMAX = 999.000);
INPUT A ( 
   LIN = 1.000;
   DELAY G ( 
      SETUP = (4.200, 4.200);
   );
);

INPUT B ( 
   LIN = 1.000;
   DELAY G ( 
      SETUP = (4.200, 4.200);
   );
);

INPUT S ( 
   LIN = 1.000;
   DELAY G ( 
      SETUP = (4.200, 4.200);
   );
);

INPUT G ( 
   LIN = 1.000;
   DELAY Q ( 
      PROP = (7.300, 7.300);
      SLOPE = (0.0, 0.0);
      DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));
      INTERCEPT = ((0, 0.1, 0.1), (1, 0.2, 0.2), (2, 0.3, 0.3), (3, 0.4, 0.4));
   );
);

OUTPUT Q ( 
   LMAX = 999.000;
 );

GATE DF1 ( 
   INPUTS CLK, D;
   OUTPUTS Q, QN;
   FUNCTION ( 
      QN = !Q;
      DFF (-, -, D, CLK, Q, -);
   );
   AREA = 2.000;
   INPUT D ( 
      LIN = 1.000;
      DELAY CLK ( 
         SETUP = (4.200, 4.200);
      );
   );
   INPUT CLK (
LIN = 2.000;
DELAY Q {
  PROP = ( 7.300, 7.300);
  SLOPE = ( 0.0, 0.0);
  DRIVE = (0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));
  INTERCEPT =((0, 0.1, 0.1),(1, 0.2, 0.2),(2, 0.3, 0.3),(3, 0.4, 0.4));
};
OUTPUT Q {
  LMAX = 999.000;
  DELAY QN {
    PROP = ( 7.300, 7.300);
    SLOPE = ( 0.0, 0.0);
    DRIVE = (0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));
    INTERCEPT =((0, 0.1, 0.1),(1, 0.2, 0.2),(2, 0.3, 0.3),(3, 0.4, 0.4));
  };
  OUTPUT QN {
    LMAX = 999.000;
  };
};

GATE DFC1 {
  INPUTS CLK, CLR, D;
  OUTPUTS Q, QN;
  FUNCTION {
    QN = !Q;
    DFF (~, CLR, D, CLK, Q, ~);
  };
  AREA = 2.000;
  INPUT D {
    LIN = 1.000;
    DELAY CLK {
      SETUP = ( 4.200, 4.200);
    };
  };
  INPUT CLR {
    LIN = 1.000;
    DELAY CLK {
      SETUP = ( 5.200, 5.200);
    };
    DELAY Q {
      PROP = ( 7.300, 7.300);
      SLOPE = ( 0.0, 0.0);
      DRIVE = (0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));
      INTERCEPT =((0, 0.1, 0.1),(1, 0.2, 0.2),(2, 0.3, 0.3),(3, 0.4, 0.4));
      PATH_TYPE = DISCONNECT;
  };
}
C

);  
);  
INPUT CLK (  
LIN = 2.000;  
DELAY Q (  
   DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));  
   INTERCEPT = ((0, 0.1, 0.1), (1, 0.2, 0.2), (2, 0.3, 0.3), (3, 0.4, 0.4));  
   PROP = ( 7.300, 7.300);  
   SLOPE = ( 0.0, 0.0);  
);  
OUTPUT Q (  
LMAX = 999.000;  
DELAY QN (  
   DRIVE = ((0, 0.5, 0.5), (1, 0.7, 0.7), (2, 0.9, 0.9), (3, 1.1, 1.1));  
   INTERCEPT = ((0, 0.1, 0.1), (1, 0.2, 0.2), (2, 0.3, 0.3), (3, 0.4, 0.4));  
   PROP = ( 8.300, 8.300);  
   SLOPE = ( 0.0, 0.0);  
);  
OUTPUT QN (  
LMAX = 999.000;  
};  
SET default_input_drive = 0.000000 0.000000;  
SET default_output_load = 1.000000;  
SET max_load = 10;  
SET opt_for_area_weights = 1.00.01 0.2 0.0;  
SET opt_for_speed_weights = 0.011.0 0.2 0.0;  
SET nominal_process = typical;  
SET fast_process = 0.60;  
SET typical_process = 1.00;  
SET worst_process = 1.03;  
SET nominal_temp = 25.0;  
SET min_temp = -55.0;  
SET max_temp = 125.0;  
SET temp_slope = 0.0025;  
SET nominal_voltage = 5.0;  
SET min_voltage = 4.5;  
SET max_voltage = 5.5;  
SET voltage_slope = -0.14;  
SET lib_version = "1.14";  
SET rep_logic = true;  
SET delay_library_model = piecewise_linear;  
SET delay_connect_model = worst;  
#defines a piece table with wire_length as its unit
PIECE_DEFINE WIRE_LENGTH (10.0, 20.0, 30.0, 40.0);
#defines a wire_load model
WIRE_TABLE "minimum" (
    wire_res = 1.0;
    wire_cap = 1.0;
    wire_area = 3.0;
    wire_slope = 1.0;
    wire_length = ((0, 0.0), (1, 1.0), (2, 2.0), (8, 8.0))
);
WIRE_TABLE "nominal" (
    wire_res = 1.0;
    wire_cap = 1.0;
    wire_area = 3.0;
    wire_slope = 1.0;
    wire_length = ((0, 0.0), (1, 2.0), (2, 4.0), (3, 6.0))
);

Nonlinear Example

LIBRARY nonlinear_lib (

    SET delay_library_model = non_linear;

    LUT_TEMPLATE "prop_5x5" (
        variable_x input_net_transition;
        variable_y total_output_net_capacitance;
        input_net_transition = ( 0.0000, 0.5000, 1.0000, 1.5000, 2.0000 );
        total_output_net_capacitance = ( 0.0000, 0.5000, 1.0000, 1.5000,2.0000 );
    );

    LUT_TEMPLATE "trans_5x1" (
        variable_x total_output_net_capacitance;
        total_output_net_capacitance = ( 0.0000, 0.5000, 1.0000, 1.5000,2.0000 );
    );

    LUT_TEMPLATE "constrain_5x5" (
        variable_x constrained_pin_transition;
        variable_y related_pin_transition;
        constrained_pin_transition = ( 0.0000, 0.5000, 1.0000, 1.5000, 2.0000 );
        related_pin_transition = ( 0.0000, 0.5000, 1.0000, 1.5000, 2.0000 );
    );

    GATE inv1 (
        inputs A;

    )

)
outputs \ Q;

function ( 
    \ Q = !A;
)
area = 1.0000;
input A ( 
    fanout_load = 0.8364;
    cap_load = 0.0485;
);
output Q ( 
    max_fanout_load = 18.1379;
    max_cap_load = 1.0520;
);
input A ( 
    delay Q ( 
        cell_delay_rise = "prop_5x5" ( 
            variable_x = (0.0072, 0.2100, 0.4128, 0.6157, 0.8185);
            variable_y = (0.0490, 0.2998, 0.5505, 0.8013, 1.0520);
            parameter = ( 
                (0.0852, 0.1204, 0.1541, 0.1861, 0.2166),
                (0.2530, 0.3244, 0.3942, 0.4625, 0.5291),
                (0.4203, 0.5156, 0.6093, 0.7015, 0.7920),
                (0.5872, 0.6941, 0.7994, 0.9031, 1.0053),
                (0.7537, 0.8599, 0.9645, 1.0675, 1.1689)
            )
        )
    )
    cell_delay_fall = "prop_5x5" ( 
        variable_x = (0.0089, 0.4116, 0.8142, 1.2169, 1.6195);
        variable_y = (0.0490, 0.2998, 0.5505, 0.8013, 1.0520);
        parameter = ( 
            (0.1236, 0.3051, 0.4650, 0.6033, 0.7199),
            (0.3384, 0.6008, 0.8415, 1.0606, 1.2580),
            (0.5526, 0.8706, 1.1670, 1.4417, 1.6947),
            (0.7663, 1.1147, 1.4414, 1.7465, 2.0300),
            (0.9794, 1.3329, 1.6649, 1.9751, 2.2637)
        )
    )
    transition_rise = "trans_5x1" ( 
        variable_x = (0.0490, 0.2998, 0.5505, 0.8013, 1.0520);
        parameter = ( 
            (0.0636, 0.3889, 0.7142, 1.0395, 1.3649)
        )
    )
    transition_fall = "trans_5x1" ( 
        variable_x = (0.0490, 0.2998, 0.5505, 0.8013, 1.0520);
    )
);
C

\begin{verbatim}

  parameter = {
      (0.0466, 0.2851, 0.5235, 0.7620, 1.0005)
  }

  GATE nand1 {
    inputs A, B;
    outputs Q;

    function {
      Q = !B + !A;
    }
    area = 1.0000;
    input A {
      fanout_load = 0.8369;
      cap_load = 0.0485;
    }
    input B {
      fanout_load = 0.8374;
      cap_load = 0.0486;
    }
    output Q {
      max_fanout_load = 10.5521;
      max_cap_load = 0.6120;
    }
    input A {
      delay Q {
        cell_delay_rise = "prop_5x5" {
          variable_x = (0.0072, 0.2100, 0.4129, 0.6157, 0.8185);
          variable_y = (0.0490, 0.1898, 0.3305, 0.4713, 0.6120);
          parameter = {
            (0.1071, 0.1504, 0.1848, 0.2102, 0.2266),
            (0.2112, 0.2841, 0.3481, 0.4032, 0.4492),
            (0.3149, 0.4087, 0.4935, 0.5694, 0.6362),
            (0.4184, 0.5241, 0.6209, 0.7087, 0.7876),
            (0.5215, 0.6304, 0.7304, 0.8213, 0.9033)
          }
        }
      }
      cell_delay_fall = "prop_5x5" {
          variable_x = (0.0089, 0.4116, 0.8142, 1.2169, 1.6195);
          variable_y = (0.0490, 0.1898, 0.3305, 0.4713, 0.6120);
          parameter = {
            (0.1575, 0.3620, 0.5476, 0.7141, 0.8617),
          }
      }
    }

\end{verbatim}

92

lGen User’s Guide
transition_rise = "trans_5x1" (
    variable_x = (0.0490, 0.1898, 0.3305, 0.4713, 0.6120);
    parameter = (
        (0.0682, 0.2641, 0.4600, 0.6560, 0.8519)
    );
);  
transition_fall = "trans_5x1" (
    variable_x = (0.0490, 0.1898, 0.3305, 0.4713, 0.6120);
    parameter = (
        (0.0652, 0.2525, 0.4397, 0.6270, 0.8142)
    );
);  
input B  
    delay Q  
        cell_delay_rise = "prop_5x5" (  
            variable_x = (0.0072, 0.2100, 0.4129, 0.6157, 0.8185);  
            variable_y = (0.0490, 0.1898, 0.3305, 0.4713, 0.6120);  
            parameter = (  
                (0.1209, 0.1817, 0.2377, 0.2890, 0.3355),  
                (0.2248, 0.3079, 0.3862, 0.4598, 0.5286),  
                (0.3285, 0.4270, 0.5208, 0.6098, 0.6941),  
                (0.4318, 0.5390, 0.6414, 0.7390, 0.8319),  
                (0.5350, 0.6439, 0.7480, 0.8474, 0.9421)
            );
        );  
        cell_delay_fall = "prop_5x5" (  
            variable_x = (0.0089, 0.4116, 0.8142, 1.2169, 1.6195);  
            variable_y = (0.0490, 0.1898, 0.3305, 0.4713, 0.6120);  
            parameter = (  
                (0.1642, 0.2931, 0.4126, 0.5225, 0.6229),  
                (0.3293, 0.4958, 0.6527, 0.8002, 0.9381),  
                (0.4941, 0.6867, 0.8698, 1.0433, 1.2074),  
                (0.6587, 0.8659, 1.0637, 1.2519, 1.4306),  
                (0.8230, 1.0335, 1.2344, 1.4259, 1.6078)
            );
        );  
        transition_rise = "trans_5x1" (  
            variable_x = (0.0490, 0.1898, 0.3305, 0.4713, 0.6120);  
        );
    );
\[
\text{parameter} = \{
\begin{array}{c}
0.0682, 0.2641, 0.4600, 0.6560, 0.8519
\end{array}
\};
\]
\[
\text{transition\_fall} = "\text{trans\_5x1}\" (\begin{array}{c}
\text{variable\_x} = (0.0490, 0.1898, 0.3305, 0.4713, 0.6120); \\
\text{parameter} = (\begin{array}{c}
0.0652, 0.2525, 0.4397, 0.6270, 0.8142
\end{array})
\};
\)
\]
\]
\[
\text{GATE} \text{ buf} \begin{array}{c}
\text{inputs} \ A; \\
\text{outputs} \ Q;
\end{array}
\]
\[
\text{function} (\begin{array}{c}
Q = A;
\end{array});
\]
\[
\text{area} = 1.7000;
\]
\[
\text{input} \ A \begin{array}{c}
\text{fanout\_load} = 1.6475; \\
\text{cap\_load} = 0.0956;
\end{array};
\]
\[
\text{output} \ Q \begin{array}{c}
\text{max\_fanout\_load} = 33.3103; \\
\text{max\_cap\_load} = 1.9320;
\end{array};
\]
\[
\text{input} \ A \begin{array}{c}
\text{delay} \ Q \begin{array}{c}
\text{cell\_delay\_rise} = "\text{prop\_5x5}\" (\begin{array}{c}
\text{variable\_x} = (0.0176, 0.4202, 0.8229, 1.2255, 1.6282); \\
\text{variable\_y} = (0.0490, 0.5197, 0.9905, 1.4613, 1.9320); \\
\text{parameter} = (\begin{array}{c}
0.1865, 0.3619, 0.5237, 0.6718, 0.8062), \\
(0.3416, 0.5215, 0.6878, 0.8404, 0.9793), \\
(0.4967, 0.6805, 0.8506, 1.0071, 1.1499), \\
(0.6518, 0.8388, 1.0122, 1.1719, 1.3180), \\
(0.8068, 0.9965, 1.1725, 1.3348, 1.4835)
\end{array})
\};
\)
\)
\]
\[
\text{cell\_delay\_fall} = "\text{prop\_5x5}\" (\begin{array}{c}
\text{variable\_x} = (0.0142, 0.2170, 0.4198, 0.6227, 0.8255); \\
\text{variable\_y} = (0.0490, 0.5197, 0.9905, 1.4613, 1.9320);
\end{array})
\]
\]
parameter = (
    (0.1904, 0.2507, 0.3027, 0.3463, 0.3817),
    (0.3959, 0.4527, 0.5012, 0.5414, 0.5733),
    (0.6015, 0.6555, 0.7013, 0.7387, 0.7679),
    (0.8071, 0.8591, 0.9029, 0.9383, 0.9654),
    (1.0128, 1.0635, 1.1060, 1.1402, 1.1660)
);  
transition_rise = "trans_5x1" (  
    variable_x = (0.0490, 0.5197, 0.9905, 1.4613, 1.9320);  
    parameter = (  
        (0.0311, 0.3304, 0.6296, 0.9289, 1.2281)
    );  
);  
transition_fall = "trans_5x1" (  
    variable_x = (0.0490, 0.5197, 0.9905, 1.4613, 1.9320);  
    parameter = (  
        (0.0223, 0.2368, 0.4512, 0.6656, 0.8801)
    );  
);  
GATE dff1 (  
    inputs C, D;  
    outputs Q, QN;  
    function (  
        DFFE (-, -, D, C, -, Q, QN);  
    );  
    area = 4.5000;  
    input C (  
        fanout_load = 2.1928;  
        cap_load = 0.1272;  
        SET clock_pin;
    );  
    input D (  
        fanout_load = 0.8636;  
        cap_load = 0.0501;
    );  
    output Q (  
        max_fanout_load = 18.1379;  
        max_cap_load = 1.0520;
    );  
    output QN (
max_fanout_load = 10.5521;
max_cap_load = 0.6120;
);
output QN ( delay Q ( 
  phase = inv;
  cell_delay_rise = "prop_5x5" ( 
    variable_x = (0.0000, 0.2729, 0.5459, 0.8188, 1.0917);
    variable_y = (0.0000, 0.2630, 0.5260, 0.7890, 1.0520);
    parameter = ( 
      (0.1058, 0.1189, 0.1320, 0.1451, 0.1582),
      (0.3511, 0.3629, 0.3747, 0.3865, 0.3984),
      (0.6446, 0.6551, 0.6656, 0.6761, 0.6867),
      (0.9861, 0.9954, 1.0046, 1.0139, 1.0231),
      (1.3758, 1.3838, 1.3918, 1.3997, 1.4077) );
    });
  cell_delay_fall = "prop_5x5" ( 
    variable_x = (0.0000, 0.3576, 0.7153, 1.0729, 1.4305);
    variable_y = (0.0000, 0.2630, 0.5260, 0.7890, 1.0520);
    parameter = ( 
      (0.2086, 0.2087, 0.2087, 0.2087, 0.2088),
      (0.6114, 0.6111, 0.6107, 0.6104, 0.6100),
      (1.0909, 1.0902, 1.0894, 1.0887, 1.0880),
      (1.6471, 1.6460, 1.6448, 1.6437, 1.6426),
      (2.2799, 2.2784, 2.2769, 2.2754, 2.2740) );
  });
transition_rise = "trans_5x1" ( 
  variable_x = (0.0000, 0.2630, 0.5260, 0.7890, 1.0520);
  parameter = ( 
    (0.0000, 0.3324, 0.6648, 0.9972, 1.3296) );
);
transition_fall = "trans_5x1" ( 
  variable_x = (0.0000, 0.2630, 0.5260, 0.7890, 1.0520);
  parameter = ( 
    (0.0000, 0.2525, 0.5049, 0.7574, 1.0099) );
);
input C ( delay QN ( 
  cell_delay_rise = "prop_5x5" ( 
    variable_x = (0.0234, 0.4260, 0.8287, 1.2313, 1.6340);
variable_y = (0.0000, 0.1530, 0.3060, 0.4590, 0.6120);
parameter = {
(0.3162, 0.4920, 0.6678, 0.8437, 1.0195),
(0.4881, 0.6667, 0.8454, 1.0241, 1.2028),
(0.6599, 0.8415, 1.0230, 1.2046, 1.3862),
(0.8318, 1.0162, 1.2006, 1.3851, 1.5695),
(1.0037, 1.1909, 1.3782, 1.5655, 1.7528)
};
}

cell_delay_fall = "prop_5x5" {
variable_x = (0.0234, 0.4260, 0.8287, 1.2313, 1.6340);
variable_y = (0.0000, 0.1530, 0.3060, 0.4590, 0.6120);
parameter = {
(0.2145, 0.3411, 0.4676, 0.5941, 0.7207),
(0.4329, 0.6048, 0.7766, 0.9485, 1.1203),
(0.6513, 0.8685, 1.0856, 1.3028, 1.5200),
(0.8697, 1.1322, 1.3947, 1.6572, 1.9197),
(1.0880, 1.3959, 1.7037, 2.0115, 2.3193)
};
}

transition_rise = "trans_5x1" {
variable_x = (0.0000, 0.1530, 0.3060, 0.4590, 0.6120);
parameter = {
(0.0000, 0.3576, 0.7153, 1.0729, 1.4305)
};
}

transition_fall = "trans_5x1" {
variable_x = (0.0000, 0.1530, 0.3060, 0.4590, 0.6120);
parameter = {
(0.0000, 0.2729, 0.5459, 0.8188, 1.0917)
};
}

input D {
delay C {
setup_rise = "constrain_5x5" {
variable_x = (0.0092, 0.4119, 0.8145, 1.2172, 1.6198);
variable_y = (0.0234, 0.4260, 0.8287, 1.2313, 1.6340);
parameter = {
(0.4726, 0.5709, 0.6692, 0.7676, 0.8659),
(0.5047, 0.5709, 0.6692, 0.7676, 0.8659),
(0.5895, 0.5895, 0.6692, 0.7676, 0.8659),
(0.6742, 0.6742, 0.7676, 0.8659),
(0.7589, 0.7589, 0.7589, 0.7676, 0.8659)
};
}
setup_fall = "constrain_5x5" {
  variable_x = (0.0075, 0.2103, 0.4131, 0.6159, 0.8187);
  variable_y = (0.0234, 0.4260, 0.8287, 1.2313, 1.6340);
  parameter = {
    (0.4726, 0.5047, 0.5895, 0.6742, 0.7589),
    (0.5709, 0.5709, 0.5895, 0.6742, 0.7589),
    (0.6692, 0.6692, 0.6692, 0.6742, 0.7589),
    (0.7676, 0.7676, 0.7676, 0.7676, 0.7676),
    (0.8659, 0.8659, 0.8659, 0.8659, 0.8659)
  };
};
Use of the lGen program is permitted through a licensing program, with the authorization code (and license key for PC users) supplied by Exemplar Logic Customer Support.

Help

For installation and setup questions, please call our Customer Support Hot Line at 1-510-337-fpga (3742).

Sun Installation

The lGen software was installed when you installed the Galileo software. If you ordered lGen with your Galileo system, the lGen authorization code was included with the other codes, and you may start working with lGen now. If lGen was ordered separately from your Galileo system, the authorization code should have been included in the package you just opened. If not, you must provide the hostid for your machine, and call the Exemplar Logic Customer Support Hot-Line at 1-510-337-fpga (3742) for your authorization code.

Once you have obtained the authorization code edit the license file license.dat with the information given you by Exemplar Logic Customer Support. Make sure this file resides in the $EXEMPLAR/license directory. The FLEX lm license manager must be made aware of the new codes. See Section 1.6 of the Galileo User Manual for more information on using the license manager.
**HP Installation**

The *lGen* software was installed when you installed the Galileo software. If you ordered *lGen* with your Galileo system, the *lGen* authorization code was included with the other codes, and you may start working with *lGen* now. If *lGen* was ordered separately from your Galileo system, the authorization code should have been included in the package you just opened. If not, you must provide the hostid for your machine, and call the Exemplar Logic Customer Support Hot-Line at 1-510-337-fpga (3742) for your authorization code.

To obtain the hostid, use the command

```
uname -i
```

to get the last 8 digits of the hexadecimal machine id (skip the first two digits).

*Activate the license system by editing the *SEXEMPLAR*/data/exemplar.lic* file to include the authorization code sent by the Customer Support Hot-Line.*

**PC Installation**

On the PC, *lGen* and *lBuild* use the same executable, with the authorization code determining the designer’s capability to generate an output synthesis library with *lGen*. If you ordered *lGen* with your Galileo system, the *lGen* authorization code was included with the other codes, and you may start working with *lGen* now. If *lGen* was ordered separately from your Galileo system, the authorization code should have been included in the package you just opened. If not, determine the hardware key id (written on the key attached to the parallel port, or type `pchostid` at the `c:\exemplar>` prompt) and call the Exemplar Logic Customer Support Hot-Line at 1-510-337-fpga (3742) for your authorization code.

*Activate the license system by editing the *EXEMPLAR*/data/exemplar.lic* file to include the authorization code sent by the Customer Support Hot-Line.* (If a printer is attached to the port while running Galileo, please make sure that this printer is turned on.)