cadence

SDF Annotator Guide

Product Version 3.2
January 2001

© 1990-2000 Cadence Design Systems, Inc. All rights reserved.
Printed in the United States of America.

Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134, USA

Trademarks: Trademarks and service marks of Cadence Design Systems, Inc. (Cadence) contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 1-800-862-4522.

All other trademarks are the property of their respective holders.

Restricted Print Permission: This publication is protected by copyright and any unauthorized use of this publication may violate copyright, trademark, and other laws. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. This statement grants you permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used solely for personal, informational, and noncommercial purposes;
2. The publication may not be modified in any way;
3. Any copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement; and
4. Cadence reserves the right to revoke this authorization at any time, and any such use shall be discontinued immediately upon written notice from Cadence.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. The information contained herein is the proprietary and confidential information of Cadence or its licensors, and is supplied subject to, and may be used only by Cadence's customer in accordance with, a written agreement between Cadence and its customer. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor.

Contents

Preface

This preface describes the following:

About This Guide

This guide describes the **Standard Delay Format (SDF) Annotator**™, which facilitates the exchange of timing data between an SDF file and a Verilog family tool. The SDF Annotator uses the SDF file as input for the annotation process. This guide assumes that you are familiar with one or more of the Verilog family tools.

Finding Information in This Guide

This guide describes the SDF Annotator and is organized as follows:

Chapter 1, "Using the SDF Annotator" describes the SDF Annotator and how to call it from the Verilog Hardware Design Language (HDL).

Chapter 2, "Using the Configuration File" describes the optional configuration file, which lets you filter timing data in the SDF file before the data is annotated to the Verilog family tool. If you donot use a configuration file, you can skip this chapter.

Chapter 3, "Using the SDF File" describes the SDF file, which stores timing data generated by the Verilog family tool. It also describes the conventions and keywords for the SDF file, including the keywords that are used by various Verilog family tools.

Chapter 4, "Annotating with Verilog-XL and Verifault-XL" describes the SDF-specific plus options you can specify on the Verilog-XL command line. It also describes the restrictions you have between the SDF Annotator and Verilog®-XL.

Appendix A, “SDF Annotator Error and Warning Messages” lists the Error and Warning messages that the SDF Annotator issues.

Appendix B, “Valid and Invalid Interconnect Combinations” lists the valid and invalid interconnect combinations.

Other Sources of Information

Related Manuals

Cadence provides the following sources of information.

The SDF Annotator is used with other Cadence products during the design process. For more information about the SDF Annotator and other related products, see the following manuals.

Programming Language Interface 1.0 User Guide and Reference
VPI User Guide and Reference
PLI Wizard User Guide
Contains information about how you can use the interface to pass information between the SDF Annotator and the Verilog family tool.

Verilog-XL Reference
Verilog-XL User Guide
Contains information about how to use the Verilog®-XL simulator.

Verifault-XL Reference
Verifault-XL User Guide
Contains information about how to use the Verifault-XL® simulator.

Customer Education Services

Cadence also offers many customer education services. Contact your sales representative for more information.

Syntax Conventions

The following table shows the conventions for the syntax code in this guide:

KEYWORDS	Uppercase text indicates a keyword, which must be typed exactly as shown.

item	An italicized item in syntax examples indicates a variable name where you must supply information to complete the syntax.
item+	The plus (+) on a item indicates that this item can be replicated one or more times.
{item}	Items in braces indicate that an item is optional.
[item|item]	Items in brackets with a vertical bar (meaning “or”) indicate that you must choose one item only.
[item]	Items in brackets, when shown without a vertical bar, are required as part of the syntax. These generally occur in text, showing bit specifications, but are noted here (and in the text) so they are not confused with syntax conventions.
(“item”)	Parentheses and quotation marks, when shown, are required as part of the syntax of an item.

1

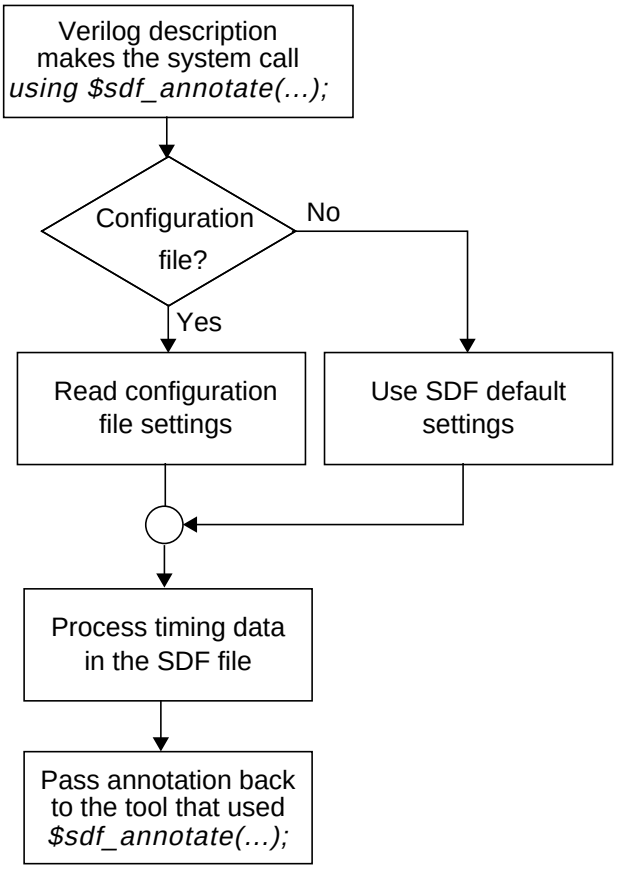
Using the SDF Annotator

This chapter describes the following:

Understanding How the SDF Annotator Works

The SDF Annotator annotates timing data for Verilog family tools. Timing data in an SDF file can come from front-end tools such as SiliconQuest™. Preview™, or Veritime™, or from Cadence's back-end tools, such as Cell3 Ensemble™and Gate Ensemble™. Timing data is annotated as illustrated in Figure 1-1 .

Figure 1-1 Standard Delay Format Annotation Process



The process shown in Figure 1-1 is described in more detail as follows:

1. A Verilog family tool responds to the `$sdf_annotate` system task, which calls the SDF Annotator. See the section "Calling the SDF Annotator from Verilog HDL" on page 12 for more information.
2. The SDF Annotator then reads the configuration file, if one exists. The configuration file filters timing data before it is annotated to a Verilog Family tool. See Chapter 2, "Using the Configuration File" for more information.
3. The SDF Annotator reads the timing data from the SDF file, which is an ASCII text file that stores the timing data generated by the Verilog family tool. See Chapter 3, "Using the SDF File" for more information about the SDF file.
4. The SDF Annotator processes the timing data according to the configuration file commands or the SDF Annotator's settings.
5. The processed data is annotated to the Verilog family tool.

The SDF Annotator operates on many aspects of a design. However, when you need to operate on only a few aspects of a design, you can achieve significant annotation process performance improvements by implementing some of the recommendations described in Chapter 4, “Annotating with Verilog-XL and Verifault-XL.” You can perform separate annotations to distinct hierarchical portions of a single design description. For example, you can annotate from multiple SDF files, each of which corresponds to a separate IC within a description of a board-level design.

Note: When doing multiple annotations, specify a different log file name for each annotation so that you can verify the results.

Calling the SDF Annotator from Verilog HDL

To call the SDF Annotator from a Verilog family tool, enter the `$sdf_annotate` system task at the interactive command line or in the Verilog family tool’s description. The `$sdf_annotate` system task specifications take precedence over specifications in the SDF file.

Note: You must use the `+annotate_any_time` option on the Verilog-XL command line to annotate after time 0 with Verilog-XL.

$sdf_annotate System Task Syntax

```
$sdf_annotate ( “sdf_file”
     {, module_instance}
     {, “config_file”}
     {, “log_file”}
     {, “mtm_spec”}
     {, “scale_factors”}
     {, “scale_type”} );
```

Note: You must specify the arguments to the `$sdf_annotate` system task in the order shown in the syntax. You can skip an argument specification, but the number of comma separators must maintain the argument sequence. For example, to specify only the first and last arguments, use the following syntax:

```
$sdf_annotate ( “sdf_file”,,,,,, “scale_type”);
```

$sdf_annotate Arguments

`“sdf_file”`	The full or relative path of the SDF file. This argument is required and must be in quotation marks. You can specify the file name with the `+sdf_file` plus option on the command line.

`module_instance` — Optional: Specifies the scope in which the annotation takes place. The names in the SDF file are relative paths to the `module_instance` with respect to the entire Verilog HDL description. The SDF Annotator uses the hierarchy level of the specified instance for running the annotation. Array indexes (`module_instance[index]`) are permitted in the scope. If you do not specify `module_instance`, the SDF Annotator uses the module containing the call to the `$sdf_annotate` system task as the `module_instance` for annotation.

`"config_file"` — Optional: The name of the configuration file, specified in quotation marks, that the SDF Annotator reads before annotating begins. If you do not specify `config_file`, the SDF Annotator uses the default settings. See Chapter 2, "Using the Configuration File" for more information.

`"log_file"` — Optional: The name of the log file, specified in quotation marks, that the SDF Annotator generates during annotation. Also, you must specify the `+sdf_verbose` plus option on the command line to generate a log file. If you do not specify a log file name, but do specify the `+sdf_verbose` plus option, the SDF Annotator creates a default log file called `sdf.log`.

`"mtm_spec"` — Optional: One of the following keywords, specified in quotation marks, indicating the delay values that are annotated to the Verilog family tool.

Keyword	Description
`MAXIMUM`	Annotates the maximum delay value.
`MINIMUM`	Annotates the minimum delay value.
`TOOL_CONTROL` (default)	Annotates the delay value that is determined by the Verilog-XL and Verifault-XL command line options (`+mindelays`, `+typdelays`, or `+maxdelays`); minimum, typical, and maximum values are always annotated to Veritime. If none of the `TOOL_CONTROL` command line options is specified, the default keyword is then `TYPICAL`.
`TYPICAL`	Annotates the typical delay value.

`"scale_factors"` — Optional: The minimum, typical, and maximum timing data values, specified in quotation marks, expressed as a set of three

positive real number multipliers (*min_mult:typ_mult:max_mult)*. For example, 1.6:1.4:1.2. If you do not specify values, the default values are 1.0:1.0:1.0 for minimum, typical, and maximum values. The SDF Annotator uses these values to scale the minimum, typical, and maximum timing data from the SDF file before they are annotated to the Verilog family tool. The *scale_factors* argument overrides the SCALE_FACTORS keyword in the configuration file. See the "SCALE_FACTORS Keyword" on page 21for an example of scaling delay values.

"scale_type" — Optional: One of the following keywords, specified in quotation marks, to scale the timing specifications in SDF, which are annotated to the Verilog family tool. The *scale_type* argument override*s* the SCALE_TYPE keyword in the configuration file.

Keyword	Description
FROM_MAXIMUM	Scales from the maximum timing specification.
FROM_MINIMUM	Scales from the minimum timing specification.
FROM_MTM (default)	Scales from the minimum, typical, and maximum timing specifications. This is the default.
FROM_TYPICAL	Scales from the typical timing specification.

See the "SCALE_FACTORS Keyword" on page 21 for an example of delay scaling.

Examples: Calling the SDF Annotator

The following examples show different ways to call the SDF Annotator to pass timing information to your Verilog family tool.

Annotation with Scaling

This example shows annotation with scaling to the top-level design. The FROM_MTM keyword overrides any scale type specifications in the configuration file.

```
module top;
    •••
    circuit m1(i1,i2,i3,o1,o2,o3);
```

```
    initial
$sdf_annotate("my.sdf",m1,"config"
                  ,,,1.6:1.4:1.2,"FROM_MTM");
    // stimulus and response-checking
        •••
endmodule
```

Separate Annotations

This example shows separate annotations to distinct portions of a design hierarchy. There is no configuration file specification, so the SDF Annotator uses the defaults.

```
module top;
    •••
    cpu m1(i1,i2,i3,o1,o2,o3);
    fpu m2        (i4,o1,o3,i2,o4,o5,o6);
    dma m3(o1,o4,i5,i6,i2);
    // perform annotation
    initial
    begin
        $sdf_annotate("cpu.sdf",m1,,"cpu.log");
        $sdf_annotate("fpu.sdf",m2,,"fpu.log");
        $sdf_annotate("dma.sdf",m3,,"dma.log");
    end
    // stimulus and response-checking
        •••
endmodule
```

Annotation with Arrays of Instances

This example shows arrays of instance in a design hierarchy. There is no configuration file specification, so the SDF Annotator uses the defaults.

```
module top;
    •••
    cpu ar[1](i1,i2,i3,o1,o2,o3);
    fpu ar[2](i4,o1,o3,i2,o4,o5,o6);
    dma ar[3](o1,o4,i5,i6,i2);
    // perform annotation
    initial
    begin
        $sdf_annotate("cpu.sdf",ar[1],,"cpu.log");
        $sdf_annotate("fpu.sdf",ar[2],,"fpu.log");
        $sdf_annotate("dma.sdf",ar[3],,"dma.log");
    end
    // stimulus and response-checking
        •••
endmodule
```

2

Using the Configuration File

This chapter describes the following:

Understanding the Configuration File

The configuration file allows you to filter timing data in the SDF file before the data is annotated to a Verilog family tool using the SDF Annotator configuration file. If you do not use a configuration file, the SDF Annotator uses default settings for annotation, and you can skip this chapter. You can do the following using the configuration file.

- Map or ignore timing constructs from the SDF file to the Verilog HDL description
- Select multiple timing specifications
- Select minimum, typical, or maximum delays values
- Specify scaling operations
- Determine turn-off delays
- Specify delay data for a specific type of module

Sample Configuration File

Many of the configuration file keywords are shown in the following example. If the SDF Annotator finds conflicting keywords, it uses the last specified keyword.

Note: Keywords must be in uppercase letters. Blank lines are not allowed.

```
PATHPULSE = IGNORE;            // Ignores all PATHPULSE constructs in SDF file.
INTERCONNECT_MIPD = MAXIMUM;   // Specifies maximum interconnect delay.
MTM = MAXIMUM;                 // Specifies maximum delays from SDF.
SCALE_FACTORS = 0.5:1:2.0;     // Scales the delays with these factors.
SCALE_TYPE = FROM_TYPICAL;     // Scales from the typical delays.
```

```
TURNOFF_DELAY = FROM_FILE;     // Specifies the turn-off delays in SDF.
MODULE AND                     // Applies to instances of type AND.
{
MAP_INNER = and1;              // Maps delays to inner module and1.
(in1 => out1) = OVERRIDE       // Uses delays between in1 and out1
 {                             // specified to override the delay paths
 (CP => Q);                    // between CP and Q in Verilog.
 }
}
```

Configuration File Keyword Syntax

This section lists the keywords you can specify in the configuration file.

Timing Keywords

The following keywords have only one option (`IGNORE`), which specify whether the SDF Annotator ignores the constructs in the SDF file. See Chapter 3, “Using the SDF File” for information about using these keywords in the SDF file.

- `DEVICE = IGNORE;`
- `HOLD = IGNORE;`
- `INTERCONNECT = IGNORE;`
- `IOPATH = IGNORE;`
- `NETDELAY = IGNORE;`
- `NOCHANGE = IGNORE;`
- `PATHPULSE = IGNORE;`
- `PATHPULSEPERCENT = IGNORE;`
- `PERIOD = IGNORE;`
- `PORT = IGNORE;`
- `RECOVERY = IGNORE;`
- `SETUP = IGNORE;`
- `SETUPHOLD = IGNORE;`
- `SKEW = IGNORE;`
- `WIDTH = IGNORE;`

Default Mapping for Verilog-XL

In Verilog-XL, if you do not specify a mapping for a timing keyword, the SDF Annotator uses the default mapping for that keyword as shown in Table 2-1 on page 18.

Table 2-1 Default mapping for Verilog-XL

SDF Timing Keywords	Path delay library	Distributed delay library
DEVICE	PATH	LUMPED OUTPUT
HOLD	HOLD	
INTERCONNECT	MIPD [a], SITD [b], MITD [c]	MIPD
IOPATH	PATH	LUMPED OUTPUT
NETDELAY	MIPD, SITD, MITD	MIPD
PERIOD	PERIOD	
PORT	MIPD, SITD, MITD	MIPD
RECOVERY	RECOVERY	
SETUP	SETUP	
SETUPHOLD	SETUP/HOLD	
SKEW	SKEW	
WIDTH	WIDTH	

a. Module Input Port Delay

b. Single-Source Interconnect Transport Delay

c. Multisource Interconnect Transport Delay

Default Mapping for Verifault-XL

In Verifault-XL, if you do not specify a mapping for a timing keyword, the SDF Annotator uses the default mapping for that keyword as shown in Table 2-2 on page 19.

Table 2-2 Default Mapping for Verifault-XL

SDF Timing Keywords	Path Delay Library	Distributed Delay Library
DEVICE	PATH	LUMPED OUTPUT
HOLD	HOLD	
INTERCONNECT	MIPD	MIPD
IOPATH	PATH	LUMPED OUTPUT
NETDELAY	MIPD	MIPD
PERIOD	PERIOD	
PORT	MIPD	MIPD
RECOVERY	RECOVERY	
SETUP	SETUP	
SETUPHOLD	SETUP/HOLD	
SKEW	SKEW	
WIDTH	WIDTH	

Default Mapping for Veritime

In Veritime, if you do not specify a mapping for a timing keyword, the SDF Annotator uses the default mapping for that keyword as shown in Table 2-3 on page 19.

Table 2-3 Default Mapping for Veritime

SDF Timing Keywords	Path delay library	Distributed delay library
DEVICE	PATH	LUMPED OUTPUT
HOLD	HOLD	
INTERCONNECT	INTERMOD PATH	INTERMOD PATH
IOPATH	PATH	LUMPED OUTPUT
NETDELAY	INTERMOD PATH	INTERMOD PATH
PERIOD	PERIOD	
PORT	MIPD	MIPD

SDF Timing Keywords	Path delay library	Distributed delay library
RECOVERY	RECOVERY	
SETUP	SETUP	
SETUPHOLD	SETUP/HOLD	
SKEW	SKEW	
WIDTH	WIDTH	

INTERCONNECT_MIPD Keyword

The INTERCONNECT_MIPD keyword selects how the interconnect delays in the SDF file are mapped to MIPDs in the Verilog family tool. You can select the options shown in the following syntax.

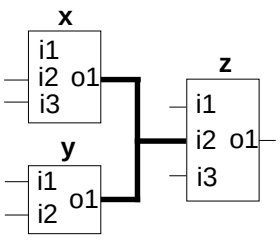
```
INTERCONNECT_MIPD = [AVERAGE | MAXIMUM | MINIMUM];
```

The different options for the INTERCONNECT_MIPD keyword are described below.

Keyword	Description
AVERAGE	Annotates the average of the overlapping timing specification.
MAXIMUM *(*Default*)*	Annotates the maximum of the overlapping timing specification.
MINIMUM	Annotates the minimum of the overlapping timing specification.

Example: INTERCONNECT_MIPD Keyword

If multiple interconnect delays fan into the same device input port, and if the interconnect delays have different values, the maximum delay is annotated to the MIPD at the common input port by default. Use MINIMUM or AVERAGE to override the default.



For example, the average value of the interconnect delay is annotated to the MIPD at *z.i2*, which is the common input port if you specify the *AVERAGE* assignment.

MTM Keyword

The `MTM` keyword specifies whether the SDF Annotator uses the minimum, typical, or maximum delays from the SDF file. The syntax for the `MTM` keyword is as follows:

```
MTM = [MAXIMUM | MINIMUM | TOOL_CONTROL | TYPICAL];
```

The different keywords for the `MTM` keyword are described in the table that follows.

Keyword	Description
`MAXIMUM`	Annotates the maximum delay value.
`MINIMUM`	Annotates the minimum delay value.
`TOOL_CONTROL`	Delay value is determined by the Verilog family tool plus options (*+mindelays*, *+typdelays*, or *+maxdelays*).
`TYPICAL` *(Default)*	Annotates the typical delay value.

Note: The *`mtm_spec`* option in *the* `$sdf_annotate` system task call overrides the `MTM` keyword. See “Calling the SDF Annotator from Verilog HDL” on page 12 for details.

SCALE_FACTORS Keyword

The `SCALE_FACTORS` keyword specifies scaling operations that the SDF Annotator performs on the timing information before it is annotated to the Verilog family tool. The syntax for the `SCALE_FACTORS` keyword is as follows:

```
SCALE_FACTORS = min_mult:typ_mult:max_mult;
```

Note: The *`scale_factors`* argument in the `$sdf_annotate` system task overrides the `SCALE_FACTORS` keyword. See “Calling the SDF Annotator from Verilog HDL” on page 12 for more information.

The *`min_mult:typ_mult:max_mult`* argument to the `SCALE_FACTORS` keyword is expressed as a set of three positive real number multipliers (*`min_mult:typ_mult:max_mult`*), for example, `1.6:1.4:1.2`. If you do not specify values, the default values are `1.0:1.0:1.0` for minimum, typical, and maximum values.

SCALE_TYPE Keyword

The SCALE_TYPE keyword specifies which of the scale types to use when performing scaling operations on the timing information before it is annotated to the Verilog family tool. The syntax for the SCALE_TYPE keyword is as follows:

```
SCALE_TYPE = [FROM_MAXIMUM | FROM_MINIMUM | FROM_MTM | FROM_TYPICAL];
```

Note: The *scale_type* argument in the $sdf_annotate system task overrides the SCALE_TYPE keyword. See "Calling the SDF Annotator from Verilog HDL" on page 12 for more information.

The following table describes the keywords for the SCALE_TYPE keyword.

Keyword	Description
FROM_MAXIMUM	Scales from the maximum timing specification in the SDF file.
FROM_MINIMUM	Scales from the minimum timing specification in the SDF file.
FROM_MTM *(Default)*	Scales directly from the minimum, typical, and maximum timing specifications in the SDF file.
FROM_TYPICAL	Scales from the typical timing specification in the SDF file.

Example: SCALE_FACTORS and SCALE_TYPE Keywords

To show how the SCALE_FACTORS and SCALE_TYPE keywords work in the SDF file that assigns rise, fall, and turn-off delays to a net in the cell instance *x,* consider the following CELL entry:

```
(CELL (CELLTYPE "adder4")
   (INSTANCE x)
   (DELAY (ABSOLUTE (NETDELAY a.o2 (6:7:8) (4:6:7) (5:8:9)))))
```

The configuration file defines the scale factors and scale type as follows:

```
SCALE_FACTORS = 0.5:1:1.5;
SCALE_TYPE = FROM_TYPICAL;
```

The typical delays in the SDF file are multiplied by the specified scaling factors to create new *min:typ:max* triplets, producing the following delays for the net:

```
(3.5:7:10.5) (3:6:9) (4:8:12)
```

TURNOFF_DELAY Keyword

The TURNOFF_DELAY keyword specifies how the SDF Annotator determines the turn-off delay that is annotated to the Verilog family tool. The syntax for the TURNOFF_DELAY keyword is as follows:

```
TURNOFF_DELAY=[AVERAGE | FROM_FILE | MAXIMUM | MINIMUM];
```

The following table describes the keywords for the TURNOFF_DELAY keyword.

Keyword	Description
AVERAGE	Average the values from the rise and fall delays.
FROM_FILE	The SDF Annotator uses the turn-off delays in the SDF file. If you do not specify FROM_FILE, or you specify FROM_FILE but the SDF file does not contain the turn-off delay, the turn-off delay is set to MINIMUM(rise, fall).
MAXIMUM	Choose the greatest values from the rise and fall delays.
MINIMUM *(Default)*	Choose the smallest values from the rise and fall delays.

Note: The TURNOFF_DELAY keyword is relevant only when both rise and fall delays are specified for a specific SDF construct. Also, when the SDF Annotator calculates the turn-off delays from the rise and fall delays, it uses that delay for all transitions to or from Z.

Example: TURNOFF_DELAY Keyword

The following cell entry in the SDF file assigns rise, fall, and turn-off delays to a net in the cell instance *x*. If the configuration file sets the TURNOFF_DELAY keyword to MAXIMUM, then the turn-off delay for annotation is (6:7:9) which is derived from the 6 rise and 4 fall minimum values, the 7 rise and 6 fall typical values, and the 8 rise and 9 fall maximum values; the (5:8:10) delay is ignored.

```
(CELL (CELLTYPE “adder4”)
(INSTANCE x)
(DELAY
      (ABSOLUTE (NETDELAY a.o2 (6:7:8) (4:6:9) (5:8:10)))
```

MODULE Keyword

The MODULE keyword maps path delays and timing checks from the SDF file to Verilog HDL description, performs scaling operations for a specific type of module, and selects *min:typ:max* delay data.

The syntax for the MODULE keyword is as follows:

```
MODULE module_name
      {
        [MTM = [MINIMUM | TYPICAL | MAXIMUM];]
        [SCALE_FACTORS = min_mult:typ_mult:max_mult;]
        [MAP_INNER = path;
          (original_timing) = [ADD | OVERRIDE | IGNORE] [{(new_timing);}]
        ]
      }
```

The following table describes the arguments to the MODULE keyword.

Argument	Description
module_name	Name of a specific type of module (not instance name) specified in the corresponding Verilog HDL description.
MTM	Specifies the minimum, typical, or maximum delays from the SDF file. See "MTM Keyword" on page 21 for more information.
SCALE_FACTORS	See "SCALE_FACTORS Keyword" on page 21 for more information.
[MAP_INNER] (Optional)	See "MAP_INNER Keyword" on page 24 for more information.

Note: The MTM and SCALE_FACTORS arguments to the MODULE keyword affect only the IOPATH, DEVICE, and TIMINGCHECK information annotated to the specified *module_name* module; they do not affect scale factors specified for other modules in the same design.

MAP_INNER Keyword

The MAP_INNER keyword is an optional argument to the MODULE keyword. It specifies a subsequent module in the hierarchy of the module specified with the MODULE keyword. You can specify the MAP_INNER keyword for each subsequent module in the module hierarchy.

The syntax for the MAP_INNER keyword is as follows:

```
MAP_INNER = path;
      (original_timing) = [ADD | OVERRIDE | IGNORE]
      { (new_timing); }
```

The following table describes the arguments to the MAP_INNER keyword.

Argument	Description
path	Verilog HDL hierarchical path of a submodule within *module_type* of the MODULE keyword. The paths specified in the SDF file are mapped to *module_type*. This path applies to all path delays and timing checks specified for this module in the SDF file including those mapped with ADD and OVERRIDE.
original_timing	The path delay or timing specification that is in the SDF file.
new_timing	The path delay in the Verilog description that corresponds to the *original_timing* delay.
ADD	Adds to the mapping specifications of the SDF file. The *original_timing* specification is mapped to *new_timing*, the Verilog HDL syntax of a path delay or timing check.
OVERRIDE	Replaces the mapping specifications of the SDF file. The *original_timing* specification is mapped to *new_timing*, the Verilog HDL syntax of a path delay or timing check.
IGNORE	Ignores the mapping specifications in the SDF file.

Note: In all cases, the path name is applied to all *new_timing* specifications before they are annotated to the Verilog family tool.

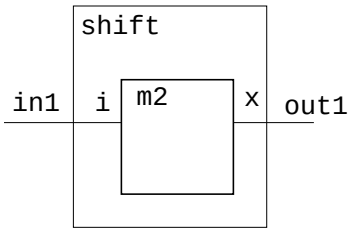
Examples of Using the MODULE and MAP_INNER Keywords

The following examples shows how the MODULE and MAP_INNER keywords work.

Example 1

This example applies module mapping to the *shift* module type, which contains a submodule called *m2* and specifies that the delay between *in1* and *out1* in the SDF file is

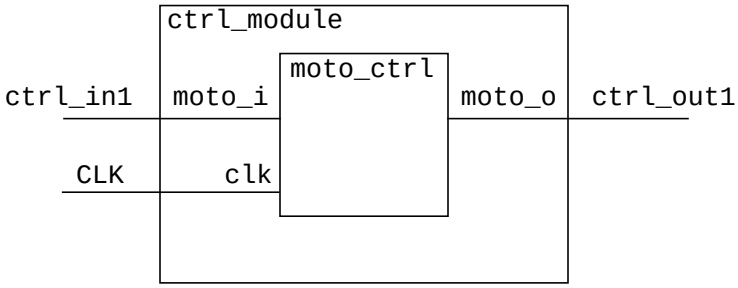
to be mapped to the delay between *i* and *x* of *m2* in Verilog, using minimum delays and a scaling factor of 2.0.




```
MODULE shift
{
MTM = MINIMUM;
SCALE_FACTORS = 2.0:2.0:2.0;
MAP_INNER = m2;
    (in1 => out1) = OVERRIDE
{
     (i => x)
 }
}
```

Example 2

In this example, two mappings are performed with the `MAP_INNER` keyword. Using the `OVERRIDE` keyword with the hierarchical design in the following figure, this example shows how to map and annotate the delay from the path `ctrl_in1=>ctrl_out1` to the path `moto_i=>moto_o`, and `clk=>moto_o`.



The Verilog design has a specify block with the following path delay specification:

```
(moto_i => moto_o) = (3, 4);
```

The SDF file for the design has the following delay specification for the path:

```
(IOPATH ctrl_in1 ctrl_out1 (5) (6))
```

A module mapping for this hierarchy is specified in the SDF configuration file as follows:

```
MODULE ctrl_module
{    MAP_INNER = moto_ctrl;
            (ctrl_in1 => ctrl_out1) = OVERRIDE
    {    (moto_i => moto_o);
        (clk => moto_o); }
}
```

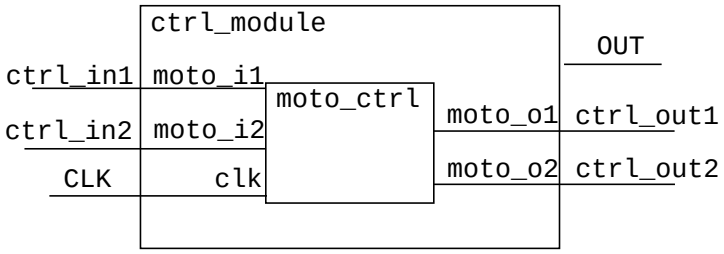
Example 3

Using the `IGNORE` keyword with the same hierarchy as shown in Example 2, the following mapping in the configuration file ignores the specification in the SDF file and continues to use the timing in the Verilog design.

```
MODULE ctrl_module
{ MAP_INNER = moto_ctrl;
    (ctrl_in1 => ctrl_out1) = IGNORE; }
```

Example 4

In this example three mappings are done using the `MAP_INNER` keyword. Using the `ADD` keyword with the hierarchy in the following figure, this example shows how to map a path from `(clk=>moto_o2)` and `(clk=>moto_o1)` in addition to the `(CLK=>OUT)` mapping.



The following mapping example allows the annotation of the delay specified for the `IOPATH (CLK =>OUT)` to `(CLK =>OUT)`, `(clk=>moto_o1)` and `(clk=>moto_o2)`.

```
MODULE ctrl_module
{ MAP_INNER = moto_ctrl;
    (CLK => OUT) = ADD
  { (clk => moto_o1);
    (clk => moto_o2); }
}
```

Example 5

This examples shows how to specify `IOPATH` mappings with a `COND` condition around them. For example, if the Verilog design has a specify block with the following conditional statement:

```
specify
    if (TI_cond0)
       (A => B) = (3:4:5);
endspecify
```

And the SDF file for the design has a statement for annotation:

```
(COND TI_cond0
    (IOPATH A B(0.2:0.3:0.4) (0.27:0.37:0.47)))
```

The configuration file for mapping can have a specification:

```
IF (TI_cond0)
     (A => B) = OVERRIDE
          { (a => b); }
```

The conditions in the SDF file are compared to the condition in the configuration file and mapping is performed if the conditions match.

Rules for Module Mapping with Conditional Delays

The rules for module mapping in the case of conditional path delays are shown in the following tables. Table 2-4 on page 28 shows different combinations of how Verilog design path delays are handled in the SDF annotation process, when combined with the `COND` statements of SDF. Table 2-5 on page 28 shows the rules for mapping paths between the SDF file and the configuration file.

Table 2-4 Annotating Path Delays in Verilog-XL

SDF	Verilog-XL	Annotate action
no condition	no condition	Annotate one path
no condition	conditional path delay	Annotate to all conditions in the design, unless an `ifnone` is present
`COND`	no_cond	No annotation
`COND`	conditional path delay	Annotate one path

Table 2-5 Module Mapping in SDF

SDF File	Config File	Map action
no condition	no condition	Mapping performed
no condition	`COND`	No mapping performed

Table 2-5 Module Mapping in SDF

SDF File	Config File	Map action
COND	no condition	Mapping performed
COND	COND	Mapping performed

3

Using the SDF File

This chapter describes the following:

Understanding the SDF File

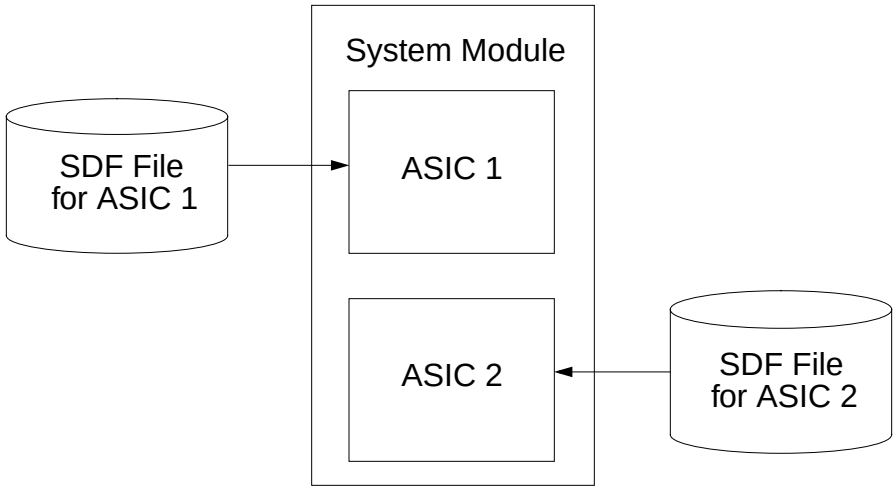
The SDF file is an ASCII text file that stores the timing data generated by the Verilog family tool. The SDF file can contain either a pre-layout or post-layout timing data.

The timing data in the SDF file is independent of the Verilog family tool and can include the following:

Delays	Timing Checks	Data	Parameters
Module path	Setup	Design	Scaling
Device	Hold	Instance	Environmental
Interconnect	Recovery	Type	Technological
Port	Removal	Library	
Incremental	Skew		
Absolute	Width		
Conditional	Conditional		
Unconditional	Unconditional		

The SDF file supports hierarchical delay annotation. A design hierarchy might include several different application-specific integrated circuits (ASICs), including cells or blocks within ASICs. Each design hierarchy has its own SDF file, as shown in Figure 3-1 .

Figure 3-1 Multiple SDF Files in a Hierarchical Design



SDF File Conventions

This section describes identifiers, character use, operators, and common expressions in SDF files.

Using Identifiers

Identifiers are names for ports or nets, depending on the syntax. Identifiers can have up to 1024 alphanumeric characters. Special characters are permitted if the escape character precedes the special character. Spaces are not allowed in identifiers. For more information about character use in SDF files, see “Using Characters” on page 32.

You can specify hierarchical identifiers by placing the hierarchy divider character (`.` or `/`) in the identifier name.

Bit specifications can be placed at the end of identifiers with no spaces between the bit specifications and the identifier. Bit specifications are specified in square braces (`[ ]`). If the bit spec is a range, use a colon to separate the range, as shown in the following examples:

```
[4]          [3:31]          [15:0]
```

Edge identifiers are specified with the following names:

```
posedge     negedge     01     10     0z     z1     lz     z0
```

Examples of Correct IDENTIFIERs

```
AMUX\+BMUX
Cache_Row_\#4
mem_array\[0\:1023\]\(0\:15\)
     //From a language where square brackets indicate arrays
     // parentheses indicate bit specs
pipe4\-done\&enb[3]
     // Unescaped square brackets is a bit spec
```

Examples of Incorrect IDENTIFIERs

```
\AMUX+BMUX
     // Do not use Verilog style name escaping
PHASE\  LOCK\  DONE
     // Spaces cannot be escaped
Ctl_Brk\
     // Do not use carriage return
MEM[4:16]_BRK\+IDLE
     // Do not include bit specs within identifiers
```

Using Characters

The following table describes the characters you can use in the SDF file.

Character Type	Characters
Alphanumeric Characters	ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 0123456789_ (underscore)
Arithmetic Characters	+ (add), - (subtract), / (divide), * (multiply)
Bit-wise binary and	& (ampersand)
Bit-wise binary equivalence	^~ or ~^ (caret tilde or tilde caret)
Bit-wise binary exclusive or	^ (caret)
Bit-wise binary inclusive or	| (vertical bar)
Bit-wise unary negation	~ (tilde)
Case equality operator	=== (triple equals signs)
Case inequality operator	!== (exclamation equals equals)

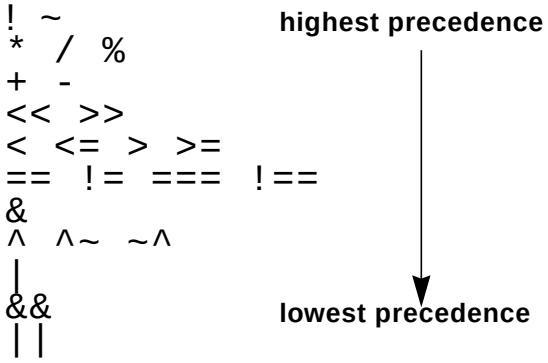
Character Type	Characters
Comment Characters	// double slash for any single line /* begins comment text ending with */
Escape Character	\ (backslash)
Hierarchy dividers	. (period) or / (slash)
Left shift	<< (double left angle brackets)
Logical and	&& (double ampersand)
Logical equality	== (double equals signs)
Logical inequality	!= (exclamation equals)
Logical negation	! (exclamation)
Logical or	|| (double vertical bar)
Modulus	% (percentage)
Relational operators	> (greater than), >= (greater than or equal to), < (less than), <= (less than or equal to),
Right shift	>> (double right angle brackets)
Space	Space, tab, and new line.
Special Characters	~ ! " # $ % & ´ ( ) * + , - . / : ; < = > ? @ [\ ] ^ ` { | }
Unary Operators	+ - ! ~ & ~& | ~| ^ ^~ ~^
Binary Operators	+ < - <= * > / >= % & == | != ^ === ^~ !== ~^ && >> || <<

Note: The escape character (\) must precede a special character in a port or net *identifier*, which enables you to use special characters in port or net *identifier* names. If you use an escape character preceding a hierarchy divider character (. or /), the characters no longer divide the hierarchy.

Operator Precedence

Figure 3-2 shows the order of precedence for SDF operators. Operators shown on the same row have the same precedence.

Figure 3-2 Operator Precedence



OVI SDF Specification Tool Compatibility

This section shows the keywords that Verilog-XL and Verifault-XL uses. Each tool can read an SDF file with all the OVI SDF Standard keyword constructs, but each tool uses a different subset of the OVI SDF Standard keyword constructs. Because a subset of OVI standard keywords apply to some tools but not others, some of the OVI keywords are ignored by the SDF Annnotator. You do not receive error messages on ignored, syntactically correct OVI standard keyword constructs. The OVI SDF Standard keywords are described in "SDF File Keyword Constructs" on page 38.

OVI Standard SDF Keywords

Figure 3-3 shows the keywords that are supported by the OVI SDF Specification, Version 3.0.

Figure 3-3 OVI Standard 3.0 SDF Keywords

```
(DELAYFILE
    (SDFVERSION ...)
    (DESIGN ...)
    (DATE ...)
    (VENDOR ...)
    (PROGRAM ...)
    (VERSION ...)
    (DIVIDER ...)
    (VOLTAGE ...)
    (PROCESS ...)
    (TEMPERATURE ...)
    (TIMESCALE ...)
    (CELL     (CELLTYPE ...)
          (INSTANCE ...)
          (DELAY
            (ABSOLUTE | INCREMENT
```

```
              (IOPATH ...)
              (COND ... (    IOPATH ... {(RETAIN ...)}...))
              (CONDELSE (    IOPATH ... {(RETAIN ...)} ...))
              (PORT ...)
               (INTERCONNECT ...    )
               (NETDELAY ...)
              (DEVICE ...)
            ) // end ABSOLUTE or INCREMENT
            (PATHPULSE ...)
            (PATHPULSEPERCENT ...)
          )  // end DELAY
          (TIMINGCHECK {COND ...)
            (SETUP ...)
            (HOLD ...)
            (SETUPHOLD ... {(SCOND ...)} {(CCOND ...)} )
            (RECOVERY ... {(SCOND ...)} {(CCOND ...)} )
            (REMOVAL ...)
            (RECREM ... {(SCOND ...)} {(CCOND ...)} )
            (SKEW ...)
            (WIDTH ...)
            (PERIOD ...)
            (NOCHANGE ...)
          )  // end TIMINGCHECK
          (TIMINGENV
            (PATHCONSTRAINT ...)
            (PERIODCONSTRAINT ... (EXCEPTION (INSTANCE ...)))
            (SKEWCONSTRAINT ...)
            (SUM ...)
            (DIFF ...)
            (ARRIVAL ...)
            (DEPARTURE ...)
            (SLACK ...)
            (WAVEFORM ...)
          )  // end TIMINGENV
    )  // end CELL
)  // end DELAYFILE
```

SDF Keywords for Verilog-XL

Figure 3-4 shows the keywords that are used by Verilog-XL.

Figure 3-4 Verilog-XL SDF Keywords

```
(DELAYFILE
    (SDFVERSION ...)
    (DIVIDER...)
    (TIMESCALE...)
    (CELL     (CELLTYPE...)
          (INSTANCE...)
          (DELAY
            (ABSOLUTE | INCREMENT
                (IOPATH...)
                (COND...(    IOPATH...))
                (CONDELSE (    IOPATH...))
                (PORT...)
                (INTERCONNECT...    )
                (NETDELAY...)
```

```
              (DEVICE...)
            )// end ABSOLUTE or INCREMENT
            (PATHPULSE...)
            (PATHPULSEPERCENT...)
          )// end DELAY
          (TIMINGCHECK{COND...)
            (SETUP...)
            (HOLD...)
            (SETUPHOLD... {(SCOND...)} {(CCOND...)})
            (RECOVERY... {(SCOND...)} {(CCOND...)})
            (SKEW...)
            (WIDTH...)
               (PERIOD...)
          ) // end TIMINGCHECK
    ) // end CELL
) // end DELAYFILE
```

SDF Keywords for Verifault-XL

Figure 3-5 shows the keywords that are used by Verifault-XL.

Figure 3-5 Verifault-XL SDF Keywords

```
(DELAYFILE
    (SDFVERSION...)
    (DIVIDER...)
    (TIMESCALE...)
    (CELL     (CELLTYPE...)
         (INSTANCE...)
         (DELAY
            (ABSOLUTE | INCREMENT
                (IOPATH...)
                (COND... (    IOPATH...))
                (CONDELSE (    IOPATH...))
                (PORT...)
                (INTERCONNECT...    )
                (NETDELAY...)
                (DEVICE...)
            ) // end ABSOLUTE or INCREMENT
            (PATHPULSE...)
            (PATHPULSEPERCENT...)
          ) // end DELAY
    ) // end CELL
) // end DELAYFILE
```

OVI SDF Specification Version Differences

The SDF Annotator supports multiple versions of the OVI SDF specifications. However, because some constructs in one version may not available in other versions, you need to specify which version you want to use in the `SDFVERSION` entry in the header section of the SDF file. For information about the header section of the SDF file, see “SDF File Keyword Constructs” on page 38.

In most cases, the difference between versions are minimal. If a construct that is not supported under the current version setting is encountered, then you will receive a syntax error.

The following sections show the constructs that exist in a specific version of the OVI SDF specifications when you specify 1.*, 2.*, or 3.* in the SDFVERSION entry.

SDF Version 1.* Constructs

The following constructs are specific to the 1.* versions of the SDF standard. Specify 1.* for any version prior to 2.0.

- If no SDFVERSION is specified, version 1.0 is used by default.
- Version 1.* specifies the conditional TIMINGCHECK construct differently than later versions, as follows:

  ```
  (COND ... (timing_check ...))
  ```

  This implies that you can supply a single condition to the timing check. By contrast, in SDF version 2.0 or greater, you can match one or more timing check terminals, if more than one is present.
- The INCLUDE keyword specifies the full or relative path of a file that contains timing specifications. The file is read as if it was inserted as a continuation of the current SDF file. If the specified file is in your current directory, you can specify just the file name else, you need to specify the full path. The example given below illustrates this.

  ```
  (INCLUDE /cds/home/dff_celldef)
  ```

SDF Version 2.* Constructs

The following constructs are specific to the 2.* versions of the SDF standard:

- The GLOBALPATHPULSE construct is supported. This was changed to PATHPULSEPERCENT in Version 3.0.
- You can optionally specify the TIMESCALE keyword in Version 2.0; in Version 2.1, the TIMESCALE keyword is required. The default for TIMESCALE is 1 nanosecond (ns).
- The timing constraints (PATHCONSTRAINT, SUM, DIFF, SKEWCONSTRAINT) are allowed within the TIMINGCHECK construct.
- The INCLUDE keyword is no longer supported.

SDF Version 3.* Constructs

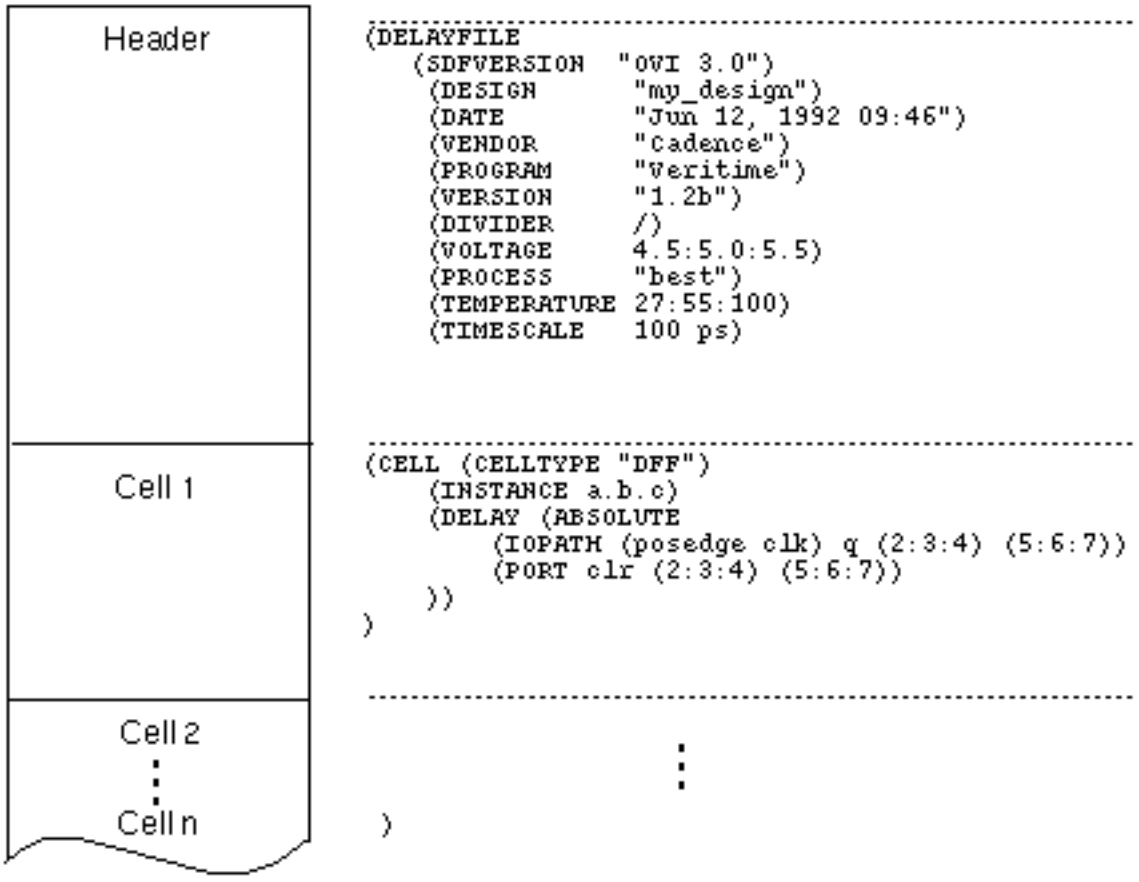
The following constructs are specific to the 3.* versions of the SDF standard:

- The `PATHPULSEPERCENT` keyword replaces the 2.* `GLOBALPATHPULSE` keyword but the functionality is the same.
- Consecutive `INSTANCE` constructs are not allowed. In addition, the `INSTANCE` construct is allowed only in the `CELL` header.
- You can specify 12 delay values, the extra 6 delay values being `X` transition delays.
- The `CONDELSE` construct is supported.
- The `RETAIN` construct is supported in `IOPATH`, `COND IOPATH`, and `CONDELSE IOPATH` constructs.
- The `REMOVAL` timing check is allowed.
- The `RECOVERY` construct allows a single limit only, and does not allow use of the `SCOND` or `CCOND` constructs. Two-limit recoveries should be annotated using the `RECREM` construct.
- The `NETDELAY` construct is no longer supported.
- The `RECREM` construct is supported.
- You can specify timing constraint constructs only in the `TIMINGENV` construct. In addition the new `TIMINGENV` entries (`ARRIVAL`, `DEPARTURE`, `SLACK`, `WAVEFORM`) are supported.

SDF File Keyword Constructs

Every SDF file contains a header section followed by one or more cell entries, as shown in the Figure 3-6 . For each cell entry, you can specify delay and timing check types.

Figure 3-6 Sample SDF File



DELAYFILE Keyword

The `DELAYFILE` keyword construct contains all header and `CELL` entries in an SDF file. You can specify any or all header entries in the `DELAYFILE` construct (but they must be in the sequence shown in the following syntax) before you specify `CELL` entries.

```
(DELAYFILE
    [ (SDFVERSION "sdf_version") ]
    [ (DESIGN "design_name") ]
    [ (DATE "date") ]
    [ (VENDOR "vendor_name") ]
    [ (PROGRAM "program_name") ]
    [ (VERSION "program_version") ]
    [ (DIVIDER hierarchy_divider) ]
    [ (VOLTAGE min:typ:max) ]
    [ (PROCESS "process_name") ]
    [ (TEMPERATURE min:typ:max) ]
```

```
    [ (TIMESCALE time_scale) ]
    (CELL cell_constructs)
)
```

Note: Verilog-XL and Verifault-XL use only the `SDFVERSION`, `DIVIDER`, and `TIMESCALE` header keywords. The other keywords are provided for completeness.

The following table describes the SDF file header keywords and arguments.

Keyword	Keyword Argument	Description
SDFVERSION	*sdf_version*	A string specified in quotation marks that specifies the SDF software version number.
DESIGN	*design_name*	A string specified in quotation marks that specifies the name of the design.
DATE	*date*	A string specified in quotation marks that specifies the date and time when SDF was generated.
VENDOR	*vendor_name*	A string specified in quotation marks that specifies the name of the vendor whose tools generated the SDF file.
PROGRAM	*program_name*	A string specified in quotation marks that specifies the name of the program used to generate the SDF file.
VERSION	*program_version*	A string specified in quotation marks that specifies the program version number used to generate the SDF file.
DIVIDER	*hierarchy_divider*	Either the period (.) which is the default, or the slash (/) which specifies which hierarchical path divider your program is using.
VOLTAGE	*min:typ:max*	Three values (*min:typ:max*) that specify the operating voltage (in volts) of the design.
PROCESS	*process_name*	A string specified in quotation marks that specifies the process operating envelope, which is a string in double quotes

Keyword	Keyword Argument	Description
TEMPERATURE	*min:typ:max*	Three values (*min:typ:max*) that specify the operating ambient temperature(s) of the design in centigrade degrees.
TIMESCALE	*time_scale*	A value followed by a time specification, such as 100 ps for 100 picoseconds. If you do not specify this keyword, the default is 1 ns. You can specify the following time specifications: ■ ns (nanoseconds; default) ■ us (microseconds) ■ ps (picoseconds)
CELL	*cell_constructs*	For more information, see CELL Keyword and Constructs on page 41.

CELL Keyword and Constructs

The cell entries identify specific design instances, paths, and nets and associate timing data with them. Cell entries are specific to a design, instance, library, or type. Each cell entry begins with the CELL keyword followed by the CELLTYPE*, and* INSTANCE keywords. These keywords, in turn, are followed by one or more timing specifications, which contain the actual timing data associated with the cell entry.

The CELL keyword specifies an instance of a cell. The syntax for the CELL keyword is as follows:

```
(CELL
    (CELLTYPE "celltype")
    (INSTANCE path)
    {(DELAY delay_keywords)}
    {(TIMINGCHECK tcheck_keywords)}
    ((TIMINGENV tenv_keywords)}
)
```

Note: The TIMINGENV keyword and its subsequent constructs are ignored by Verilog-XL and Verifault-XL, but are included in this syntax diagram for completeness. You do not receive error messages if you specify TIMINGENV keywords.

You can specify one or more timing specifications using the DELAY, TIMINGCHECK, and TIMINGENV keywords. For information about the DELAY keyword, see "DELAY Keyword and Constructs" on page 43. For information about the TIMINGCHECK keyword, see

“TIMINGCHECK Keyword and Constructs” on page 60. The other `CELL` keyword constructs are described in this section.

CELLTYPE Keyword

The `CELLTYPE` keyword specifies the type of a cell in a quoted string. For example, `"DFF"` specifies a D flip-flop.

Note: This keyword is equivalent to the HDL module name.

INSTANCE Keyword

The `INSTANCE` keyword specifies an instance of the specified cell type. Specify a full hierarchical path such as `a1.b1.c1` or a path relative to the scope of annotation. A full hierarchical path can be represented in either of the following formats:

```
(CELL
    (CELLTYPE "DFF")
    (INSTANCE a1.b1.c1)
        timing_specification
        timing_specification  )
(CELL
    (CELLTYPE "DFF")
    (INSTANCE a1)
    (INSTANCE b1)
    (INSTANCE c1)
        timing_specification
        timing_specification  )
```

The timing data in the timing specification applies only to the specified cell instance. You can also specify an arrayed instance for the cell instance, as shown in the following format:

```
(CELL     (CELLTYPE "DFF")
      (INSTANCE a1.b1[3].c1)
        timing_specification
        timing_specification
)
```

To associate the timing data with all instances of the specified cell type, you can place a wildcard character (`*`) after the `INSTANCE` keyword, as shown in the following example. Only instances in or below the current top level are affected.

```
(CELL (CELLTYPE "DFF")
      (INSTANCE *)
        timing_specification
        timing_specification
        ...
)
```

If you do not specify a path, the default is the current top level.

DELAY Keyword and Constructs

The DELAY keyword specifies the delay values associated with module paths, nets, interconnects, devices, and ports. You can specify the keyword entries listed in the following syntax.

```
(DELAY
    {(ABSOLUTE
        (IOPATH port_spec port_path delay_list)
        (COND cond_port_expr
            (IOPATH port_spec     port_path
                {(RETAIN delay_list)} delay_list)    )
        (CONDELSE
            (IOPATH port_spec    port_path
                {(RETAIN delay_list)} delay_list)    )
        (PORT port_path delay_list)
        (INTERCONNECT port_path1 port_path2 delay_list    )
        (NETDELAY name delay_list)
        (DEVICE {port_path} delay_list))}
{(INCREMENT
    (IOPATH port_spec port_path rdelay_list)
    (COND cond_port_expr
        (IOPATH port_spec port_path
            {(RETAIN rdelay_list)} rdelay_list)    )
    (CONDELSE
        (IOPATH port_spec     port_path
            {(RETAIN rdelay_list)} rdelay_list)    )
    (PORT port_path rdelay_list)
    (INTERCONNECT port_path1 port_path2 rdelay_list)
    (NETDELAY name rdelay_list)
    (DEVICE {port_instance} rdelay_list))}
{(PATHPULSE port_path1 {port_path2} (reject) {(error)})}
{(PATHPULSEPERCENT port_path1 {port_path2} (reject) {(error)} )}
)
```

Note: The *delay_list* variable in the syntax descriptions of this guide can be specified as any of the syntaxes for delays shown in the following table. Also, a delay can be a single value or three values representing minimum, typical, and maximum delays in the form *min:typ:max*. Although you can specify up to 12 delays, Verilog-XL and Verifault-XL use only the first six.

Transitions	Delay Syntax
All transitions	*delay*
Rise and fall	*(delay, delay)*
Rise, fall, and Z	*(delay, delay, delay)*
01, 10, 0Z, Z1, 1Z, Z0	*(delay, delay, delay, delay, delay, delay)*

Transitions	Delay Syntax
01, 10, 0Z, Z1, 1Z, Z0, 0X, X1, 1X, X0, XZ, ZX	*(delay, delay, delay, delay, delay, delay, delay, delay, delay, delay, delay, delay)*

ABSOLUTE Keyword

The ABSOLUTE keyword specifies the delay values that replace the existing delay values in the design. The syntax for the ABSOLUTE keyword construct is as follows:

```
(ABSOLUTE
    (IOPATH port_spec port_path delay_list)
    (COND cond_port_expr
        (IOPATH port_spec     port_path
            {(RETAIN delay_list)} delay_list))
    (CONDELSE
        (IOPATH port_spec     port_path
            {(RETAIN delay_list)} delay_list))
    (PORT port_path delay_list)
    (INTERCONNECT port_path1 port_path2 delay_list    )
    (NETDELAY name delay_list)
    (DEVICE {port_path} delay_list)
)
```

In the following example, the delay values in the examples are specified as two *min:typ:max* triplets. The first triplet is the delay for a rising edge transition and the second triplet is the delay for a falling edge transition.

```
(CELL
    (CELLTYPE "DFF")
    (INSTANCE a.b.c)
    (DELAY
        (ABSOLUTE
            (IOPATH (posedge clk) q (22:28:33) (25:30:37))
            (PORT clr (32:39:49) (35:41:47))
        )
    )
)
```

INCREMENT Keyword

The INCREMENT keyword specifies delay values that are positive or negative and that are added to the existing delay values. The syntax for the INCREMENT keyword construct is as follows:

```
(INCREMENT
    (IOPATH port_spec port_path rdelay_list)
    (COND cond_port_expr
        (IOPATH port_spec     port_path
```

```
            {(RETAIN rdelay_list)} rdelay_list)    )
    (CONDELSE
        (IOPATH port_spec     port_path
            {(RETAIN rdelay_list)} rdelay_list)    )
    (PORT port_path rdelay_list)
    (INTERCONNECT port_instance1 port_instance2 rdelay_list)
    (NETDELAY name rdelay_list)
    (DEVICE {port_instance} rdelay_list)
)
```

Note: You can only use positive numbers (*delay_list*) with the ABSOLUTE keyword; you can use positive or negative numbers (*rdelay_list*) with the INCREMENT keyword. Other than this difference, the keywords have the same syntax. The following sections describe the constructs that you can use within the ABSOLUTE and INCREMENT keyword constructs.

In the following example, the delay values in the examples are specified as two *min:typ:max* triplets. The first triplet is the delay for a rising edge transition and the second triplet is the delay for a falling edge transition.

```
(CELL     (CELLTYPE "DFF")
    (INSTANCE a.b.c)
    (DELAY (INCREMENT
                (IOPATH (posedge clk) q (-4::2) (-7::5))
                (PORT clr (2:3:4) (5:6:7))
             )
      )
)
```

IOPATH Keyword

The IOPATH keyword specifies delays on a path from an input port to an output port of a device and optional reject limits and error limits on the path. The syntax for the IOPATH keyword is as follows:

```
(IOPATH port_spec port_path {(RETAIN rdelay_list)} rdelay_list)
```

The following table describes the arguments of theIOPATH keyword

Keyword Argument	Description
port_spec	*Input* or *inout* (bidirectional) port. An edge identifier can be included.
port_path	*Output* or *inout* port. Where applicable, a port path can have an array index. For example: x.y[3].p.
RETAIN	Ignored by Verilog-XL and Verifault-XL because retain delays are not supported by these tools. It is part of the syntax only for completeness.

Keyword Argument	Description
`rdelay_list`	`IOPATH` delay from *`port_spec`* to *`port_path`*. The delay can also include optional reject and error limit specifications. You can specify negative numbers only within the `INCREMENT` keyword construct.

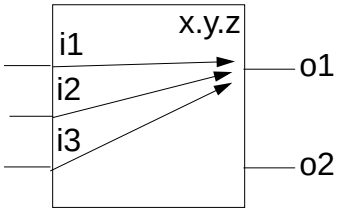
Each delay value is associated with a unique input port/output port pair.

Note: The SDF Annotator calculates the reject and error limit values as follows to determine the level of acceptance for a delay value on a module path.

```
error_limit  = (error%/100)  * (module_path_delay)
reject_limit = (reject%/100) * (module_path_delay)
```

In the following example, the *`delay_list`* for each `IOPATH` is a set of three *`min:typ:max`* triplets that specifies the delays for rise, fall, and turn-off transitions. This example also includes a conditional `IOPATH` using the `COND` construct to represent state-dependent path delays.

```
(INSTANCE x.y.z)
(DELAY (ABSOLUTE
         (IOPATH (posedge i1) o1 (2:3:4) (4:5:6) (3:5:6))
         (IOPATH i2 o1 (2:4:5) (5:6:7) (4:6:7))
         (COND i1 (IOPATH i3 o1 (2:4:5) (4:5:6) (4:5:7))
       )
)
```



To specify optional reject and error limits, enclose the entire *`delay_list`* in parentheses and enclose the delay, reject limit, and error limit in their own parentheses. For example, the following construct specifies one delay. For all transitions, the delay is 12, the reject limit is 6, and the error limit is 10.

```
(IOPATH A B ((12:12:12) (6:6:6) (10:10:10)))
```

To specify that a current delay is to be maintained, use an empty set of parentheses. For example, the following `IOPATH` statement would annotate a delay of `3:5:7` and an error limit of `2:3:6`, while keeping the current setting for the reject limit.

```
(IOPATH A B ((3:5:7) ( ) (2:3:6)))
```

The following commented examples illustrate the syntax for the `IOPATH` keyword construct.

```
(IOPATH A B (12:12:12))
    // Delay=12 for all transitions
    // Reject and error limits are not specified,
    // and are set equal to the delay.
(IOPATH A B (12:12:12)
(10:10:10))
    // Delay=12 for rise transition.
    // Delay=10 for fall transition.
    // Reject and error limits are not specified,
    // and are set equal to the delay.
(IOPATH A B ((12:12:12) (10:10:10)))
    // Delay=12 and reject=10 for all transitions.
    // Error limit is not included,
    // so it is set equal to reject limit.
(IOPATH A B ((12:12:12) (6:6:6) (10:10:10)))
    // Delay=12, reject=6, error=10
    // for all transitions.
(IOPATH A B ((12:12:12) ( ) (10:10:10)))
    // Delay=12, reject=current value,
    //error=10 for all transitions.
(IOPATH A B (12:12:12)
((10:10:10) (5:5:5) (9:9:9)))
    // Delay=12, reject=12, error=12
    // for rise transition.
    // Delay=10, reject=5, error=9
    // for fall transition.
(IOPATH A B ((12:12:12) (6:6:6) (8:8:8))
((10:10:10) (5:5:5) (9:9:9)))
    // Delay=12, reject=6, error=8
    // for rise transition.
    //  Delay=10, reject=5, error=9
    // for fall transition.
(IOPATH A B ((12:12:12) (6:6:6))
((10:10:10) (5:5:5) (9:9:9)))
    // Delay=12, reject=6, error=6
    // for rise transition.
    // Delay=10, reject=5, error=9
//   for fall transition.
(IOPATH A B ((5:5:5) (2:2:2) (3:3:3))
       ((6:6:6) (3:3:3) (4:4:4))
       ((15:15:15) (7:7:7) (10))
    ((14:14:14) (6:6:6) (9:9:9))
    ((12:12:12) (7:7:7) (9:9:9))
    ((13:13:13) (5:5:5) (8:8:8)))
    // Delay=5,  reject=2, error=3  for 01 transition.
    // Delay=6,  reject=3, error=4  for 10 transition.
    // Delay=15, reject=7, error=10 for 0Z transition.
    // Delay=14, reject=6, error=9  for Z1 transition.
    // Delay=12, reject=7, error=9  for 1Z transition.
    // Delay=13, reject=5, error=8  for Z0 transition.
```

COND Keyword

The COND keyword specifies conditional module path delays. The syntax is as follows:

```
(COND cond_port_expr
    (IOPATH port_spec port_path
        {(RETAIN rdelay_list)} rdelay_list
     )
)
```

The following table describes the arguments of the COND keyword.

Keyword Argument	Description
cond_port_expr	Boolean description of the state dependencty of the delay. The delay values apply only if *cond_port_expr* is true(logical one).
IOPATH	See IOPATH Keyword on page 45 for more information.
port_spec	*Input* or *inout* port that can have an edge identifier.
port_path	*Output* or *inout* port. Where applicable, a port path can have an array index. For example: x.y[3].p.
RETAIN	Ignored by Verilog-XL and Verifault-XL because both these tools do not support retain delays. It is part of the syntax only for completeness.
rdelay_list	IOPATH delay from *port_spec* to *port_path*.

CONDELSE Keyword

The CONDELSE keyword specifies path delays when a signal change must be propagates to an output or inout, but none of the conditions for module paths to it are true. The syntax is as follows:

```
(CONDELSE
    (IOPATH port_spec port_path
        {(RETAIN rdelay_list)} rdelay_list
     )
)
```

The following table describes the arguments of the CONDELSE keyword.

Keyword Argument	Description
IOPATH	See IOPATH Keyword on page 45 for more information.

Keyword Argument	Description
port_spec	*Input* or *inout* port that can have an edge identifier.
port_path	*Output* or *inout* port. Where applicable, a port path can have an array index. For example: x.y[3].p.
RETAIN	Ignored by Verilog-XL and Verifault-XL because both these tools do not support retain delays. It is part of the syntax only for completeness.
rdelay_list	IOPATH delay from *port_spec* to *port_path*.

Use the CONDELSE keyword to cover all cases for a path that have not been specified in COND keyword constructs.

RETAIN Keyword

The RETAIN keyword specifies the time for which an *output* or *inout* port retains its previous logic value after a change at a related *input* or *inout* port. It is specified inside an IOPATH keyword construct. Use the RETAIN keyword on paths that proceed from the address or select *inputs* to the data *outputs* of memory and register file circuits. Also, you should use this keyword only where the cell timing model includes an explicit mechanism for providing retention times. The syntax is as follows:

```
(RETAIN delay_list)
```

The *delay_list* variable specifies the retention time data from the *port_spec* to the *port_path* variables in the IOPATH keyword construct. Consecutive delays in the *delay_list* must be increasing numerically.

The following example shows the retain time of the bus do[7:0] with respect to changes on the bus addr[13:0]. The rise time (4:5:7) proceeds from low to X; the fall time (5:6:9) proceeds from high to X.

```
(IOPATH addr[13:0] do[7:0] (RETAIN (4:5:7) (5:6:9))
```

PORT Keyword

The PORT keyword specifies estimated or actual interconnect delay values you can place on the input port, without having to specify a start point for the wire path. The syntax is as follows:

```
(PORT port_path delay_list)
```

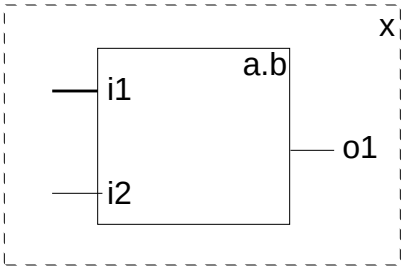
The following table describes the arguments of the `PORT` keyword.

Keyword Argument	Description
`port_path`	*Input* or *inout* port. Where applicable, a port path can have an array index. For example: `x.y[3].p`.
`delay_list`	`PORT` delay of the *`port_path`*. Optional reject and error limits can be included.

If you specify interconnect delays and port delays for the same input of a module, the SDF Annotator uses whichever specification appears last in the SDF file.

In the following example, the delay consists of one *`min:typ:max`* triplet specifying the minimum, typical, and maximum delays for all transitions. The `PORT` delay is specified as incremental, which means the existing delay data values are increased or decreased rather than replaced.

```
(INSTANCE x)
(DELAY
    (INCREMENT (PORT a.b.i1 (-2:0:2)))
)
```



Port delays are mapped to Module Input Port Delays (MIPDs). By default, MIPDs use inertial delays. Verilog-XL has interconnect transport delay functionality with pulse control, which is enabled by using the `+transport_int_delays` and the `+multisource_int_delays` plus options.

Verilog-XL generates Single-source Interconnect Transport Delays (SITDs) or Multi-source Interconnect Transport Delays (MITDs), as shown in the following table:

Plus Option	Single-source Nets	Multi-source Nets
`+transport_int_delays`	SITD	SITD

Plus Option	Single-source Nets	Multi-source Nets
`+multisource_int_delays`	MIPD	MITD
Both plus options	SITD	MITD
Neither plus option	MIPD	MIPD

For more information and examples of valid and invalid interconnect combinations, see Appendix B, “Valid and Invalid Interconnect Combinations.”

To specify optional reject and error limits, enclose the entire *delay_list* in parentheses and enclose the delay, reject limit, and error limit in their own parentheses. For example, the following command specifies one delay. For all transitions, the delay is 12, the reject limit is 6, and the error limit is 10.

```
(PORT A ((12:12:12) (6:6:6) (10:10:10)))
```

To specify that a current delay is to be maintained, use an empty set of parentheses. For example, the following PORT keyword would annotate a delay of 3:5:7 and an error limit of 2:3:6, while keeping the current setting for the reject limit.

```
(PORT A ((3:5:7) ( ) (2:3:6)))
```

The following commented examples show how to use the PORT keyword.

```
(PORT A (12:12:12))
    // Delay=12 for all transitions.
    // Reject and error limits are not specified,
    // and are set equal to the delay.

(PORT A (12:12:12)
(10:10:10))
    // Delay=12 for rise transition.
    // Delay=10 for fall transition.
    // Reject and error limits are not specified,
    // and are set equal to the delay.

(PORT A ((12:12:12) (6:6:6) (10:10:10)))
    // Delay=12, reject=6, error=10
    // for all transitions.

(PORT A ((12:12:12) ( ) (10:10:10)))
    // Delay=12, reject=current value,
    // error=10 for all transitions.

(PORT A ((12:12:12) (10:10:10)))
    // Delay=12 and reject=10 for all transitions.
    // Error limit is not included,
    // so it is set equal to reject limit.

(PORT A ((5:5:5) (2:2:2) (3:3:3))
        ((6:6:6) (3:3:3) (4:4:4))
        ((15:15:15) (7:7:7) (10))
        ((14:14:14) (6:6:6) (9:9:9))
        ((12:12:12) (7:7:7) (9:9:9))
        ((13:13:13) (5:5:5) (8:8:8)))
```

```
    // Delay=5,  reject=2, error=3  for 01 transition.
    // Delay=6,  reject=3, error=4  for 10 transition.
    // Delay=15, reject=7, error=10 for 0Z transition.
    // Delay=14, reject=6, error=9  for Z1 transition.
    // Delay=12, reject=7, error=9  for 1Z transition.
    // Delay=13, reject=5, error=8  for Z0 transition.
```

INTERCONNECT Keyword

The `INTERCONNECT` keyword specifies estimated or actual delays in the wire paths between devices. The syntax is as follows:

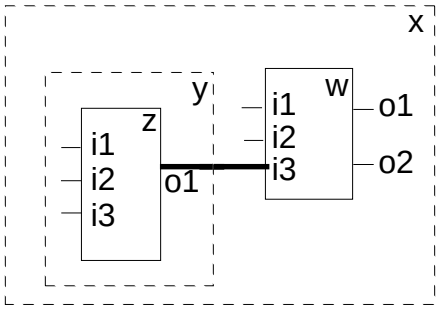
`(INTERCONNECT` *`port_path1 port_path2 delay_list`*`)`

The following table describes the arguments of the `INTERCONNECT` keyword

Keyword Argument	Description
`port_path1`	*Output* or *inout* port. Where applicable, a port path can have an array index. For example: `x.y[3].p`.
`port_path2`	*Input* or *inout* port. Where applicable, a port path can have an array index. For example: `x.y[3].p`.
`delay_list`	Interconnect delay between the *output* and *input* ports. Unique delays can be specified for multi-source nets. The delay can also include optional reject and error limits.

In the following example, the *`delay_list`* consists of two *`min:typ:max`* triplets specifying the delays for rise and fall transitions.

```
(INSTANCE x)
(DELAY
    (ABSOLUTE
        (INTERCONNECT y.z.o1 w.i3 (5:6:7) (5.5:6:6.5))
    )
)
```



`INTERCONNECT` delays are mapped to Module Input Port Delays (MIPDs) by default. MIPDs use inertial delays. Verilog-XL has interconnect transport delay functionality with pulse control, which is enabled by using the `+transport_int_delays` and/or the `+multisource_int_delays` plus options.

Verilog-XL generates Single-source Interconnect Transport Delays (SITDs) or Multi-source Interconnect Transport Delays (MITDs), as shown in the following table:

Plus Option	Single-source Nets	Multi-source Nets
`+transport_int_delays`	SITD	SITD
`+multisource_int_delays`	MIPD	MITD
Both plus options	SITD	MITD
Neither plus option	MIPD	MIPD

For more information and examples of valid and invalid interconnect combinations, see Appendix B, “Valid and Invalid Interconnect Combinations.”

To specify optional reject and error limits, enclose the entire *`delay_list`* in parentheses and enclose the delay, reject limit, and error limit in their own parentheses. For example, the following command specifies one delay. For all transitions, the delay is 12, the reject limit is 6, and the error limit is 10.

```
(INTERCONNECT A B ((12:12:12) (6:6:6) (10:10:10)))
```

To specify that a current delay is to be maintained, use an empty set of parentheses. For example, the following `INTERCONNECT` statement annotates a delay of 3:5:7 and an error limit of 2:3:6, while keeping the current setting for the reject limit.

```
(INTERCONNECT A B ((3:5:7) ( ) (2:3:6)))
```

The following commented examples illustrate the syntax for *INTERCONNECT*.

```
(INTERCONNECT A B (12:12:12))
    // Delay=12 for all transitions.
    // Reject and error limits are not specified,
    // and are set equal to the delay.
(INTERCONNECT A B ((12:12:12) (6:6:6) (10:10:10)))
    // Delay=12, reject=6, error=10
    // for all transitions.
(INTERCONNECT A B ((12:12:12) ( ) (10:10:10)))
    // Delay=12, reject=current value,
    // error=10 for all transitions.
(INTERCONNECT A B ((12:12:12) (10:10:10)))
    // Delay=12 and reject=10 for all transitions.
    // Error limit is not included,
    // so it is set equal to reject limit.
(INTERCONNECT A B (12:12:12)
```

```
((10:10:10) (5:5:5) (9:9:9)))
    // Delay=12, reject=12, error=12
    // for rise transition.
    // Delay=10, reject=5, error=9
    // for fall transition.
(INTERCONNECT A B ((5:5:5) (2:2:2) (3:3:3))
     ((6:6:6) (3:3:3) (4:4:4))
     ((15:15:15) (7:7:7) (10))
     ((14:14:14) (6:6:6) (9:9:9))
     ((12:12:12) (7:7:7) (9:9:9))
     ((13:13:13) (5:5:5) (8:8:8)))
    // Delay=5,  reject=2, error=3  for 01 transition.
    // Delay=6,  reject=3, error=4  for 10 transition.
    // Delay=15, reject=7, error=10 for 0Z transition.
    // Delay=14, reject=6, error=9  for Z1 transition.
    // Delay=12, reject=7, error=9  for 1Z transition.
    // Delay=13, reject=5, error=8  for Z0 transition.
(INTERCONNECT A D ((5:5:5) (2:2:2) (3:3:3)))
(INTERCONNECT B D ((6:6:6) (3:3:3) (4:4:4)))
(INTERCONNECT C D ((7:7:7) (4:4:4) (5:5:5)))
    // Unique delays, reject limits, and error limits
    // for multi-source net.
```

NETDELAY Keyword

The `NETDELAY` keyword specifies delay for a complete net, where delays from all the source port(s) on the net to all destination port(s) have the same value. `NETDELAY` is a short form of `INTERCONNECT` delay.

The syntax is as follows:

`(NETDELAY` *`name delay_list`*`)`

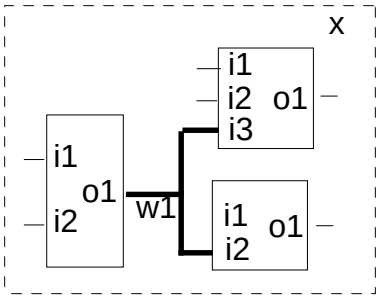
The following table describes the arguments of the `NETDELAY` keyword.

Keyword Argument	Description
`name`	Name of the net or the output port driving the net. Where applicable, a port name can have array index (for example, x.y[3].p).
`delay_list`	Delay associated with the net or port specified by name. The value specifies the same delay for all source/load pairs. The delay can include optional reject and error limits.

In the following example, the net is identified by name. The *`delay_list`* consists of three *`min:typ:max`* triplets specifying the rise, fall, and turn-off delays.

```
(INSTANCE x)
(DELAY
```

```
    (ABSOLUTE
        (NETDELAY w1 (2.5:3.0:3.5) (2.9:3.4:4.2) (6.3:8:9.9))
    )
)
```



NETDELAY delays are mapped to Module Input Port Delays (MIPDs). By default, MIPDs use inertial delays. Verilog-XL has interconnect transport delay functionality with pulse control, which is enabled by using the `+transport_int_delays` and/or the `+multisource_int_delays` plus options.

Verilog-XL generates Single-source Interconnect Transport Delays (SITDs) or Multi-source Interconnect Transport Delays (MITDs), as shown in the following table:

Plus Option	Single-Source Nets	Multi-source Nets
`+transport_int_delays`	SITD	SITD
`+multisource_int_delays`	MIPD	MITD
Both plus options	SITD	MITD
Neither plus option	MIPD	MIPD

For more information and examples of valid and invalid interconnect combinations, see Appendix B, “Valid and Invalid Interconnect Combinations.”

To specify optional reject and error limits, enclose the entire *delay_list* in parentheses and enclose the delay, reject limit, and error limit in their own parentheses. For example, the following command specifies one delay. For all transitions, the delay is 12, the reject limit is 6, and the error limit is 10.

```
(NETDELAY A ((12:12:12) (6:6:6) (10:10:10)))
```

To specify that a current delay is to be maintained, use an empty set of parentheses. For example, the following NETDELAY statement would annotate a delay of 3:5:7 and an error limit of 2:3:6, while keeping the current setting for the reject limit.

```
(NETDELAY A ((3:5:7) ( ) (2:3:6)))
```

The following commented examples illustrate the syntax for NETDELAY.

```
(NETDELAY A (12:12:12))
     // Delay=12 for all transitions.
     // Reject and error limits are not specified,
     // and are set equal to the delay.

(NETDELAY A (12:12:12)
(10:10:10))
    // Delay=12 for rise transition.
    // Delay=10 for fall transition.
    // Reject and error limits are not specified,
    // and are set equal to the delay.

(NETDELAY A ((12:12:12) (6:6:6) (10:10:10)))
    // Delay=12, reject=6, error=10
    // for all transitions.

(NETDELAY A ((12:12:12) ( ) (10:10:10)))
    // Delay=12, reject=current value,
    // error=10 for all transitions.

(NETDELAY A ((12:12:12) (10:10:10)))
    // Delay=12 and reject=10 for all transitions.
    // Error limit is not included,
    // so it is set equal to reject limit.

(NETDELAY A ((5:5:5) (2:2:2) (3:3:3))
            ((6:6:6) (3:3:3) (4:4:4))
            ((15:15:15) (7:7:7) (10))
            ((14:14:14) (6:6:6) (9:9:9))
            ((12:12:12) (7:7:7) (9:9:9))
            ((13:13:13) (5:5:5) (8:8:8)))

      // Delay=5,  reject=2, error=3  for 01 transition.
      // Delay=6,  reject=3, error=4  for 10 transition.
      // Delay=15, reject=7, error=10 for 0Z transition.
      // Delay=14, reject=6, error=9  for Z1 transition.
      // Delay=12, reject=7, error=9  for 1Z transition.
      // Delay=13, reject=5, error=8  for Z0 transition.
```

DEVICE Keyword

The DEVICE keyword specifies the intrinsic delay of a module or gate. Intrinsic delay is specific to the type of the object and has the same value for every instance of that module or gate. Conceptually, this represents all path delays through the object, independent of loading or input slope. At the gate level, the delay is associated with the output. If a module has more than one output, specify the delays to each output port by using additional *port_instance* specifications. If you do not specify any port, SDF assumes that all output ports have the same delay values. The syntax is as follows:

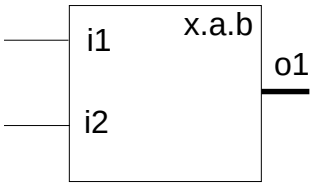
```
(DEVICE {port_path} delay_list)
```

The following table describes the arguments of the DEVICE keyword.

Keyword Argument	Description
port_path	Output port. Where applicable, a port path can have array index (for example, x.y[3].p).
delay_list	Device delay (specific to a type).

In the following example, the *delay_list* consists of three *min:typ:max* triplets specifying the rise, fall, and turn-off delays.

```
(INSTANCE x.a.b)
(DELAY
    (ABSOLUTE
        (DEVICE o1 (6:7:8) (4:6:7) (5:8:9))
    )
)
```



Verilog-XL and Verifault-XL have path pulse control functionality. To specify optional reject and error limits, enclose the entire *delay_list* in parentheses and enclose the delay, reject limit, and error limit in their own parentheses. For example, the following command specifies one delay. For all transitions, the delay is 12, the reject limit is 6, and the error limit is 10.

```
(DEVICE A ((12:12:12) (6:6:6) (10:10:10)))
```

To specify that you want a current delay maintained, use an empty set of parentheses. For example, the following DEVICE statement would annotate a delay of 3:5:7 and an error limit of 2:3:6, while keeping the current setting for the reject limit.

```
(DEVICE A ((3:5:7) ( ) (2:3:6)))
```

The following commented examples illustrate the syntax for DEVICE.

```
(DEVICE A (12:12:12))
    // Delay=12 for all transitions.
    // Reject and error limits are not specified,
    // and are set equal to the delay.
(DEVICE A (12:12:12)
(10:10:10))
    // Delay=12 for rise transition.
    // Delay=10 for fall transition.
```

```
    // Reject and error limits are not specified,
    // and are set equal to the delay.
(DEVICE A ((12:12:12) (10:10:10)))
    // Delay=12 and reject=10 for all transitions.
    // Error limit is not included,
    // so it is set equal to reject limit.
(DEVICE A ((12:12:12) (6:6:6) (10:10:10)))
    // Delay=12, reject=6, error=10
    // for all transitions.
(DEVICE A ((12:12:12) ( ) (10:10:10)))
    // Delay=12, reject=current value,
    // error=10 for all transitions.
(DEVICE A (12:12:12)
((10:10:10) (5:5:5) (9:9:9)))
    // Delay=12, reject=12, error=12
    // for rise transition.
    // Delay=10, reject=5, error=9
    // for fall transition.
(DEVICE A ((12:12:12) (6:6:6) (8:8:8))
((10:10:10) (5:5:5) (9:9:9)))
    // Delay=12, reject=6, error=8
    // for rise transition.
    // Delay=10, reject=5, error=9
    // for fall transition.
(DEVICE A ((5:5:5) (2:2:2) (3:3:3))
          ((6:6:6) (3:3:3) (4:4:4))
          ((15:15:15) (7:7:7) (10))
          ((14:14:14) (6:6:6) (9:9:9))
          ((12:12:12) (7:7:7) (9:9:9))
          ((13:13:13) (5:5:5) (8:8:8)))
     // Delay=5,  reject=2, error=3  for 01 transition.
     // Delay=6,  reject=3, error=4  for 10 transition.
     // Delay=15, reject=7, error=10 for 0Z transition.
     // Delay=14, reject=6, error=9  for Z1 transition.
     // Delay=12, reject=7, error=9  for 1Z transition.
     // Delay=13, reject=5, error=8  for Z0 transition.
```

PATHPULSE Keyword

The PATHPULSE keyword specifies the limits associated with a legal path between an input port and an output port of a device. These limits determine whether a pulse at the input can pass through the device to the output.

The syntax for the PATHPULSE keyword is as follows. If you specify only one value for *reject* or *error*, both limits are set to that value.

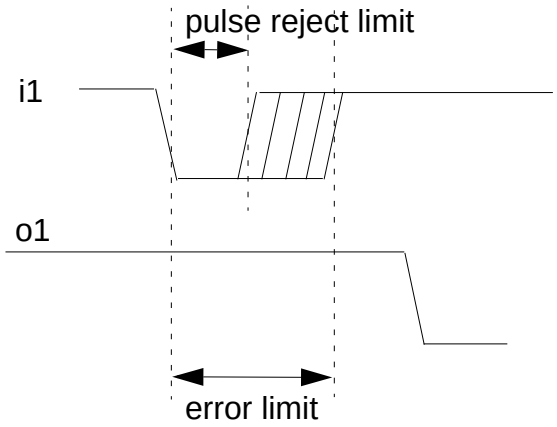
```
(PATHPULSE port_path1 {port_path2} (reject) {(error)} )
```

The following table describes the arguments of the PATHPULSE keyword.

Keyword Argument	Description
port_path1	*Input* or *inout* port. Where applicable, a port path can have array index (for example, x.y[3].p).
port_path2	*Output* or *inout* port. Where applicable, a port path can have array index (for example, x.y[3].p).
reject	Pulse rejection limit, in time units. This limit defines the minimum pulse width required for the pulse to pass through to the output. Anything smaller does not affect the output.
error	The error limit, in time units. This limit defines the minimum pulse width necessary to drive the output to a known state. Anything smaller causes the output to enter the unknown (e) state or is rejected (if smaller than the pulse reject limit). The error limit must be equal to or greater than the pulse reject limit.

In the following example, the first value (13) is the pulse reject limit and the second value (21) is the error limit.

```
(INSTANCE x)
(DELAY
    (PATHPULSE y.i1 y.o1 (13) (21))
)
```



PATHPULSEPERCENT Keyword

The PATHPULSEPERCENT keyword is the same as the PATHPULSE keyword except that reject and error limits are expressed in percentages (%) of the cell path delay from the input to the output. If you specify only one value, both reject and error limits are set to that value. See the PATHPULSE keyword for a description of the syntax.

Note: The PATHPULSEPERCENT keyword supersedes the GLOBALPATHPULSE keyword.

In the following example, the first value (25) is the pulse reject limit and the second value (35) is the error limit.

```
(INSTANCE x)
(DELAY
    (PATHPULSEPERCENT y.i1 y.o1 (25) (35))
)
```

The error limit must be equal to or greater than the reject limit.

If you omit both the reject limit and error limit, both specifications are set to 100. If the reject limit exceeds the error limit or if you omit the error limit, the error limit is set equal to the reject limit.

TIMINGCHECK Keyword and Constructs

The TIMINGCHECK keyword assigns timing check limits to specific cell instances and assigns layout constraints to critical paths in the design that determine how the signals can change in relation to each other. Verilog family tools use this information during the design process, as follows:

- Simulation tools are notified of signal transitions that violate timing checks.
- Timing analysis tools identify paths that might violate timing checks and determine the timing constraints for those paths.
- Layout tools use the timing constraints from timing analysis tools to generate layouts that do not violate any timing checks.

The syntax of the TIMINGCHECK keyword is as follows:

```
(TIMINGCHECK
     {(SETUP data_sig clk_sig setup_time)}
     {(HOLD data_event clk_event hold_time)}
     {(SETUPHOLD data_event clk_event setup_time hold_time
          {(SCOND tstamp_cond)} {(CCOND tcheck_cond)} )}
     {(RECOVERY asynch_ctl_sig clk_or_gate recovery_time
          {(SCOND tstamp_cond)} {(CCOND tcheck_cond)} )}
     {(REMOVAL asynch_ctl_sig clk_or_gate removal_time)}
```

```
    {(RECREM asynch_ctl_port clk_or_gate recovery_time removal_time
         {(SCOND tstamp_cond)} {(CCOND tcheck_cond)} )}
    {(SKEW lower_clk upper_clk max_skew)}
    {(WIDTH edge_clk max_width)}
    {(PERIOD edge_clk max_period)}
    {(NOCHANGE clk_event data_event start_offset end_offset)} )
```

Note: The following syntax applies to SDF standards prior to OVI Version 2.0, where *tcheck* applies to the TIMINGCHECK keyword constructs.

```
(TIMINGCHECK
    (COND tcheck_cond tcheck)
)
```

The following example shows a cell entry that assigns setup and hold timing checks:

```
(CELL (CELLTYPE "DFF")
    (INSTANCE a.b.c)
    (TIMINGCHECK
        (SETUP din (posedge clk) (3:4:5.5))
        (HOLD din (posedge clk) (4:5.5:7))))
```

COND Keyword

The COND keyword specifies conditional timing check. The syntax is as follows for OVI SDF Versions previous to 2.0:

```
(COND tcheck_cond tcheck)
```

The COND keyword in OVI SDF Version 2.0 and higher places the condition on the signal itself as shown in the following syntax examples:

```
(SETUP
  (COND tcheck_cond data_sig) clk_sig setup_time
)
(SETUP
  (data_sig (COND tcheck_cond clk_sig) setup_time
)
```

The following table describes the arguments of the COND keyword.

Keyword Argument	Description
tcheck_cond	Boolean expression.
data_sig	Data signal.

Note: In SETUPHOLD and RECOVERY timing checks, you can use the SCOND and CCOND constructs to condition the time stamp and time check events. See "SETUPHOLD Keyword"

on page 64 and "RECOVERY Keyword" on page 65 for more information about these conditional keywords.

SETUP Keyword

The SETUP keyword specifies the minimum interval before a clock transition. The syntax is as follows:

```
(SETUP data_sig clk_sig setup_time)
```

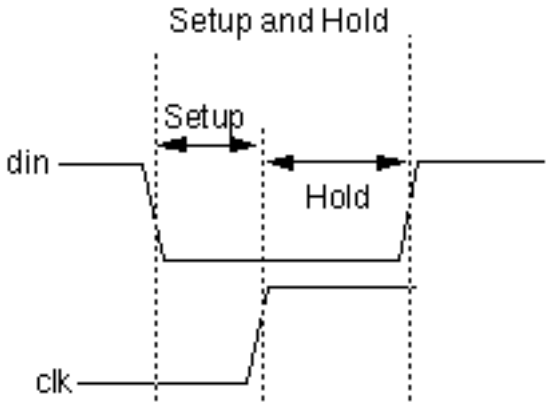
The following table describes the arguments of the SETUP keyword.

Keyword Argument	Description
data_sig	Data signal.
clk_sig	Clock signal.
setup_time	Specifies the minimum interval between the data and clock event (that is, *before* clock transition). Any change to the data signal within these intervals results in a timing violation.

The following examples show how SETUP and HOLD works:

```
(INSTANCE x.a)
(TIMINGCHECK
    (SETUP din (posedge clk) (12))
)
(INSTANCE x.a)
(TIMINGCHECK
```

```
    (HOLD din (posedge clk) (9.5))
)
```



HOLD Keyword

The `HOLD` keyword specifies the minimum interval after a clock transition. The syntax is as follows:

`(HOLD` *`data_event clk_event hold_time`*`)`

The following table describes the arguments of the `HOLD` keyword.

Keyword Argument	Description
`data_event`	Data event.
`clk_event`	Clock event.
`hold_time`	Specifies the minimum interval between the clock and data events (that is, *after* clock transition). Any change to the data signal within these intervals results in a timing violation.

See the example in "SETUPHOLD Keyword" on page 64 to see how you use `SETUP` and `HOLD`.

SETUPHOLD Keyword

The SETUPHOLD keyword specifies the same information with one keyword as both the SETUP and HOLD keywords do. The syntax is as follows:

```
(SETUPHOLD data_event clk_event setup_time hold_time
     {(SCOND tstamp_cond)} {(CCOND tcheck_cond)} )
```

The following table describes the arguments of the SETUPHOLD keyword.

Keyword Argument	Description
data_event	Data event.
clk_event	Clock event.
setup_time	Minimum interval between the data and clock events (that is, *before* clock transition). Any change to the data signal within these intervals results in a timing violation.
hold_time	Minimum interval between the clock and data events (that is, *after* clock transition). Any change to the data signal within these intervals results in a timing violation.
(SCOND *tstamp_cond*)	Event that triggers the timing check. If the *tstamp_cond* is true, the Verilog family tool accepts the timing check. If false, the Verilog family tool ignores the timing check.
(CCOND *tcheck_cond*)	Event that causes the timing check to be validated. If the *tcheck_cond* is true, the Verilog family tool accepts the timing check. If false, the Verilog family tool ignores the timing check.

Note: The *setup_time* or the *hold_time* can be negative, but their sum must be 0 or more. To perform SDF back annotation to SETUPHOLD timing checks with negative values, you must use the +neg_tchk plus option on the command line.

In addition to performing the SETUP and HOLD operations, the SETUPHOLD keyword can have different conditions specified for the event that triggers the check and the event that causes the check to be validated.

If SCOND or CCOND is used with the COND construct, the COND construct is overruled.

The following examples show how to use the SETUPHOLD keyword:

```
(INSTANCE x.a)
(TIMINGCHECK
    (SETUPHOLD din (posedge clk) (12) (9.5))
)
(INSTANCE x.a)
(TIMINGCHECK
    (SETUPHOLD din (posedge clk) (12) (9.5) (SCOND !rst) )
)
(INSTANCE x.a)
(TIMINGCHECK
    (SETUPHOLD din (posedge clk) (12) (9.5) (CCOND !rst) )
)
```

RECOVERY Keyword

The RECOVERY keyword limits a change in an asynchronous control signal and the next clock pulse (for example, between clearbar and the clock for a flip-flop). If the clock signal violates the constraint, the output value is unknown. The syntax is as follows:

```
(RECOVERY asynch_ctl_sig clk_or_gate recovery_time
          {(SCOND tstamp_cond)} {(CCOND tcheck_cond)})
```

The following table describes the arguments of the COND keyword.

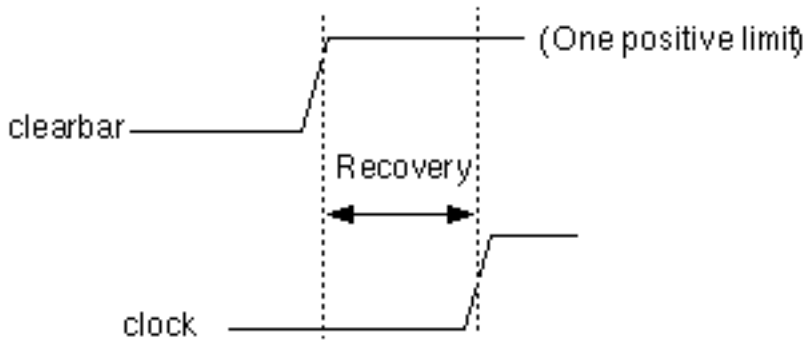
Keyword Argument	Description
asynch_ctl_sig	Asynchronous control signal, which normally has an edge identifier associated with it to indicate which transition corresponds to the release from the active state.
clk_or_gate	Clock (flip-flops) or gate (latches) signal, which normally has an edge identifier to indicate the active edge of the clock or the closing edge of the gate.
recovery_time	Minimum interval between the release of the asynchronous control signal and the next active edge of the clock/gate event. The simulator uses the recovery limit for deterministic comparisons and does not admit x values.
*(*SCOND *tstamp_cond)*	Event that triggers the timing check. If the tstamp_cond is true, the Verilog family tool accepts the timing check. If false, the Verilog family tool ignores the timing check.
*(*CCOND *tcheck_cond)*	Event that causes the timing check to be validated. If the tcheck_cond is true, the Verilog family tool accepts the timing check. If false, the Verilog family tool ignores the timing check.

The recovery timing check can have different conditions specified for the event that triggers the check and the event that causes the check to be validated.

If SCOND or CCOND is used with the COND construct, the COND construct is overruled.

The following example shows how to use the RECOVERY keyword:

```
(INSTANCE x.b)
(TIMINGCHECK
            (RECOVERY (posedge clearbar) (posedge clk) (11.5)))
```




```
(INSTANCE x.a)
(TIMINGCHECK
    (RECOVERY (posedge rst) (posedge clk) (12) (9.5)
              (SCOND       !clear)       )
)
(INSTANCE x.a)
(TIMINGCHECK
    (RECOVERY (posedge rst) (posedge clk) (12) (9.5)
               (CCOND        !clear)        )
)
```

Note: Prior to the release of SDF version 3.0, the RECOVERY keyword was used to annotate two limits - the recovery limit and the removal limit. Two-limit recoveries are now annotated using the RECREM keyword. For more information on two-limit recoveries, refer to "RECREM Keyword" on page 67.

REMOVAL Keyword

The REMOVAL keyword is similar to the HOLD keyword and specifies the time between an active clock edge and the release of an asynchronous control signal from the active state. The syntax is as follows:

```
(REMOVAL asynch_ctl_sig clk_or_gate removal_time)
```

The following table describes the arguments of the REMOVAL keyword.

Keyword Argument	Description
asynch_ctl_sig	Asynchronous control signal, which normally has an edge identifier associated with it to indicate which transition corresponds to the release from the active state.
clk_or_gate	Clock (flip-flops) or gate (latches) signal, which normally has an edge identifier to indicate the active edge of the clock or the closing edge of the gate.
removal_time	Positive time value for which an extraordinary operation (such as a `set` or `reset`) must persist to ensure that a device ignores any normal operation (such as, clocking in new data).

For example, if the release of the clearbar occurs too soon after the edge of the clock, the state of a flip-flop between the clock and the clearbar becomes uncertain. That is, it could be the value set by the clearbar, or it could be the value clocked into the flip-flop from the data input. The following example shows how to use the REMOVAL keyword to avoid this problem.

```
(INSTANCE x.b)
(TIMINGCHECK
    (REMOVAL (posedge clearbar) (posedge clk) (6.3))
)
```

RECREM Keyword

The RECREM keyword specifies both recovery and removal limits in a single keyword. The syntax is as follows:

```
(RECREM asynch_ctl_sig clk_or_gate recovery_time removal_time
     {(SCOND tstamp_cond)} {(CCOND tcheck_cond)})
```

The following table describes the arguments of the RECREM keyword.

Keyword Argument	Description
asynch_ctl_sig	Asynchronous control signal, which normally has an edge identifier associated with it to indicate which transition corresponds to the release from the active state.

Keyword Argument	Description
clk_or_gate	Clock (flip-flop) or gate (latch) signal, which normally has an edge identifier to indicate the active edge of the clock or the closing edge of the gate.
recovery_time	Minimum interval between the release of the asynchronous control signal and the next active edge of the clock/gate event. The simulator uses the recovery limit for deterministic comparisons and does not admit x values.
removal_time	Minimum interval between the clock and data events (that is, *after* clock transition). Any change to the data signal within these intervals results in a timing violation.
*(*SCOND *tstamp_cond)*	Event that triggers the timing check. If the tstamp_cond is true, the Verilog family tool accepts the timing check. If false, the Verilog family tool ignores the timing check.
*(*CCOND *tcheck_cond)*	Event that causes the timing check to be validated. If the tcheck_cond is true, the Verilog family tool accepts the timing check. If false, the Verilog family tool ignores the timing check.

When two time limits (*recovery_time* and *removal_time*) are specified, the *recovery_time* can be negative, but the sum of both must be 0 or more. To perform SDF annotation to recovery timing checks with negative values, you must use the +neg_tchk plus option on the command line.

The following example specifies a recovery time of 1.5 before the clock transition and a removal time of 0.8 after the clock transition. Any change to the clearbar signal within this interval results in a timing violation.

```
(INSTANCE x.b)
(TIMINGCHECK
    (RECREM (posedge clearbar) (posedge clk) (1.5) (0.8)))
```

Note: Prior to the release of SDF version 3.0, the RECOVERY keyword was used to annotate two-limit recoveries.

SKEW Keyword

The SKEW keyword specifies the limits for signal skew timing checks. A signal skew limit is the maximum delay allowable between two signals. You can specify these timing checks only between two signals existing at the same design hierarchy level. The syntax is as follows:

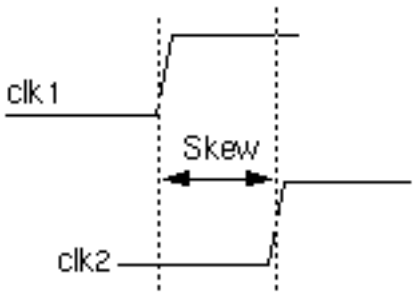
```
(SKEW lower_clk upper_clk max_skew)
```

The following table describes the arguments of the SKEW keyword.

Keyword Argument	Description
lower_clk	Lower-bound clock event which can include an edge specification.
upper_clk	Upper-bound clock event which can include an edge specification.
max_skew	Maximum delay allowed between the upper- and lower-bound clock signals.

The following example shows how to use the SKEW keyword:

```
(INSTANCE x)
(TIMINGCHECK
    (SKEW (posedge clk1) (posedge clk2) (6))  )
```



WIDTH Keyword

The WIDTH keyword specifies the duration of signal levels from one clock edge to the opposite clock edge. If a signal has unequal phases, specify a separate width check for each phase. The syntax is as follows:

```
(WIDTH edge_clk max_width)
```

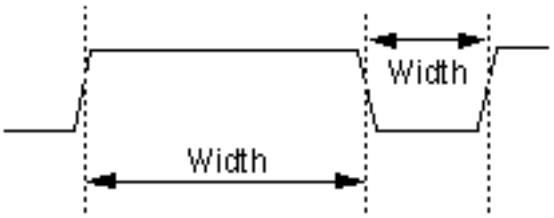
The following table describes the arguments of the WIDTH keyword.

Keyword Argument	Description
edge_clk	Edge-triggered clock event.

Keyword Argument	Description
max_width	Maximum time for the positive or negative phase of each cycle.

The following example shows how to use the WIDTH keyword:

```
(INSTANCE x.b)
(TIMINGCHECK
             (WIDTH (posedge clk) (30))
             (WIDTH (negedge clk) (16.5))
)
```



The first width check is the phase beginning with the positive clock edge, and the second width check is the phase beginning with the negative clock edge. The data event is equal to the clock event with the opposite edge.

PERIOD Keyword

The PERIOD keyword specifies limit values for a minimum period timing check. The minimum period timing check is the minimum allowable time for one complete cycle.

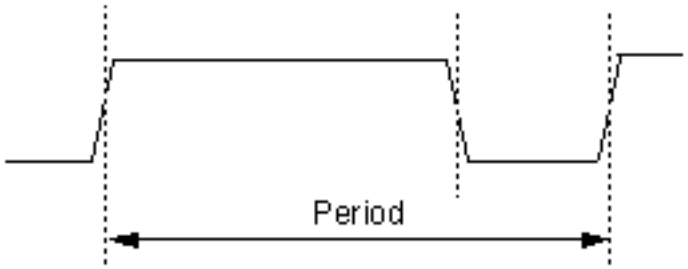
```
(PERIOD edge_clk max_period))
```

The following table describes the arguments for the PERIOD keyword.

Keyword Argument	Description
edge_clk	Edge-triggered clock event.
max_period	Minimum period for complete signal cycle.

The following example shows how to use the PERIOD keyword. The period is the interval between two positive clock edges. The data event is equal to the clock event with the same edge.

```
(INSTANCE x.b)
(TIMINGCHECK
        (PERIOD (posedge clk) (46.5))
```



NOCHANGE Keyword

The NOCHANGE keyword specifies a signal constraint relative to the width of a clock pulse. You can use this construct to model the timing constraints of memory devices, for example, when address lines must remain stable during a write pulse. The syntax is as follows:

```
(NOCHANGE clk_event data_event start_offset end_offset)
```

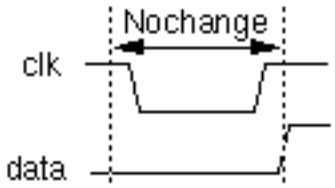
The following table describes the arguments of the NOCHANGE keyword.

Keyword Argument	Description
clk_event	Clock event.
data_event	Data event.
start_offset	Start edge event.
end_offset	End edge event.

The following example defines a period beginning at 4.5 time units before the negative clk edge and ending at 4.5 time units after the subsequent positive clk edge. Both clock and data events can be edge-triggered and conditional. During this time period, the data signal must not change.

```
(INSTANCE x)
(TIMINGCHECK
```

```
          (NOCHANGE (negedge clk) data (4.5) (4.5))
)
```



TIMINGENV Keyword and Constructs

The TIMINGENV keyword and its constructs are ignored by Verilog-XL and Verifault-XL. However, the following syntax is included for completeness. Specifying the TIMINGENV keyword constructs will not generate an error message. The TIMINGENV keyword associates constraint values with critical paths in the design and provides information about the timing environment in which the circuit operates. The syntax is as follows:

```
(TIMINGENV
     {(PATHCONSTRAINT {"path_name"} start_path {int_path+} end_path max_rise
     max_fall)}
     {(PERIODCONSTRAINT edge_clk max_period {(EXCEPTION (INSTANCE path+))}
     {(SKEWCONSTRAINT port_path max_skew)}
     {(SUM (start_path end_path) {(start_path end_path)+} (start_end_sum)
     {(start_end_sum)+})}
     {(DIFF (start_path end_path) {(start_path end_path)+} (start_end_diff)
     {(start_end_diff)+})}
     {(ARRIVAL {(edge port)} port_name early_rise late_rise early_fall late_fall)}
     {(DEPARTURE {(edge port)} port_name early_rise late_rise early_fall
     late_fall)}
     {(SLACK port rise_setup fall_setup rise_hold fall_hold {clk_period} )
     {(WAVEFORM port wave_period (edge num1 {num2}) + (edge num1 {num2})+ )
     )
```

Constraint entries provide information about the timing properties that a design is required to have to meet certain design objectives. A tool that is synthesizing some aspect of the design (logic synthesis, layout, and so on) will.

PATHCONSTRAINT Keyword

The PATHCONSTRAINT keyword specifies the maximum allowable delay for a path, which is typically identified by the two ports at each end of the path. Path constraints are the critical paths in a design identified during timing analysis. You can also specify intermediate ports to

uniquely identify path(s). Layout tools use these constraints to direct the physical design. The syntax is as follows:

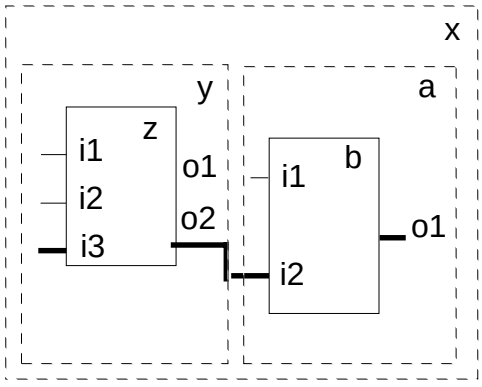
```
(PATHCONSTRAINT {"path_name"} start_path {int_path+} end_path max_rise max_fall)
```

The following table describes the arguments of the PATHCONSTRAINT keyword.

Keyword Argument	Description
path_name	Optional symbolic or actual name of the path.
start_path	Start of port path. Where applicable, a port path can have array index (for example, x.y[3].p).
int_path	Intermediate points to describe the path. You can have multiple *int_path* arguments. Where applicable, a port path can have array index (for example, x.y[3].p).
end_path	End of port path. Where applicable, a port path can have array index (for example, x.y[3].p).
max_rise	Maximum rise delay between the start and end path points.
max_fall	Maximum fall delay between the start and end path points.

The following example shows how to use the PATHCONSTRAINT keyword:

```
(INSTANCE x)
     (TIMINGENV (PATHCONSTRAINT y.z.i3 y.z.o2 a.b.o1 (25.1) (15.6)))
```



The following example shows a cell entry specifying a path constraint.

```
(CELL (CELLTYPE “DFF”)
    (INSTANCE a.b.c)
    (TIMINGCHECK
```

```
      (PATHCONSTRAINT y.z.i3 a.b.o1 (25)(15))
    )
)
```

PERIODCONSTRAINT Keyword

The PERIODCONSTRAINT keyword specifies the maximum time allowable for one complete cycle of the signal. The syntax is as follow:

```
(PERIODCONSTRAINT edge_clk max_period {(EXCEPTION (INSTANCE path+))})
```

The following table describes the arguments of the PERIODCONSTRAINT keyword.

Keyword Argument	Description
edge_clk	Edge-triggered clock event.
max_period	Maximum period for complete signal cycle.
EXCEPTION	A list of one or more cell instances to be excluded from the group.
INSTANCE	Cell instance.

Note: You must specify the PERIODCONSTRAINT keyword at a code-hierarchy level that includes the cell instance that drives the common clock inputs of the flip-flops and any cell instances to be placed in the *exception* list.

The following example shows how to use the PERIODCONSTRAINT keyword.

```
(INSTANCE x)
(TIMINGENV
    (PERIODCONSTRAINT bufa.y (10)
        (EXCEPTION (INSTANCE dff3))
    )
```

SKEWCONSTRAINT Keyword

The SKEWCONSTRAINT keyword specifies the clock event signal which is constrained against the associated port signals. For example, in a chain of devices such as counters that are connected to the same clock signal, the clock ideally changes at exactly the same time at all counters. However, there is some delay between the change at one device and the change at another. If this delay exceeds the signal skew limit, the data passed along the chain is unreliable. The syntax is as follows:

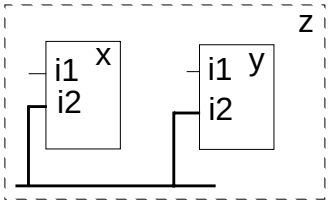
```
(SKEWCONSTRAINT port_path max_skew)
```

The following table describes the arguments of the SKEWCONSTRAINT keyword.

Keyword Argument	Description
port_path	Port that drives the net. Where applicable, a port path can have array index (for example, x.y[3].p).
max_skew	Maximum skew between signals is identified as a design constraint by timing analysis tools. This information can be passed through the SDF file to layout tools to ensure that the physical design is laid out within these constraints.

The following example shows how to use the SKEWCONSTRAINT keyword:

```
(INSTANCE z)
(TIMINGENV (SKEWCONSTRAINT (posedge i2) (7.5)))
```



SUM Keyword

The SUM keyword specifies the maximum sum of two or more path delays. The syntax is as follows:

```
(SUM (start_path end_path) {(start_path end_path)+}
    (start_end_sum) {(start_end_sum)+}
)
```

The following table describes the arguments of the SUM keyword.

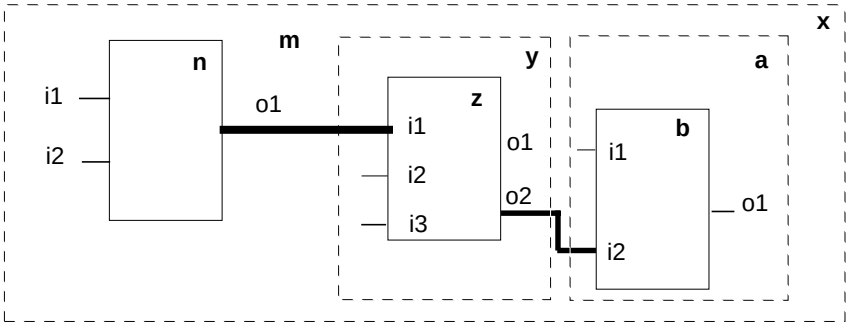
Keyword Argument	Description
start_path	Start of port path. Where applicable, a port path can have array index (for example, x.y[3].p).
end_path	End of port path. Where applicable, a port path can have array index (for example, x.y[3].p).

Keyword Argument	Description
start_end_sum	Sum of the individual delays associated with each start and end port pair. The sum of all port pair delays must be less than the value of *start_end_path*.

Note: You can specify additional paths by using additional pairs of *start_path* and *end_path* arguments with corresponding *start_end_sum* arguments.

The following example shows how to use the SUM keyword:

```
(INSTANCE x)
(TIMINGENV
    (SUM (m.n.o1 y.z.i1) (y.z.o2 a.b.i2) (67.3))
)
```



DIFF Keyword

The DIFF keyword specifies the maximum allowable difference between the delays of two paths in a design. The syntax is as follows:

```
(DIFF (start_path end_path) {(start_path end_path)+}
     (start_end_diff) {(start_end_diff)+}
)
```

The following table describes the arguments of the DIFF keyword.

Keyword Argument	Description
start_path	Start of port path. Where applicable, a port path can have array index (for example, x.y[3].p).
end_path	End of port path. Where applicable, a port path can have array index (for example, x.y[3].p).

Keyword Argument	Description
start_end_diff	Maximum difference between two path delays. The difference of the individual delays in the two circuit paths must be less than the value of *start_end_diff*.

Note: You can specify additional paths by using additional pairs of *start_path* and *end_path* arguments with corresponding *start_end_diff* arguments.

The following example shows how to use the DIFF keyword:

```
(INSTANCE x)
(TIMINGCHECK
    (DIFF (m.n.o1 y.z.i1) (y.z.o2 a.b.i2) (8.3))
)
```

ARRIVAL Keyword

The ARRIVAL keyword specifies the time when a primary input signal is applied during the intended circuit operation. You use this keyword to analyze the timing behavior for a circuit and to compute logic constraints for logic synthesis and layout. The syntax is as follows:

```
(ARRIVAL {(edge port)} port_name early_rise late_rise early_fall late_fall)
```

The following table describes the arguments of the ARRIVAL keyword.

Keyword Argument	Description
edge	Either posedge or negedge.
port	The input port from which the time reference is specified. This is the required primary input signal is a fan-out from a sequential element (usually an active edge of a clock signal).
port_name	The input or inout port for which the arrival time is to be defined. This port must be a primary (external) *input* of the top-level module.
early_rise	Earliest rise value relative to the time reference. This value must be less than the *late_rise* value.
late_rise	Latest rise value relative to the time reference. This value must be greater than the *early_rise* value.
early_fall	Earliest fall value relative to the time reference. This value must be less than the *late_fall* value.

Keyword Argument	Description
late_fall	Latest fall value relative to the time reference. This value must be greater than the *early_fall* value.

The following example shows how to use the ARRIVAL keyword. It applies rising transitions at D[15:0] no sooner than 10 and no later than 40 time units after the rising edge of the reference time MCLK. It applies falling transitions no sooner than 12 and no later than 45 time units after the edge.

```
(INSTANCE top)
(TIMINGENV
    (ARRIVAL (posedge MCLK) D[15:0] (10) (40) (12) (45))
)
```

DEPARTURE Keyword

The DEPARTURE keyword specifies the time when a primary output signal is applied during the intended circuit operation. You use this keyword to analyze the timing behavior for a circuit and to compute logic constraints for logic synthesis and layout. The syntax is as follows:

```
(DEPARTURE {(edge port)} port_name early_rise late_rise early_fall late_fall)
```

The following table describes the arguments of the DEPARTURE keyword.

Keyword Argument	Description
edge	Either posedge or negedge.
port	The input port from which the time reference is specified. This is the required primary input signal is a fan-out from a sequential element (usually an active edge of a clock signal).
port_name	The output or inout port for which the departure time is to be defined. This port must be a primary (external) *output* of the top-level module.
early_rise	Earliest rise value relative to the time reference. This value must be less than the *late_rise* value.
late_rise	Latest rise value relative to the time reference. This value must be greater than the *early_rise* value.
early_fall	Earliest fall value relative to the time reference. This value must be less than the *late_fall* value.

Keyword Argument	Description
late_fall	Latest fall value relative to the time reference. This value must be greater than the *early_fall* value.

The following example shows how to use the DEPARTURE keyword. It applies rising transitions at A[15:0] no sooner than 8 and no later than 20 time units after the rising edge of the reference time SCLK. It applies falling transitions no sooner than 12 and no later than 34 time units after the edge.

```
(INSTANCE top)
(TIMINGENV
    (DEPARTURE (posedge SCLK) D[15:0] (8) (20) (12) (34))
)
```

SLACK Keyword

The SLACK keyword compares the calculated delay over a path to the delay constraints imposed upon the path and determines the available slack or margin in the delay path. Positive slack indicates that the constraints are met with additional time units to spare. Negative slack indicates a failure to construct a circuit according to the constraints. The syntax is as follows:

```
(SLACK port rise_setup fall_setup rise_hold fall_hold {clk_period})
```

The following table describes the arguments of the SLACK keyword.

Keyword Argument	Description
port	Input port that provides the slack or margin information.
rise_setup	Additional rise delay that can be tolerated in all paths ending at the *port* without causing the design constraint to be violated.
fall_setup	Additional fall delay that can be tolerated in all paths ending at the *port* without causing the design constraint to be violated.
rise_hold	Reduction of rise delay that can be tolerated in all paths ending at the *port* without causing the design constraint to be violated.
fall_hold	Reduction of fall delay that can be tolerated in all paths ending at the *port* without causing the design constraint to be violated.
clk_period	Optionally represents the clock period on which the slack or margin values are based. The clock period is specified by the WAVEFORM keyword construct.

Note: You can specify multiple SLACK keyword constructs for the same port and are distinct as long as the value of *clk_period* is different.

The following example shows that the signal arrives at port macro.AOI6.B in time to meet the setup time requirement of a flip-flop down the path with 3 time units to spare. Therefore, the delay of any and all paths leading to port macro.AOI6.B can be increased by an additional 3 time units without violating a setup requirement. The example also shows that the delay of any data paths leading to port macro.AOI6.B can be decreased by 7 time units without violating a hold requirement.

```
(CELL
    (CELLTYPE "cpu"
    (INSTANCE macro.AOI6)
    (TIMINGENV
        (SLACK B (3) (3) (7) (7))
    )
)
```

WAVEFORM Keyword

The WAVEFORM keyword specifies a periodic waveform that is applied to a circuit during its intended operation. Typically, you use this to define a clock signal. You use this keyword to analyze the timing behavior for a circuit and to compute logic constraints for logic synthesis and layout. The syntaxes are as follows:

```
(WAVEFORM port wave_period (posedge num1 {num2})
     (negedge num1 {num2}) )

(WAVEFORM port wave_period (negedge num1 {num2})
     (posedge num1 {num2}) )
```

Note: Specifying posedge or negedge first determines the direction of the transition.

The following table describes the arguments of the WAVEFORM keyword.

Keyword Argument	Description
port	Input or inout port where the waveform will appear.
wave_period	Number that specifies the waveform which repeats indefinitely at the interval.
num1	When specified alone, defines the transition offset. When specified with *num2*, defines the beginning of the uncertainty region in which the transition takes place.

Keyword Argument	Description
num2	Defines the end of the uncertainty region in which the transition takes place.

If the port is not a primary input of the circuit, (if it is driven by the output of some other circuit element in the scope of the analysis), then the signal driven in the circuit should be ignored and the specified waveform should replace it in the analysis.

The following example shows the specification of a waveform of period 15 to be applied to port `top.clka`. Within each period, a rising edge occurs at somewhere between 0 and 2 and a falling edge somewhere between 5 and 7.

```
(CELL     (CELLTYPE "cpu")
      (INSTANCE top)
      (TIMINGENV (WAVEFORM clka 15 (posedge 0 2) (negedge 5 7))
      )
)
```

The following example shows the specification of a waveform of period 25 to be applied to port `top.clkb`. Within each period, a falling edge occurs at 0, a rising edge at 5, a falling edge at 10, and a rising edge at 15.

```
(CELL (CELLTYPE "cpu")
    (INSTANCE top)
    (TIMINGENV (WAVEFORM clkb 25 (negedge 0) (posedge 5) (negedge 10) (posedge 15))
    )
)
```

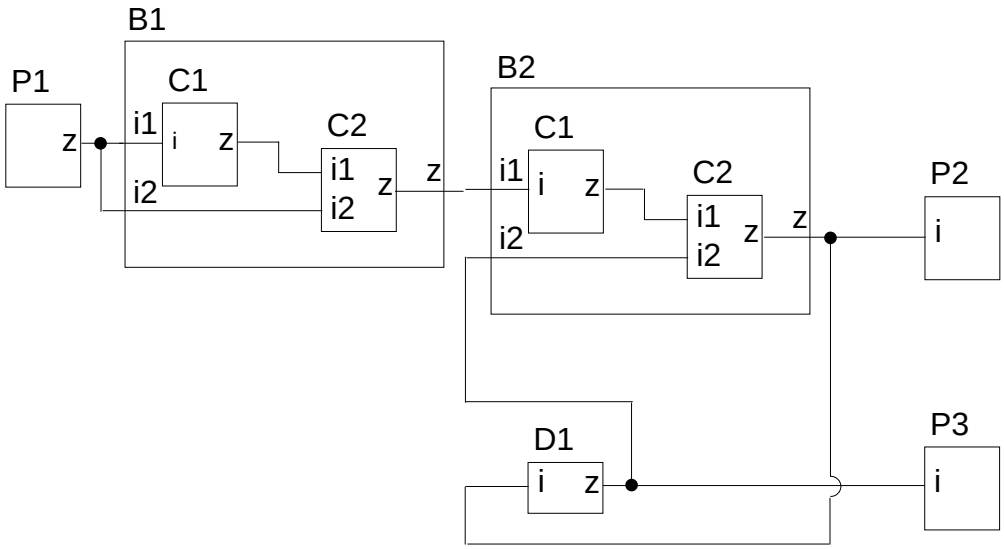
The following example shows negative numbers in defining a waveform.

```
(CELL (CELLTYPE "cpu")
    (INSTANCE top)
    (TIMINGENV (WAVEFORM clkb 50 (negedge -10) (posedge 20))
    )
)
```

SDF File Examples

Example 1

The SDF file example that follows is based on the following schematic
.




```
(DELAYFILE
    (SDFVERSION "2.1")
    (DESIGN "system")
    (DATE "Saturday December 14 08:30:33 PST 1996")
    (VENDOR "Cadence")
    (PROGRAM "delay_calc")
    (VERSION "1.5")
    (DIVIDER /)
    (VOLTAGE 4.5:5.0:5.5)
    (PROCESS "worst")
    (TEMPERATURE 55:85:125)
    (TIMESCALE 1ns)

(CELL (CELLTYPE "system") (INSTANCE block_1)
    (DELAY (ABSOLUTE
            (INTERCONNECT P1/z B1/C1/i (.145::.145) (.125::.125))
            (INTERCONNECT P1/z B1/C2/i2 (.135::.135) (.130::.130))
            (INTERCONNECT B1/C1/z B1/C2/i1 (.095::.095) (.095::.095))
            (INTERCONNECT B1/C2/z B2/C1/i (.145::.145) (.125::.125))
            (INTERCONNECT B2/C1/z B2/C2/i1 (.075::.075) (.075::.075))
            (INTERCONNECT B2/C2/z P2/i (.055::.055) (.075::.075))
            (INTERCONNECT B2/C2/z D1/i (.255::.255) (.275::.275))
            (INTERCONNECT D1/z B2/C2/i2 (.155::.155) (.175::.175))
            (INTERCONNECT D1/z P3/i (.155::.155) (.130::.130)))))
```

```
(CELL (CELLTYPE "INV") (INSTANCE B1/C1)
    (DELAY (ABSOLUTE
            (IOPATH i  z (.345::.345) (.325::.325)))))
(CELL (CELLTYPE "OR2") (INSTANCE B1/C2)
    (DELAY (ABSOLUTE
            (IOPATH i1  z (.300::.300) (.325::.325))
            (IOPATH i2  z (.300::.300) (.325::.325)))))
(CELL (CELLTYPE "INV") (INSTANCE B2/C1)
    (DELAY (ABSOLUTE
            (IOPATH i  z (.345::.345) (.325::.325)))))
(CELL (CELLTYPE "AND2") (INSTANCE B2/C2)
    (DELAY (ABSOLUTE
            (IOPATH i1  z (.300::.300) (.325::.325))
            (IOPATH i2  z (.300::.300) (.325::.325)))))
(CELL (CELLTYPE "INV") (INSTANCE D1)
    (DELAY (ABSOLUTE
            (IOPATH i  z (.380::.380) (.380::.380)))))
) // end delayfile
```

Example 2

This example shows how you can use the `COND` construct with the `IOPATH` and `TIMINGCHECK` constructs.

```
(DELAYFILE
    (SDFVERSION "2.0")
    (DESIGN "top")
    (DATE "Nov 12, 1996 11:30:10")
    (VENDOR "Cool New Tools")
    (PROGRAM "Delay Obfuscator")
    (VERSION "v1.0")
    (DIVIDER .)
    (VOLTAGE :5:)
    (PROCESS "typical")
    (TEMPERATURE :25:)
    (TIMESCALE 1ns)
    (CELL (CELLTYPE     "CDS_GEN_FD_P_SD_RB_SB_NO")
        (INSTANCE     top.ff1)
        (DELAY
            (ABSOLUTE (COND (TE == 0 && RB == 1 && SB == 1)
                (IOPATH (posedge CP) Q (2:2:2) (3:3:3))))
            (ABSOLUTE (COND (TE == 0 && RB == 1 && SB == 1)
                (IOPATH (posedge CP) QN (4:4:4) (5:5:5))))
            (ABSOLUTE (COND (TE == 1 && RB == 1 && SB == 1)
                (IOPATH (posedge CP) Q (6:6:6) (7:7:7))))
            (ABSOLUTE (COND (TE == 1 && RB == 1 && SB == 1)
                (IOPATH (posedge CP) QN (8:8:8) (9:9:9))))
            (ABSOLUTE
                (IOPATH (negedge RB) Q (1:1:1) (1:1:1)))
            (ABSOLUTE
                (IOPATH (negedge RB) QN (1:1:1) (1:1:1)))
            (ABSOLUTE
                (IOPATH (negedge SB) Q (1:1:1) (1:1:1)))
            (ABSOLUTE
                (IOPATH (negedge SB) QN (1:1:1) (1:1:1)))
        ) // end delay
```

```
        (DELAY
            (ABSOLUTE (PORT D (0:0:0) (0:0:0) (5:5:5)))
            (ABSOLUTE (PORT CP (0:0:0) (0:0:0) (0:0:0)))
            (ABSOLUTE (PORT RB (0:0:0) (0:0:0) (0:0:0)))
            (ABSOLUTE (PORT SB (0:0:0) (0:0:0) (0:0:0)))
            (ABSOLUTE (PORT TI (0:0:0) (0:0:0) (0:0:0)))
            (ABSOLUTE (PORT TE (0:0:0) (0:0:0) (0:0:0)))
        ) // end delay
        (TIMINGCHECK
            (COND D_ENABLE (SETUP D (posedge CP) (1:1:1)))
            (COND D_ENABLE (HOLD D (posedge CP) (1:1:1)))
            (COND TI_ENABLE (SETUPHOLD TI (posedge CP)) (1:1:1)
                (1:1:1))
            (COND ENABLE (WIDTH (posedge CP) (1:1:1)))
            (COND ENABLE (WIDTH (negedge CP) (1:1:1)))
            (WIDTH (negedge SB) (1:1:1))
            (WIDTH (negedge RB) (1:1:1))
            (COND SB (RECOVERY (posedge RB) (negedge CP) (1:1:1)))
            (COND RB (RECOVERY (posedge SB) (negedge CP) (1:1:1)))
        ) // end timingcheck
    ) // end cell
) // end delayfile
```

Example 3

This example shows how State Dependent Path Delays (SDPDs) can be annotated using `COND` and `IOPATH` constructs.

```
(DELAYFILE
    (SDFVERSION "2.0")
    (DESIGN "top")
    (DATE "May 12, 1996 17:25:18")
    (VENDOR "Slick Trick Systems")
    (PROGRAM "Viability Tester")
    (VERSION "v3.0")
    (DIVIDER .)
    (VOLTAGE :5:) (PROCESS "typical") (TEMPERATURE :25:)
    (TIMESCALE 1ns)
    (CELL (CELLTYPE "XOR2") (INSTANCE top.x1)
        (DELAY
            (INCREMENT (COND i1 (IOPATH i2 o1 (2:2:2) (2:2:2))))
            (INCREMENT (COND i2 (IOPATH i1 o1 (2:2:2) (2:2:2))))
            (INCREMENT (COND ~i1 (IOPATH i2 o1 (3:3:3) (3:3:3))))
            (INCREMENT (COND ~i2 (IOPATH i1 o1 (3:3:3) (3:3:3))))
        )
    )
)
```

4

Annotating with Verilog-XL and Verifault-XL

This chapter describes the following:

SDF-Specific Plus Options

Table 4-1 on page 86 shows the plus options you can use to control the SDF Annotator. The SDF-specific plus options are described in more detail later in this section. "Additional Plus Options that Control the SDF Annotator" on page 89 briefly describes the Verilog-XL and Verifault-XL plus options that control the SDF Annotator. For complete information about the plus options that control the SDF Annotator, refer to the *Verilog-XL Reference* or the *Verifault-XL Reference.*

Table 4-1 Plus Options that Control the SDF Annotator

SDF-Specific	For Verilog-XL	For Verifault-XL
+sdf_cputime +sdf_error_info +sdf_file +sdf_ign_timing_edge +sdf_nocheck_celltype +sdf_no_errors +sdf_nomsrc_int +sdf_no_warnings +sdf_splitvlog+splitrecrem +sdf_splitvlog_splitsuh +sdf_split_two_timing_check +sdf_verbose	+annotate_any_time +maxdelays +mindelays +multisource_int_delays +neg_tchk +no_pulse_int_backanno +notimingchecks +pulse_e/n and +pulse_r/m +pulse_int_e/n and +pulse_int_r/m +transport_int_delays +typdelays	+annotate_any_time +maxdelays +mindelays +notimingchecks +typdelays +vfaddtchk

See “SDF Keywords for Verilog-XL” on page 35 for the OVI SDF standard keywords that Verilog-XL uses.

See “SDF Keywords for Verifault-XL” on page 36 for the OVI SDF standard keywords that Verifault-XL uses.

+sdf_cputime

The +sdf_cputime plus option logs the number of central processing unit (CPU) seconds that it takes to complete the annotation. The CPU time is written to the log file.

+sdf_error_info

The +sdf_error_info plus option displays PLI error messages.

Note: SDF Annotator errors are classified as fatal and non-fatal errors. Fatal errors cause the SDF Annotator to stop. Non-fatal errors do not stop the SDF Annotator, but cause it to skip the action that caused the error. An example of a non-fatal error is when a condition specified in the SDF file cannot be matched in the Verilog description.

+sdf_file<filename>

The `+sdf_file` plus option with a corresponding appended file name (no space in between) specifies the SDF file that the SDF Annotator uses. This plus option overrides the file specified as an argument to the `$sdf_annotate` system task.

+sdf_ign_timing_edge

Note: This option is applicable for Verilog-XL only.

The `+sdf_ign_timing_edge` plus option annotates the last edge without any extra overheads for `SETUP, HOLD and SETUPHOLD`. By default, Verilog-XL generates an error message during annotation if the verilog file contains a timing check without an edge and the sdf file contains an edge. To facilitate annotation, you can use the `+sdf_ign_timing_edge` plus option. Consider the example given below.

Verilog File:

```
$setup(data, clk, 2);
$hold(clk, data, 1);
```

SDF File:

```
(SETUP (posedge data) clk (3))
(SETUP (negedge data) clk (3))
(HOLD  (posedge data) clk (2))
(HOLD  (negedge data) clk (2))
```

In this example, the timing check signal `data` does not contain an edge in the Verilog file but contains both a posedge and a negedge in the SDF file. Using the `+sdf_ign_timing_edge` plus option, the timing check signal `data` will first be annotated with a `posedge` and then a `negedge`. In effect, the last edge defined is annotated.

+sdf_nocheck_ celltype

The `+sdf_nocheck_celltype` plus option disables celltype validation between the SDF Annotator and the Verilog description. By default, the SDF Annotator validates the type specified in the `CELLTYPE` construct against the type of the cell instance that is specified in the `INSTANCE` keyword construct.

+sdf_no_errors

The `+sdf_noerrors` plus option disables error messages from the SDF Annotator.

+sdf_nomsrc_int

If you have no multisource interconnect transport delays (MITDs) in the design, the `+sdf_nomsrc_int` plus option increases performance and reduces memory consumption by not maintaining information about the various interconnects that map to the same port. If you have multiple interconnects in the design that map to the same input port, the SDF Annotator must resolve these delays using a resolution function prior to annotating the port. The SDF Annotator provides three resolution functions (`AVERAGE`, `MAXIMUM`, and `MINIMUM`). For the SDF Annotator to correctly resolve the delays, it must maintain the interconnect information until the end of annotation.

+sdf_no_warnings

The `+sdf_no_warnings` plus option disables warning messages from the SDF Annotator.

+sdf_split_two_timing_check
+sdf_splitvlog_splitsuh
+sdf_splitvlog_splitrecrem

Note: These options are applicable for Verilog-XL only.

SDF Annotator attempts to match the one-timing checks (`SETUP`, `HOLD`, `REMOVAL`, and `RECOVERY`) to their corresponding one-timing checks in the Verilog source. If no match is found, then the SDF annotator splits the two-timing checks (`$setuphold` and `$recrem`) in the Verilog source into corresponding one-timing checks and attempts to match. For example, `$setuphold` is split into `$setup` and `$hold` and then matched to `SETUP` and `HOLD`.

You can split SDF two-timing checks (`SETUPHOLD` and `RECREM`) using the `+sdf_split_two_timing_check` plus option into their corresponding one-timing checks. The conditions specified with `SETUPHOLD` and `RECREM` are ignored after the split.

If you have used `+sdf_split_two_timing_check` plus option and no two-timing checks are found, the SDF Annotator reports errors in terms of corresponding split timing checks.

The options `+sdf_splitvlog_splitsuh` and `+sdf_splitvlog_splitrecrem` can be used to perform splitting of `SETUPHOLD` only and `RECREM` only respectively.

+sdf_verbose

The `+sdf_verbose` plus option writes the following detailed information about the annotation process to the SDF Annotator's log file:

- Annotated delays
- Configuration information about the annotator
- Assumptions made during annotation
- Warnings or errors due to inconsistencies found during annotation

(4-1) The SDF Annotator also prints warning and error messages to standard output.

Additional Plus Options that Control the SDF Annotator

The following table briefly describes the plus options that control the SDF Annotator. For complete information about these plus options, see the *Verilog-XL Reference* and the *Verifault-XL Reference*. See the "SDF-Specific Plus Options" on page 85 for information about SDF-specific plus options.

Plus Option	Description
`+annotate_any_time`	Allows SDF backannotation to occur at times other than time 0.
`+maxdelays`	Selects the maximum delay.
`+mindelays`	Selects the minimum delay.
`+multisource_int_delays`	(Verilog-XL only) Affecting only nets with more than one source, provides transport delays with full pulse control and the ability to specify unique source/load delays. MIPDs are inserted on all single-source nets. Using the `+multisource_int_delays` plus option with the `+transport_int_delays` plus option provides transport delays with full pulse control for interconnect delays with one or more sources and unique source/load delays for such nets.

Plus Option	Description
`+neg_tchk`	(Verilog-XL only) The `+neg_tchk` plus option enables negative timing check arguments in the `$recrem` and `$setuphold` timing checks. When you do not use the `+neg_tchk` plus option, any limits that are negative, either in the description or annotation, are set to 0, and a warning is issued. You can specify negative time arguments for `$recrem` only if you are specifying values for both the *<recovery_limit>* and *<removal_limit>* arguments. When either the *<recovery_limit>* or *<removal_limit>* argument is negative, the sum of the two limits must be 0 or greater.
`+no_pulse_int_backanno`	(Verilog-XL only) Prevents PLI annotation of pulse limits for interconnect delays. Only one warning message is issued on the first attempt.
`+notimingchecks`	Disables all timing checks. When you disable timing checks, processing speed improves and the circuit data structure requires less memory. For Verifault-XL, this is set by default. **Note:** Module path delays remain active.
`+pulse_e/n and +pulse_r/m`	(Verilog-XL only) The `+pulse_e/n` and `+pulse_r/m` plus options control the way in which pulse rejection and pulse error limits are annotated to module paths. If you do not specify values for these limits in an `IOPATH` construct, the SDF Annotator applies the percentages specified in these plus options to calculate a pulse reject and error limit to annotate. Also, if you do not specify these plus options, then the SDF Annotator defaults to 100% for both the reject and error percentages. See the *Verilog-XL Reference* for more information about pulse rejection for module paths.

Plus Option	Description
`+pulse_int_e/n and +pulse_int_r/m`	(Verilog-XL only)The `+pulse_int_e/n` and `+pulse_int_r/m` plus options control the way in which pulse rejection and pulse error limits are annotated to interconnects. If you do not specify values for these limits in an `INTERCONNECT` construct, the SDF Annotator applies the percentages specified with these plus options to calculate a pulse reject and error limit to annotate. Also, if you do not specify these plus options, then the SDF Annotator defaults to 100% for both the reject and error percentages. See the *Verilog-XL Reference* for more information about pulse rejection for interconnects.
`+transport_int_delays`	(Verilog-XL only) Provides transport delays with full pulse control for interconnect delays with one or more sources. Using the `+transport_int_delays` plus option with `+multisource_int_delays`, provides transport delays with full pulse control for interconnect delays with one or more sources and unique source/load delays.
`+typdelays`	Selects typical delays.
`+vfaddtchk`	(Verifault-XL only) Enables timing checks.

Improving SDF Annotator Performance and Memory Use

Annotation is an important and time consuming process for simulating designs. Because many annotation operations are design dependant, you can optimize the annotation process to achieve maximum performance The SDF Annotator Version 2.0 has significantly improved its performance from the previous version. The first table summarizes some of the

performance improvements. The first table shows the sample specifications. The second table describes the performance improvements

Design Description	SDF File Description
193,343 Gates 90,873 Module Instances 51,669 Primitive Instances	78.2 MB 1,303,258 lines 331,253 IOPATHs 250,228 Interconnects 237,603 Timing Checks

Performance Improvements	CPU secs	Incremental Speedup	Memory
SDF Annotator 1.7	8262.9	--	355.3 MB
SDF Annotator 2.0	2395.5	345%	19.6 MB
`+sdf_nocheck_celltype`	2223.3	7%	--
`+sdf_nomsrc_int`	1510.8	37%	5.9 MB

Note: The information described in this section is deliberately broad and generic. Requirements for your specific design may dictate procedures slightly different from those described here.

Removing Module Mapping

If your design does not require module mapping, you can achieve a significant improvement in annotation performance by removing `MODULE` keyword constructs.

You can map timing constructs that do not correspond exactly to the design specification. However, for every cell encountered, the SDF Annotator determines if the cell is mapped and if it is, performs a second check to determine the timing construct to which timing information is mapped. In the configuration file, annotation information is retained in memory by the SDF Annotator in an internal data structure. The retained information can get large enough to cause swapping and affect performance.

Disabling Multisource Interconnect Timing Resolution

You should use the `+sdf_nomsrc_int` plus option only when you do not need to resolve between multiple timing specifications. If you know that multisource interconnects do not exist

in the design and are using MIPDs to annotate interconnect delays, you can significantly improve performance by using the `+sdf_nomsrc_int` plus option.

An interconnect delay in the SDF file can be mapped to a MIPD, SITD, or MITD. When a multisource interconnect delay is mapped to a MIPD, the SDF Annotator resolves it using average, maximum, and minimum values. The annotator retains interconnect information in cache memory to annotate timing information into the design after processing the SDF file. The `+sdf_nomsrc_int` plus option disables the retention of interconnect information.

Using Pre-scaled Delays

You can specify scaling operations for the timing information in the SDF file before the delays are annotated into Verilog-XL. However, you can improve performance if your design contains prescaled delays. Because the SDF Annotator must perform floating point operations for each delay value, scaling can affect the performance of the SDF Annotator.

Synchronizing Time Scales

When the time scales for the Verilog-XL description and the SDF file are the same, you can improve performance. If the time scales in the SDF File and the Verilog description are different, the SDF Annotator performs a multiplication operation for each delay to obtain the correct scaling.

Synchronizing Precision

When the precision of the time (degree of accuracy) for the Verilog-XL description and the SDF file are the same, you can improve performance. If the precision is different, the SDF Annotator performs rounding operations.

Disabling Cell Type Verification

The `CELLTYPE` keyword specifies the type of cell that contains the timing information in the SDF file. The SDF Annotator verifies that a type of cell in the SDF file corresponds to the type of cell in the Verilog-XL description. If there are many cells, the verification can be time consuming. The `+sdf_nocheck_celltype` plus option disables this verification.

Note: You should use the `+sdf_nocheck_celltype` plus option only when you are confident that the syntax of the SDF file is correct.

This check is useful when one of the cells does not match, because the SDF Annotator will not attempt to annotate any of the delays in the invalid cell

Processing Without Verbose Annotation

The `+sdf_verbose` plus option generates a detailed log about the annotated delays. Annotating at this level of detail is I/O intensive and may generate a large log file, requiring more disk space. Use the `+sdf_verbose` plus option only when necessary.

Using (INSTANCE *)

You can specify similar delays for a certain type of module using the `(INSTANCE *)` construct. The SDF Annotator annotates the delays that follow `(INSTANCE *)` to all the instances of the particular `CELLTYPE`. This is especially effective in a design with a large gate count but with only a few unique cells.

Depending on the delay calculation algorithm, sometimes cell-specific delays (such as timing checks and path delays) are equal for all instances of the cell. If this is the case, using `(INSTANCE *)` is very effective because this annotation algorithm has been optimized in the SDF Annotator.

Grouping Redundant Constructs

When you can group all delays of a certain type under one keyword, you reduce the parsing of the SDF file, improving performance. For example, all timing checks should be specified in the same block. The `ABSOLUTE`, `INCREMENT`, `DELAY`, `TIMINGCHECK` and `TIMINGENV` keywords pertain to a subsequent set of timing constructs. These keywords should be sparingly used.

Removing Zero-Delay MIPDs, MITDs, and SITDs

You can improve performance by removing interconnect or port delays that have a value of 0 from the SDF file. The SDF Annotator parses and interprets zero-delay timing information but does not annotate it. By removing the zero-delay information from the SDF file, you eliminate unnecessary processing of this information.

Note: This performance improvement recommendation applies only to MIPDs, MITDs, and SITDs.

Working with Verilog-XL SDF Annotator Restrictions

The following restrictions apply between Verilog-XL and the SDF Annotator.

Reverting to Original Timing Limitation

You cannot revert to the original timing information after you perform annotation. The SDF Annotator does not retain information that distinguishes annotated timing information from the original timing information in the Verilog HDL source description.

To revert to original timing, you must exit Verilog-XL and compile the design again. If all of the original timing came from an SDF file, you can annotate from that SDF file to restore the original timing.

PATHPULSE Limitation for Interconnect Delays

You cannot use `PATHPULSE` to annotate interconnect delays. The SDF Annotator uses the `PATHPULSE` information from the SDF file if the Verilog HDL description contains module path delay timing specifications. For information about the `PATHPULSE` keyword, see "PATHPULSE Keyword" on page 58.

COND Keyword Matching Condition Restriction

Whenever you specify a condition using the `COND` keyword for an `IOPATH` or `TIMINGCHECK` construct in an SDF file, the target path or timing check in Verilog-XL must have a matching condition for the data to be annotated.

When an `IOPATH` or `TIMINGCHECK` is specified in an SDF file without any conditions, and a corresponding path or timing check exists in Verilog-XL, annotation is done regardless of whether the target path or timing check has a condition specified. For information about the `COND` keyword, see "COND Keyword" on page 48. For `IOPATH`, see "IOPATH Keyword" on page 45. For `TIMINGCHECK`, see "TIMINGCHECK Keyword and Constructs" on page 60.

TIMESCALE Keyword Restriction in SDF File Header

The SDF Annotator uses and converts the timescale information in the SDF file header to the timescales specified in the Verilog HDL source description before it annotates this information.

The default Verilog-XL timescale is in seconds. The default SDF timescale is in nanoseconds (ns). If the `TIMESCALE` specification in the HDL source description is seconds, and if the SDF timescale is below this resolution, the SDF delays are annotated as 0.

Edge Identifier Limitations

Verilog-XL does not support some of the edge identifiers in the SDF file. Verilog-XL only supports the `posedge` and `negedge` edge identifiers for path delays. If an edge identifier other than `posedge` or `negedge` (01, 10, 0z, z1, 1z, z0) is specified in an SDF file, the SDF Annotator issues a warning and does not annotate that path delay to Verilog-XL.

An edge identifier for a timing check event in the SDF file must have a matching edge identifier for the target timing check event in Verilog-XL.

A positive edge identifier (`posedge`) in the SDF file maps to any rising edge identifiers (01, 0x, x1) in Verilog-XL. A negative edge identifier (`negedge`) in the SDF file maps to any falling edge identifiers (10, 1x, x0). Also, SDF does not support an edge *`(01,0x)`* specification in Verilog-XL.

When a timing check event in the SDF file is specified without an edge identifier, and a corresponding timing check event exists in Verilog-XL, annotation occurs regardless of the edge specified for the target event.

Multiple Delay Data Limitations

The SDF Annotator interprets different numbers of delays in the following manner, where each delay can be a *`min:typ:max`* triplet:

One delay	Same delay for all transitions
Two delays	Rise and fall delays
Three delays	Rise, fall, and turn-off delays (all transitions to and from Z
Six delays	Rise, fall, 0 to Z, Z to 1, 1 to Z, and Z to 0 delays (only relevant for delays mapped to Verilog HDL module path delays and to transport interconnect delays)

When you specify a delay with a single value, the SDF Annotator propagates the value to *`min:typ:max`*. For example, `(2)` is propagated to `(2:2:2)`.

During annotation to Verilog-XL and Verifault-XL, only one value from each *`min:typ:max`* triplet is annotated to the tool. Veritime uses all three values from a *`min:typ:max`* triplet.

When you specify only rise and fall delay values, the SDF Annotator maps the following:

rise maps to X1, Z1, 0X, 0Z	max(rise, fall) maps to XZ

fall maps to X0, Z0, 1X, 1Z	min(rise, fall) maps to ZX

When you specify only rise, fall, and turn off delay values, the SDF Annotator maps the following:

rise maps to Z1	fall maps to Z0
0Z maps to 1Z	0X maps to min(rise, 0Z)
ZX maps to 0Z	1X maps to min(fall, 0Z)
X0 maps to fall	XZ maps to 0Z
X1 maps to rise	

When you specify only the rise value, the SDF Annotator maps to all six delay values. If you do not want the SDF Annotator to map the delay values, use placeholders.

Escape Identifier Restrictions

Identifiers that are not supported by Verilog-XL are escaped by placing a backslash (\) before the identifier and a blank space after the identifier. Characters in identifiers that are not supported in the SDF file are escaped by placing a backslash before each character. Instance identifiers that start with numbers must be escaped. The SDF Annotator maps an identifier with unsupported characters in SDF to a Verilog Family tool by removing each backslash within the identifier and then escaping the entire identifier according to the Verilog name-escaping convention. For example, `mem_array\[01:1023].\(m1\.\)` in SDF maps to `\mem_array[01:1023] .\(m1.)` in Verilog.

A

SDF Annotator Error and Warning Messages

This appendix describes the following:

Error Messages

Error Message	Reason
Condition cannot be matched for IOPATH, skipping annotation	The condition specified in `(COND c (IOPATH))` construct cannot be matched to any SDPD condition in the given instance.
Could not annotate INTERCONNECT to cell driver *`port-name`* due to lack of driver in module	An `INTERCONNECT` is annotated to an output source port that has no drivers in the cell.
Could not find path *`input-path-name`* to *`output-path-name`* in instance *`instance-name`*	You specified a path that does not exist in the `IOPATH` construct.
Failed to find `HOLD` timingcheck	You specified a timing check that does not exist in a `HOLD` construct.
Failed to find `PERIOD` timingcheck	You specified a timing check that does not exist in a `PERIOD` construct.
Failed to find `RECOVERY` timingcheck	You specified a timing check that does not exist in a `RECOVERY` construct.
Failed to find `SETUP` timingcheck	You specified a timing check that does not exist in a `SETUP` construct.
Failed to find `SETUPHOLD` timingcheck	You specified a timing check that does not exist in a `SETUPHOLD` construct.
Failed to find `SKEW` timingcheck	You specified a timing check that does not exist in a `SKEW` construct.
Failed to find `WIDTH` timingcheck	You specified a timing check that does not exist in a `WIDTH` construct.
`INSTANCE` * specified with no `CELLTYPE`	There is no corresponding `CELLTYPE` to a `CELL` that contains the `(INSTANCE *)` construct.
Output port *`port-name`* encountered in `PORT`	You specified an output port in a `PORT` construct.

Type of INSTANCE *instance-name* (*instance-type*) does not match CELLTYPE *cell-type*	The cell type in the CELLTYPE construct that does not match the type of the cell in the INSTANCE construct.
Unable to annotate NETDELAY *net-name* due to lack of driver	A NETDELAY is annotated to an output source port that has no drivers in the cell.
Unable to find cells of CELLTYPE *cell-type*	There are no instances of the type specified in the corresponding CELLTYPE entry when the (INSTANCE *) construct is specified.
Unable to find PATHPULSE path input-port-name to output-port-name in instance instance-name	You specified a path that does not exist in the PATHPULSE, GLOBALPATHPULSE, or PATHPULSEPERCENT construct.
Unable to find input port *port-name*	You specified an input port that does not exist in the instance of an IOPATH, PATHPULSE, GLOBALPATHPULSE, or PATHPULSEPERCENT construct.
Unable to find instance *instance-name*	You specified the name of an instance that does not exist in the INSTANCE construct.
Unable to find net *net-name*	You specified a net that does not exist in the instance of a NETDELAY construct.
Unable to find output port *port-name*	You specified an output port that does not exist in the instance of an IOPATH, PATHPULSE, GLOBALPATHPULSE, or PATHPULSEPERCENT construct.
Unable to find port *port-name*	You specified a port that does not exist in the instance of a DEVICE or PORT construct.

Warning Messages

Warning Message	**Reason**
Annotating INTERCONNECT to cell driver	The SDF Annotator attempted to annotate an INTERCONNECT delay onto the cell that is driving the source port.

Annotating NETDELAY to cell driver	The SDF Annotator attempted to annotate a `NETDELAY` onto the cell that is driving the net.
Illegal limit specified, setting to 0	You specified a negative limit for a `SETUPHOLD` or `RECOVERY` construct and did not specify `+neg_tchk` on the command line. This message occurs also when the sum of the two limits to a `SETUPHOLD` or `RECOVERY` construct was less than 0.
Timescale not specified in SDF file, defaulting to 1ns	You did not specify a `TIMESCALE` construct in the header section of the SDF file.

B

Valid and Invalid Interconnect Combinations

This appendix describes the following:

Overview

This appendix lists the valid combinations of source and destination ports and the manner in which the SDF Annotator performs the action for each valid combination. When annotating *interconnect* delays, only certain combinations of ports are allowed by the SDF Annotator. In addition, certain combinations of ports may be handled in different manners.

The following sections describe the valid and invalid interconnect combinations.

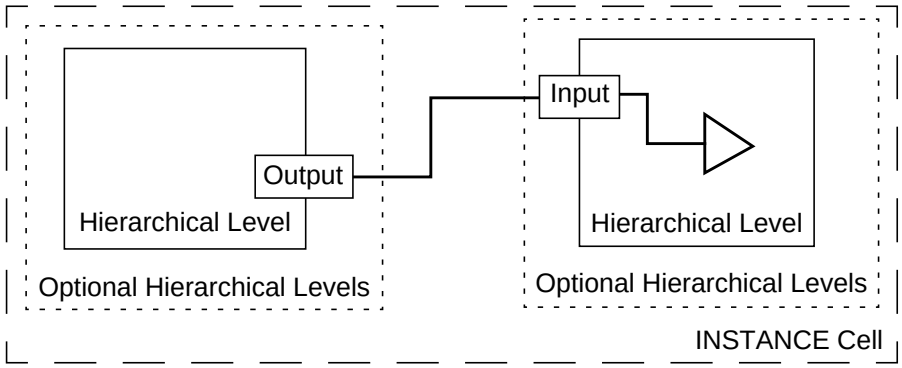
Valid Interconnect Combinations

The following table shows the valid interconnect combinations.

Source Port	Destination Port	Fanout	Drivers	Schematic
Output (lower)	Input (lower)	Yes	Single	Figure B-1
Output (lower)	Input (lower)	Yes	Multiple	Figure B-2
Output (lower)	Input (lower)	No	Any number	Figure B-3
Output (lower)	Inout (lower)	Yes	Single	Figure B-4
Output (lower)	Inout (lower)	No	Multiple	Figure B-5

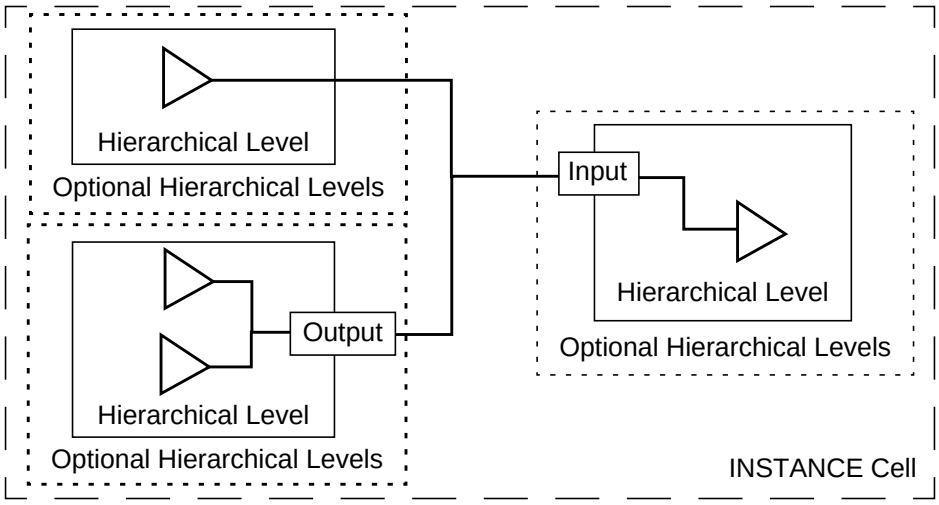
Source Port	Destination Port	Fanout	Drivers	Schematic
Output (lower)	Inout (lower)	No	Any number	Figure B-6
Output (lower)	Inout (same)	No	Any number	Figure B-7
Output (lower)	Output (same)	No	Any number	Figure B-8
Inout (lower)	Input (lower)	Yes	Single	Figure B-9
Inout (lower)	Input (lower)	Yes	Multiple	Figure B-10
Inout (lower)	Input (lower)	No	Any number	Figure B-11
Inout (lower)	Inout (lower)	Yes	Single	Figure B-12
Inout (lower)	Inout (lower)	Yes	Multiple	Figure B-13
Inout (lower)	Inout (lower)	No	Any number	Figure B-14
Inout (lower)	Inout (same)	No	Any number	Figure B-15
Inout (lower)	Output (same)	No	Any number	Figure B-16
Input (same)	Input (lower)	Yes	Single or none	Figure B-17
Input (same)	Input (lower)	Yes	Multiple	Figure B-18
Input (same)	Inout (lower)	Yes	Single or none	Figure B-19
Input (same)	Inout (lower)	Yes	Multiple	Figure B-20
Inout (same)	Input (lower)	Yes	Single or none	Figure B-21
Inout (same)	Input (lower)	Yes	Multiple	Figure B-22
Inout (same)	Inout (lower)	Yes	Single or none	Figure B-23
Inout (same)	Inout (lower)	Yes	Multiple	Figure B-24

Figure B-1 Output (lower) -> Input (lower) with XL Fanout; Single Driver



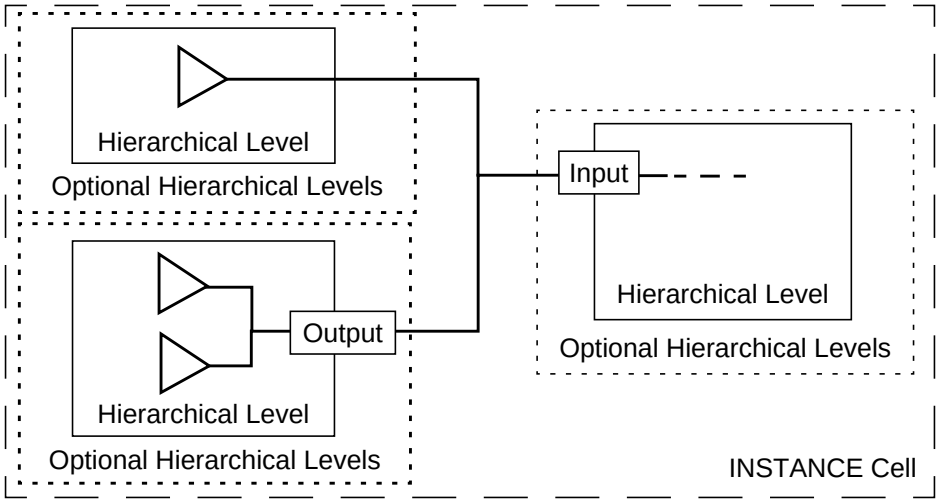
A MIPD is placed on the destination port. However, if you specify the `+transport_int_delays` plus option, then SITD is placed on the net.

Figure B-2 Output (lower) -> Input (lower) with XL Fanout; Multiple Drivers



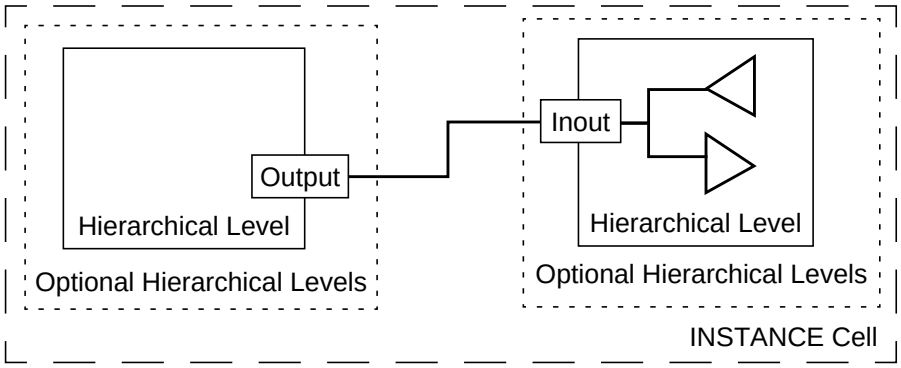
A MIPD is placed on the destination port unless you specify the `+multisource_int_delays` or `+transport_int_delays` plus option. If you specify the `+multisource_int_delays` plus option and all destination loads are accelerated, then a MITD is placed on the net and the delay is annotated from all sources in the source module to the destination. If you specify the `+transport_int_delays` plus option, a SITD is placed on the destination port.

Figure B-3 Output (lower) -> Input (lower); Any Number of Drivers



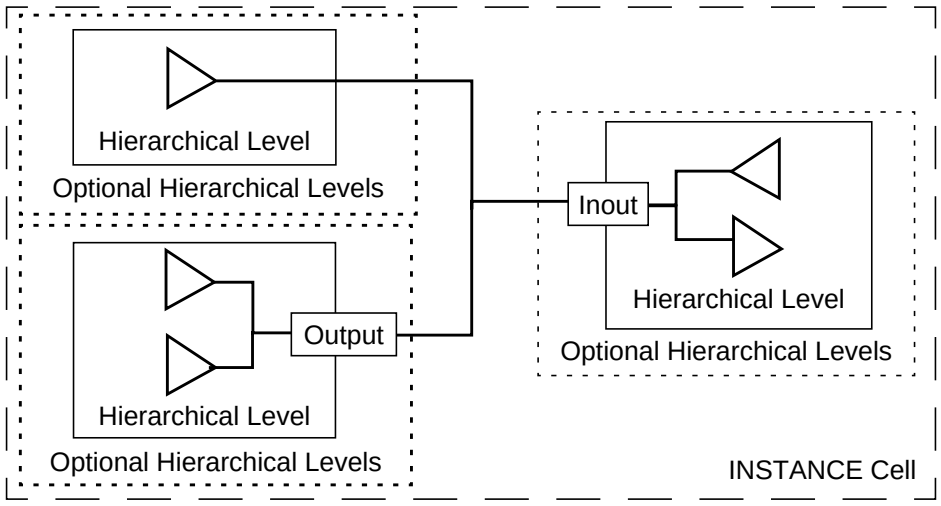
The delay is annotated onto the cell output corresponding to the source port, which is either a path delay driving the port or all gates driving the port.

Figure B-4 Output (lower) -> Inout (lower) with XL Fanout; Single Driver



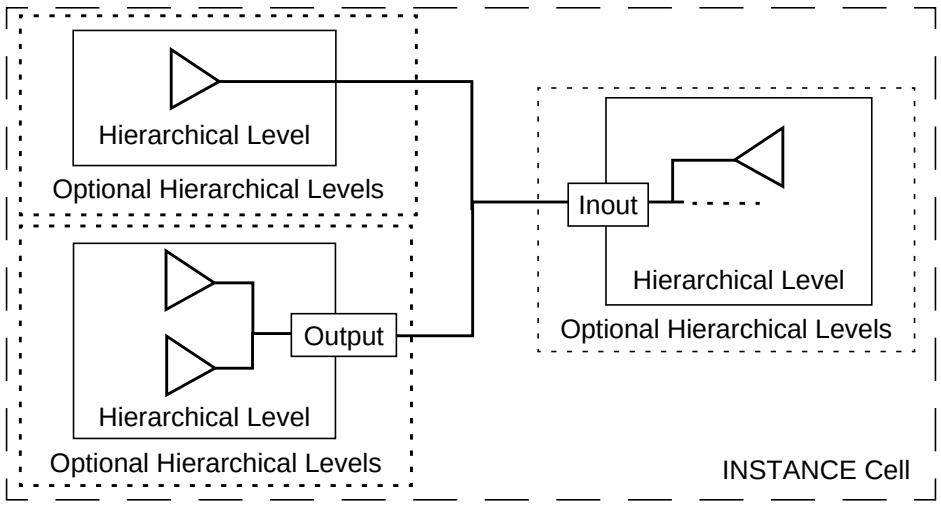
A MIPD is placed on the destination port. However, if you specify the `+transport_int_delays` plus option, then SITD is placed on the net.

Figure B-5 Output (lower) -> Inout (lower); Multiple Drivers



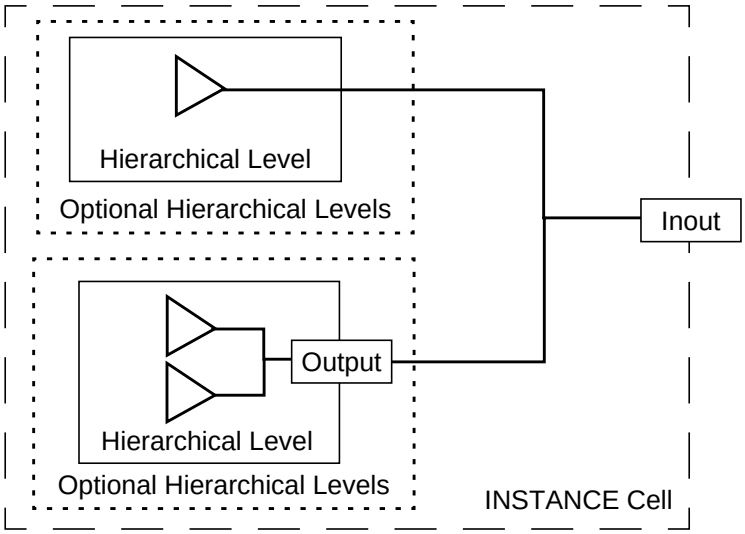
A MIPD is placed on the destination port unless you specify the `+multisource_int_delays` or `+transport_int_delays` plus option. If you specify the `+multisource_int_delays` plus option and all destination loads are accelerated, then a MITD is placed on the net and the delay is annotated from all sources in the source module to the destination. If you specify the `+transport_int_delays` plus option, a SITD is placed on the destination port.

Figure B-6 Output (lower) -> Inout (lower); Any Number of Drivers



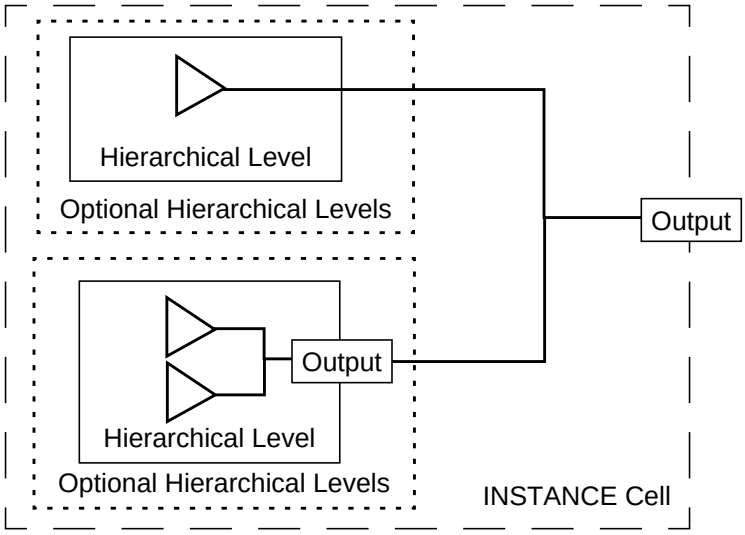
The delay is annotated onto the cell output corresponding to the source port, which is either a path delay driving the port or all gates driving the port.

Figure B-7 Output (lower) -> Inout (same); Any Number of Drivers



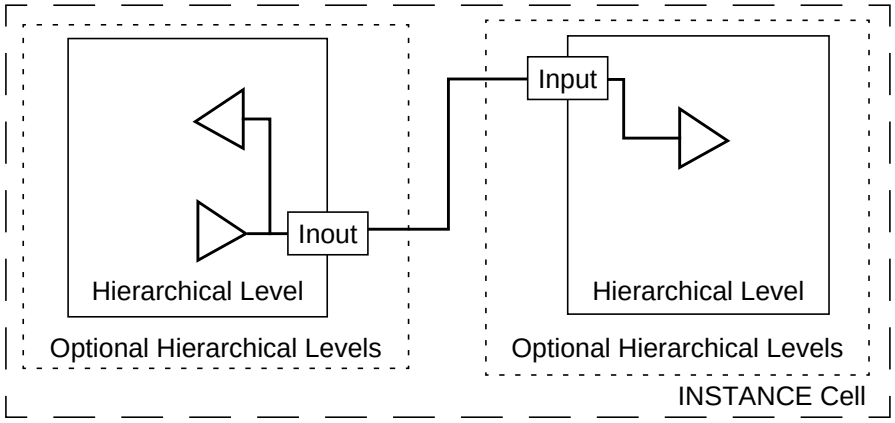
The delay is annotated onto the cell output corresponding to the source port, which is either a path delay driving the port or all gates driving the port.

Figure B-8 Output (lower) -> Output (same); Any Number of Drivers



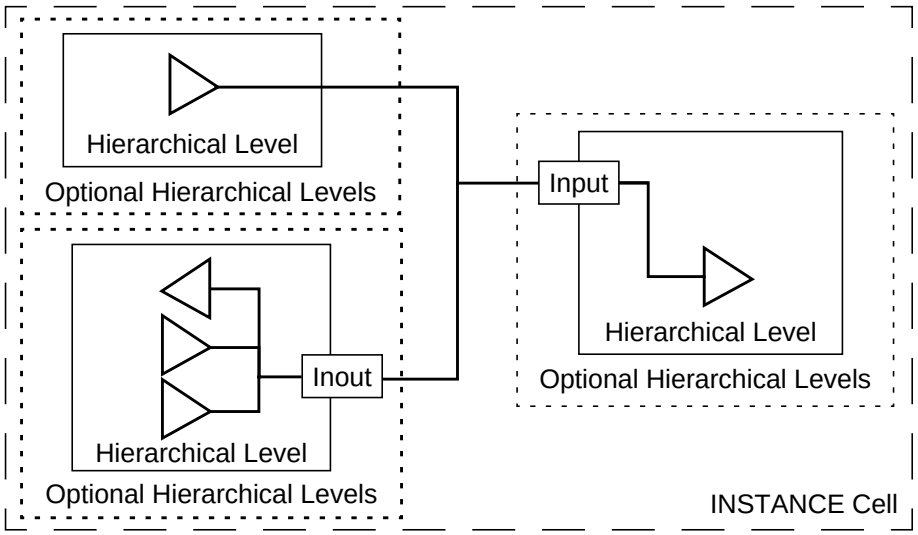
The delay is annotated onto the cell output corresponding to the source port, which is either a path delay driving the port or all gates driving the port.

Figure B-9 Inout (lower) -> Input (lower) with XL Fanout; Single Driver



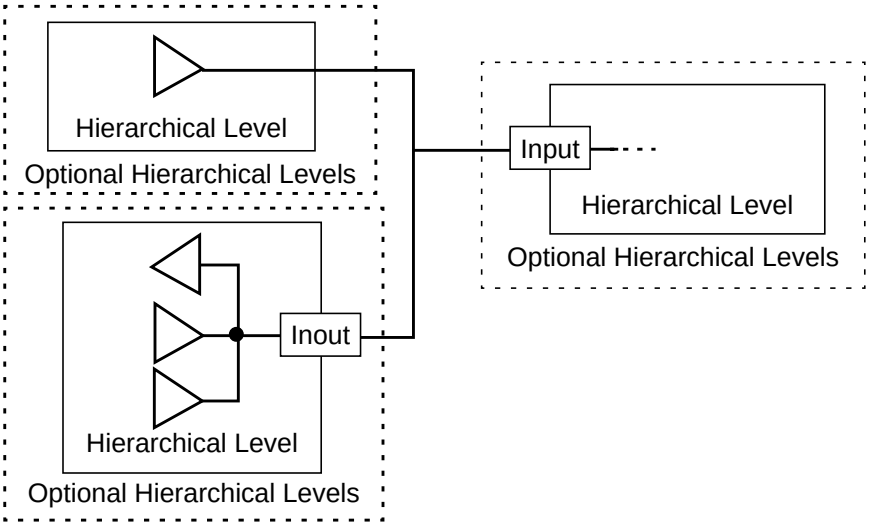
A MIPD is placed on the destination port. However, if you specify the `+transport_int_delays` plus option, then SITD is placed on the net.

Figure B-10 Inout (lower) -> Input (lower) with XL Fanout; Multiple Drivers



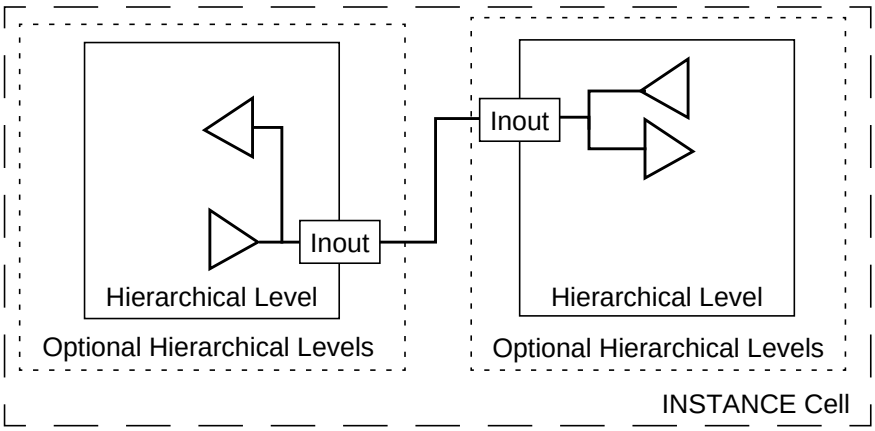
A MIPD is placed on the destination port unless you specify the `+multisource_int_delays` or `+transport_int_delays` plus option. If you specify the `+multisource_int_delays` plus option and all destination loads are accelerated, then a MITD is placed on the net and the delay is annotated from all sources in the source module to the destination. If you specify the `+transport_int_delays` plus option, a SITD is placed on the destination port.

Figure B-11 Inout (lower) -> Input (lower); Any Number of Drivers



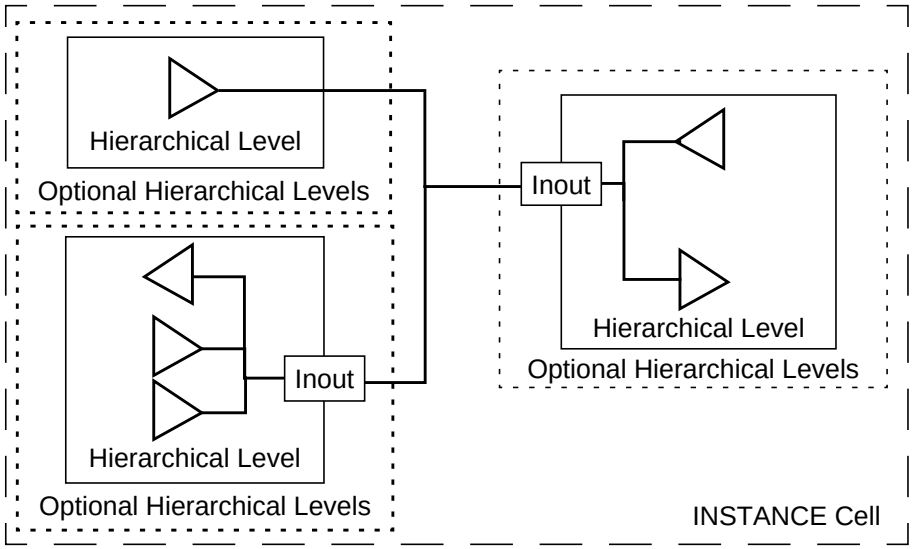
The delay is annotated onto the cell output corresponding to the source port, which is either a path delay driving the port or all gates driving the port.

Figure B-12 Inout (lower) -> Inout (lower) with XL Fanout; Single Driver



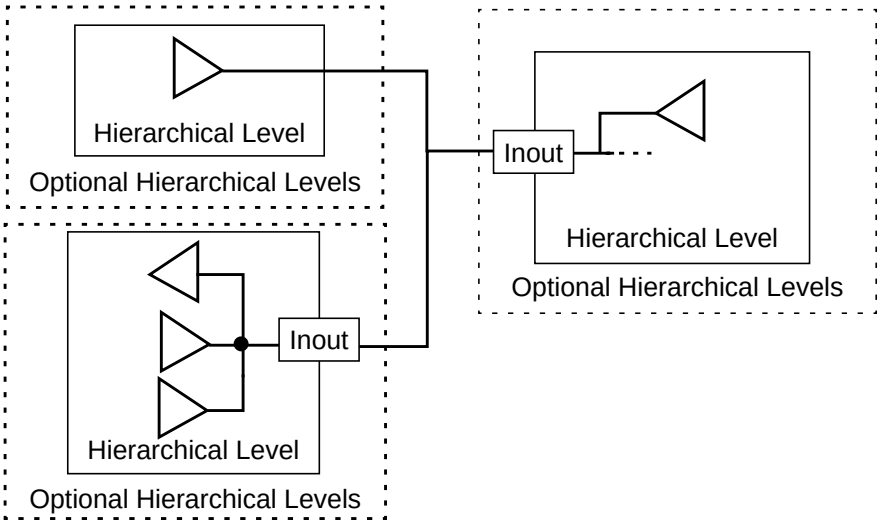
A MIPD is placed on the destination port. However, if you specify the `+transport_int_delays` plus option, then SITD is placed on the net.

Figure B-13 Inout (lower) -> Inout (lower) with XL Fanout; Multiple Drivers



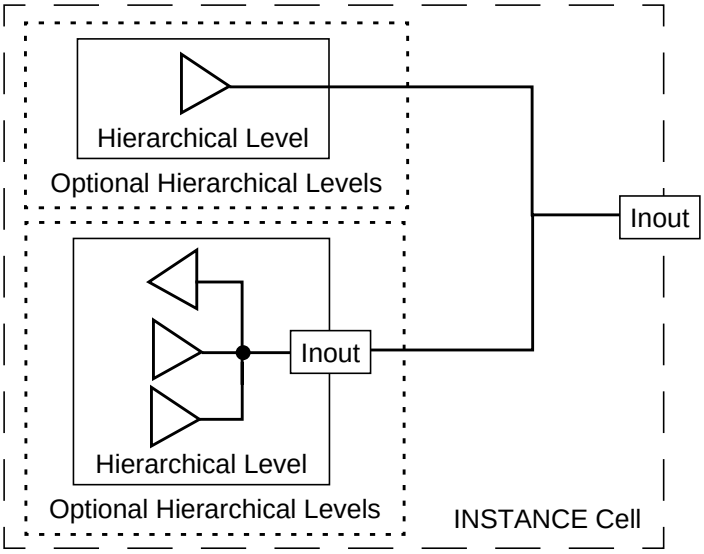
A MIPD is placed on the destination port unless you specify the `+multisource_int_delays` or `+transport_int_delays` plus option. If you specify the `+multisource_int_delays` plus option and all destination loads are accelerated, then a MITD is placed on the net and the delay is annotated from all sources in the source module to the destination. If you specify the `+transport_int_delays` plus option, a SITD is placed on the destination port.

Figure B-14 Inout (lower) -> Inout (lower); Any Number of Drivers



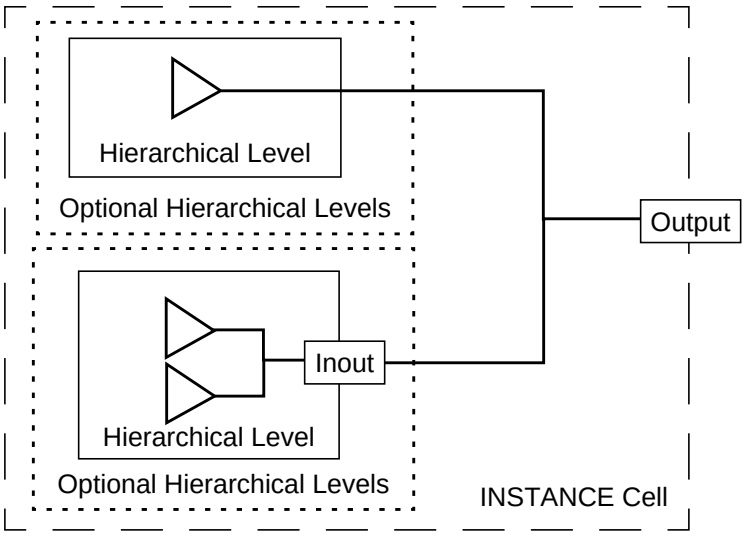
The delay is annotated onto the cell output corresponding to the source port, which is either a path delay driving the port or all gates driving the port.

Figure B-15 Inout (lower) -> Inout (same); Any Number of Drivers



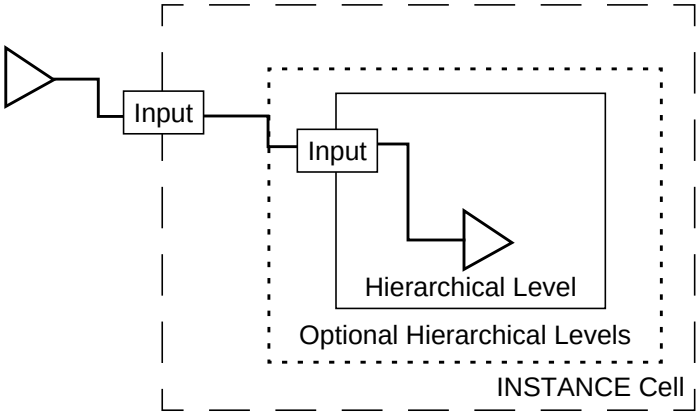
The delay is annotated onto the cell output corresponding to the source port, which is either a path delay driving the port or all gates driving the port.

Figure B-16 Inout (lower) -> Output (same); Any Number of Drivers



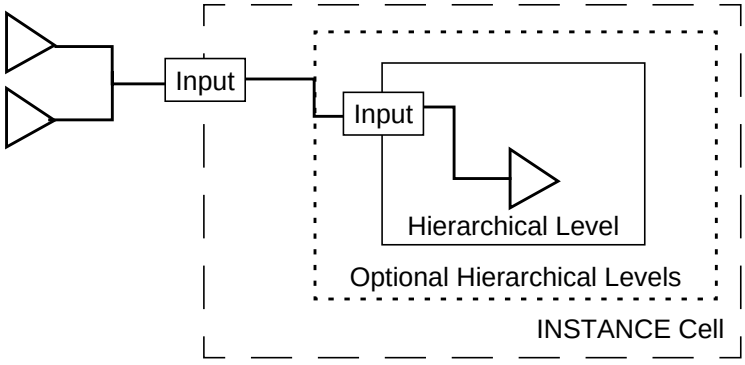
The delay is annotated onto the cell output corresponding to the source port, which is either a path delay driving the port or all gates driving the port.

Figure B-17 Input (same) -> Input (lower) with XL Fanout; Single or No Driver



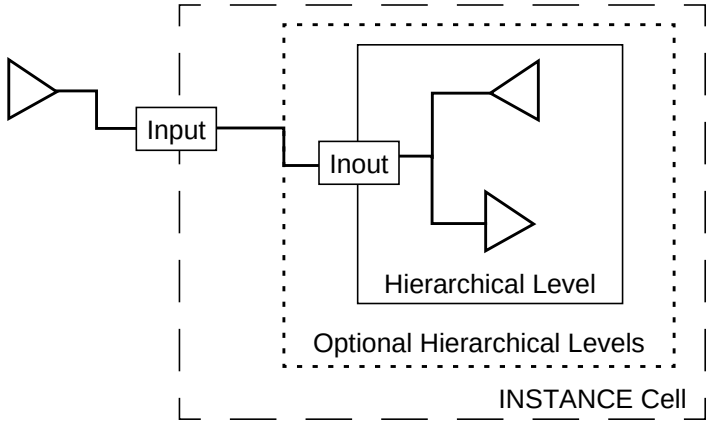
A MIPD is placed on the destination port. However, if you specify the `+transport_int_delays` plus option, then SITD is placed on the net.

Figure B-18 Input (same) -> Input (lower) with XL Fanout; Multiple Drivers



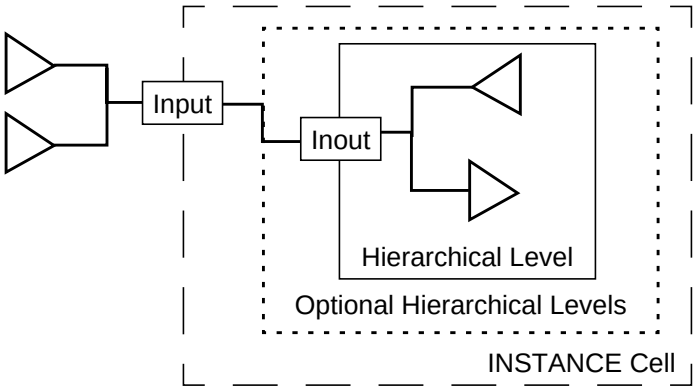
A MIPD is placed on the destination port. However, if you specify the `+multisource_int_delays` plus option a MITD is placed on the destination port.

Figure B-19 Input (same) -> Inout (lower) with XL Fanout; Single or No Drivers



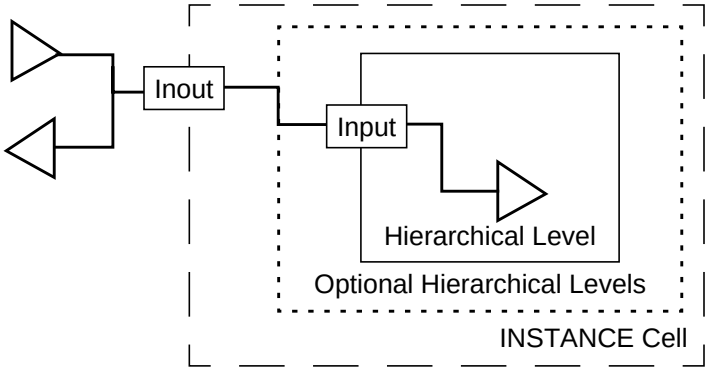
A MIPD is placed on the destination port. However, if you specify the `+transport_int_delays` plus option, then SITD is placed on the net.

Figure B-20 Input (same) -> Inout (lower) with XL Fanout; Multiple Drivers



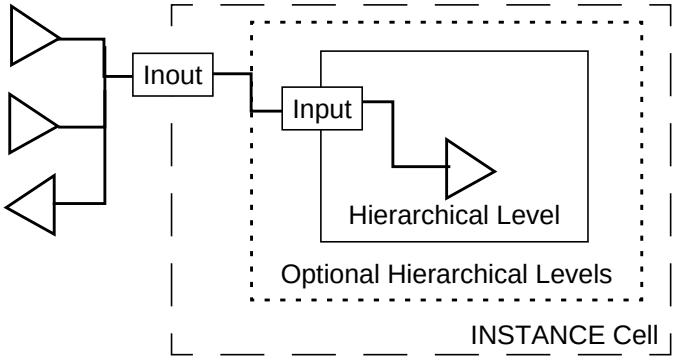
A MIPD is placed on the destination port. However, if you specify the `+multisource_int_delays` plus option a MITD is placed on the destination port.

Figure B-21 Inout (same) -> Input (lower) with XL Fanout; Single or No Driver



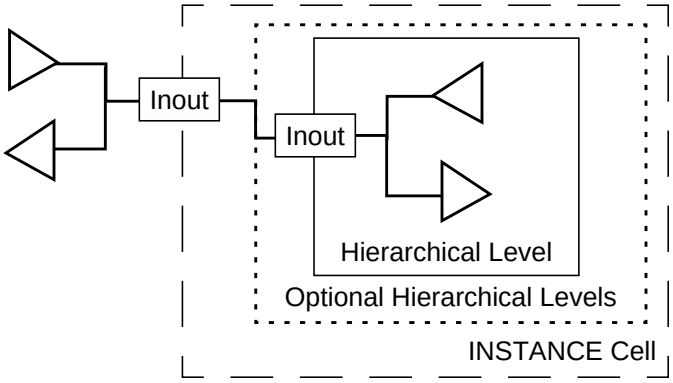
A MIPD is placed on the destination port. However, if you specify the `+transport_int_delays` plus option, then SITD is placed on the net.

Figure B-22 Inout (same) -> Input (lower) with XL Fanout; Multiple Drivers



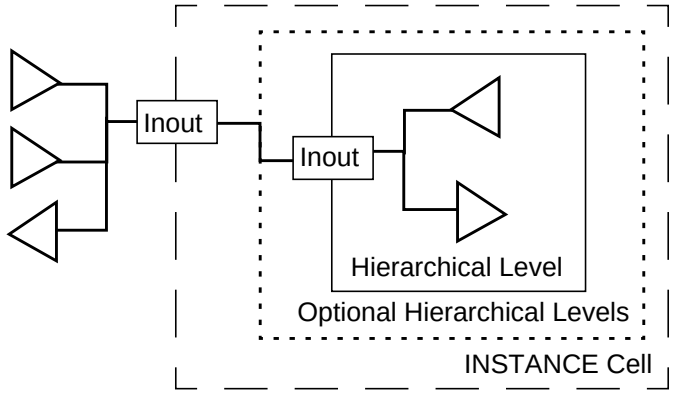
A MIPD is placed on the destination port. However, if you specify the `+multisource_int_delays` plus option a MITD is placed on the destination port.

Figure B-23 Inout (same) -> Inout (lower) with XL Fanout; Single or No Driver



A MIPD is placed on the destination port. However, if you specify the `+transport_int_delays` plus option, then SITD is placed on the net.

Figure B-24 Inout (same) -> Inout (lower) with XL Fanout; Multiple Drivers



A MIPD is placed on the destination port. However, if you specify the `+multisource_int_delays` plus option a MITD is placed on the destination port.

Invalid Interconnect Combinations

The following table shows the combinations that are illegal or unsupported.

Source Port	Driver	Destination Port	Fanout
Output (lower)	Single	Output (lower)	Yes and No

Source Port	Driver	Destination Port	Fanout
Output (lower)	Multiple	Output (lower)	Yes and No
Output (same)	Single	Output (lower)	Yes and No
Output (same)	Single	Output (same)	Yes and No
Output (same)	Single	Input (lower)	Yes and No
Output (same)	Single	Input (same)	Yes and No
Output (same)	Single	Inout (lower)	Yes and No
Output (same)	Single	Inout (same)	Yes and No
Output (same)	Multiple	Output (lower)	Yes and No
Output (same)	Multiple	Output (same)	Yes and No
Output (same)	Multiple	Input (lower)	Yes and No
Output (same)	Multiple	Input (same)	Yes and No
Output (same)	Multiple	Inout (lower)	Yes and No
Output (same)	Multiple	Inout (same)	Yes and No
Input (lower)	Single	Output (lower)	Yes and No
Input (lower)	Single	Output (same)	Yes and No
Input (lower)	Single	Input (lower)	Yes and No
Input (lower)	Single	Input (same)	Yes and No
Input (lower)	Single	Inout (lower)	Yes and No
Input (lower)	Single	Inout (same)	Yes and No
Input (lower)	Multiple	Output (lower)	Yes and No
Input (lower)	Multiple	Output (same)	Yes and No
Input (lower)	Multiple	Input (lower)	Yes and No
Input (lower)	Multiple	Input (same)	Yes and No
Input (lower)	Multiple	Inout (lower)	Yes and No
Input (lower)	Multiple	Inout (same)	Yes and No
Input (same)	Single	Output (lower)	Yes and No
Input (same)	Single	Output (same)	Yes and No

Source Port	Driver	Destination Port	Fanout
Input (same)	Single	Input (lower)	No
Input (same)	Single	Input (same)	Yes and No
Input (same)	Single	Inout (same)	Yes and No
Input (same)	Multiple	Output (lower)	Yes and No
Input (same)	Multiple	Output (same)	Yes and No
Input (same)	Multiple	Input (lower)	No
Input (same)	Multiple	Input (same)	Yes and No
Input (same)	Multiple	Inout (same)	Yes and No
Inout (lower)	Single	Output (lower)	Yes and No
Input (same)	Single	Input (same)	Yes and No
Input (same)	Multiple	Output (lower)	Yes and No
Input (same)	Multiple	Input (same)	Yes and No
Inout (same)	Single	Output (lower)	Yes and No
Inout (same)	Single	Output (same)	Yes and No
Inout (same)	Single	Input (lower)	No
Inout (same)	Single	Input (same)	Yes and No
Inout (same)	Single	Inout (lower)	No
Inout (same)	Single	Inout (same)	Yes and No
Inout (same)	Multiple	Output (lower)	Yes and No
Inout (same)	Multiple	Output (same)	Yes and No
Inout (same)	Multiple	Input (lower)	No
Inout (same)	Multiple	Input (same)	Yes and No
Inout (same)	Multiple	Inout (lower)	No
Inout (same)	Multiple	Inout (same)	Yes and No

Index

Symbols

A

B

C

D

E

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Z