Standard Delay Format
# Contents

## Introduction

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview to SDF</td>
<td>1</td>
</tr>
<tr>
<td>General Overview</td>
<td>1</td>
</tr>
<tr>
<td>SDF in the Design Process</td>
<td>1</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>3</td>
</tr>
<tr>
<td>Version History</td>
<td>3</td>
</tr>
<tr>
<td>OVI Plans For Deletion from SDF Version 3.0</td>
<td>4</td>
</tr>
<tr>
<td>Items Unsupported by SSE</td>
<td>4</td>
</tr>
</tbody>
</table>

## SDF File

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessing SDF Files</td>
<td>5</td>
</tr>
<tr>
<td>Syntax Conventions</td>
<td>6</td>
</tr>
<tr>
<td>SDF File Characters and Conventions</td>
<td>7</td>
</tr>
<tr>
<td>SDF Characters</td>
<td>7</td>
</tr>
<tr>
<td>Lexical Tokens</td>
<td>7</td>
</tr>
<tr>
<td>Common Expressions</td>
<td>12</td>
</tr>
<tr>
<td>SDF File Content</td>
<td>15</td>
</tr>
<tr>
<td>SDF File Content Example</td>
<td>15</td>
</tr>
<tr>
<td>Header Entries</td>
<td>16</td>
</tr>
<tr>
<td>sdf_version</td>
<td>16</td>
</tr>
<tr>
<td>design_name</td>
<td>17</td>
</tr>
<tr>
<td>date</td>
<td>17</td>
</tr>
<tr>
<td>vendor</td>
<td>17</td>
</tr>
<tr>
<td>program_name</td>
<td>17</td>
</tr>
<tr>
<td>program_version</td>
<td>17</td>
</tr>
<tr>
<td>hierarchy_divider</td>
<td>17</td>
</tr>
<tr>
<td>voltage</td>
<td>17</td>
</tr>
<tr>
<td>process</td>
<td>17</td>
</tr>
<tr>
<td>temperature</td>
<td>17</td>
</tr>
<tr>
<td>time_scale</td>
<td>17</td>
</tr>
<tr>
<td>cell</td>
<td>17</td>
</tr>
<tr>
<td>Cell Entries</td>
<td>18</td>
</tr>
<tr>
<td>cell</td>
<td>18</td>
</tr>
<tr>
<td>celltype</td>
<td>18</td>
</tr>
<tr>
<td>instance</td>
<td>18</td>
</tr>
<tr>
<td>correlation</td>
<td>19</td>
</tr>
<tr>
<td>corr_factor</td>
<td>19</td>
</tr>
<tr>
<td>timing_spec</td>
<td>20</td>
</tr>
<tr>
<td>Delay Entries</td>
<td>20</td>
</tr>
<tr>
<td>timing_spec</td>
<td>21</td>
</tr>
<tr>
<td>deltype</td>
<td>21</td>
</tr>
<tr>
<td>PATHPULSE</td>
<td>22</td>
</tr>
<tr>
<td>GLOBALPATHPULSE</td>
<td>23</td>
</tr>
</tbody>
</table>
Syntax and Examples

SDF File Syntax ................................................. 38
SDF File Example 1 ............................................ 41
SDF File Example 2 ............................................ 44
SDF File Example 3 ............................................ 45
SDF File Example 4 ............................................ 46

Delay Model Recommendation

Introduction ..................................................... 47
The Delay Model ............................................. 47
Timing Objects .................................................. 47
Rules .............................................................. 48
Introduction

Overview to SDF

General Overview

Simucad Note:

SSE uses the Standard Delay Format (SDF) as defined by the Open Verilog International (OVI) Standard Delay Format (SDF) Manual version 2.0. The SDF version 2.0 manual has been converted into an on-line help file by Simucad for distribution and use with the SDF product only. For your convenience, the "$sdf_anotate" system task to invoke an SDF file has been added to the OVI SDF manual (for more information, see “Accessing SDF Files” on page 5).

OVI SDF 2.0 Manual

The Standard Delay Format (SDF) file stores the timing data generated by EDA tools for use at any stage in the design process. The data in the SDF file is represented in a tool-independent way and can include:

- Delays: module path, device, interconnect, and port.
- Timing checks: setup, hold, recovery, skew, width, and period.
- Timing constraints: path and skew.
- Incremental and absolute delays.
- Conditional and unconditional module path delays and timing checks.
- Design/instance-specific or type/library-specific data.
- Scaling, environmental, technology, and user-defined parameters.

Throughout a design process, you can use several different SDF files. Some of these files can contain prelayout timing data. Others can contain path constraint or postlayout timing data.

The name of each SDF file is determined by the EDA tool. There are no conventions for naming the SDF files.

SDF in the Design Process

By accessing the SDF file, EDA tools are assured of consistent, accurate, and up-to-date data. This means that EDA tools can use data created by other tools as input to their own processes. Sharing data in this way, layout tools can use design constraints identified during timing analysis, and simulation tools can use the postlayout delay data.

The EDA tools create, read (to update their design), and write to the SDF file. SDF file support hierarchical delay annotation. A design hierarchy might include several different ASICs (and/or cells or blocks within ASICs), each with its own SDF file, see Figure 1-1.
Figure 1-1  Multiple SDF Files in a Hierarchical Design

OVI has developed this SDF specification to enable accurate and unambiguous transfer of delay data between tools that require timing. All parties utilizing the SDF should interpret and manipulate delay data according to this specification. The specification will be provided free of charge to all interested members of OVI. ASIC Vendors and 3rd party tool suppliers that desire copies of the SDF specification should request it from the OVI headquarters. Please direct your requests to:

Lynn Horobin  Open Verilog International
Suite 408
1016 East El Camino Real
Sunnyvale, CA 94087

Tel:  (408) 776-1684
Fax:  (408) 776-0124

Open Verilog International makes no warranties whatsoever with respect to the completeness, accuracy, or applicability of the information in this document to a user’s requirements.

Open Verilog International reserves the right to make changes to the Standard Delay Format Specification at any time without notice.

Open Verilog International does not endorse any particular CAE tool that is based on the Verilog hardware description language.
Acknowledgments

The OVI Library Modeling Committee acknowledges the individual and team efforts invested in establishing this version of the SDF specification:

John Busco (Toshiba/Vertex)
Irene Chang (Mitsubishi)
Yukari Chin (Ikos)
Mark Clem (Ikos)
Shir-Shen Chang (Synopsys)
Graham Davies (Viewlogic)
Paul Duvoisin (Texas Instruments)
Brian Erickson (Compaq)
Rick Eversole (Cadence)
Bob Foglesong (Cadence)
Gary Gasper (Texas Instruments)
Vasillios Gerousis (Motorola)
Anling Hsu (Cadence)
Kevin Kranen (Synopsys)
Hector Lai (Quad Design)
Jimmy Lin (Fujitsu)
Jeff Mason (Ikos)
Mike Murray (Acuson)
Joel Paston (Chronologic)
Ying-li Ren (Altera)
Paul Traynar (Nextwave)
Charles White (Quad Design)
Susan Wong (Mitsubishi)
James Wu (Toshiba/Vertex)

Version History

The keywords, USERDEF and INCLUDE, which were in version 1.0, are no longer supported by OVI.
OVI Plans For Deletion from SDF Version 3.0

The next release of SDF, we (OVI) proposes to clean up the timing model and make it consistent for its application in ASIC design and verification. You are advised to report to your OVI LM-TSC member for disapproval. The following timing objects are planned for removal in the next release of SDF:

1. PORT delay (for more information, see PORT).
2. NETDELAY (for more information, see “NETDELAY” on page 27).
3. DEVICE delay (for more information, see “DEVICE” on page 27).

Items Unsupported by SSE

SSE does not support the following keywords for OVI SDF version 2.0:

Delay keywords:
1. NETDELAY keyword (for more information, see “NETDELAY” on page 27). (Planned for deletion from OVI SDF version 3.0)
2. PATHPULSE keyword (for more information, see “PATHPULSE” on page 22).
3. GLOBALPATHPULSE keyword (for more information, see “GLOBALPATHPULSE” on page 23).

Timing Check keywords:
1. SKEWCONSTRAINT keyword (for more information, see “SKEWCONSTRAINT” on page 37).
2. NOCHANGE keyword (for more information, see “NOCHANGE” on page 34).
3. PATHCONSTRAINT keyword (for more information, see “PATHCONSTRAINT” on page 35). (Please note: This keyword does not apply to simulation, it only applies to static timing analysis)
4. SUM keyword (for more information, see “SUM” on page 36). (Please note: This keyword does not apply to simulation, it only applies to static timing analysis)
5. DIFF keyword (for more information, see “DIFF” on page 37). (Please note: This keyword does not apply to simulation, it only applies to static timing analysis)
SDF File

Accessing SDF Files

SSE supports the Standard Delay File (SDF) format. SDF is a text file that contains the instance names and delay values necessary to back-annotate delays into a Verilog HDL description. SDF is usually generated by another tool such as a place and route tool.

The $sdf_annotate system task is used to specify the SDF file (do not input the SDF file). The format specification for the $sdf_annotate system task is:

```verbatim
$sdf_annotate("file_name", module_instance);
```

where:

- **file_name** represents any valid file path and file name specification.
- **module_instance** represents the name of the module instance. The hierarchy of this instance is used for back annotation. The names in the SDF file are relative paths to the module_instance or full paths with respect to the entire Verilog HDL description. For example, if you use the module_instance name “top.dff1” then the instance names in the SDF file are relative to “top.dff1”. If you omit module_instance, SSE uses the module containing the call to the $sdf_annotate system task as the module_instance for annotation.

For the below example and diagram, if the SDF file contained the following instance name:

```
(INSTANCE name2.name4)
(DELAY
  (ABSOLUTE
   (IOPATH IN0 OUT (2420:2420:2420) (2420:2420:2420))
```

then the $sdf_annotate system task to specify the SDF file and its relative position in the design’s hierarchy would be:

```verilog
module testbench;
  initial $sdf_annotate("filename.sdf", testbench.name1);
```

When SSE reads file “filename.sdf” to update the design, the path “testbench.name1” is concatenated with path “name2.name4” to form path “testbench.name1.name2.name4”. Signals “IN0” and “OUT” are then updated as specified in the above example.
Syntax Conventions

- **ITEM** is a lexical token. Lexical token is the last item in the syntax hierarchy.
- **item** is a keyword. Keywords are case insensitive and are shown in bold for easy identification.
- **item** is a syntax construct item.
- **item** is a syntax definition for the item.
- **||=** is an alternative syntax construct.
SDF File Characters and Conventions

The characters you can use in the SDF file, lexical conventions, common expressions, and operator definitions are described in this section.

SDF Characters

The characters you can use in the SDF file are the following:

- Syntax characters
  Are non-alphanumeric characters required by the syntax.

- Alphanumeric characters
  Are upper and lower case alphabetic characters, all the numbers, and underscore (_).

- Special characters
  Are any character other than alphanumerics and underscore (_). The following is a list of special characters: ! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [ \ ] ^ ` { | } ~

- Escape character
  Is a single backslash (\). Escaping is defined as placing a single backslash (\) before any special characters, when you include the characters in an IDENTIFIER. (See “Lexical Tokens” on page 7 for a description of an IDENTIFIER.)

- Hierarchy divider characters
  Is one of the following: the period(.), or the slash(/). Any escaped hierarchy divider character loses its meaning as a hierarchy divider.

- White space
  Use white space to separate lexical tokens. Tabs, spaces and newlines are also considered white space. Do not use the white space as a hierarchy divider character or in an IDENTIFIER.

- Comment characters
  Comments can begin with /* and end with */, or begin with // until the end of the line.

Keywords, identifiers, characters, and numbers must be delimited either by syntax characters or by white space.

Lexical Tokens

QSTRING

QSTRING is a string of any legal SDF characters and spaces, excluding tabs and newlines, enclosed by double quotes.
HCHAR

HCHAR is used with the DIVIDER keyword and can be either the period (.), or the slash (/). The default is the period (.).

NUMBER

NUMBER is a positive real number, for example: 1, 3.4, .7, 0.3, 2.4e2, 5.3e-1

RNUMBER

RNUMBER is a positive or negative real number, for example: 1, -3.4, .7, -0.3

DNUMBER

DNUMBER is a non-negative integer number, for example: +12, 23

IDENTIFIER

IDENTIFIER is a port or net depending on the syntax. Identifiers can be up to 1024 characters long.

• Characters you can use in an identifier are
  • Alphanumeric characters
    Alphabetic letters – uppercase and lowercase letters are considered different, all the numbers, and the underscore (_).
  • Hierarchy divider character
    You can use the hierarchy divider character in IDENTIFIERS without escaping it. (For more information, see “SDF Characters” on page 7.)
  • Escape character
    Each special character used as a part of an IDENTIFIER must be escaped. Examples are shown below. This escaping mechanism is different from Verilog HDL where the IDENTIFIER is escaped by placing one escape character (\) before the IDENTIFIER and a white space after the IDENTIFIER.
  • Bit specs
    Place bit specs at the end of identifiers with no separating white space. Include bit specs between square brackets ([ )and (]). If bit spec is a range, use the colon (:) to separate the range, for example, [4], [3:31] and [15:0].
  • Do not use a white space in an identifier.
• Examples of correct IDENTIFIERS
  AMUX\+BMUX
Cache_Row \#4
mem_array\[0:1023\]\(0:15\) ; From a language where square
; brackets indicates arrays
; parentheses indicates; bit specs
pipe4\-done\&enb[3] ; Unescaped square brackets
; represent a bit spec

PATH

PATH is a hierarchical IDENTIFIER or a special character (*). Do not use a white space in a PATH.

EDGE_IDENTIFIER

EDGE_IDENTIFIER posedge
negedge
01
10
0z
z1
1z
z0

SCALAR_CONSTANT

SCALAR_CONSTANT 1'b0
1'b1
1'B0
1'B1
'b0
'b1
'B0
'B1
0
1

TSVALUE

TSVALUE is 1, 10, or 100 integers numbers or is 1.0, 10.0, or 100.0 real numbers, which represent a single unit of time. unit is either us, ns, or ps for microseconds, nanoseconds, and picoseconds, respectively. The default is 1 ns.
QM
QM a literal question mark ?

CLN
CLN a literal colon :

UNARY_OPERATOR
UNARY_OPERATOR is one of the following tokens:
+ 
- 
! 
~ 
& 
~& 
| 
~| 
^ 
^~ 
~^ 

BINARY_OPERATOR
BINARY_OPERATOR is one of the following tokens:
+ 
- 
* 
/ 
% 
== 
!= 
=== 
!== 
&& 
|| 
< 
<= 
> 
>= 
&
Operator Definitions

The operator definitions for the SDF file is described in this section.

+    -    *    / arithmetic
%= modulus
>    >=    <    <= relational
!    logical negation
&&   logical and
||   logical or
==   logical equality
!=   logical inequality
===  case equality
!==  case inequality
~    bit-wise unary negation
&    bit-wise binary and
|    bit-wise binary inclusive or
^    bit-wise binary exclusive or
^~   or ^~ bit-wise binary equivalence
&    reduction unary and
~&   reduction unary nand
|    reduction unary or
~|   reduction unary nor
^    reduction unary xor
^~   or ^~ reduction unary xnor
<<   left shift
>>   right shift
The Equality Operators

a == b  a equals b, result can be unknown
a != b  a is not equal to b, result can be unknown
a === b  a equals b, including x and z
a !== b  a is not equal to b, including x and z

Precedence Rules in Descending Order

!    ~  highest precedent
*    /    %
+    -
<<    >>
<    <=    >    >=
==    !=    ===    !==
&
^    ^~
|    
&&    
||  lowest precedent

Common Expressions

conditional_port_expr ::= simple_expression
||= ( conditional_port_expr )
||= UNARY_OPERATOR ( conditional_port_expr )
||= conditional_port_expr BINARY_OPERATOR conditional_port_expr

simple_expression ::= ( simple_expression )
||= UNARY_OPERATOR ( simple_expression )
||= port
||= UNARY_OPERATOR port
||= SCALAR_CONSTANT
||= UNARY_OPERATOR SCALAR_CONSTANT
||= simple_expression QM simple_expression CLN simple_expression
||= { simple_expression concat_expression? } }
concat_expression ::= , simple_expression

name ::= port_instance
|:= net_instance

net_instance ::= net
|:= instance net

net ::= IDENTIFIER

port_spec ::= port_path
|:= port_edge

port_path ::= PATH? port

port_edge ::= ( EDGE IDENTIFIER port_path )

port ::= IDENTIFIER
|:= IDENTIFIER [ DNUMBER ]
|:= IDENTIFIER [ DNUMBER : DNUMBER ]

scalar_port ::= IDENTIFIER
|:= IDENTIFIER [ DNUMBER ]

port_instance ::= port_path
|:= instance port_path

tc_value ::= ( NUMBER )
|:= ( expression )

value ::= ( NUMBER )
|:= exp_list

exp_list ::= ( expression )
|:= ( expression ? ) ( expression )
|:= ( expression ? ) ( expression ? ) ( expression )
|:= ( expression ? ) ( expression ? ) ( expression ? ) ( expression )
|:= ( expression ) ( expression ) ( expression ) ( expression ) ( expression )
The number of expressions you can specify is limited to one through six or twelve space-separated items. These expressions correspond to 0→1, 1→0, 0→Z, Z→1, 1→Z, Z→0, 0→X, X→1, 1→X, X→0, X→Z, Z→X transitions, in that order. Insert a place holder ( ) at the location in the expression sequence where the expression for a preceding transition is null. For example, if the delay values for 0→Z and Z→1 are (0.1:0.2:0.3) and (0.4:0.5:0.6), respectively, and the values for 0→1 and 1→0 are null, the expressions would be “() () (0.1:0.2:0.3) (0.4:0.5:0.6)”. Null transition expressions following known values do not need place holders. As shown in this example, the 1→Z and Z→0 transition expressions do not need place holders ( ).

expression ::= NUMBER : NUMBER ? : NUMBER ?
||= NUMBER ? : NUMBER : NUMBER ?
||= NUMBER ? : NUMBER ? : NUMBER

Consists of either a single NUMBER or a triple of NUMBERs. These triples correspond to minimum, typical, and maximum, respectively.

At least one number and two colons (;) are required.

tc_rvalue ::= ( RNUMBER )
||= ( rexpression )

rvalue ::= ( RNUMBER )
||= rex_list

rex_list ::= ( rexpression )
||= ( rexpression ? ) ( rexpression )
||= ( rexpression ? ) ( rexpression ? ) ( rexpression )
||= ( rexpression ? ) ( rexpression ? ) ( rexpression ? ) ( rexpression )

rexpression ::= RNUMBER : RNUMBER ? : RNUMBER ?
||= RNUMBER ? : RNUMBER : RNUMBER ?
||= RNUMBER ? : RNUMBER ? : RNUMBER

Standard Delay Format SDF File
SDF File Content

The SDF files are ASCII text files. Every SDF file contains a header section followed by one or more cell entries (Figure 1-2). For each cell entry, you can specify delay and timing check types.

- The header entries
  Contain information relevant to the entire file. The header entries define the design name, tool used to generate the SDF file, parameters used to identify the design, and operating conditions.

- The cell entries
  Identify specific design instances, paths, and nets and associate timing data with them. Cell entries are design/instance-specific or library/type-specific.
  Each cell entry begins with the CELL keyword followed by the CELLTYPE, INSTANCE, and optionally CORRELATION 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 delay entries
  Specify the delay values associated with module paths, nets, interconnects, and devices.
  The delay types include: module paths, device, interconnect, port, pathpulse and global pathpulse limits.

- The timing check entries
  Specify timing checks as constraints between signals that determine how they can change in relation to each other. EDA tools use this information in different ways during the design process:
    - Simulation tools are warned 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 timing check types include setup and hold, recovery, width and period, nochange, skew and skew constraint, path constraints, sum, and difference.

SDF File Content Example

```
(DELAYFILE
  (DESIGN    "my_design")
  (DATE      "Jun 12, 1992 09:46")
```
(VENDOR      "Southwestern ASIC")
(PROGRAM     "Fast program")
.VERSION     "1.2a"
(DIVIDER     /
(VOLTAGE     4.5:5.0:5.5)
(PROCESS     "best")
(TEMPERATURE 27:55:100)
(TIMESCALE   100 ps)
(CELL (CELLTYPE "DFF")
(INSTANCE a.b.c)
(Delay (ABSOLUTE
(IOPATH (posedge clk) q (2:3:4) (5:6:7))
(PORT clr (2:3:4) (5:6:7))
)
)

Header Entries

delay_file ::= DELAYFILE sdf_version? design_name? date? vendor?
program_name? program_version? hierarchy_divider? voltage? process?
temperature? time_scale? cell+ )

sdf_version ::= ( SDFVERSION QSTRING? )
design_name ::= ( DESIGN QSTRING? )
date ::= ( DATE QSTRING? )
vendor ::= ( VENDOR QSTRING? )
program_name ::= ( PROGRAM QSTRING? )
program_version ::= ( VERSION QSTRING? )
hierarchy_divider ::= ( DIVIDER HCHAR? )
voltage ::= ( VOLTAGE rexpression? )
process ::= ( PROCESS QSTRING? )
temperature ::= ( TEMPERATURE rexpression? )
time_scale ::= ( TIMESCALE TSVALUE? )

sdf_version

sdf_version is the SDF software version.
design_name

design_name is a name that identifies the design.

date

date is the date and time when SDF was generated.

vendor

vendor is the name of the vendor whose tools generated the SDF file.

program_name

program_name is the name of the program used to generate the SDF file.

program_version

program_version is the program version number used to generate the SDF file.

hierarchy_divider

hierarchy_divider is the hierarchical path divider. HCHAR is either a period (.), or a slash (/). The default is the period (.).

voltage

voltage is the operating voltage (in volts) of the design.

process

process is the process operating envelope which is a string in double quotes.

temperature

temperature is the operating ambient temperature(s) of the design in degrees centigrade.

time_scale

time_scale is the definition of time units. The default is \textit{ns}.

cell

cell is the definition of delay annotation information for a single cell.
Cell Entries

cell ::= ( CELL celltype instance+ correlation? timing_spec+ )
celltype ::= ( CELLTYPE QSTRING )
instance ::= ( INSTANCE PATH? )
correlation ::= ( CORRELATION QSTRING corr_factor? )
corr_factor ::= ( NUMBER )
||= ( NUMBER : NUMBER : NUMBER )
timing_spec ::= ( DELAY deltype+ )
||= ( TIMINGCHECK tc_def+ )

cell

The CELL keyword represents an instance of a cell.

celltype

The CELLTYPE keyword represents the type of a cell.

QSTRING is typically a cell from a cell library. For example, QSTRING can be "DFF" (for a D flip-flop).

instance

The INSTANCE keyword represents a particular instance of the specified cell type.

PATH is a full hierarchical path such as a1.b1.c1, which can be represented in either format shown below:

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

(CELL (CELLTYPE "DFF")
  (INSTANCE a1)
  (INSTANCE b1)
  (INSTANCE c1)
  <timing_specification>
  <timing_specification>
)
If you do not specify PATH, the default is the current top level.

The timing data in the timing specification applies only to the specified cell instance. To associate the timing data with all instances of the specified cell type, you can specify a wildcard character (*) after the INSTANCE keyword, as shown below. Note that only instances in or below the wildcard (*) specification are affected.

```
(CELL (CELLTYPE "DFF")
  (INSTANCE *)
  <timing_specification>
  <timing_specification>
)
```

correlation

correlation

CORRELATION is an optional parameter associated with an instance of a cell. QSTRING is a correlation group name in double quotes into which this instance is placed. All the cell instances with the same QSTRING and all their child instances are placed in the same correlation group. The correlation of a child instance can be overridden using a different cell definition statement.

corr_factor

corr_factor

The correlation factor is a single positive real number or a triplet which represents the correlation in percentage (%).

If a single value is used, all delays on all the instances in the specified correlation group are correlated by this amount, except the correlation between the rising and the falling edges of the same instance will be defaulted to 100%.

If a triplet is used, the first value refers to the correlation between the rising and the falling edges of an instance, the second value refers to the correlation between the same edges of different instances in the same correlation group, and the third value refers to the correlation between the rising and the falling edges of different instances in the same correlation group.

If a group is not specified with a corr_factor, a correlation of 100% is assumed. A cell instance is not correlated with any other cell in any way if CORRELATION is not specified, unless it is a child instance of a correlated cell instance.

```
(CELL (CELLTYPE "DFF")
  (INSTANCE a1)
  (CORRELATION "target" (50:50:50))
)
Each cell entry in the SDF file includes one or more timing specifications, which contains the actual timing data associated with that cell entry. There are two types of timing specifications that are identified by the DELAY, and TIMINGCHECK.

- The DELAY keyword (for more information, see “Delay Entries” on page 20).
- The TIMINGCHECK keyword (for more information, see “Timing Check Entries” on page 28).

### Delay Entries

\[
\text{timing_spec} \::= \quad (\text{DELAY} \text{ deltype}+) \\
\mid \mid = \quad (\text{TIMINGCHECK} \text{ tc_def}+) \\
\text{deltype} \::= \quad (\text{PATHPULSE} \text{ port_path} \text{ port_path}? (\text{expression}) (\text{expression})?) \\
\mid \mid = \quad (\text{GLOBALPATHPULSE} \text{ port_path} \text{ port_path}? (\text{expression}) (\text{expression})?) \\
\mid \mid = \quad (\text{ABSOLUTE} \text{ absvals}+) \\
\mid \mid = \quad (\text{INCREMENT} \text{ incvals}+) \\
\text{absvals} \::= \quad (\text{IOPATH} \text{ port_spec} \text{ port_path} \text{ rvalue}) \\
\mid \mid = \quad (\text{COND} \text{ conditional_port_expr} (\text{IOPATH} \text{ port_spec} \text{ port_path} \text{ rvalue})) \\
\mid \mid = \quad (\text{PORT} \text{ port_path} \text{ rvalue}) \\
\mid \mid = \quad (\text{INTERCONNECT} \text{ port_instance} \text{ port_instance} \text{ rvalue}) \\
\mid \mid = \quad (\text{NETDELAY} \text{ name} \text{ rvalue}) \\
\mid \mid = \quad (\text{DEVICE} \text{ port_instance}? \text{ rvalue}) \\
\text{incvals} \::= \quad (\text{IOPATH} \text{ port_spec} \text{ port_path} \text{ rvalue}) \\
\mid \mid = \quad (\text{COND} \text{ conditional_port_expr} (\text{IOPATH} \text{ port_spec} \text{ port_path} \text{ rvalue})) \\
\mid \mid = \quad (\text{PORT} \text{ port_path} \text{ rvalue}) \\
\mid \mid = \quad (\text{INTERCONNECT} \text{ port_instance} \text{ port_instance} \text{ rvalue}) \\
\mid \mid = \quad (\text{NETDELAY} \text{ name} \text{ rvalue}) \\
\mid \mid = \quad (\text{DEVICE} \text{ port_instance}? \text{ rvalue})
\]
Timing specifications that start with the DELAY keyword associate delay values with module paths, nets, interconnects, and devices. The delay values are absolute or incremental. Absolute delays replace existing delay values in the design and incremental delays are positive or negative values that are added to the existing delay values. Absolute and incremental delays are indicated with the ABSOLUTE and INCREMENT keywords, as in the following examples:

```sdf
(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))
  ))
)

(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))
  ))
)
```

The delay values in the examples above 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. You can specify delay values of one through six or twelve min:typ:max triplets, which represent the delays in the following order:

- 0→1, 1→0, 0→Z, Z→1, 1→Z, Z→0,
- 0→X, X→1, 1→X, X→0, X→Z, Z→X

If you do not want to change a minimum, typical, or maximum delay value in the design, omit that component in the min:typ:max triplet. This is true for both absolute and incremental delays.

deltype

deltype Use the following keywords for “deltype” to specify the type of delay.
**PATHPULSE**

- The **PATHPULSE** keyword represents 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 first *port_path* is an input, or a bi-directional port.

  The second *port_path* is an output or a bi-directional port.

  The first *expression*, in time units, is the pulse rejection limit. The second *expression*, in time units, is the X limit. If you specify only one *expression*, both limits are set to that value. You can associate two types of limits with the path.

- **Pulse rejection limit**

  This limit defines the minimum pulse width required for the pulse to pass through to the output; anything smaller does not affect the output.

- **X limit**

  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 (X) state or is rejected (if smaller than the pulse rejection limit). Note that the X limit must be greater than the pulse rejection limit to carry any significance.

Specify PATHPULSE in time units.

**Example**

```lisp
(INSTANCE x)
(Delay
  (PATHPULSE y.i1 y.o1 (13) (21))
)
```
The first value (13) is the pulse rejection limit and the second (21) is the X limit.

GLOBALPATHPULSE

- The **GLOBALPATHPULSE** keyword is the same as **PATHPULSE** but the values are in percentage (%) of cell path delay from the input to the output.

**Example**

(INSTANCE x)

(DELAY

(GLOBALPATHPULSE y.i1 y.o1 (25) (35))

)

The first value (25) is the pulse rejection limit and the second (35) is the X limit.

ABSOLUTE

- The **ABSOLUTE** keyword represents the delay values that replace existing delay values.

INCREMENT

- The **INCREMENT** keyword represents the delay values that are added to existing delay values.

**absvals**

absvals represents the absolute delay values that replaces the existing delay values.

You can specify *rvalue* (that is, positive or negative numbers) for both *absvals* and *incvals*.

**incvals**

incvals represents the incremental delay values that are added to existing delay values.

You can specify *rvalue* (that is, positive or negative numbers) for both *absvals* and *incvals*. 
The `IOPATH` keyword represents the delays on a legal path from an input/bidirectional port to an output port of a device. Each delay value is associated with a unique input port/output port pair.

`port_spec` is an input, or a bi-directional port and can have an edge identifier.

`port_path` is an output or a bi-directional port.

`rvalue` is the IOPATH delay from `port_spec` to `port_path`.

If conditions exist between the two specified ports, the specified `rvalue` applies to all of them. This situation is similar to specifying all existing conditional paths (using the `COND` clause form of IOPATH below) with the same IOPATH delay `rvalue`.

**Example**

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

This example includes a conditional IOPATH to represent state-dependent path delays using the `COND` construct.

**COND**

The `COND` keyword represents conditional module path delays.

`conditional_port_expression` is the Boolean description of the state dependency of the delay. The delay values apply if the `conditional_port_expression` is true (logical one) and the delay
values do not apply if the `conditional_port_expression` is false (logical zero).

`port_spec` is an input, or a bi-directional port and can have an edge identifier. `port_path` is an output or bi-directional port. `rvalue` is the IOPATH delay from `port_spec` to `port_path`.

**PORT**
• The **PORT** keyword represents the actual or estimated interconnect signal delay values you can place on the input port, without having to specify a start point for the delay path.

```
port_path is an input or bi-directional port. rvalue is the PORT delay of the port_path.
```

**Example**

```
(INSTANCE x)
(DELAY (ABSOLUTE
    (PORT a.b.i1 (2:0:3))
))
```

**INTERCONNECT**

• The **INTERCONNECT** keyword represents actual or estimated delays in the wire paths between devices.

The first *port_instance* is an output or bi-directional port. The second *port_instance* is an input or bi-directional port. *rvalue* is the INTERCONNECT delay between the output and input ports.

**Example**

```
```

Standard Delay Format
The **NETDELAY** keyword represents a 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.

`name` is the name of the net or you can specify it by the output port driving it. `rvalue` is the delay associated with the net or port specified by `name`.

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

In the example, the net is identified by name.

**Example**

```
(INSTANCE x)
(DELAY (ABSOLUTE
       (NETDELAY w1 (2.5:3:3.5) (2.9:4:5) (6.3:8:9.9))
))
```

In the example, the net is identified by name.
• The **DEVICE** keyword represents the intrinsic delay of a module or gate. Intrinsic delay is specific to the type of the object and so 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, you can attach 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.

```
x.a.b
    i1
    i2
      o1
```

**port_instance** is an output port.

**rvalue** is a device (type-specific) delay.

**Example**

```
(INSTANCE x.a.b)
(DELAY (INCREMENT
    (DEVICE o1 (-1:3:8) (-4:-2:7) (-2:8:9)
      (-1:7:8) (-1:5:7) (-1:8:9))
))
```

**Timing Check Entries**

```
timing_spec ::= ( DELAY deltype+ )
  ||= ( TIMINGCHECK tc_def+ )
tc_def ::= ( SETUP port_tchk port_tchk tc_rvalue )
  ||= ( HOLD port_tchk port_tchk tc_rvalue )
  ||= ( SETUPHOLD port_tchk port_tchk tc_rvalue tc_rvalue )
  ||= ( RECOVERY port_tchk port_tchk tc_rvalue )
  ||= ( SKEW port_tchk port_tchk tc_rvalue )
  ||= ( WIDTH port_tchk tc_rvalue )
  ||= ( PERIOD port_tchk tc_rvalue )
  ||= ( NOCHANGE port_tchk port_tchk tc_rvalue tc_rvalue )
```
Timing specifications that start with the TIMINGCHECK keyword assign timing check limits to specific cell instances and assign constraints to critical paths in the design.

In the SDF file you either have timing data (for back annotation) or layout constraints (for forward annotation).

- Timing Checks

You can specify many different timing check types including setup, hold, recovery, skew, width, period, and nochange checks. The following example shows a cell entry which 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))
 )
)
```

- Layout Constraints

There are four types of layout constraints in the SDF file — PATHCONSTRAINT, SUM, DIFF, and SKEWCONSTRAINT. 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))
    )
)

port_tchk

The COND keyword represents conditional timing checks.

timing_check_condition is a conditional signal. The timing check is
triggered by the occurrence of the associate event if the
timing_check_condition is true (logical one).

tc_def

Use the following keywords for “tc_def” to delimit the definition of
timing check information.

SETUP

- The SETUP keyword represents limits for setup timing checks.

The basic definition for setup and hold requirements is that the data
signal must remain stable for a specified minimum time interval in
order for a transition of the clock signal to store the data
successfully. Setup time defines the minimum interval before the
clock transition; hold time defines the minimum interval after the
clock transition. Any change to the data signal within these intervals
results in a timing violation. To shift the violation intervals with
respect to the clock transition, either the setup time or hold time can
be negative; however, their sum must always be greater than zero.

Setup and hold

\[
\begin{array}{c}
\text{Setup} \\
\text{Hold}
\end{array}
\]

\[
\begin{array}{c}
\text{din} \\
\text{clk}
\end{array}
\]

The first port_tchk is the data event.
The second port_tchk is the clock event. tc_rvalue is the SETUP time between the data and clock events.

Both events can be edge-triggered and conditional.

**Example**

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

**HOLD**

- The HOLD keyword represents limits for hold timing checks.
  
The first port_tchk is the data event.
  
The second port_tchk is the clock event. tc_rvalue is the HOLD time between the data and clock events.
  
Both events can be edge-triggered and conditional.

**Example**

(INSTANCE x.a)
(TIMINGCHECK
  (HOLD din (posedge clk) (9.5))
)

**SETUPHOLD**

- The SETUPHOLD keyword represents setup and hold limits in a single statement.
  
The first port_tchk is the data event.
  
The second port_tchk is the clock event. The first tc_rvalue is the setup time.
  
The second tc_rvalue is the hold time.
  
Both events can be edge-triggered and conditional.

**Example**

(INSTANCE x.a)
(TIMINGCHECK
  (SETUPHOLD (COND !reset din) (posedge clk) (12) (9.5))
)

Note the conditional timing check in the example occurs upon an event on din only when the reset signal is in the zero state.
RECOVERY

- The **RECOVERY** keyword represents the limits for recovery timing checks. A recovery timing check is a constraint between 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—it can be the value set by the clearbar, or it can be the value clocked into the device from the data input.

![Diagram of recovery timing check](image)

The first `port_tchk` is the edge-triggered clock event. The second `port_tchk` is the upper bound data event. `tc_rvalue` is the recovery limit. Both events can be edge-triggered and conditional.

**Example**

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

SKEW

- The **SKEW** keyword represents the limits for signal skew timing checks. A signal skew limit is the maximum allowable delay between two signals, which if exceeded causes devices to behave unreliably. The first `port_tchk` is the lower bound clock event.

The second `port_tchk` is the upper bound clock event. `tc_rvalue` is the skew limit.

Both events can be edge-triggered and conditional.
Example
(INSTANCE x)
(TIMINGCHECK
  (SKEW (posedge clk1) (posedge clk2) (6))
)

WIDTH

- The WIDTH keyword represents the limits for width timing checks. The width timing check is the minimum allowable time for the positive or negative phase of each cycle. If a signal has unequal phases, you can specify a separate width check for each phase.

```
port_tchk is the edge-triggered clock event. tc_rvalue is the width limit. The data event is equal to the clock event with the opposite edge.

You can use the COND construct with the WIDTH construct.
Example
(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.

**PERIOD**

- The **PERIOD** keyword represents the limits for period timing checks.

The period timing check is the minimum allowable time for one complete cycle of the signal.

\[ \text{port_tchk} \text{ is the edge-triggered clock event. } \text{tc_rvalue} \text{ is the period limit.} \]

The data event is equal to the clock event with the same edge. You can use the COND construct with the PERIOD construct.

**Example**

(INSTANCE x.b)

(TIMINGCHECK

  (PERIOD (posedge clk) (46.5))

)

The period is the interval between two positive clock edges.

**NOCHANGE**

- The **NOCHANGE** keyword represents the limits for nochange timing checks. The nochange timing check is a signal constraint relative to the width of a clock pulse. For example, you can use this construct to model the timing constraints of memory devices, as when address lines must remain stable during a write pulse.
The first port_tchk is the reference event. The second port_tchk is the data event. The first tc_rvalue is the start edge offset. The second tc_rvalue is the end edge offset. Both events can be edge-triggered and conditional.

**Example**

(INSTANCE x)
(TIMINGCHECK
  (NOCHANGE (negedge clk) data (4.5) (4.5))
)

This example defines a period beginning 4.5 time units before the negative clk edge and ending 4.5 time units after the subsequent positive clk edge. During this time period, the data signal must not change.

**PATHCONSTRAINT**

- The **PATHCONSTRAINT** keyword represents the path constraints. Path constraints are the critical paths in a design identified during timing analysis. Layout tools can use these constraints to direct the physical design. The constraint specifies the maximum allowable delay for a path, which is typically identified by the two ports at each end of the path. You can also specify intermediate ports to uniquely identify path(s).
The first *port_instance* is the beginning of the path. The last *port_instance* is the end of the path. You can specify intermediate points to describe the path by using additional *port_instance* specifications. The first *tc_rvalue* is the maximum rise delay between the beginning and end points. The second *tc_rvalue* is the maximum fall delay between the beginning and end points.

**Example**

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

**SUM**

- The **SUM** keyword represents two or more paths in a design and limits the sum of their delays to a specified maximum.

The first *port_instance* is the beginning of the path. The second *port_instance* is the end of the path. You can specify additional paths by using additional beginning/end port pairs. *tc_rvalue* is the total (SUM) of the individual delays associated with each beginning/end port pair. The sum of all port pair delays must be less than *tc_rvalue*.

**Example**

(INSTANCE x)
(TIMINGCHECK
  (SUM (m.n.o1 y.z.i1) (y.z.o2 a.b.i2) (67.3))
)
**DIFF**

- The **DIFF** keyword represents two paths in a design and the maximum allowable difference between their delays.

  The first *port_instance* is the beginning of the path.
  The second *port_instance* is the end of the path. You can specify additional paths by using additional beginning/end port pairs. 
  *tc_rvalue* is the maximum difference between two path delays. The difference of the individual delays in the two circuit paths must be less than *tc_rvalue*.

**Example**

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

**SKEWCONSTRAINT**

- The **SKEWCONSTRAINT** keyword represents the same skew limit for multiple signal pairs.

  The signal in the SKEWCONSTRAINT construct is the reference signal (that is, the clock signal) against which all associated port signals are constrained.

  *port_spec* is the clock event. All associated ports are data events.*tc_rvalue* is the maximum skew delay associated with *port_spec*.

**Example**

```
(INSTANCE x)
(TIMINGCHECK
  (SKEWCONSTRAINT (posedge clk1) (7.5))
)
```
Syntax and Examples

SDF File Syntax

The syntax for the Standard Delay File is shown here.

sdf_version ::= ( SDFVERSION QSTRING? )
design_name ::= ( DESIGN QSTRING? )
date ::= ( DATE QSTRING? )
vendor ::= ( VENDOR QSTRING? )
program_name ::= ( PROGRAM QSTRING? )
program_version ::= ( VERSION QSTRING? )
hierarchy_divider ::= ( DIVIDER HCHAR? )
voltage ::= ( VOLTAGE reexpression? )
process ::= ( PROCESS QSTRING? )
temperature ::= ( TEMPERATURE reexpression? )
time_scale ::= ( TIMESCALE TSVALUE? )
cell ::= ( CELL celltype instance+ correlation? timing_spec+ )
celltype ::= ( CELLTYPE QSTRING )
instance ::= ( INSTANCE PATH? )
correlation ::= ( CORRELATION QSTRING corr_factor? )
corr_factor ::= ( NUMBER )
||= ( NUMBER : NUMBER : NUMBER )
timing_spec ::= ( DELAY deltype+ ) ||= ( TIMINGCHECK tc_def+ )
deltype ::= ( PATHPULSE port_path port_path? ( expression ) ( expression )? )
||= ( GLOBALPATHPULSE port_path port_path? ( expression ) ( expression )? )
||= ( ( ABSOLUTE absvals+ )
||= ( INCREMENT incvals+ )
absvals ::= ( IOPATH port_spec port_path rvalue )
Standard Delay Format Syntax and Examples

\[
\begin{align*}
&||= ( \text{COND} \ \text{conditional\_port\_expr} ( \ \text{IOPATH} \ \text{port\_spec} \ \text{port\_path} \ rvalue ) ) \\
&||= ( \ \text{PORT} \ \text{port\_path} \ rvalue ) \\
&||= ( \ \text{INTERCONNECT} \ \text{port\_instance} \ \text{port\_instance} \ rvalue ) \\
&||= ( \ \text{NETDELAY} \ \text{name} \ rvalue ) \\
&||= ( \ \text{DEVICE} \ \text{port\_instance}\? \ rvalue ) \\
&\text{incvals ::= ( IOPATH \ \text{port\_spec} \ \text{port\_path} \ rvalue )} \\
&||= ( \ \text{COND} \ \text{conditional\_port\_expr} ( \ \text{IOPATH} \ \text{port\_spec} \ \text{port\_path} \ rvalue ) ) \\
&||= ( \ \text{PORT} \ \text{port\_path} \ rvalue ) \\
&||= ( \ \text{INTERCONNECT} \ \text{port\_instance} \ \text{port\_instance} \ rvalue ) \\
&||= ( \ \text{NETDELAY} \ \text{name} \ rvalue ) \\
&||= ( \ \text{DEVICE} \ \text{port\_instance}\? \ rvalue ) \\
&\text{conditional\_port\_expr ::= simple\_expression} \\
&||= ( \ \text{conditional\_port\_expr} ) \\
&||= \ \text{UNARY\_OPERATOR} ( \ \text{conditional\_port\_expr} ) \\
&||= \ \text{conditional\_port\_expr} \ \text{BINARY\_OPERATOR} \ \text{conditional\_port\_expr} \\
&\text{simple\_expression ::= ( simple\_expression )} \\
&||= \ \text{UNARY\_OPERATOR} ( \ \text{simple\_expression} ) \\
&||= \ \text{port} \\
&||= \ \text{UNARY\_OPERATOR} \ \text{port} \\
&||= \ \text{SCALAR\_CONSTANT} \\
&||= \ \text{UNARY\_OPERATOR} \ \text{SCALAR\_CONSTANT} \\
&||= \ \text{simple\_expression} \ \text{QM} \ \text{simple\_expression} \ \text{CLN} \\
&\text{simple\_expression} \\
&||= \ \{ \ \text{simple\_expression} \ \text{concat\_expression}\? \ \} \\
&||= \ \{ \ \text{simple\_expression} \ \{ \ \text{simple\_expression} \ \text{concat\_expression}\? \ \} \ \} \\
&\text{concat\_expression ::= , simple\_expression} \\
&\text{name ::= port\_instance} \\
&||= \ \text{net\_instance} \\
&\text{net\_instance ::= net} \\
&||= \ \text{instance} \ \text{net} \\
&\text{net ::= IDENTIFIER} \\
&\text{tc\_def ::= ( SETUP port\_tchk port\_tchk tc\_rvalue )} \\
&||= ( HOLD port\_tchk port\_tchk tc\_rvalue ) \\
&||= ( SETUPHOLD port\_tchk port\_tchk tc\_rvalue tc\_rvalue )
\end{align*}
\]
\[ \text{port_tchk} ::= \text{port_spec} \]
\[ \text{timing_check_condition ::= scalar_port} \]
\[ \text{UNARY_OPERATOR scalar_port} \]
\[ \text{scalar_port==SCALAR_CONSTANT} \]
\[ \text{scalar_port===SCALAR_CONSTANT} \]
\[ \text{scalar_port!=SCALAR_CONSTANT} \]
\[ \text{scalar_port!==SCALAR_CONSTANT} \]

\[ \text{port_spec} ::= \text{port_path} \]
\[ \text{port_edge} ::= \text{EDGE_IDENTIFIER port_path} \]

\[ \text{port ::= IDENTIFIER} \]
\[ \text{IDENTIFIER [ DNUMBER ]} \]
\[ \text{IDENTIFIER [ DNUMBER : DNUMBER ]} \]

\[ \text{scalar_port ::= IDENTIFIER} \]
\[ \text{IDENTIFIER [ DNUMBER ]} \]

\[ \text{port_instance ::= port_path} \]
\[ \text{instance port_path} \]

\[ \text{tc_value ::= ( NUMBER )} \]
\[ \text{value ::= ( NUMBER )} \]
\[ \text{exp_list ::= ( expression )} \]
\[ \text{( expression ? ) ( expression )} \]
\[ \text{( expression ? ) ( expression ? ) ( expression )} \]
expression ::= NUMBER : NUMBER ? : NUMBER ?
||= NUMBER ? : NUMBER : NUMBER ?
||= NUMBER ? : NUMBER ? : NUMBER

tc_rvalue ::= ( RNUMBER )
||= ( rexpression )
rvalue ::= ( RNUMBER )
||= rexp_list

rexp_list ::= ( rexpression )
||= ( rexpression ? ) ( rexpression )
||= ( rexpression ? ) ( rexpression ? ) ( rexpression )

rexpansion ::= RNUMBER : RNUMBER ? : RNUMBER ?
||= RNUMBER ? : RNUMBER : RNUMBER ?
||= RNUMBER ? : RNUMBER ? : RNUMBER

SDF File Example 1

The SDF file example, shown on the next page, is based on the schematic shown below.
Figure A-3 SDF Example Schematic

(DELAYFILE
  (SDFVERSION "1.0")
  (DESIGN "system")
  (DATE "Saturday September 30 08:30:33 PST 1990")
  (VENDOR "Yosemite Semiconductor")
  (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))))
)
)
SDF File 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 "Feb 21, 1992 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))
    )
  ))
```
SDF File Example 3

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

(DELAYFILE
  (SDFVERSION "2.0")
  (DESIGN "top")
  (DATE "Nov 25, 1991 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
       (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))
       )
    )
    (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 (negedge CP) (posedge RB) (1:1:1)))
     (COND RB (RECOVERY (negedge CP) (posedge SB) (1:1:1)))
    )))
)
SDF File Example 4

This example shows how to forward annotate timing constraints. The key to specifying SDF constraints is to identify INSTANCE-PINS of library cells. In the example shown below I2 is an instance and H01 is a PIN (port) on that instance.

(DELAYFILE
  (SDFVERSION "1.0")
  (DESIGN "testchip")
  (DATE "Dec 17, 1991 14:49:48")
  (VENDOR "Big Chips Inc.")
  (PROGRAM "Chip Analyzer") (VERSION "1.3b")
  (DIVIDER )
  (VOLTAGE :3.8:) (PROCESS "worst") (TEMPERATURE :37: )
  (TIMESCALE 10ps)
  (CELL (CELLTYPE "XOR") (INSTANCE )
    (TIMINGCHECK
      (PATHCONSTRAINT I2.H01 I1.N01 (989:1269:1269))
      (PATHCONSTRAINT I2.H01 I3.N01 (904:1087:1087))
    )))
Delay Model Recommendation

Introduction

The delay model provides a guideline for using SDF in ASIC application tools. All constructs in SDF should be directly applicable to the delay model. ASIC timing is divided into forward annotation and backannotation. Although SDF supports both timing concepts, this section concentrates on ASIC timing backannotation model. Future release of SDF will provide an abstract model for forward annotation.

The following section define the delay model and provide rules that should be adhered to ensure proper interpretations and usage of SDF constructs.

The Delay Model

Timing Objects

The delay model consists the following timing objects:
1. Interconnect delay (INT), represented by the INTERCONNECT delay construct in SDF.
2. Path delay (PD), represented by IOPATH delay construct in SDF.
3. State-dependent path delay (SDPD), represented by COND keyword in SDF.
4. Port delay (IPD), represented by PORT delay construct in SDF.
5. Net delay (ND), represented by NETDELAY construct in SDF. Note this timing object is a degenerate interconnect delay.
6. Device delay (DEV), represented by DEVICE construct in SDF. Note when specified with a cell output port, this timing object is a degenerate path delay; when specified with a primitive instance, this timing object is its intrinsic delay.

7. Path pulse (PP), represented by PATHPULSE construct in SDF.

8. Timing checks (TC), represented with several keywords in SDF depending on the type of the timing checks.

Rules

1. Path delay is described between any input (or bi-directional) port to any output (or bi-directional) port in the same cell.

2. Multiple path delays can be defined for any output (or bi-directional) port.

3. Multiple path delays can be defined between any pair of ports only by using state dependent delays.

4. Path delay can have up to twelve transition states with twelve different delay values.

5. Negative timing values for absolute path delays may default to zero in certain application tools.

6. Interconnect delay is described between any output (bi-directional) port of a cell to any input (bi-directional) port of any cell.

7. Multiple interconnect delays from different sources can be described for any input (bi-directional) port, destination port.

8. Depending on the type of the timing check, it can be applied to a single or a pair of ports.

9. Timing checks are allowed from an output port to another output port.

10. Timing checks are applied after the interconnect delays are applied.

11. Negative timing values in timing checks are allowed. Some application tools may use the negative values while others may compile them as zero values.

12. INTERCONNECT delays cannot be used if NETDELAY and/or PORT delays are specified between the same source and destination signals.

13. Similarly, NETDELAY and/or PORT delays cannot be used if INTERCONNECT delay is specified between the same source and destination signals.

14. IOPATH delay cannot be used if DEVICE delay is specified between the same ports within the same cell.

15. Similarly, DEVICE delay cannot be used if IOPATH delay is specified between the same ports within the same cell.

16. All timing objects using the internal nodes may be ignored by application tools that have no concept of the internal nodes.
17. For the same timing object, delay annotation is executed in the sequential order as encountered in a single SDF file.
Index

$  
$sdfl_annotate system task 5

A  
absolute delays 21  
ABSOLUTE keyword 20, 23  
example 21  
alphanumeric characters 7

C  
CELL keyword 17  
example 18  
CELLTYPE keyword 18  
example 18  
COND keyword 20, 25, 30  
conditional timing checks 30

D  
DATE keyword 16  
defining  
nochange timing check 34  
path constraints 35  
period timing check 34  
signal skew limit 32  
width timing checks 33  
DELAY keyword 18, 20  
example 21  
delay values  
adding to existing values 22  
replacing existing values 22  
DELAYFILE keyword 16  
delays  
absolute 21  
edge transitions 21  
incremental 21  
DESIGN keyword 16  
design name, identifying 17  
device delay 27  
DEVICE keyword 20  
example 28  
DIFF keyword 29  
example 37  
DIVIDER keyword 16

E  
escape character 7
G
GLOBALPATHPULSE keyword 20
example 23

H
HCHAR 17
hierarchical design 1
hierarchy divider character 7
HOLD keyword 28
example 31

I
I/O port delay 24
INCREMENT keyword 20, 23
example 21
incremental delays 21
INSTANCE keyword, example 18
INTERCONNECT keyword 20
example 26
interconnect signal delay 26
IOPATH keyword 20
example 21, 24

K
keywords 6, 7
ABSOLUTE 20, 23
CELL 17
CELLTYPE 18
COND 20, 25, 30
DATE 16
DELAY 18, 20
DELAYFILE 16
DESIGN 16
DEVICE 20
DIFF 29
DIVIDER 16
GLOBALPATHPULSE 20
HOLD 28
INCREMENT 20, 23
INTERCONNECT 20
IOPATH 20
NETDELAY 20
NOCHANGE 28
PATHCONSTRAINT 29, 35
PATHPULSE 20, 22
PERIOD 28
PORT 20
PROCESS 16
PROGRAM 16
RECOVERY 28, 32
SDFVERSION 16
SETUP 28
SETUPHOLD 28
SKEW 28
SKEWCONSTRAINT 29
SUM 29
TEMPERATURE 16
TIMESCALE 16
TIMINGCHECK 18, 28, 29
VENDOR 16
VERSION 16
VOLTAGE 16
WIDTH 28

L

layout constraints 29
lexical tokens 6, 7
   BINARY_OPERATOR 10, 12
   DNUMBER 8, 13
   EDGE_IDENTIFIER 9, 13
   HCHAR 8, 13, 16, 17
   IDENTIFIER 8, 13
   NUMBER 8, 13
   PATH 9, 13
   QSTRING 7, 16, 18
   RNUMBER 8, 13, 14
   SCALAR_CONSTANT 9, 12, 29
   TSVALUE 9, 16
   UNARY_OPERATOR 10, 12, 29

M

min,typ,max triples 14, 21
module delay 27
multiple signal pairs 37

N

naming the SDF files 1
net delay 27
NETDELAY keyword 20
   example 27
NOCHANGE keyword 28
   example 35

P

PATH 18
path delays
   adding two or more 37
   difference between two paths 37
PATHCONSTRAINT keyword 29, 35
   example 36
PATHPULSE keyword 20, 22
   example 23
percentage of device path delay 23
PERIOD keyword 28
   example 34
place holder 14
PORT keyword 20
   example 21, 26
PROCESS keyword 16
PROGRAM keyword 16
pulse rejection limit 22

R
RECOVERY keyword 28, 32
   example 32
recovery timing check defining 32

S
SDF
   $sdf_annotate system task 5
   $sdf_annotate system task 5
SDF file
   cell entries 15
   Cell entries 18
   content 1
delay entries 15
   example 42, 45, 46, 47
   format 15
   format example 16
   header 15, 16
   syntax 39, 49
time generated 17
timing check entries 15
   version 17
SDFVERSION keyword 16
setup and hold, defining 30
SETUP keyword 28
   example 31
SETUPHOLD keyword 28
   example 31
SKEW keyword 28
   example 33
skew timing checks 32
SKEWCONSTRAINT keyword 29
   example 37
special character 7
SUM keyword 29
   example 36
syntax
   SDF file 39, 49
   syntax characters 7

T
temperature 17
TEMPERATURE keyword 16
time units 17
TIMESCALE keyword 16
timing checks 29
TIMINGCHECK keyword 18, 28, 29
transition expressions 14

V

VENDOR keyword 16
VERSION keyword 16
voltage 17
VOLTAGE keyword 16

W

white space 7
WIDTH keyword 28
  example 33
wildcard character 19
wire path delays 26

X

X limit 22