Section 8

Electrical Performance Modeling

8.1 Overview of modeling keywords

8.1.1 Timing models

The following tables show the set of keywords used for timing measurements and timing constraints, respectively. All keywords have implied semantics that restrict their capability to describe general temporal relations between arbitrary signals. For unrestricted purpose, the keyword TIME shall be used.

Table 8-1 Timing measurements

Keyword	Value type	Base Units	Default Units	Description
DELAY	number	Second	n (nano)	time between two threshold crossings within two consecutive events on two pins. A causal relationship between the two events is implied.
RETAIN	number	Second	n (nano)	time during which an output pin will retain its value after an event on the related input pin. RETAIN appears always in conjunction with DELAY for the same two pins.
SLEWRATE	non-negative number	Second	n (nano)	time between two threshold crossings within one event on one pin

Table 8-2 Timing constraints

Keyword	Value type	Base Units	Default Units	Description
HOLD	number	Second	n (nano)	minimum time limit for hold between two threshold crossings within two consecutive events on two pins
NOCHANGE	optional[a] non-negative number	Second	n (nano)	minimum time limit between two threshold crossings within two arbitrary consecutive events on one pin, in conjunction with SETUP and HOLD

Table 8-2 Timing constraints

Keyword	Value type	Base Units	Default Units	Description
PERIOD	non-negative number	Second	n (nano)	minimum time limit between two identical events within a sequence of periodical events
PULSEWIDTH	number	Second	n (nano)	minimum time limit between two threshold crossings within two consecutive and complementary events on one pin
RECOVERY	number	Second	n (nano)	minimum time limit for recovery between two threshold crossings within two consecutive events on two pins
REMOVAL	number	Second	n (nano)	minimum time limit for removal between two threshold crossings within two consecutive events on two pins
SETUP	number	Second	n (nano)	minimum time limit for setup between two threshold crossings within two consecutive events on two pins
SKEW	number	Second	n (nano)	absolute value is maximum time limit between two threshold crossings within two consecutive events on two pins, the sign indicates positive or negative direction

a. The associated SETUP and HOLD measurements provide data. NOCHANGE itself need not provide data

Table 8-3 : Generalized timing measurements

Keyword	Value type	Base Units	Default Units	Description
TIME	number	Second	1 (unit)	time point for waveform modeling, time span for average, RMS, peak modeling
FREQUENCY	non-negative number	Hz	meg (mega)	frequency
JITTER	non-negative number	Second	n (nano)	uncertainty of arrival time

Table 8-4 : Normalized measurements

Keyword	Value type	Base Units	Default Units	Description
THRESHOLD	non-negative number between 0 and 1	Normalized signal voltage swing	1 (unit)	Fraction of signal voltage swing, specifying a reference point for timing measurement data. The threshold is the voltage for which the timing measurement is taken.
NOISE_MARGIN	non-negative number between 0 and 1	Normalized signal voltage swing	1 (unit)	Fraction of signal voltage swing, specifying the noise margin. The noise margin is a deviation of the actual voltage from the expected voltage for a specified signal level

8.1.2 Analog models

Table 8-5 : Analog measurements

Keyword	Value type	Base Units	Default Units	Description
CURRENT	number	Ampere	m (milli)	electrical current drawn by the cell. A pin may be specified as annotation.[a]
ENERGY	number	Joule	p (pico)	electrical energy drawn by the cell, including charge and discharge energy, if applicable.
POWER	number	Watt	u (micro)	electrical power drawn by the cell, including charge and discharge power, if applicable.
TEMPERATURE	number	o Celsius	1 (unit)	temperature
VOLTAGE	number	Volt	1 (unit)	voltage

Table 8-5 : Analog measurements

Keyword	Value type	Base Units	Default Units	Description
FLUX	non-negative number	Coulomb per Square Meter	1 (unit)	amount of hot electrons in units of electrical charge per gate oxide area
FLUENCE	non-negative number	Second times Coulomb per Square Meter	1 (unit)	integral of FLUX over time

a. If the annotated PIN has PINTYPE=supply, the CURRENT measurement qualifies for power analysis. In this case, the current includes charge/discharge current, if applicable.

Table 8-6 : Electrical components

Keyword	Value type	Base Units	Default Units	Description
CAPACITANCE	non-negative number	Farad	p (pico)	pin, wire, load, or net capacitance
INDUCTANCE	non-negative number	Henry	n (nano)	pin, wire, load, or net inductance
RESISTANCE	non-negative number	Ohm	K (kilo)	pin, wire, load, or net resistance

8.1.3 Supplementary models

Table 8-7 : Abstract measurements

Keyword	Value type	Base Units	Default Units	Description
DRIVE_STRENGTH	non-negative number	None	1 (unit)	drive strength of a pin, abstract measure for $(\text{drive resistance})^{-1}$
SIZE	non-negative number	None	1 (unit)	abstract cost function for actual or estimated area of a cell or a block

Table 8-8 : Discrete measurements

Keyword	Value type	Base Units	Default Units	Description
SWITCHING_BITS	non-negative number	None	1	number of switching bits on a bus
FANOUT	non-negative number	None	1	number of receivers connected to a net
FANIN	non-negative number	None	1	number of drivers connected to a net
CONNECTIONS	non-negative number	None	1	number of pins connected to a net, where CONNECTIONS = FANIN+FANOUT

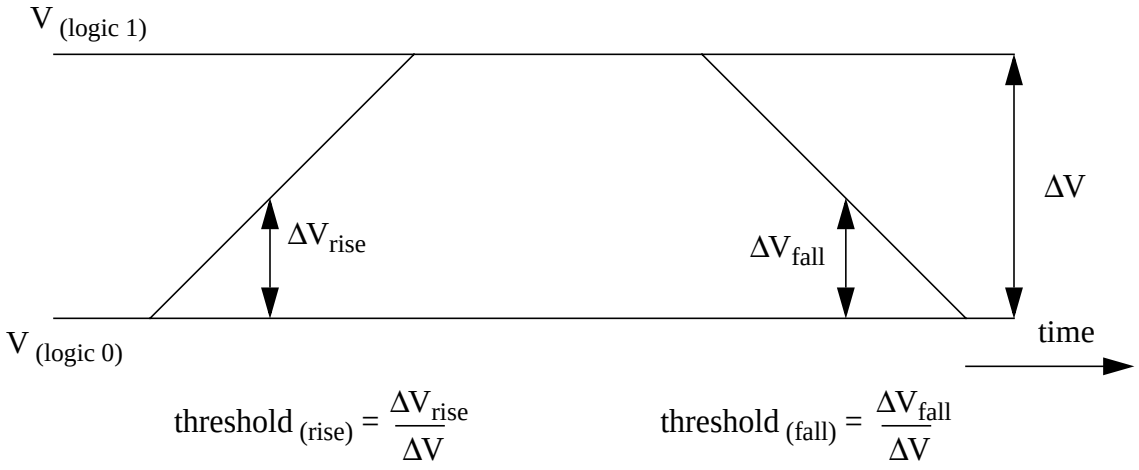
Actual values for discrete measurements are always integer numbers, however, estimated values may be non-integer numbers (e.g. average fanout of a net =2.4).

8.2 Auxiliary statements for timing models

8.2.1 THRESHOLD definition

The THRESHOLD represents a reference voltage level for timing measurements, normalized to the signal voltage swing and measured with respect to the logic 0 voltage level.

Figure 8-1: THRESHOLD measurement definition



The voltage levels for logic 1 and 0 represent a full voltage swing.

Different threshold data for RISE and FALL may be specified or else the data shall apply for both rising and falling transistions.

The THRESHOLD statement has the form of an arithmetic model. If the submodel keywords RISE and FALL are used, it has the form of an arithmetic model container.

Examples:

```
THRESHOLD = 0.4;
THRESHOLD { RISE = 0.3;  FALL = 0.5; }
THRESHOLD { HEADER { TEMPERATURE {TABLE{ 0 50 100 }}} TABLE { 0.5 0.4 0.3}}
```

8.2.2 FROM and TO container

A FROM container and a TO container shall be used inside timing measurements and timing constraints. Depending on the semantics of the timing model (see section ??), they may contain a THRESHOLD statement, PIN annotation and EDGE_NUMBER annotation. The data in the FROM and TO containers define the measurement start and end point, respectively.

Example:

```
DELAY {
      FROM {PIN = data_in;  THRESHOLD { RISE = 0.4; FALL = 0.6;} }
      TO   {PIN = data_out; THRESHOLD = 0.5;}
}
```

The delay is measured from pin `data_in` to pin `data_out`. The threshold for `data_in` is 0.4 for rising signal and 0.6 for falling signal. The threshold for `data_out` is 0.5, which applies for both rising and falling signal.

8.2.3 PIN annotation

If the timing measurements or timing constraints, respectively, apply semantically for two pins (see section ??), the FROM, TO containers shall each contain the PIN annotation.

Example:

```
DELAY {
      FROM { PIN = A ; }
      TO { PIN = Z ; }
}
```

Otherwise, if the timing measurements or timing constraints, respectively, apply semantically only for one pin (see section ??), the PIN annotation shall be outside the FROM, TO container.

Example:

```
SLEWRATE {
      PIN = A ;
}
```

8.2.4 EDGE_NUMBER annotation

The EDGE_NUMBER annotation within the context of a timing model shall specify the edge for which the timing measurement applies. The timing model must be in the context of a VECTOR. The EDGE_NUMBER shall have an unsigned value pointing to exactly one of subsequent `vector_single_event` expressions applicable to the referenced pin. The

EDGE_NUMBER shall be counted individually for each pin which appears in the VECTOR, starting with zero.

If the timing measurements or timing constraints, respectively, apply semantically for two pins (see section ??), the EDGE_NUMBER annotation shall be legal inside the FROM or TO container in conjunction with the PIN annotation.

Example:

```
DELAY {
      FROM { PIN = A ; EDGE_NUMBER = 0; }
      TO { PIN = Z ; EDGE_NUMBER = 0; }
}
```

Otherwise, if the timing measurements or timing constraints, respectively, apply semantically only for one pin (see section ??), the EDGE_NUMBER annotation shall be legal outside the FROM or TO container in conjunction with the PIN annotation.

Example:

```
SLEWRATE {
      PIN = A ;  EDGE_NUMBER = 0;
}
```

Default values for EDGE_NUMBER are specific for each timing model keyword (see section ??).

The EDGE_NUMBER annotation is necessary for complex timing models involving multiple transitions on the same pin. This is illustrated by the following examples.

Figure 8-2: Schematic of a pulse generator

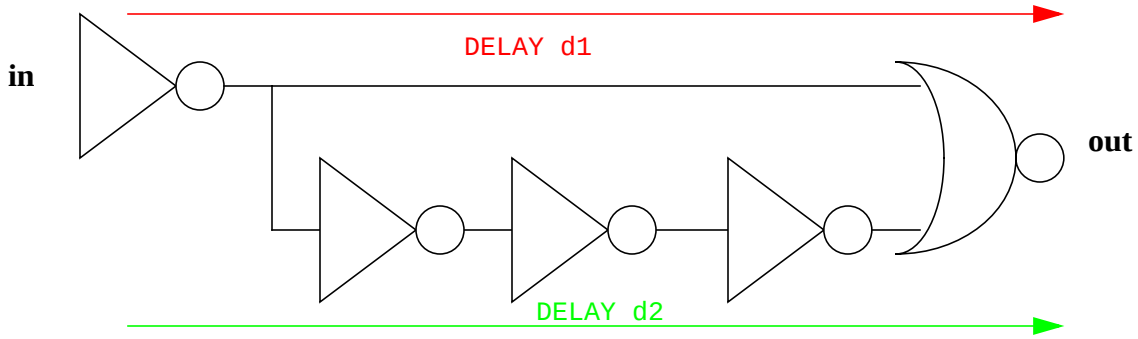
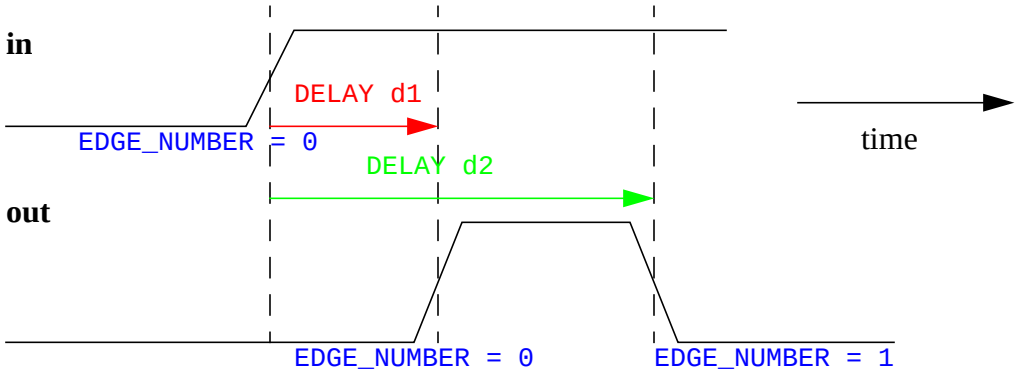
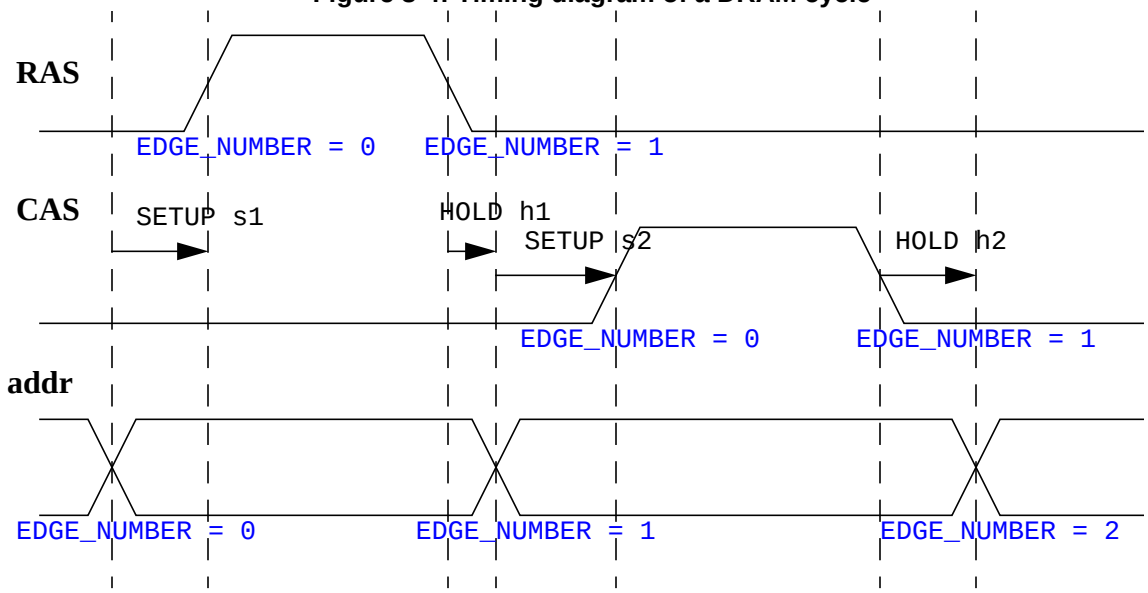


Figure 8-3: Timing diagram of a pulse generator




```
VECTOR ( 01 in -> 01 out -> 10 out ) {
   DELAY d1 {
         FROM { PIN = in; }
         TO { PIN = out; EDGE_NUMBER = 0; }
   }
   DELAY d2 {
         FROM { PIN = in; }
         TO { PIN = out; EDGE_NUMBER = 1; }
   }
}
```

Figure 8-4: Timing diagram of a DRAM cycle



```
VECTOR(?! addr ->01 RAS ->10 RAS ->?! addr ->01 CAS ->10 CAS ->?! addr){
   SETUP s1 {
         FROM { PIN = addr; EDGE_NUMBER = 0; }
         TO { PIN = RAS; EDGE_NUMBER = 0; }
   }
   HOLD h1 {
         FROM { PIN = RAS; EDGE_NUMBER = 1; }
         TO { PIN = addr; EDGE_NUMBER = 1; }
   }
   SETUP s2 {
         FROM { PIN = addr; EDGE_NUMBER = 1; }
         TO { PIN = CAS; EDGE_NUMBER = 0; }
   }
   HOLD h2 {
         FROM { PIN = CAS; EDGE_NUMBER = 1; }
         TO { PIN = addr; EDGE_NUMBER = 2; }
   }
}
```

8.2.5 Context of THRESHOLD definitions

The THRESHOLD statement may appear in the context of a FROM or TO container. In this case it specifies the applicable reference for the start and end point of the timing measurement, respectively.

Example:

```
SLEWRATE {
      FROM { THRESHOLD = 0.2; }
      TO { THRESHOLD = 0.8; }
}
```

The THRESHOLD statement may also appear in the context of a PIN. In this case, it specifies the applicable reference for the start or end point of timing measurements indicated by the PIN annotation inside a FROM or TO container, unless a THRESHOLD is specified explicitely inside the FROM or TO container.

If both RISE and FALL thresholds are specified and the switching direction of the applicable pin is clearly indicated in the context of a VECTOR, the RISE or FALL data shall be applied accordingly.

Example:

```
PIN A { THRESHOLD { RISE = 0.3;  FALL = 0.5; } }
PIN Z { THRESHOLD = 0.4; }
// other statements ...
VECTOR ( 01 A -> 10 Z ) {
   DELAY { FROM { PIN=A; } TO { PIN=Z; } }
// the applicable threshold for A is 0.3
// the applicable threshold for Z is 0.4
```

If thresholds are needed for exact definition of the model data, the FROM, TO containers shall each contain an arithmetic model for THRESHOLD.

A THRESHOLD statement may also appear as argument of an arithmetic model for timing measurements. In this case, it must contain a PIN annotation matching another PIN annotation in the FROM or TO container.

Example:

```
DELAY {
      FROM { PIN = A; THRESHOLD = 0.5; }
      TO { PIN = Z; }
      HEADER { THRESHOLD { PIN = Z; TABLE { 0.3 0.4 0.5 } }
      TABLE { 1.23 1.45 1.78 }
}
/* The measurement reference for pin A is always 0.5. The delay from A to
Z is expressed as a function of the measurement reference for pin Z. */
```

FROM and TO containers with THRESHOLD definitions yet without PIN annotations may appear within unnamed timing model definitions in the context of VECTOR, CELL, WIRE, SUBLIBRARY, or LIBRARY object for the purpose of specifying global threshold definitions for all timing models within scope of the definition. The following priorities apply:

1. THRESHOLD in the HEADER of the timing model
2. THRESHOLD in the FROM, TO statement within the timing model
3. THRESHOLD for timing model definition in the context of the same VECTOR
4. THRESHOLD within the PIN definition
5. THRESHOLD for timing model definition in the context of the same CELL or WIRE
6. THRESHOLD for timing model definition in the context of the same SUBLIBRARY
7. THRESHOLD for timing model definition in the context of the same LIBRARY
8. THRESHOLD for timing model definition outside LIBRARY

Example:

```
LIBRARY my_library {
   DELAY {
      FROM { THRESHOLD = 0.4;  }
      TO   { THRESHOLD = 0.4;  }
   }
   SLEWRATE {
      FROM { THRESHOLD { RISE = 0.2; FALL = 0.8; } }
      TO   { THRESHOLD { RISE = 0.8; FALL = 0.2; } }
   }
   CELL my_cell {
      PIN A { DIRECTION=input; THRESHOLD { RISE = 0.3; FALL = 0.5; } }
      PIN Z { DIRECTION=output; }
      VECTOR (01 A -> 10 Z) {
```

```
            DELAY { FROM { PIN=A; } TO { PIN=Z; } }
            SLEWRATE { PIN = Z; }
         }
      }
   }
   // delay is measured from A (threshold=0.3) to Z (threshold=0.4)
   // slewrate on Z is measured from threshold=0.8 to threshold=0.2.
```

8.3 Specification of timing models

Timing models shall be specified in the context of a VECTOR statement. Details are provided in the following subsections.

8.3.1 TEMPLATE for timing measurements and timing constraints

The following templates show a general timing measurement and a general timing constraint description, respectively, applicable for two pins.

```
TEMPLATE TIMING_MEASUREMENT {
   <timeKeyword> = <timeValue> {
      FROM {
         PIN=<fromPin>;
         THRESHOLD=<fromThreshold>;
         EDGE_NUMBER=<fromEdge>;
      }
      TO {
         PIN=<toPin>;
         THRESHOLD=<toThreshold>;
         EDGE_NUMBER=<toEdge>;
      }
   }
}

TEMPLATE TIMING_CONSTRAINT {
   LIMIT {
      <timeKeyword> {
         FROM {
            PIN=<fromPin>;
            THRESHOLD=<fromThreshold>;
            EDGE_NUMBER=<fromEdge>;
         }
         TO {
            PIN=<toPin>;
            THRESHOLD=<toThreshold>;
            EDGE_NUMBER=<toEdge>;
         }
         MIN = <timeValueMin>;
         MAX = <timeValueMax>;
      }
   }
}
```

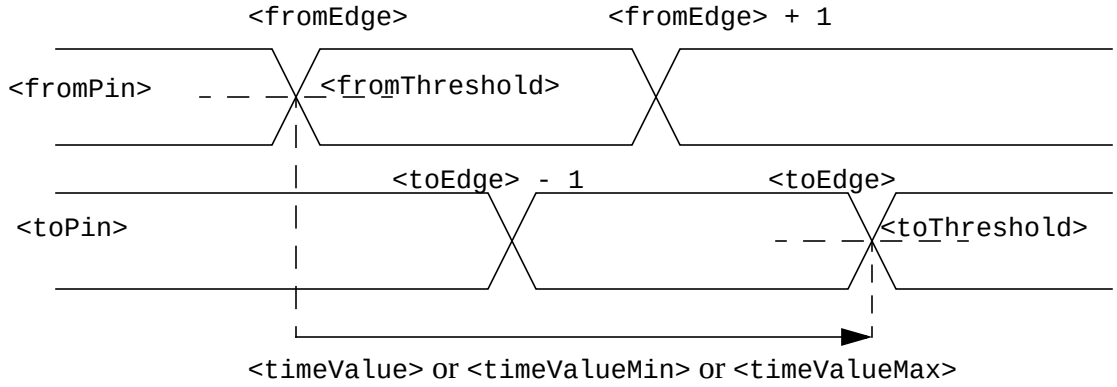
Notes:

For simplicity, trivial arithemetic models shown here. In general, a HEADER, TABLE, EQUATION construct could be used for calculation of <timeValue>, <timeValueMin> or <timeValueMax>.

A particular timing constraint does not necessarily contain both <timeValueMin> and <timeValueMax>.

The <fromThreshold> and <toThreshold> can be globally predefined as explained in section 8.2.4.

Figure 8-5: general timing measurement or timing constraint



The vector_expression in the context of which the <timeKeyword> appears shall contain at least two expressions of the type vector_single_event with the <fromPin> and <toPin>, respectively, as operand. The <fromEdge> and <toEdge> point to the respective vector_single_event.

The direction of the respective transition shall be identified by the respective edge_literal, i.e., the operator of the respective vector_single_event.

The temporal order of the LHS and RHS vector_single_event expressions within the vector_expression is indicated by a vector_binary operator.

The implications on the range of <timeValue> or <<refPin>> or <timeValueMax> are shown in the following table.

Table 8-9 Range of time value depending on VECTOR

LHS	operand	RHS	range of <timeValue> or <timeValueMin> or <timeValueMax>
<fromPin>	-> or ~>	<toPin>	positive
<toPin>	-> or ~>	<fromPin>	negative
<fromPin>	&>	<toPin>	positive or zero
<toPin>	&>	<fromPin>	negative or zero
<fromPin>	<->	<toPin>	positive or negative
<toPin>	<->	<fromPin>	positive or negative

Table 8-9 Range of time value depending on VECTOR

LHS	operand	RHS	range of <timeValue> or <timeValueMin> or <timeValueMax>
<fromPin>	<&>	<toPin>	positive or negative or zero
<toPin>	<&>	<fromPin>	positive or negative or zero

Note: This table does not apply for models with CALCULATION=incremental. Incremental values can always be positive, negative or zero.

8.3.2 Partially defined timing measurements and constraints

A partially defined timing measurement or timing constraint contains only a FROM statement or a TO statement, but not both. This construct can be used to specify measurements from any point to a specific point (only TO is specified) or from a specific point to any point (only FROM is specified).

This is summarized in the following table:

Table 8-10 Partially specified timing measurements and constraints

DIRECTION of PIN	FROM or TO specified	specified model applicable for
input	FROM only	cell timing arcs starting at this pin
input	TO only	interconnect timing arcs ending at this pin
output	FROM only	interconnect timing arcs starting at this pin
output	TO only	cell timing arcs ending at this pin

It is recommended to use the constructs for interconnect timing arcs only in conjunction with CALCULATION=incremental. In that case the <timeValue> or <timeValueMin> or <timeValueMax> from this model is added to the <timeValue> or <timeValueMin> or <timeValueMax> from timing arcs starting or ending at this pin, respectively. If the construct is used with CALCULATION=absolute, the timing model can only be used, if completely specified interconnect timing models are not available, and the result will not be accurate in general.

8.3.3 TEMPLATE for same-pin timing measurements and constraints

The following templates show a timing measurement and a timing constraint description, respectively, applicable for the same pin.

```
TEMPLATE SAME_PIN_TIMING_MEASUREMENT {
   <timeKeyword> = <timeValue> {
      PIN=<refPin>;
      EDGE_NUMBER=<refEdge>;
      FROM { THRESHOLD=<fromThreshold>; }
      TO { THRESHOLD=<toThreshold>; }
   }
}
TEMPLATE SAME_PIN_TIMING_CONSTRAINT {
   LIMIT {
      <timeKeyword> {
         PIN=<refPin>;
         EDGE_NUMBER=<refEdge>;
         FROM { THRESHOLD=<fromThreshold>; }
         TO { THRESHOLD=<toThreshold>; }
         MIN = <timeValueMin>;
         MAX = <timeValueMax>;
      }
   }
}
```

Depending on the `<timeKeyword>`, the `<timeValue>` or `<timeValueMin>` or `<timeValueMax>` is measured on the same `<refEdge>` or between `<refEdge>` and `<refEdge>` plus 1.
Only the `->` or `~>` operators are applicable between subsequent edges. Therefore the `<timeValue>` or `<timeValueMin>` or `<timeValueMax>` are positive by definition.

Note:

The `<fromThreshold>` and `<toThreshold>` can be globally predefined as explained in section 8.2.4. However, the THRESHOLD in the context of a PIN does not apply for *`SAME_PIN_TIMING_MEASUREMENT`* or *`SAME_PIN_TIMING_CONSTRAINT`*, since the `<refPin>` is not within a FROM or TO statement.

8.3.4 Absolute and incremental evaluation of timing models

As mentioned in the previous sections, the calculation models for *`TIMING_MEASUREMENT`*, *`TIMING_CONSTRAINT`*, *`SAME_PIN_TIMING_MEASUREMENT`*, *`SAME_PIN_TIMING_CONSTRAINT`* may have the annotation `CALCULATION=absolute` (default) or `CALCULATION=incremental`. These annotations are only relevant, if there exists more than one calculation model for the same timing arc.

Calculation models for the same timing arc with `CALCULATION=absolute` must be within the context of mutually exclusive VECTORs. The `vector_expression` specifies, which model to use under which condition.

Example:

```
VECTOR ( (01 A -> 01 Z) && B & !C ) {
   DELAY { CALCULATION=absolute; FROM { PIN=A; } TO { PIN=Z; }
   /* fill in HEADER, TABLE */ }
}
VECTOR ( (01 A -> 01 Z) && !B & C ) {
   DELAY { CALCULATION=absolute; FROM { PIN=A; } TO { PIN=Z; }
   /* fill in HEADER, TABLE */ }
}
```

The vectors `( (01 A -> 01 Z) && B & !C )` and `( (01 A -> 01 Z) && !B & C )` are mutually exclusive. They describe the same timing arc with two mutually exclusive conditions.

In the case of a VECTOR containing a calculation model for a timing arc with `CALCULATION=incremental`, there must be another VECTOR with a calculation model for the same timing arc with `CALCULATION=absolute`, and both vectors must be compatible. The `vector_expression` of the latter must be necessarily true, when the `vector_expression` of the former is true.

Example:

```
VECTOR (01 A -> 01 Z) {
   DELAY { CALCULATION=absolute; FROM { PIN=A; } TO { PIN=Z; }
   /* fill in HEADER, TABLE */ }
}
VECTOR ( (01 A -> 01 Z) && B & !C ) {
   DELAY { CALCULATION=incremental; FROM { PIN=A; } TO { PIN=Z; }
   /* fill in HEADER, TABLE */ }
}
VECTOR ( (01 A -> 01 Z) && !B & C ) {
   DELAY { CALCULATION=incremental; FROM { PIN=A; } TO { PIN=Z; }
   /* fill in HEADER, TABLE */ }
}
```

The vectors `( (01 A -> 01 Z) && B & !C )` and `( (01 A -> 01 Z) && !B & C )` are both compatible with the vector `(01 A -> 01 Z)` and mutually exclusive with each other. The latter describe the same timing arc with two mutually exclusive conditions. The former describes the same timing arc without conditions. This modeling style is useful for timing analysis tools with or without support for conditions. The vectors with condition, if supported, add accuracy to the calculation. However, the vector without condition is always available for basic calculation.

8.3.5 RISE and FALL submodels

For timing models in the context of a VECTOR, submodels for RISE and FALL are only applicable, if the `vector_expression` does not specify the switching direction of the referenced PIN and EDGE_NUMBER. This is the case, when symbolic `vector_unary` operators are used, i.e., "`?!`", "`??`", "`?*`", or "`*?`" instead of "`01`", "`10`" etc.

For *`SAME_PIN_TIMING_MEASUREMENT`* or *`SAME_PIN_TIMING_CONSTRAINT`*, the RISE and FALL submodels apply for the `<refEdge>`.

For a partially specified *`TIMING_MEASUREMENT`* or *`TIMING_CONSTRAINT`*, the RISE and FALL submodels apply for the `<fromEdge>` or `<toEdge>`, whichever is specified.

For a completely specified *TIMING_MEASUREMENT* or *TIMING_CONSTRAINT*, it is not possible to apply a RISE and FALL submodel for both <fromEdge> and <toEdge>. The vector_unary operator should specify the switching direction for at least one edge. If the switching direction for both edges is unspecified, the RISE and FALL submodel shall apply for the <toEdge>.

Example:

```
VECTOR ( 01 CLK -> ?! Q ) {
   DELAY { FROM { PIN = CLK; } TO { PIN = Q; }
      RISE = 0.76; FALL = 0.58;
   }
}
// If Q is a scalar pin, the following construct is equivalent:
VECTOR ( 01 CLK -> 01 Q ) {
   DELAY = 0.76 { FROM { PIN = CLK; } TO { PIN = Q; } }
}
VECTOR ( 01 CLK -> 10 Q ) {
   DELAY = 0.58 { FROM { PIN = CLK; } TO { PIN = Q; } }
}
```

8.3.6 TIME

The <timeKeyword> TIME is describes a general *TIMING_MEASUREMENT* or *TIMING_CONSTRAINT* without implying any particular relationship between <fromEdge> and <toEdge>.

In general, <fromPin> and <toPin> refer to two different pins. However, it is legal that both <fromPin> and <toPin> refer to the same pin.

Default value for <fromEdge> and <toEdge> shall be 0.

8.3.7 DELAY

The <timeKeyword> DELAY describes a *TIMING_MEASUREMENT* implying a causal relationship between <fromEdge> and <toEdge>.

Usually <fromPin> refers to an input pin and <toPin> refers to an output pin. However, it is legal that both <fromPin> and <toPin> refer to an output pin.

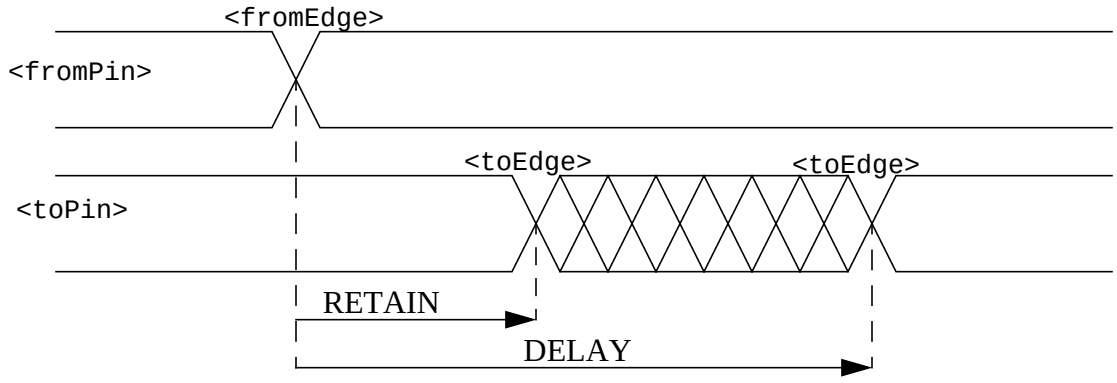
Default value for <fromEdge> and <toEdge> shall be 0, unless the DELAY statement appears in conjunction with a RETAIN statement within the context of the same VECTOR.

8.3.8 RETAIN

The <timeKeyword> RETAIN describes a *TIMING_MEASUREMENT* implying a causal relationship between <fromEdge> and <toEdge> in the same way as DELAY.

RETAIN is used to describe the elapsed time until the output changes its old value, whereas DELAY is used to describe the eleapsed time until the output settles to a stable new value.

Figure 8-6: RETAIN and DELAY



When DELAY appears in conjunction with RETAIN, the <fromEdge> for both measurements shall be the same. The <toEdge> for DELAY shall be the <toEdge> for RETAIN plus 1.

Default value for <fromEdge> and <toEdge> for RETAIN shall be 0. Default value for <toEdge> for DELAY shall be 1.

8.3.9 SLEWRATE

The <timeKeyword> SLEWRATE describes a *SAME_PIN_TIMING_MEASUREMENT* for <timeValue> defining the duration of a signal transition or a fraction thereof.

The SLEWRATE applies for the <refEdge> on the <refPin>. The default value for <refEdge> shall be 0.

8.3.10 SETUP

The <timeKeyword> SETUP describes a *TIMING_CONSTRAINT* for <timeValueMin> defining the minimum stable time required for the data signal on the <fromPin> before it is sampled by the strobe signal on the <toPin>.

The <fromPin> usually is an input pin with SIGNALTYPE=data. The <toPin> is an input pin with SIGNALTYPE=clock.

Default value for <fromEdge> and <toEdge> for SETUP shall be 0.

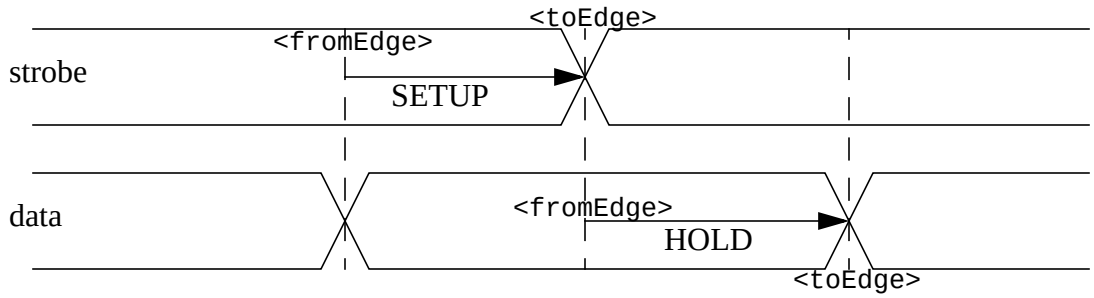
8.3.11 HOLD

The <timeKeyword> HOLD describes a *TIMING_CONSTRAINT* for <timeValueMin> defining the minimum stable time required for the data signal on the <toPin> after it is sampled by the strobe signal on the <fromPin>.

The <toPin> usually is an input pin with SIGNALTYPE=data. The <fromPin> is an input pin with SIGNALTYPE=clock.

Default value for <fromEdge> shall be 0. Default value for <toEdge> shall be 0, unless HOLD appears in conjunction with SETUP in the context of the same VECTOR. In that case, the default value for <toEdge> shall be 1.

Figure 8-7: SETUP and HOLD

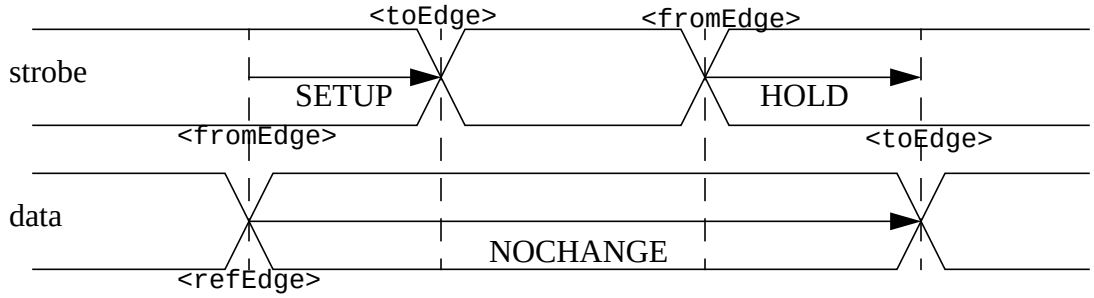


The <timeValueMin> for SETUP or the <timeValueMin> for HOLD with respect to the same strobe may be negative. However, the sum of both values must be positive. The sum represents the minimum duration of a valid data signal around a strobe signal.

8.3.12 NOCHANGE

The <timeKeyword> NOCHANGE describes a *SAME_PIN_TIMING_CONSTRAINT* defining the requirement for a stable signal on a pin subjected to SETUP and HOLD on subsequent edges of a strobe signal.

Figure 8-8: NOCHANGE, SETUP and HOLD



The NOCHANGE applies between the <refEdge> and the subsequent edge, i.e., <refEdge> plus 1 on the <refPin>. The default value for <refEdge> shall be 0.

When NOCHANGE appears in conjunction with SETUP and HOLD within the context of the same VECTOR, the default value for <fromEdge> and <toEdge> of SETUP shall be 0, the default value for <fromEdge> and <toEdge> of HOLD shall be 1.

8.3.13 RECOVERY

The <timeKeyword> RECOVERY describes a *TIMING_CONSTRAINT* for <timeValueMin> defining the minimum stable time required for an asynchronous control signal on the <fromPin> to be inactive before a strobe signal on the <toPin> can be active.

The <fromPin> usually is an input pin with SIGNALTYPE=set|clear. The <toPin> is an input pin with SIGNALTYPE=clock.

Default value for <fromEdge> and <toEdge> for RECOVERY shall be 0.

8.3.14 REMOVAL

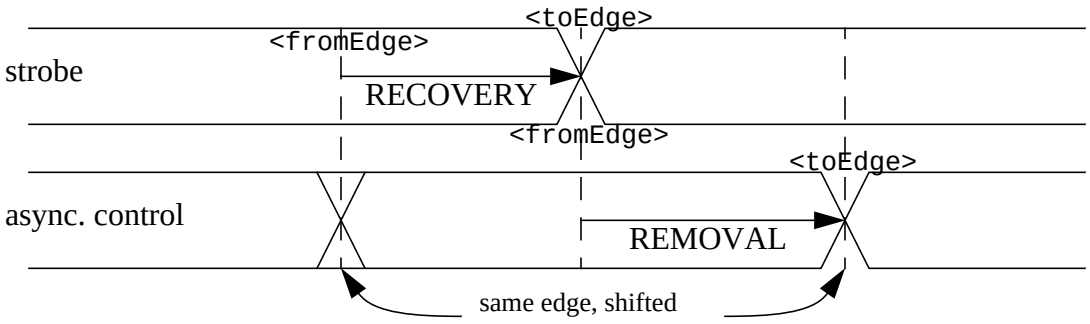
The <timeKeyword> REMOVAL describes a *TIMING_CONSTRAINT* for <timeValueMin> defining the minimum stable time required for an asynchronous control signal on the <toPin> to remain active after overriding a strobe signal on the <fromPin>.

The <toPin> usually is an input pin with SIGNALTYPE=set|clear. The <fromPin> is an input pin with SIGNALTYPE=clock.

Default value for <fromEdge> and <toEdge> for REMOVAL shall be 0.

REMOVAL may appear in conjunction with RECOVERY within the context of the same VECTOR.

Figure 8-9: RECOVERY and REMOVAL



The <timeValueMin> for RECOVERY or the <timeValueMin> for REMOVAL with respect to the same strobe may be negative. However, the sum of both values must be positive. The sum represents the time window around the clock signal during which the asynchronous control signal must not switch.

8.3.15 SKEW between two signals

The <timeKeyword> SKEW describes a *TIMING_CONSTRAINT* for <timeValueMax> defining the maximum allowed time separation between <fromEdge> on <fromPin> and <toEdge> on <toPin>.

Default value for <fromEdge> and <toEdge> for SKEW shall be 0.

8.3.16 SKEW between multiple signals

SKEW may also describe the maximum time distortion between signals on multiple pins. In this case, a list of pins appears in form of a multivalue annotation. No FROM or TO container appear.

```
SKEW {
   PIN { <pinList> }
   EDGE_NUMBER { <edgeList> }
   <skewData>
}
```

Default for EDGE_NUMBER in SKEW for multiple signals shall be a list of 0s.

A special case of multiple pins is a single bus. ALF 1.1 allows the following syntax:

```
SKEW { PIN = my_bus_pin[8:1]; }
```

as opposed to

```
SKEW { PIN { my_bus_pin[8:1] } }
```

Shall we still allow the old form for backward compatibility?

8.3.17 PULSEWIDTH

The <timeKeyword> PULSEWIDTH describes a *SAME_PIN_TIMING_CONSTRAINT* for <timeValueMin> defining the minimum duration of the signal before changing state.

The PULSEWIDTH statement is applicable for both input and output pins. In the case of an input pin, it represents a timing check against the minimum duration. In case of an output pin it represents the minimum possible duration of the signal.

The PULSEWIDTH applies between the <refEdge> and the subsequent edge, i.e., <refEdge> plus 1 on the <refPin>. The default value for <refEdge> shall be 0.

8.3.18 PERIOD

The <timeKeyword> PERIOD describes a *SAME_PIN_TIMING_CONSTRAINT* for <timeValueMin> defining the minimum time between subsequent repetitions of a signal. Because of periodicity, <fromThreshold> and <toThreshold> are not required. Therefore FROM and TO statements do not appear.

If the VECTOR describes a completely specified event sequence, <refPin> and <refEdge> are not required. PERIOD applies for the complete event sequence.

If the VECTOR describes a partially specified event sequence, involving the "~>" operator, <refPin> and <refEdge> are required.

8.3.19 JITTER

The <timeKeyword> JITTER describes a *SAME_PIN_TIMING_MEASUREMENT* for <timeValue> defining the actual uncertainty of arrival time for a periodical signal at a pin.

The JITTER applies for the <refEdge> on the <refPin>. The default value for <refEdge> shall be 0.

Threshold definitions, i.e., <fromThreshold> or <toThreshold> do not apply.

A limit for tolerable jitter at a pin can be expressed using the LIMIT construct, as shown in the template for *SAME_PIN_TIMING_CONSTRAINT*.

8.4 VIOLATION container

A VIOLATION statement may appear within an ILLEGAL statement (see section ??) and also within a *TIMING_CONSTRAINT* or a *SAME_PIN_TIMING_CONSTRAINT*. The VIOLATION statement may contain the BEHAVIOR object (Section 13.17), since the behavior in case of timing constraint violation cannot be described in the FUNCTION. The VIOLATION statement may also contain the following annotations:

Table 8-11 : Annotations within VIOLATION

Keyword	Value type	Description
MESSAGE_TYPE	string	specifies the type of the message. It can be one of information, warning or error.
MESSAGE	string	specifies the message itself.

Example:

```
VECTOR (01 d <&> 01 cp) {
      SETUP {
            VIOLATION {
                        MESSAGE_TYPE = error;
                        MESSAGE = “setup violation 01 d <&> 01 cp“;
                        BEHAVIOR {q = 'bx;}
            }
      }
}
```

8.5 EARLY and LATE container

The EARLY and LATE containers define the boundaries of timing measurements in one single analysis. Only applicable to DELAY and SLEWRATE. Both of them must appear in both containers.

The quadruple

```
EARLY {
      DELAY { FROM {...} TO { ...} /* data */ }
      SLEWRATE { /* data */ }
LATE {
      DELAY { FROM {...} TO { ...} /* data */ }
      SLEWRATE { /* data */ }
```

is used to calculate the envelope of the timing waveform at the TO point of a delay arc with respect to the timing waveform at the FROM point of a delay arc.

The EARLY DELAY is of course a smaller number (or a set of smaller numbers) than the LATE DELAY. However, the EARLY SLEWRATE is not necessarily smaller than the LATE SLEWRATE, since the SLEWRATE of the EARLY signal may be larger than the SLEWRATE of the LATE signal.

8.6 Environmental dependency for electrical data

The following table describes the arguments for arithmetic models to describe environmental dependency:

Table 8-12 : Environmental data

Annotation string	Value type	Description
DERATE_CASE	string	derating case, i.e. combination of process, supply voltage, temperature
PROCESS	string	process corner
TEMPERATURE	number	environmental temperature

8.6.1 PROCESS

The following identifiers can be used as predefined process corners:

`?n?p` process definition with transistor strength

where `?` can be

`s` strong
`w` weak

The possible process name combinations are

Table 8-13 : Predefined process names

Process name	Description
snsp	strong NMOS, strong PMOS
snwp	strong NMOS, weak PMOS
wnsp	weak NMOS, strong PMOS
wnwp	weak NMOS, weak PMOS

8.6.2 DERATE_CASE

The following identifiers can be used as predefined derating cases:

`nom`	nominal case
`bc?`	prefix for best case
`wc?`	prefix for worst case

where `?` can be

`com`	suffix for commercial case
`ind`	suffix for industrial case
`mil`	suffix for military case

The possible derating case combinations are defined in Table 3-62.

Table 8-14 : Predefined derating cases

Derating case	Description
`bccom`	best case commercial
`bcind`	best case industrial
`bcmil`	best case military
`wccom`	worst case commercial
`wcind`	worst case military
`wcmil`	worst case military

8.6.3 Lookup table without interpolation

The PROCESS or DERATE_CASE can be used in a TABLE within the HEADER of an arithmetic model for electrical data, e.g. DELAY. Data can not be interpolated in the dimension of this table.

Example:

```
DELAY {
      UNIT = ns;
      HEADER {
            PROCESS { TABLE { nom snsp wnwp } }
      }
      TABLE { 0.4  0.3  0.6 }
}
```

The DELAY is 0.4 ns for nominal process, 0.3 ns for `snsp`, 0.6 ns for `wnwp`. A delay "in-between" `snsp` and `wnwp` can not be interpolated.

8.6.4 Lookup table for process-or derating-case coefficients

A nested arithmetic model construct can be used to describe lookup tables for coefficients, based on PROCESS or DERATE_CASE. These coefficients can be used in an EQUATION to calculate electrical data, for example DELAY.

Example:

```
DELAY {
      UNIT = ns;
      HEADER {
            PROCESS { HEADER { nom snsp wnwp } TABLE {0.0 -0.25 0.5} }
      }
      EQUATION { (1 + PROCESS)*0.4 }
}
```

The equation uses the PROCESS coefficient 0.0 for nominal, -0.25 for `snsp`, 0.5 for `wnwp`. Therefore the DELAY is 0.4 ns for nominal process, 0.3 ns for `snsp`, 0.6 ns for `wnwp`. Conceivably, the DELAY can be calculated for any value of the coefficient.

8.6.5 TEMPERATURE

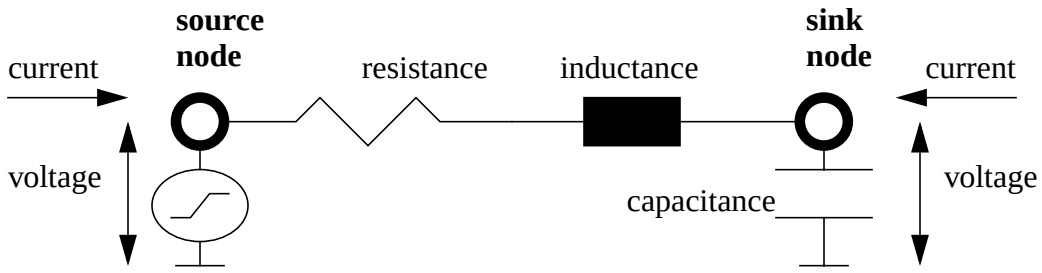
TEMPERATURE can be used as argument in the HEADER of an arithmetic model for timing or electrical data. It can also be used as an arithmetic model with DERATE_CASE as argument, in order to describe what temperature applies for the specified derating case.

8.7 PIN-related arithmetic models for electrical data

8.7.1 Principles

Arithmetic models for electrical data can be associated with a pin of a cell. Their meaning is illustrated in the following picture.

Figure 8-10: General representation of electrical models around a pin



A pin is represented as a source node and a sink node. For pins with DIRECTION=input, the source node is externally accessible. For pins with DIRECTION=output, the sink node is externally accessible.

8.7.2 CAPACITANCE, RESISTANCE, INDUCTANCE

RESISTANCE and INDUCTANCE apply between source and sink node. CAPACITANCE applies between sink node and ground. Per default, the values for resistance, inductance and capacitance shall be zero.

8.7.3 VOLTAGE, CURRENT

VOLTAGE and CURRENT may be measured at either source or sink node, depending on which node is externally accessible. However, a voltage source may only be connected to a source node. The sense of measurement for voltage shall be from node to ground. The sense of measurement for current shall be *into* the node.

8.7.4 PIN-related timing models

SAME_PIN_TIMING_MEASUREMENT and *SAME_PIN_TIMING_CONSTRAINT* (see section ??) are pin-related timing models. They are defined with reference to the externally accessible node.

8.7.5 Submodels for RISE, FALL, HIGH, LOW

RISE, FALL contain data characterized in transient measurements. HIGH, LOW contain data characterized in static measurements.

```
<modelKeyword> { RISE=<modelValueRise>; FALL=<modelValueFall>; }
<modelKeyword> { HIGH=<modelValueHigh>;  LOW=<modelValueLow>; }
```

It is generally not required that both RISE and FALL or both HIGH and LOW, respectively, appear as arithmetic submodel.

HIGH and LOW qualify states with the logic value 1 and 0, respectively. RISE and FALL qualify transitions between states with initial logic value 0 and 1, respectively and final value 1 and 0, respectively. For other states and their mapping to logic values, see section ??. If the arithmetic model is within the scope of a vector which describes the logic values without ambiguity, the use of RISE, FALL, HIGH, LOW does not apply.

HIGH, LOW, RISE, FALL applies for all pin-related arithmetic models with the following exceptions:

RISE and FALL does not apply for VOLTAGE.
HIGH and LOW does not apply for *SAME_PIN_TIMING_MEASUREMENT* and *SAME_PIN_TIMING_CONSTRAINT*.

Note: For states that cannot be mapped to logic 1 or 0, RISE, FALL, HIGH, LOW cannot be used. The use of VECTOR with unambiguous description of the relevant states is mandatory in such cases.

8.7.6 Context-specific semantics

An arithmetic model for VOLTAGE, CURRENT, SLEWRATE, RESISTANCE, INDUCTANCE, CAPACITANCE may be associated with a PIN in one of the following ways.

1. model in the context of a PIN

Example:

```
PIN my_pin {
   CAPACITANCE = 0.025;
```

2. model in the context of a CELL, WIRE or VECTOR with PIN annotation

Example:

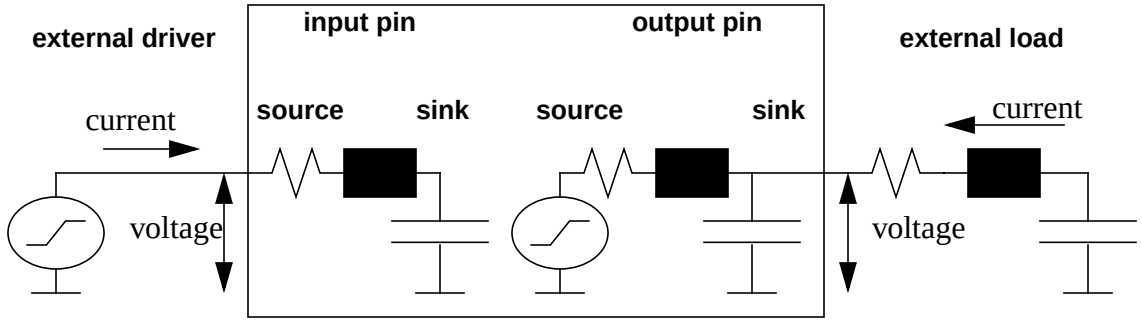
```
VOLTAGE = 1.8 { PIN = my_pin; }
```

The model in the context of a PIN shall be used, if the data is completely confined to the pin. That means, no argument of the model shall make reference to any pin, since such reference would imply an external dependency. A model with dependency only on environmental data not associated with a pin (e.g. TEMPERATURE, PROCESS, DERATE_CASE) may be described within the context of the PIN.

A model with dependency on external data applied to a pin (e.g. load capacitance) shall be described outside the context of the PIN, using a PIN annotation. In particular, if the model involves a dependency on logic state or logic transition of other PINs, the model shall be described within the context of a VECTOR.

The following figure illustrates electrical models associated with input and output pins.

Figure 8-11: Electrical models associated with input and output pins



The following tables define, in which way the models shall be associated with the pin, depending on the context.

Table 8-15 Direct assocation of models with a PIN

Model	Model in context of PIN	Model in context of CELL, WIRE, VECTOR with PIN annotation
CAPACITANCE	pin self-capacitance	externally controlled capacitance at the pin, e.g. voltage-dependent
INDUCTANCE	pin self-inductance	externally controlled inductance at the pin, e.g. voltage-dependent
RESISTANCE	pin self-resistance	externally controlled resistance at the pin, e.g. voltage-dependent in context of VECTOR for timing-arc specific driver resistance
VOLTAGE	operational voltage measured at pin	externally controlled voltage at the pin
CURRENT	operational current measured into pin	externally controlled current into pin

Table 8-15 Direct assocation of models with a PIN

Model	Model in context of PIN	Model in context of CELL, WIRE, VECTOR with PIN annotation
SAME_PIN_TIMING_MEASUREMENT	for model definition, default etc., not for the timing arc	in context of VECTOR for timing arc, other context for definition, default etc.
SAME_PIN_TIMING_CONSTRAINT	for model definition, default etc., not for the timing arc	in context of VECTOR for timing arc, other context for definition, default etc.

Table 8-16 External association of models with a PIN

Model / Context	LIMIT within PIN or with PIN annotation	model argument with PIN annotation
CAPACITANCE	min or max limit for applicable load	load for model characterization
INDUCTANCE	min or max limit for applicable load	load for model characterization
RESISTANCE	min or max limit for applicable load	load for model characterization
VOLTAGE	min or max limit for applicable voltage	voltage for model characterization
CURRENT	min or max limit for applicable current	current for model characterization
SAME_PIN_TIMING_MEASUREMENT	currently applicable for min or max limit for SLEWRATE	stimulus with SLEWRATE for model characterization
SAME_PIN_TIMING_CONSTRAINT	N/A, since the keyword means a min or max limit by itself	N/A

Example:

```
CELL my_cell {
      PIN pin1 { DIRECTION=input; CAPACITANCE = 0.05; }
      PIN pin2 { DIRECTION=output; LIMIT { CAPACITANCE { MAX=1.2; } } }
      PIN pin3 { DIRECTION=input; }
      PIN pin4 { DIRECTION=input; }
      CAPACITANCE {
            PIN=pin3;
            HEADER { VOLTAGE { PIN=pin4; } }
            EQUATION { 0.25 + 0.34*VOLTAGE }
      }
}
```

The capacitance on pin1 is 0.05. The maximum allowed load capacitance on pin2 is 1.2. The capacitance on pin3 depends on the voltage on pin4.

8.8 Other PIN-related arithmetic models

8.8.1 DRIVE_STRENGTH

DRIVE_STRENGTH is a unit-less, abstract measure for the drivability of a PIN. It may be used as a substitute of driver RESISTANCE. The higher the DRIVE_STRENGTH, the lower the driver RESISTANCE. However, DRIVE_STRENGTH may only be used within a coherent

system of calculation models, since it does not represent an absolute quantity, as opposed to RESISTANCE. For example, the weakest driver of a library may have drive strength 1, the next stronger driver may have drive strength 2 and so forth. This does not necessarily mean that the resistance of the stronger driver is exactly half of the resistance of the weaker driver.

To relate the quantity DRIVE_STRENGTH across technology libraries, an arithmetic model for conversion from DRIVE_STRENGTH to RESISTANCE may be given.

Example:

```
SUBLIBRARY high_speed_library {
   RESISTANCE {
      HEADER { DRIVE_STRENGTH } EQUATION { 800 / DRIVE_STRENGTH }
   }
   CELL high_speed_std_driver {
      PIN Z { DIRECTION = output; DRIVE_STRENGTH = 1; }
   }
}
SUBLIBRARY low_power_library {
   RESISTANCE {
      HEADER { DRIVE_STRENGTH } EQUATION { 1600 / DRIVE_STRENGTH }
   }
   CELL low_power_std_driver {
      PIN Z { DIRECTION = output; DRIVE_STRENGTH = 1; }
   }
}
```

Drive strength 1 in the high speed library corresponds to 800 ohm. Drive strength 1 in the low power library corresponds to 1600 ohm. Note: any particular arithmetic model for RESISTANCE in either library will locally override the conversion formula from drive strength to resistance.

8.8.2 SWITCHING_BITS

The quantity SWITCHING_BITS applies only for bus pins. The range is from 0 to the width of the bus. Usually, the quantity SWITCHING_BITS is not calculated by an a arithmetic model, since the number of switching bits on a bus depends on the functional specification rather than the electrical specification. However, SWITCHING_BITS can be used as argument in the HEADER of an arithmetic model to calculate electrical quantities, for instance energy consumption.

Example:

```
CELL my_rom {
   PIN [3:0] addr { DIRECTION=input;  SIGNALTYPE=address; }
   PIN [7:0] dout { DIRECTION=output; SIGNALTYPE=data; }
   VECTOR ( ?! addr -> ?! dout ) {
      ENERGY {
         HEADER {
            SWITCHING_BITS addr_bits { PIN = addr; }
            SWITCHING_BITS dout_bits { PIN = dout; }
         }
         EQUATION { 0.45*LOG(addr_bits) + 2.6*dout_bits }
      }
   }
}
```

The energy consumption of `my_rom` depends on the number of switching data bits and on the logarithm of the number of switching address bits.

8.9 Annotations for arithmetic models

8.9.1 MEASUREMENT annotation

Arithmetic models describing analog measurements (see Table 8-5) can have a MEASUREMENT annotation. This annotation indicates the type of measurement used for the computation in arithmetic model.

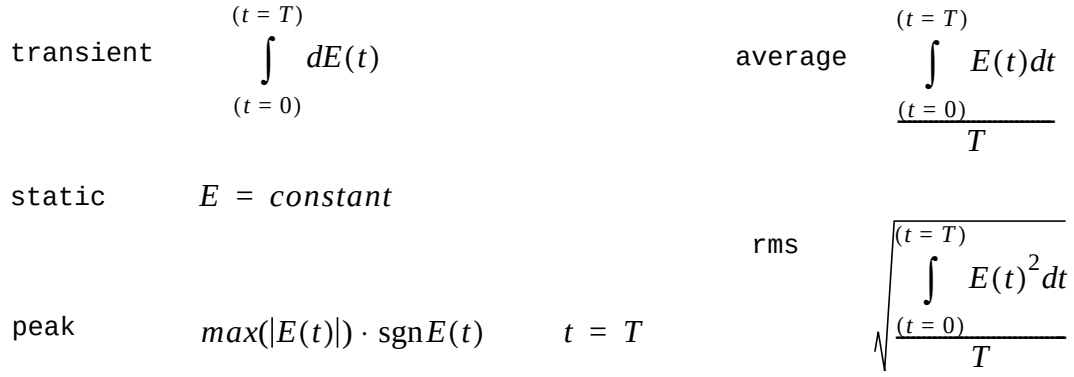
```
MEASUREMENT = string ;
```

The string can take the following values:

Table 8-17 : MEASUREMENT annotation

Annotation string	Description
`transient`	measurement is a transient value
`static`	measurement is a static value
`average`	measurement is an average value
`rms`	measurement is an root mean square value
`peak`	measurement is a peak value

Figure 8-12: Mathematical definitions for MEASUREMENT annotations



Examples:

transient measurement of ENERGY
static measurement of VOLTAGE, CURRENT, POWER
average measurement of POWER, CURRENT
rms measurement of POWER, CURRENT
peak measurement of VOLTAGE, CURRENT, POWER

8.9.2 TIME and FREQUENCY annotation

Arithmetic models with certain values of MEASUREMENT annotation can also have *either* TIME *or* FREQUENCY as annotations.

The semantics are defined as follows:

Table 8-18 : Semantic interpretation of MEASUREMENT, TIME or FREQUENCY annotation

MEASUREMENT annotation	Semantic meaning of TIME annotation	Semantic meaning of FREQUENCY annotation
`transient`	integration of analog measurement is done during that time window	integration of analog measurement is repeated with that frequency
`static`	N/A	N/A
`average`	average value is measured over that time window	average value measurement is repeated with that frequency
`rms`	root-mean-square value is measured over that time window	root-mean-square measurement is repeated with that frequency
`peak`	peak value occurs at that time (only within context of VECTOR)	observation of peak value is repeated with that frequency

In the case of `average`, `rms`, the interpretation FREQUENCY = 1 / TIME is valid. Either one or the other annotation shall be mandatory. The values for `average` measurements and for `rms` measurements scale linearly with FREQUENCY and 1 / TIME, respectively.

In the case of `transient`, `peak`, the interpretation FREQUENCY = 1 / TIME is not valid. Either one or the other annotation shall be optional. The values do not necessarily scale with TIME or FREQUENCY. The TIME or FREQUENCY annotations for `transient` measurements are purely informational.

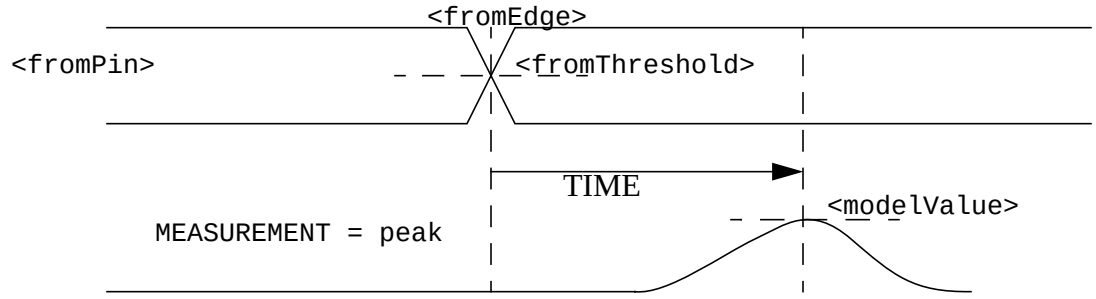
8.9.3 TIME to peak measurement

For a model in the context of a VECTOR, featuring a `peak` measurement, the TIME annotation shall define the time between a reference event within the `vector_expression` and the instant when the peak value occurs.

For that purpose, either the FROM or the TO statement shall be used in the context of the TIME annotation, containing a PIN annotation and, if necessesary, a THRESHOLD and / or an EDGE_NUMBER annotation.

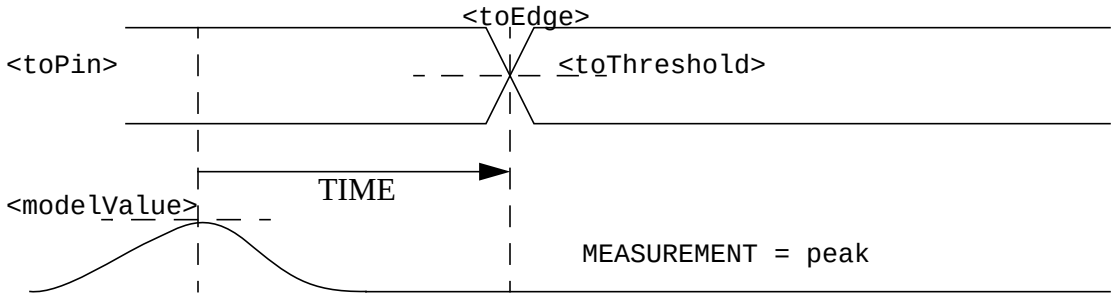
If the FROM statement is used, the start point shall be the reference event, and the end point shall be the occurence time of the peak.

Figure 8-13: Illustration of time to peak using FROM statement



If the TO statement is used, the start point shall be the occurence time of the peak, and the end point shall be the reference event.

Figure 8-14: Illustration of time to peak using TO statement



Example:

```
VECTOR (01 A -> 01 B -> 10 B) {
   CURRENT peak1 = 10.8 {
      PIN = Vdd;
      MEASUREMENT = peak;
      TIME = 3.0 { UNIT=ns; FROM { PIN=A; EDGE_NUMBER=0; } }
   }
   CURRENT peak2 = 12.3 {
      PIN = Vdd;
      MEASUREMENT = peak;
      TIME = 2.0 { UNIT=ns; TO { PIN=B; EDGE_NUMBER=1; } }
   }
}
```

The peak with magnitude 10.8 occurs 3 nanoseconds after the event "`01 A`".

The peak with magnitude 12.3 occurs 2 nanoseconds before the event "`10 B`".

8.10 Waveform description

8.10.1 Principles

In order to describe an arithmetic model representing a waveform, TIME must be be an argument in the HEADER. Other arguments may appear in the HEADER as well. The model can be described as a TABLE or EQUATION.

Example for TABLE:

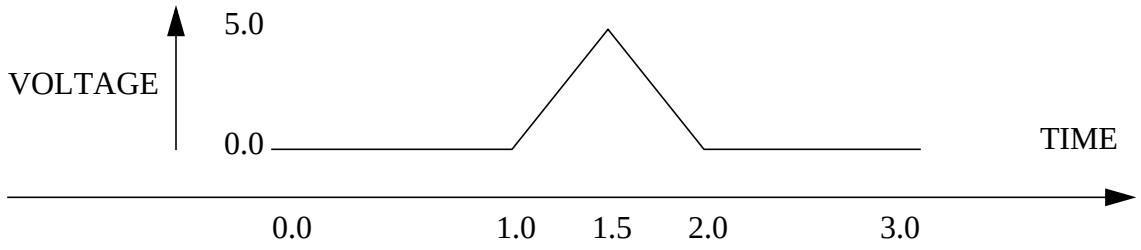
```
VOLTAGE {
   HEADER {
      TIME {
         UNIT = ns;
         INTERPOLATION=linear;
         TABLE { 0.0  1.0  1.5  2.0  3.0 }
      }
   }
   TABLE { 0.0  0.0  5.0  0.0  0.0 }
}
```

Example for EQUATION:

```
VOLTAGE {
   HEADER {
      TIME { UNIT = ns; }
   }
   EQUATION {
      (TIME < 1.0) ? 0 :
      (TIME < 1.5) ? 5.0*(TIME - 1.0) :
      (TIME < 2.0) ? 5.0*(2.0 - TIME) :
      0.0
   }
}
```

Both models describe the same piece-wise linear waveform.

Figure 8-15: Illustration of a piece-wise linear waveform



If the model is within the context of a VECTOR, either the FROM or the TO statement may be used in the context of TIME, pointing to a reference event which occurs at TIME = 0 relative to the waveform description. See previous section for the definition of start and end points of measurements.

Example:

```
VECTOR (01 A -> 01 B -> 10 B) {
   VOLTAGE {
      HEADER {
         TIME {
            FROM { PIN = B; EDGE_NUMBER = 1; }
            TABLE { 0.0  1.0  1.5  2.0  3.0 }
// alternative description:
//          TO { PIN = B; EDGE_NUMBER = 1; }
//          TABLE { -3.0  -2.0  -1.5  -1.0  0.0 }
         }
      }
      TABLE { 0.0  0.0  5.0  0.0  0.0 }
   }
}
```

Note: It is recommended to use the FROM statement. If the TO statement is used, TIME is measured backwards, which is counter-intuitive. For dynamic analysis, it is also recommended to use the last event in the `vector_expression` as reference. Otherwise the analysis tool must remember the occurence time of previous events in order to place the waveform into the context of absolute time.

8.10.2 Annotations within a waveform

The MEASUREMENT annotation `transient` shall apply as a default for waveforms.

The FREQUENCY annotation is applicable, specifying a repetition frequency of the waveform. The following boundary restrictions are imposed in order to make the waveform repeatable:

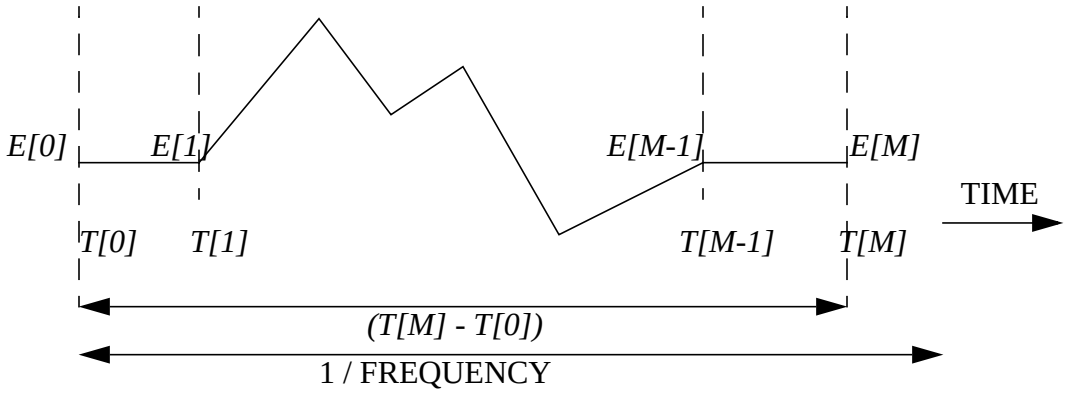
- The initial value and the final value of waveform must be the same.
- The extrapolation beyond the initial and the final value of the waveform must yield the

same result. Per consequence, the first, second, last and second last point of the waveform must be the same.

- The time window between the first and the last measurement must be smaller or equal to 1 / FREQUENCY

This is illustrated in the following figure.

Figure 8-16: TIME and FREQUENCY in a waveform



The MEASUREMENT annotation `average` , `rms` for waveforms (see ALF 1.1, chapter 3.6.10.6) are almost rendered obsolete by the INTERPOLATION annotation. The `average` , `rms` waveforms were previously defined as piece-wise constant waveforms representing piece-wise average or rms value, respectively. Using INTERPOLATION=floor|ceiling, piece-wise constant waveforms now can be described in the same general way as other arithmetic models. Is there any requirement to maintain the ALF 1.1 definition in chapter 3.6.10.6 or can it be phased out?

8.11 Arithmetic models for Power calculation

8.11.1 Principles

The purpose of power calculation is to evaluate the electrical power supply demand and electrical power dissipation of an electronic circuit. In general, both power supply demand and power dissipation are the same, due to the energy conservation law. However, there are scenarios where power is supplied and dissipated locally in different places. The power models in ALF shall be specified in such a way, that the total power supply and dissipation of a circuit adds up correctly to the same number.

Example: A capacitor C is charged from 0 volt to V volt by a switched DC source. The energy supplied by the source is $C*V^2$. The energy stored in the capacitor is $1/2*C*V^2$. Hence the dissipated energy is also $1/2*C*V^2$. Later the capacitor is discharged from V volt to 0 volt. The supplied energy is 0. The dissipated energy is $1/2*C*V^2$. A supply-oriented power model can

associate the energy $E_1=C*V^2$ with the charging event and $E_2=0$ with the discharging event. The total energy is $E=E_1+E_2=C*V^2$. A dissipation-oriented power model can associate the energy $E_3=1/2*C*V^2$ with both the charging and discharging event. The total energy is also $E=2*E_3=C*V^2$.

In many cases, it is not so easy to decide when and where the power is supplied and where it is dissipated. The choice between a supply-oriented and dissipation-oriented model or a mixture of both is subjective. Hence the ALF language provides no means to specify, which modeling approach is used. The choice is up to the model developper, as long as the energy conservation law is respected.

8.11.2 POWER and ENERGY

POWER and/or ENERGY models shall be in the context of a CELL or within a VECTOR. Total energy and/or power of a cell shall be calculated by combining the data of all models within the scope of the CELL or the VECTORs within the cell.

The data for POWER and/or ENERGY shall be positive, when energy is actually supplied to the CELL and/or dissipated within the CELL. The data shall be negative, when energy is actually supplied or restored by the CELL.

The following table shows the mathematical relationship between ENERGY and POWER and the applicable MEASUREMENT annotations.

Table 8-19 Relations between ENERGY and POWER

MEASUREMENT for ENERGY	MEASUREMENT for POWER	formula to calculate POWER from ENERGY	formula to calculate ENERGY from POWER
`transient`	`transient`	$\frac{d}{dt}\text{ENERGY}$	$\int \text{POWER}\, dt$
`transient`	`average`	$\frac{\text{ENERGY}}{\text{TIME}}$	$\text{POWER} \cdot \text{TIME}$
`transient`	`peak`	$\max\left(\left\lvert\frac{d}{dt}\text{ENERGY}\right\rvert\right)$	N/A
`transient`	`rms`	$\frac{1}{\text{TIME}} \cdot \int\left(\frac{d}{dt}\text{ENERGY}\right)^2 dt$	N/A

Table 8-19 Relations between ENERGY and POWER

MEASUREMENT for ENERGY	MEASUREMENT for POWER	formula to calculate POWER from ENERGY	formula to calculate ENERGY from POWER
N/A	`static`	N/A	POWER · TIME
`static`	N/A	0	N/A

Comments:

In order to establish a meaningful relationship between energy and power, the measurement for energy must be `transient`. A `static` measurement for energy is conceivable, modeling a state with constant energy, but no power is dissipated during such a state. A `static` measurement for power models a state during which constant power dissipation occurs. Although it is not meaningful to describe an energy model for such a state, it is conceivable to calculate the energy by multiplying the power with the duration of the state. A 1-to-1 correspondence between power and energy can be established for `transient` and `average` power measurements, modeling instantaneous and average power, respectively. Therefore it is redundant to specify both energy and power in such case. Also, `peak` and `rms` power can be conceivably calculated from a transient energy or power waveform, but transient energy can not be calculated from a `peak` or `rms` power measurement.

8.12 Arithmetic models for Hot Electron calculation

8.12.1 Principles

The purpose of hot electron calculation is to evaluate the damage done to the performance of an electronic device due to the hot electron effect. The hot electron effect consists in accumulation of electrons trapped in the gate oxide of a transistor. The more electrons are trapped, the more the device will slow down. At a certain point, the performance specification will no longer be met, and the device is considered to be damaged.

8.12.2 FLUX and FLUENCE

FLUX and/or FLUENCE models shall be in the context of a CELL or within a VECTOR. Total fluence and/or flux of a cell shall be calculated by combining the data of all models within the scope of the CELL or the VECTORs within the cell.

Both FLUX and FLUENCE are measures for hot electron damage. FLUX relates to FLUENCE in the same way as POWER relates to ENERGY.

The following table shows the mathematical relationship between FLUENCE and FLUX and the applicable MEASUREMENT annotations.

Table 8-20 Relations between FLUENCE and FLUX

MEASUREMENT for FLUENCE	MEASUREMENT for FLUX	formula to calculate FLUX from FLUENCE	formula to calculate FLUENCE from FLUX
`transient`	`transient`	$\frac{d}{dt}\text{FLUENCE}$	$\int \text{FLUX}\, dt$
`transient`	`average`	$\frac{\text{FLUENCE}}{\text{TIME}}$	$\text{FLUX} \cdot \text{TIME}$
N/A	`static`	N/A	$\text{FLUX} \cdot \text{TIME}$
`static`	N/A	0	N/A

Since hot electron damage is purely accumulative, the only meaningful MEASUREMENT annotations are `transient`, `average` and `static`.

8.13 Reliability calculation

In general, reliability is modeled by arithmetic models using the LIMIT construct.

8.13.1 TIME within the LIMIT construct

Within a LIMIT construct, TIME can be used in the following ways:

1. TIME itself is subjected to a LIMIT (see section 8.3)
2. TIME is the argument of a model subjected to a LIMIT

When TIME is used as argument of a model within the LIMIT construct, it shall mean the amount of time during which the device is exposed to the quantity modeled within the LIMIT construct. This amount of time is also called lifetime.

Example:

```
LIMIT {
   CURRENT {
      PIN = my_pin;
      MEASUREMENT = static;
      MAX {
         HEADER { TIME TEMPERATURE }
         EQUATION { 6.5*EXP(-10/(TEMPERATURE+273))*TIME**(-0.3) }
      }
   }
}
```

The limit for maximum current depends on the temperature and the expected lifetime of the device.

8.13.2 FREQUENCY within a LIMIT construct

Within a LIMIT construct, FREQUENCY can be used in the following ways:

1. FREQUENCY itself is subjected to a LIMIT
2. FREQUENCY is the argument of a model subjected to a LIMIT

FREQUENCY can be subjected to a LIMIT within the context of a VECTOR. The LIMIT construct specifies an upper and/or lower limit for the repetition frequency of the event sequence described by the `vector_expression`.

Example:

```
VECTOR ( 01 A -> 01 Z ) {
   LIMIT {
      FREQUENCY {
         MAX {
            HEADER {
               SLEWRATE { PIN = A; TABLE { 0.1 0.5 1.0 5.0 } }
               CAPACITANCE { PIN = Z; TABLE { 0.1 0.4 1.6 } }
            }
            TABLE {
               200 190 180 120
               150 150 145 130
                80  80  80  70
            }
         }
      }
   }
}
```

The maximum allowed switching frequency for rising edge on A followed by rising edge on Z depends on the slewrate on A and the load capacitance on Z.

A LIMIT for a quantity with MEASUREMENT annotation `average` or `rms` or `peak` may be frequency-dependent. The FREQUENCY specifies the repetion frequency for the measurement.

Example:

```
LIMIT {
   CURRENT {
      PIN = Vdd;
      MEASUREMENT = average;
      MAX {
         HEADER { FREQUENCY TIME TEMPERATURE }
         EQUATION {
            (FREQUENCY<1)? 6.5*EXP(-10/(TEMPERATURE+273))*TIME**(-0.3) :
            7.8*EXP(-9/(TEMPERATURE+273))*TIME**(-0.2) :
         }
      }
   }
}
```

The limit for average current is specified for low frequencies (< 1MHz) and for higher frequencies. In both cases, the limit depends on temperature and lifetime.

8.13.3 Global LIMIT specifications

Global limits can be specified for electrical quantities, even if they are related to CELLs, PINs or VECTORs. Such global limits apply, unless local limits are specified within the context of CELLs, PINs or VECTORs. The priorities are given below.

1. LIMIT within the context of the VECTOR
2. LIMIT within the context of a PIN (if the LIMIT in the VECTOR has PIN annotation)
3. LIMIT within the context of the CELL
4. LIMIT within the context of the SUBLIBRARY
5. LIMIT within the context of the LIBRARY
6. LIMIT outside LIBRARY

The arguments in the HEADER of the LIMIT model can only be items that are visible within the scope of the LIMIT model. In particular, arguments with PIN annotations are only legal for LIMIT models in the context of a CELL or a VECTOR within the CELL.

8.13.4 LIMIT specification and model specification in the same context

An arithmetic model for a physical quantity and a limit specification for the same physical quantity may appear within the same context, for example an arithmetic model for FLUENCE calculation and a LIMIT for FLUENCE within the context of a VECTOR. In such a case, the calculated quantity shall be checked against the limit of the quantity within that context.

On the other hand, if multiple arithmetic models are given within the context for which the limit applies, the limit shall be checked against the combination of all arithmetic models in the case of accumulative quantities or against the minimum or maximum calculated value in the case of non-accumulative or mutually exclusive quantities.

For example, a LIMIT for FLUENCE may be given in the context of a CELL. Calculation models for FLUENCE may be given for multiple VECTORs within the context of the CELL. The LIMIT for FLUENCE shall be checked against the accumulated FLUENCE calculated for all VECTORs.

Example:

```
CELL my_cell {
   PIN A { DIRECTION = input; }
   PIN B { DIRECTION = input; }
   PIN C { DIRECTION = input; }
   PIN Z { DIRECTION = output; }
   LIMIT { FLUENCE { MAX = 1e20; } } }
   VECTOR ( 01 A -> 10 Z ) {
      FLUENCE = 1e-5;
   }
   VECTOR ( 01 B -> 10 Z ) {
      FLUENCE = 1e-5;
   }
   VECTOR ( 01 C -> 10 Z ) {
      FLUENCE = 1e-6;
      LIMIT { FLUENCE { MAX = 1e18; } }
   }
}
```

The fluence limit for the cell is reached after 10^{25} occurences of `VECTOR ( 01 A -> 10 Z )` or `VECTOR ( 01 B -> 10 Z )` counted together.
The fluence limit for the `VECTOR ( 01 C -> 10 Z )` is reached after 10^{24} occurences of that vector.

An example for a non-accumulative quantity is SLEWRATE. The VECTORs in the context of which SLEWRATE is modeled describe timing arcs with mutually exclusive conditions. Therefore, if a minimum or maximum LIMIT for SLEWRATE is given for a PIN in the context of a CELL, this SLEWRATE shall be checked against the minimum or maximum value of any calculated SLEWRATE applicable to that PIN.

Example:

```
CELL my_cell {
   PIN A { DIRECTION = input; }
   PIN B { DIRECTION = input; }
   PIN C { DIRECTION = input; }
   PIN Z { DIRECTION = output; LIMIT { SLEWRATE { MAX = 5; } } }
   VECTOR ( 01 A -> 10 Z ) {
      SLEWRATE { PIN = Z; /* fill in HEADER, TABLE */ }
   }
   VECTOR ( 01 B -> 10 Z ) {
      SLEWRATE { PIN = Z; /* fill in HEADER, TABLE */ }
   }
   VECTOR ( 01 C -> 10 Z ) {
      SLEWRATE { PIN = Z; /* fill in HEADER, TABLE */ }
   }
}
```

The slewrate on pin Z calculated in the context of any vector is checked against the same maximum limit.

8.13.5 Model specification and argument specification in the same context

An accumulative quantity may also be an argument in the HEADER of an arithmetic model. If the model for calculation of that quantity is within the same context as the argument of the other model, than the value of the calculated quantity shall be used. Otherwise, the value of the accumulated quantity shall be used.

For example, SLEWRATE may be modeled as a function of FLUENCE in the context of a VECTOR. If a calculation model for FLUENCE appears in the context of the same VECTOR, the value for FLUENCE shall be used for SLEWRATE calculation. On the other hand, if there is no calculation model for FLUENCE in the context of the same VECTOR, but in the context of other VECTORs, the accumulated value of FLUENCE from the other calculation models shall be used for SLEWRATE calculation.

Example:

```
CELL my_cell {
   PIN A { DIRECTION = input; }
   PIN B { DIRECTION = input; }
   PIN C { DIRECTION = input; }
   PIN Z { DIRECTION = output; }
   VECTOR ( (01 A | 01 B) -> 10 Z ) { FLUENCE = 1e-5; }
   VECTOR ( 01 A -> 10 Z ) {
      SLEWRATE { CALCULATION=incremental; PIN = Z;
         HEADER { FLUENCE } EQUATION { 1e-8 * FLUENCE }
      }
   }
   VECTOR ( 01 B -> 10 Z ) {
      SLEWRATE { CALCULATION=incremental; PIN = Z;
         HEADER { FLUENCE } EQUATION { 1e-8 * FLUENCE }
      }
   }
   VECTOR ( 01 C -> 10 Z ) {
      FLUENCE = 1e-6;
      SLEWRATE { CALCULATION=incremental; PIN = Z;
         HEADER { FLUENCE } EQUATION { 1e-9 * FLUENCE }
      }
   }
}
```

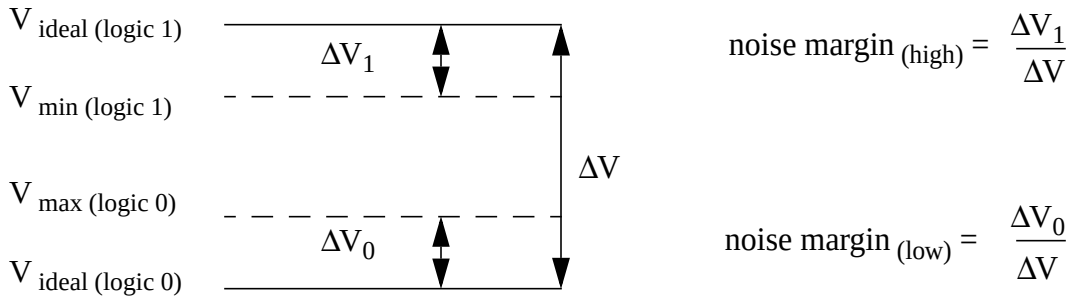
After $10^{13} = 10^5 * 10^8$ occurences of `VECTOR ( (01 A | 01 B) -> 10 Z )`, the slewrate at pin Z for `VECTOR ( 01 A -> 10 Z )` and `VECTOR ( 01 B -> 10 Z )` is increased by 1 unit. After $10^{15} = 10^6 * 10^9$ occurences of `VECTOR ( 01 C -> 10 Z )`, the slewrate at pin Z for `VECTOR ( 01 C -> 10 Z )` is increased by 1 unit.

8.14 Noise calculation

8.14.1 NOISE_MARGIN definition

Noise margin is defined as the maximal allowed differerence between the ideal signal voltage under a well-specified operation condition and the actual signal voltage, normalized to the ideal voltage swing. This is illustrated in the following picture.

Figure 8-17: Definition of noise margin



Noise margin is measured at a signal input pin of a digital cell. The terms “ideal signal voltage” and “actual signal voltage” apply from the standpoint of that particular pin. In CMOS technology, the ideal signal voltage at a pin is the actual supply voltage of the cell, which is not necessarily identical to the nominal supply voltage of the chip.

The NOSE_MARGIN statement has the form of an arithmetic model. If the submodel keywords HIGH and LOW are used, it has the form of an arithmetic model container.

Examples:

```
NOISE_MARGIN = 0.3;

NOISE_MARGIN { HIGH = 0.2;  LOW = 0.4; }

NOISE_MARGIN {
   HEADER { TEMPERATURE { TABLE { 0 50 100 } } }
   TABLE { 0.4 0.3 0.2 }
}
```

NOISE_MARGIN can be related to signal VOLTAGE by the following statement:

```
   VOLTAGE {
      LOW = 0;
      HIGH = 2.5;
   }
   NOISE_MARGIN {
      LOW = 0.4;
      HIGH = 0.3;
   }
}
```

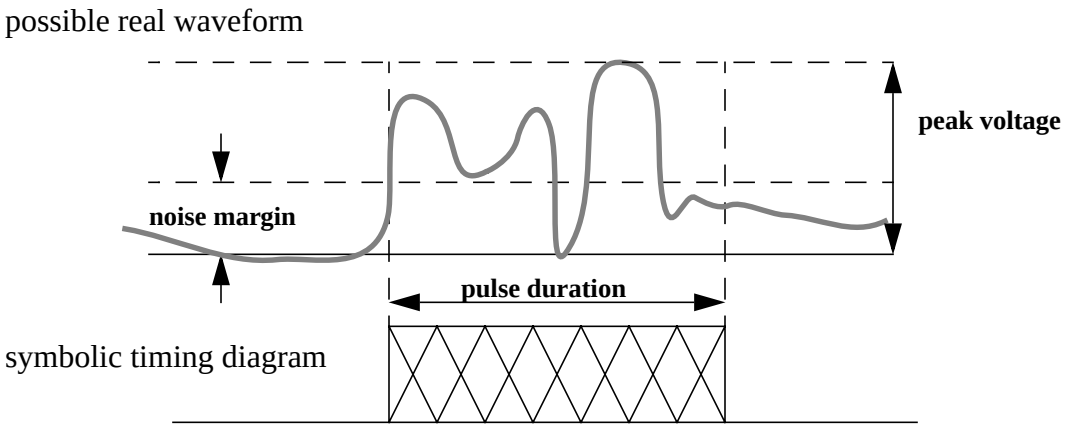
In this example, the valid signal voltage levels are bound by
1 volt = 2.5 volt * 0.4 for logic 0 and 1.75 volt = 2.5 volt * (1 - 0.3) for logic 1.

8.14.2 Representation of noise in a VECTOR

In order to describe timing diagrams involving noisy signals, the symbolic state "*" (see section ??) shall be used. This state represents arbitrary transitions between arbitrary states, which corresponds to the nature of noise.

An illustration is shown in the following figure.

Figure 8-18: Timing diagram of a noisy signal



As illustrated, the signal can be above or below noise margin during the state "*", but it must be within noise margin during the state "0" or "1". During the state "*", the signal is bound by an enveloppe defined by the pulse duration and the peak voltage.

A description of the noisy signal is given in the following template:

```
VECTOR ( 0* my_pin -> *0 my_pin ) {
   TIME = <pulse_duration> {
      FROM { PIN=my_pin; EDGE_NUMBER=0; }
      TO   { PIN=my_pin; EDGE_NUMBER=1; }
   }
   VOLTAGE = <peak_voltage> {
      CALCULATION = incremental;
      MEASUREMENT = peak;
      PIN = my_pin;
   }
}
```

The VECTOR describes the symbolic timing diagram. The TIME statement specifies the duration of the pulse. The VOLTAGE statement specifies the peak voltage. The annotation `CALCULATION=incremental` specifies that the voltage is measured from the nominal signal voltage level rather than from an absolute reference level and that noise voltage may add up.

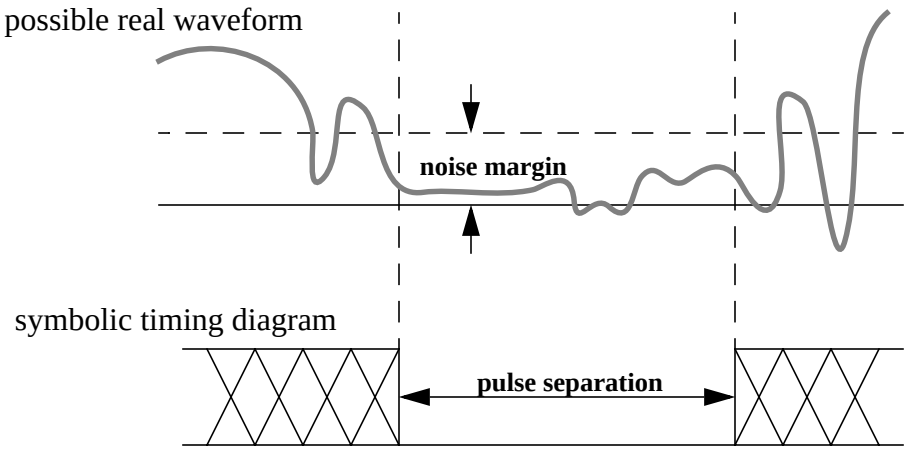
It is also necessary to specify whether a noisy signal (which may oscillate above and below the noise margin) should be considered as one symbolic noise pulse or sperated into multiple symbolic noise pulses.

The LIMIT statement for TIME shall be used for that purpose, as shown in the following example:

```
VECTOR ( *0 my_pin -> 0* my_pin ) {
   LIMIT {
      TIME {
         FROM { PIN = my_pin; EDGE_NUMBER = 0; }
         TO   { PIN = my_pin; EDGE_NUMBER = 1; }
         MIN = <minimum_pulse_separation> ;
      }
   }
}
```

This example corresponds to the following timing diagram.

Figure 8-19: Separation between two noise pulses



When the minimum pulse separation is not met, consecutive noise pulses shall be symbolically marged into one pulse.

8.14.3 Context of NOISE_MARGIN

NOISE_MARGIN is a pin-related quantity. It may appear either in the context of a PIN statement or in the context of a VECTOR statement with PIN annotation. It may also appear in the global context of a CELL, SUBLIBRARY or LIBRARY statement.

If a NOISE_MARGIN statement appears in multiple contexts, the following priorities apply:

1. NOISE_MARGIN with PIN annotation in the context of the VECTOR or NOISE_MARGIN with PIN annotation in the context of the CELL or NOISE_MARGIN in the context of the PIN
2. NOISE_MARGIN without PIN annotation in the context of the CELL

3. NOISE_MARGIN in the context of the SUBLIBRARY
4. NOISE_MARGIN in the context of the LIBRARY
5. NOISE_MARGIN outside the LIBRARY

If the noise margin is either constant or depends only on environmental quantities, the NOISE_MARGIN statement shall appear within the context of the PIN. The noise margin shall relate to the signal VOLTAGE levels applicable for that pin.

Example:

```
PIN my_signal_pin {
   PINTYPE = digital;
   DIRECTION = input;
   VOLTAGE { LOW = 0; HIGH = 2.5; }
   NOISE_MARGIN { LOW = 0.4; HIGH = 0.3; }
}
```

If the noise margin depends on electrical quantities related to other pins, e.g. the supply voltage, the NOISE_MARGIN statement shall have a PIN annotation and appear in the context of the CELL.

Example:

```
CELL my_cell {
   PIN my_signal_pin { PINTYPE = digital; DIRECTION = input; }
   PIN my_power_pin  { PINTYPE = supply; SUPPLYTYPE = power; }
   PIN my_ground_pin { PINTYPE = supply; SUPPLYTYPE = ground; }
   NOISE_MARGIN {
      PIN = my_signal_pin;
      HEADER {
         VOLTAGE vdd { PIN = my_power_pin;  }
         VOLTAGE vss { PIN = my_ground_pin; }
      }
      EQUATION { 0.16 * (vdd - vss ) }
   }
}
```

If the noise margin depends on the logical states and/or the timing of other pins, the NOISE_MARGIN statement shall have a PIN annotation and appear in the context of a VECTOR, describing the state-and/or timing dependency.

Example for state-dependent noise margin:

```
CELL my_latch {
   PIN Q { DIRECTION = output; SIGNALTYPE = data; }
   PIN D { DIRECTION = input; SIGNALTYPE = data; }
   PIN CLK { DIRECTION = input; SIGNALTYPE = clock; POLARITY = high; }
   VECTOR ( CLK && ! D ) { NOISE_MARGIN = 0.4 { PIN = D; } }
   VECTOR ( CLK && D ) { NOISE_MARGIN = 0.3 { PIN = D; } }
}
```

The pin D is only noise-sensitive, when CLK is high. No noise margin is given for the case when CLK is low.

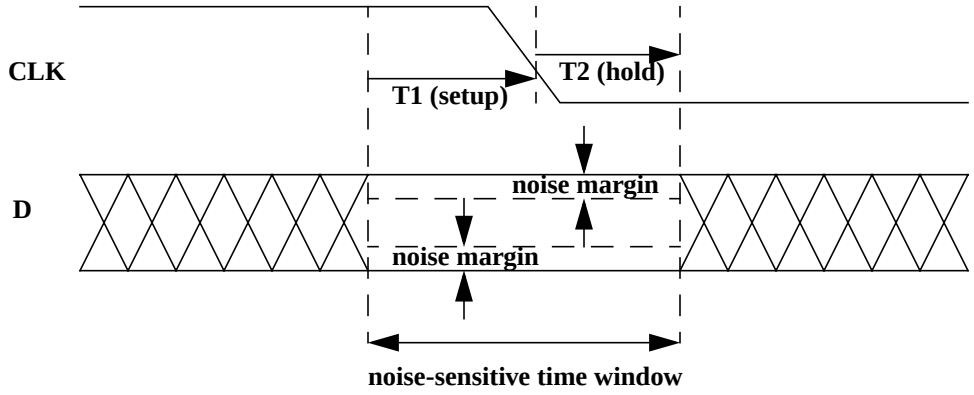
In the case of timing-dependency, the `vector_expression` shall indicate the time window where noise is allowed and where noise is disallowed for the applicable pin. The symbolic state "*" (see section ??) shall be used to indicate a noisy signal.

Example for timing-dependent noise margin:

```
VECTOR ( *? D -> 10 CLK -> ?* D ) {
   TIME T1 = 0.35 {
      FROM { PIN = D;   EDGE_NUMBER = 0; }
      TO   { PIN = CLK; EDGE_NUMBER = 0; }
   }
   TIME T2 = 0.28 {
      FROM { PIN = CLK; EDGE_NUMBER = 0; }
      TO   { PIN = D;   EDGE_NUMBER = 1; }
   }
   NOISE_MARGIN = 0.44 { PIN = D; }
}
```

This example corresponds to the following timing diagram.

Figure 8-20: Example for timing-dependent noise margin



Noise on pin D is allowed 0.35 time units before and 0.28 time units after the falling edge of CLK. during the time window in-between, the noise margin is 0.44.

8.14.4 Noise propagation

Noise propagation from input to output can be modeled in a similar way as signal propagation, using the concept of timing arcs. This is illustrated in the following figure.

Figure 8-21: Prinicple of signal propagation

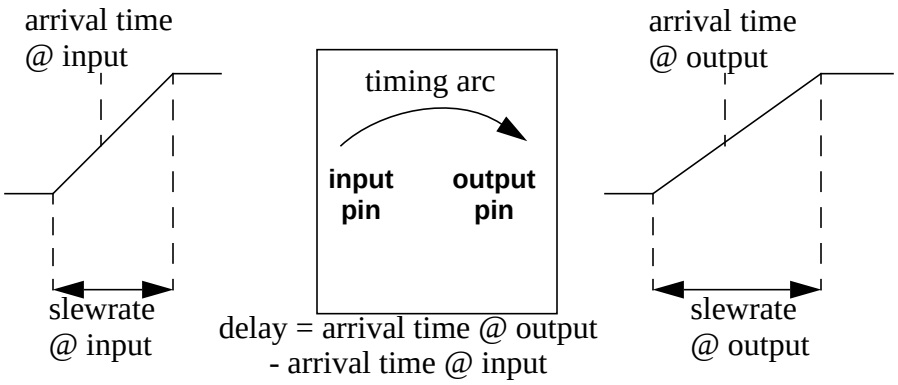
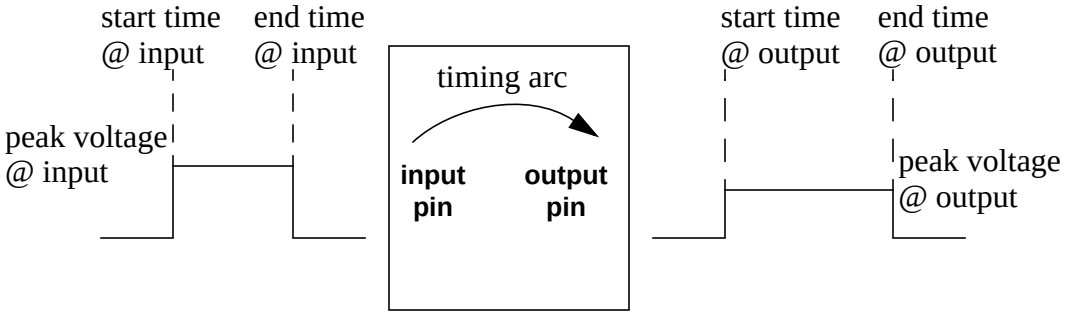


Figure 8-22: Principle of noise propagation



The principle of signal propagation is to calculate the output arrival time and slewrate from the input arrival time and slewrate. In a more abstract way, two points in time propagate from input to output. The same principle applies for noise propagation. Two points in time, start and end time of the noise waveform propagate from input to output. In addition, the noise peak voltage also propagates from input to output.

A VECTOR shall be used to describe the timing of the noise waveform. Again, the symbolic state "*" (see section ??) shall be used to indicate a noisy signal.

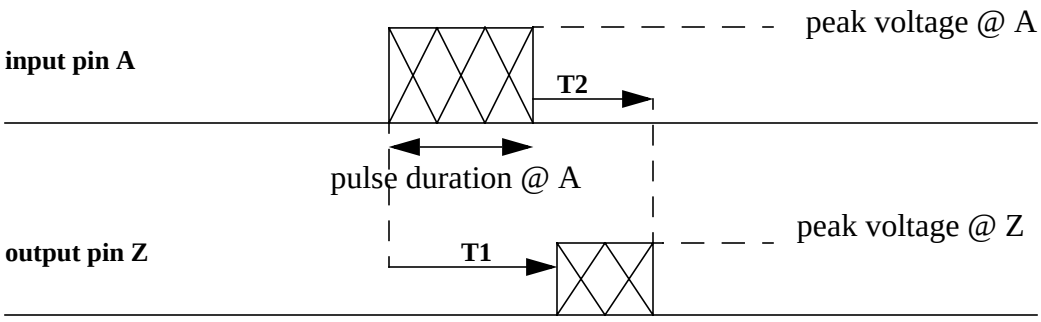
Example:

```
CELL my_cell {
   PIN A { DIRECTION = input; }
   PIN Z { DIRECTION = output; }
   VECTOR ( 0* A -> *0 A <&> 0* Z -> *0 Z ) {
      DELAY T1 {
         FROM { PIN = A; EDGE_NUMBER = 0; }
         TO   { PIN = Z; EDGE_NUMBER = 0; }
         /* fill in HEADER, TABLE or EQUATION */
      }
      DELAY T2 {
```

```
            FROM { PIN = A; EDGE_NUMBER = 1; }
            TO   { PIN = Z; EDGE_NUMBER = 1; }
            /* fill in HEADER, TABLE or EQUATION */
         }
         VOLTAGE { PIN = Z; MEASUREMENT = peak;
            /* fill in HEADER, TABLE or EQUATION */
         }
      }
```

This example corresponds to the following timing diagram.

Figure 8-23: Example of noise propagation



The input to output delay of the leading edge of the noise pulse may depend on the peak voltage at pin A, the load capacitance at pin Z and other electrical quantities. In addition, the input to output delay of the trailing edge of the noise pulse as well as the peak voltage at pin Z may also depend on the duration of the pulse at pin A.

Note: The time measurement from start to end of the noise pulse shall be represented by the keyword TIME (no causality between start and end time), whereas the time measurement from input to output shall be represented by the keyword DELAY (causality between input and output arrival time).

8.14.5 Noise rejection

Noise rejection is a limit case for noise propagation, when the output peak voltage is so low that the noise is considered rejected. In this case, the input peak voltage can still be above noise margin, whereas the output peak voltage is way below noise margin.

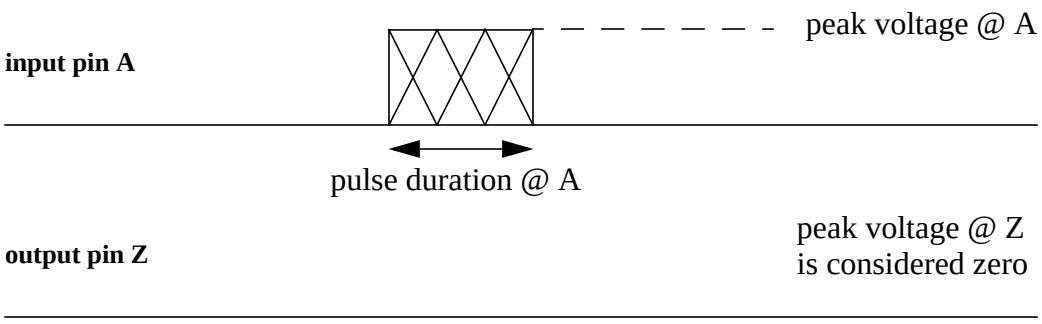
Example:

```
CELL my_cell {
   PIN A { DIRECTION = input; }
   PIN Z { DIRECTION = output; }
   VECTOR ( 0* A -> *0 A -> 00 Z ) {
      LIMIT {
         VOLTAGE {
            PIN = A; MEASUREMENT = peak;
            MAX { /* fill in HEADER, TABLE or EQUATION */ }
         }
      }
   }
}
```

Note: The `vector_expression 00 Z` says explicitely that a transition at pin Z does *not* happen.

This example corresponds to the following timing diagram.

Figure 8-24: Example of noise rejection



The peak voltage limit for noise rejection may depend on the duration of the noise pulse at pin A and other electrical quantities, e.g. the load capacitance at pin Z. If the peak voltage limit does not depend on the duration of the noise pulse, the NOISE_MARGIN statement should be used rather than the vector-specific LIMIT construct for noise rejection.

8.15 Annotations for a PIN object

This chapter may be combined with section 6.4.

8.15.1 DRIVETYPE annotation

```
DRIVETYPE = string ;
```

annotates the drive type for the pin, which can take the following values:

Table 8-21 : DRIVETYPE annotations for a PIN object

Annotation string	Description
`cmos` (default)	standard cmos signal
`nmos`	nmos or pseudo nmos signal
`pmos`	pmos or pseudo pmos signal
`nmos_pass`	nmos passgate signal
`pmos_pass`	pmos passgate signal
`cmos_pass`	cmos passgate signal, i.e. full transmission gate
`ttl`	TTL signal
`open_drain`	open drain signal
`open_source`	open source signal

8.15.2 SCOPE annotation

```
SCOPE = string ;
```

annotates modeling scope of a pin, which can take the following values:

Table 8-22 : SCOPE annotations for a PIN object

Annotation string	Description
`behavior`	Pin is used for modeling functional behavior. Events on the pin are monitored for vector expressions in BEHAVIOR statement
`measure`	Measurements related to the pin can be described, e.g. timing or power characterization. Events on the pin are monitored for vector expressions in VECTOR statements
`both` (default)	Pin is used for functional behavior as well as for characterization measurements
`none`	no model, only pin exists

8.15.3 PULL annotation

```
PULL = string ;
```

which can take the following values:

Table 8-23 : PULL annotations for a PIN object

Annotation string	Description
`up`	pullup device connected to pin
`down`	pulldown device connected to pin
`both`	pullup and pulldown device connected to pin
`none` (default)	no pull device

8.16 Annotations for a VECTOR

This chapter may be moved into section 6.

A `VECTOR` object may contain the following annotations:

8.16.1 LABEL annotation

```
LABEL = string ;
```

to be used to ensure SDF matching with conditional delays across Verilog, VITAL etc.

8.16.2 EXISTENCE_CONDITION annotation

```
EXISTENCE_CONDITION = boolean_expression ;
```

For false-path analysis tools, the existence condition shall be used to eliminate the vector from further analysis if and only if the existence condition evaluates to “false”. For applications other than false-path analysis, the existence condition shall be treated as if the boolean expression was a cofactor to the vector itself. Default existence condition is “true”.

Example:

```
VECTOR (01 a -> 01 z & (c | !d) ) {
      EXISTENCE_CONDITION = !scan_select;
      DELAY { FROM { PIN=a; } TO { PIN=z; } /* data */ }
}
VECTOR (01 a -> 01 z & (!c | d) ) {
      EXISTENCE_CONDITION = !scan_select;
      DELAY { FROM { PIN=a; } TO { PIN=z; } /* data */ }
}
```

Each vector contains state-dependent delay for the same timing arc. If "`!scan_select`" evaluates "true", both vectors are eliminated from timing analysis.

8.16.2.1 EXISTENCE_CLASS

```
EXISTENCE_CLASS = string ;
```

Reference to the same existence class by multiple vectors has the following effects:

- A common mode of operation is established between those vectors, which can be used for selective analysis, for instance mode-dependent timing analysis. Name of the mode is the name of the class.
- A common existence condition is inherited from that existence class, if there is one.

Example:

```
CLASS non_scan_mode {
      EXISTENCE_CONDITION = !scan_select;
}
VECTOR (01 a -> 01 z & (c | !d) ) {
      EXISTENCE_CLASS = non_scan_mode;
      DELAY { FROM { PIN=a; } TO { PIN=z; } /* data */ }
}
VECTOR (01 a -> 01 z & (!c | d) ) {
      EXISTENCE_CLASS = non_scan_mode;
      DELAY { FROM { PIN=a; } TO { PIN=z; } /* data */ }
}
```

Each vector contains state-dependent delay for the same timing arc. If the mode "`non_scan_mode`" is turned off or if "`!scan_select`" evaluates "true", both vectors are eliminated from timing analysis.

8.16.3 EXISTENCE_CLASS annotation

```
EXISTENCE_CLASS = string ;
```

Reference to the same existence class by multiple vectors has the following effects:

- A common mode of operation is established between those vectors, which can be used for selective analysis, for instance mode-dependent timing analysis. Name of the mode is the name of the class.
- A common existence condition is inherited from that existence class, if there is one.

Example:

```
CLASS non_scan_mode {
      EXISTENCE_CONDITION = !scan_select;
}
VECTOR (01 a -> 01 z & (c | !d) ) {
      EXISTENCE_CLASS = non_scan_mode;
      DELAY { FROM { PIN=a; } TO { PIN=z; } /* data */ }
}
VECTOR (01 a -> 01 z & (!c | d) ) {
      EXISTENCE_CLASS = non_scan_mode;
      DELAY { FROM { PIN=a; } TO { PIN=z; } /* data */ }
}
```

Each vector contains state-dependent delay for the same timing arc. If the mode "`non_scan_mode`" is turned off or if "`!scan_select`" evaluates "true", both vectors are eliminated from timing analysis.

8.16.4 CHARACTERIZATION_CONDITION annotation

```
CHARACTERIZATION_CONDITION = boolean_expression ;
```

For characterization tools, the characterization condition shall be treated as if the boolean expression was a cofactor to the vector itself. For all other applications, the characterization condition shall be disregarded. Default characterization condition is “true”.

Example:

```
VECTOR (01 a -> 01 z & (c | !d) ) {
      CHARACTERIZATION_CONDITION = c & !d;
      DELAY { FROM { PIN=a; } TO { PIN=z; } /* data */ }
}
```

The delay value for the timing arc applies for any of the following conditions (`c & !d`) or (`c & d`) or (`!c & !d`), since they all satisfy (`c | !d`) .
However, the only condition chosen for delay characterization is (`c & !d`).

8.16.5 CHARACTERIZATION_VECTOR annotation

```
CHARACTERIZATION_VECTOR = ( vector_expression ) ;
```

The characterization vector is provided for the case that the vector expression cannot be constructed using the vector and a boolean cofactor. The use of the characterization vector is restricted to characterization tools in the same way as the use of the characterization condition. Either a characterization condition or a characterization vector may be provided, but not both. If none is provided, the vector itself will be used by the characterization tool.

Example:

```
VECTOR (01 A -> 01 Z) {
      CHARACTERIZATION_VECTOR = ((01 A & 10 inv_A) -> (01 Z & 10 inv_Z));
}
```

Analysis tools see the signals "`A`" and "`Z`". The signals "`inv_A`" and "`inv_Z`" are visible to the characterization tool only.

8.16.6 CHARACTERIZATION_CLASS annotation

```
CHARACTERIZATION_CLASS = string ;
```

Reference to the same characterization class by multiple vectors has the following effects:

- A commonality is established between those vectors, which can be used for selective characterization in a way defined by the library characterizer, for instance to share the characterization task between different teams or jobs or tools.
- A common characterization condition or characterization vector is inherited from that characterization class, if there is one.

8.17 Interconnect parasitics and analysis

8.17.1 Principles of the WIRE statement

Parasitic descriptions shall be in the context of a WIRE statement. The following fundamental modeling styles are supported.

- Statistical wireload models

- Boundary parasitics

Statistical wireload models as well as interconnect analysis calculation models may be within the context of a LIBRARY, SUBLIBRARY or CELL statement. The latter applies only for cells with CELLTYPE=block, i.e., hierarchical cells. Boundary parasitics apply exclusively for hierarchical cells. Statistical wireload models can be mixed with boundary parasitics within the same WIRE statement.

Interconnect analysis models shall also be within a WIRE statement. However, they shall not be mixed with statistical wireload models or boundary parasitic descriptions.

The purpose of interconnect analysis is to calculate electrical quantities such as DELAY, SLEWRATE, noise VOLTAGE in the context of a netlist consisting of electrical components, such as CAPACITANCE, RESISTANCE, INDUCTANCE.

As opposed to boundary parasitics, where the components are connected to physical nodes and pins of a cell, the components represent an abstract network targeted for analysis. The interconnect analysis model specifies a directive for the parasitic extraction / delay calculation tool as to which order an arbitrary network shall be reduced. In addition, the model specifies the calculation models for delay, noise etc. in the context of the reduced network.

8.17.2 Statistical wireload models

A statistical wireload model is a collection of arithmetic models for estimated electrical quantities CAPACITANCE, RESISTANCE, INDUCTANCE, representing interconnect load and for estimated AREA and SIZE of interconnect nets.

These arithmetic models shall have no PIN annotation. Only environmental quantities such as PROCESS, DERATE_CASE, TEMPERATURE shall be allowed as arguments in the HEADER.

In addition, the quantities AREA, SIZE, FANOUT, FANIN, CONNECTIONS shall be allowed as arguments in the HEADER.

FANOUT and FANIN represent the number of receiver pins and driver pins, respectively, connected to the net. CONNECTIONS is the total number of pins connected to the net. CONNECTIONS equals to the sum of FANOUT and FANIN.

AREA represents a physically measurable area of an object, whereas SIZE represents an abstract symbolic quantity or cost function for area. When AREA or SIZE is used as argument within the HEADER, it shall represent the total area or size, respectively, allocated for place & route of the block for which the wireload model applies. An arithmetic model given for AREA or SIZE itself shall represent the estimated or actual area or size, respectively, of the object in the context of which the model appears. Applicable objects for AREA or SIZE models are CELL and WIRE.

In order to convert SIZE to AREA (analogous to converting DRIVE_STRENGTH to RESISTANCE, see section ??), an arithmetic model for SIZE with AREA as argument can be used outside the WIRE statement. Arithmetic models for SIZE inside the WIRE statement will be interpreted as a calculation model rather than a conversion model.

The total area or size of a block must be larger or equal to the area or size, respectively, of all objects within the block, i.e., cells and wires. Note: The area or size of a block is design-specific data, whereas the area or size of cells and wires is given in the library.

Example:

```
LIBRARY my_library {
   WIRE my_wlm {
      CAPACITANCE {
         HEADER {
            CONNECTIONS { TABLE { 2 3 4 5 10 20 } }
            AREA { TABLE { 1000 10000 100000 } }
         }
         TABLE {
            0.03 0.06 0.08 0.10 0.15 0.25
            0.05 0.10 0.15 0.18 0.25 0.35
            0.10 0.18 0.25 0.32 0.50 0.65
         }
      }
      AREA {
         HEADER {
            CONNECTIONS { TABLE { 2 3 4 5 10 20 } }
            AREA { TABLE { 1000 10000 100000 } }
         }
         TABLE {
            0.3 0.6 0.8 1.0 1.5 2.5
            0.5 1.0 1.5 1.8 2.5 3.5
            1.0 1.8 2.5 3.2 5.0 6.5
         }
      }
   }
   CELL my_cell {
      AREA = 1.5;
      PIN my_input  { DIRECTION = input;  CAPACITANCE = 0.1; }
      PIN my_output { DIRECTION = output; CAPACITANCE = 0.0; }
   }
}
```

A net routed in a block of AREA=10000, driven by an instance of `my_cell` connecting to 4 receivers (i.e., CONNECTIONS=5), each of which is an instance of `my_cell`, will have an estimated capacitance of 0.18+4*0.1=0.58 and wire area of 1.8. The 5 cell instances together will have an area of 7.5.

Note: CAPACITANCE, RESISTANCE, AREA can each be independent arithmetic models within the WIRE statement. No multiplication factor between area and capacitance or between area and resistance is assumed.

8.17.3 Boundary parasitics

Boundary parasitics for a CELL may be given within a WIRE statement in the context of the CELL. The parasitics shall be identified by arithmetic models for CAPACITANCE, RESISTANCE, INDUCTANCE containing a NODE annotation.

The syntax is as follows:

```
two_node_multi_value_assignment ::=
   NODE { node_identifier node_identifier }
four_node_multi_value_assignment ::=
   NODE { node_identifier node_identifier node_identifier node_identifier }
```

where *node*_identifier is one of the following:

a simple identifier, refering to a declared PIN of the CELL.

a hierarchical_identifier, refering to a declared PORT of a PIN of the CELL (see section 10.10 and 12.3.9)

a simple identifier, refering to a declared NODE of the WIRE (see section 8.17.3)

a simple identifier, not refering to a declared object. Can be used for connectivity inside the WIRE only.

The *two_node*_multi_value_assignment applies for capacitance, resistance and self-inductance. These components imply the following relationship between voltage and current accross the nodes:

$$\text{VOLTAGE(node1, node2)} = \text{RESISTANCE(node1, node2)} \cdot \text{CURRENT(node1, node2)}$$

$$\text{CURRENT(node1, node2)} = \text{CAPACITANCE(node1, node2)} \cdot \frac{d}{dt}\text{VOLTAGE(node1, node2)}$$

$$\text{VOLTAGE(node1, node2)} = \text{INDUCTANCE(node1, node2)} \cdot \frac{d}{dt}\text{CURRENT(node1, node2)}$$

The *four_node*_multi_value_assignment applies for mutual inductance. This component imples the following relationship between voltage and current accross the nodes:

$$\text{VOLTAGE(node1, node2)} = \text{INDUCTANCE(node1, node2, node3, node4)} \cdot \frac{d}{dt}\text{CURRENT(node3, node4)}$$

Note:

Both PIN assignments (e.g. `PIN=A;`) and NODE assignments (e.g. `NODE { A B }`) may refer to PINs or PORTs.
The fundamental semantic difference between a PIN assignment and a NODE assignment is the following: The PIN assignment within an object defines that the object is *applied* or *measured* at the PIN or PORT. (e.g. DELAY, SLEWRATE). The NODE assignment within an object defines that the object is fundamentally *connected* with the PIN or PORT, in the same way as an object inside a PIN is also fundamentally connected with the PIN.
Therefore, the CAPACITANCE with NODE assignment is a more detailed way of describing a CAPACITANCE of a PIN, whereas a CAPACITANCE with PIN assignment describes a load capacitance, which is applied externally to the pin.

A CELL may contain a WIRE statement describing boundary parasitics as well as PIN statements containing arithmetic models for CAPACITANCE, RESISTANCE or INDUCTANCE. In this case the latter shall be considered as a reduced form of the former. An

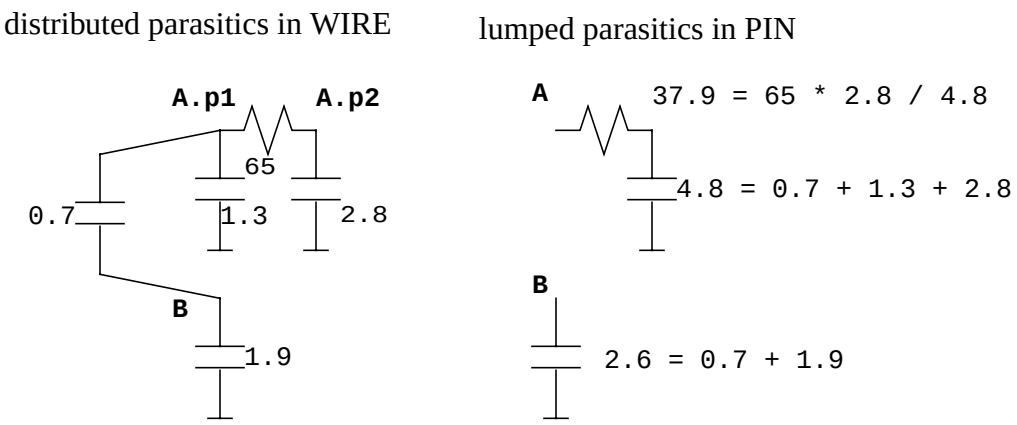
analysis tool shall either use the set of components inside the PIN or inside the WIRE, but not a combination of both.

Example:

```
CELL my_cell {
   PIN A { PINTYPE = digital; CAPACITANCE = 4.8; RESISTANCE = 37.9;
      PORT p1 { VIEW = physical; } // see section 10.10
      PORT p2 { VIEW = none; } // see section 10.10
   }
   PIN B { PINTYPE = digital; CAPACITANCE = 2.6; }
   PIN gnd { PINTYPE = supply; SUPPLYTYPE = ground; }
   WIRE my_boundary_parasitics {
      CAPACITANCE = 1.3 { NODE { A.p1 gnd } }
      CAPACITANCE = 2.8 { NODE { A.p2 gnd } }
      RESISTANCE = 65 { NODE { A.p1 A.p2 } }
      CAPACITANCE = 0.7 { NODE { A.p1 B } }
      CAPACITANCE = 1.9 { NODE { B gnd } }
   }
}
```

This example corresponds to the following netlist:

Figure 8-25: Example of boundary parasitic description



The distributed parasitics in the WIRE statement can be reduced to the lumped parasitics in the PIN statement.

8.17.4 NODE declaration

The nodes used for interconnect analysis shall be declared within the WIRE statement, using the following syntax.

```
node ::=
      NODE node_identifier { all_purpose_items }
```

Specific applicable to NODE are the NODETYPE annotation and the NODE_CLASS annotation.

```
nodetype_annotation ::=
   NODETYPE = nodetype_identifier ;

nodetype_identifier ::=
   ground
|  power
|  source
|  sink
|  driver
|  receiver
```

- A driver node is the interface between a cell output pin and interconnect
- A receiver node is the interface between interconnect and a cell input pin
- A source node is a virtual start point of signal propagation. It may be collapsed with a driver node
- A sink node is a virtual end point of signal propagation. It may be collapsed with a receiver node
- A power node provides the current for rising signals at the source / driver side and a reference for logic high signals at the sink / receiver side
- A ground node provides the current for falling signals at the source / driver side and a reference for logic low signals at the sink / receiver side

The arithmetic models for electrical components which are part of the network shall have names and NODE annotations, refering either to the predeclared nodes or to internal nodes which need not be declared.

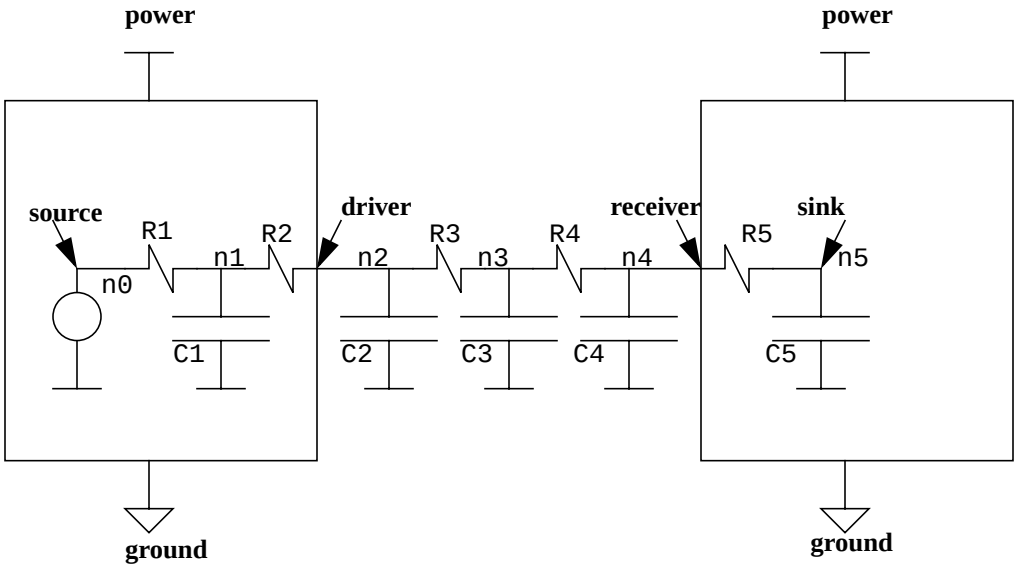
Example:

```
WIRE my_interconnect_model {
   NODE n0  { NODETYPE = source;   }
   NODE n2  { NODETYPE = driver;   }
   NODE n4  { NODETYPE = receiver; }
   NODE n5  { NODETYPE = sink;     }
   NODE vdd { NODETYPE = power;    }
   NODE vss { NODETYPE = ground;   }
   RESISTANCE R1 { NODE { n0 n1 } }
   RESISTANCE R2 { NODE { n1 n2 } }
   RESISTANCE R3 { NODE { n2 n3 } }
   RESISTANCE R4 { NODE { n3 n4 } }
```

```
   RESISTANCE R5 { NODE { n4 n5 } }
   CAPACITANCE C1 { NODE { n1 vss } }
   CAPACITANCE C2 { NODE { n2 vss } }
   CAPACITANCE C3 { NODE { n3 vss } }
   CAPACITANCE C4 { NODE { n4 vss } }
   CAPACITANCE C5 { NODE { n5 vss } }
}
```

This example is illustrated in the following figure.

Figure 8-26: Example for interconnect description



The NODE_CLASS annotation is optional and orthogonal to the NODETYPE annotation.

*node_class*_annotation ::=
NODE_CLASS = *node_class*_identifier **;**

The NODE_CLASS annotation shall refer to a predeclared CLASS within the WIRE statement to indicate, which node belongs to which device, in the case of separate power supplies.

Example:

```
WIRE my_interconnect_model {
   CLASS driver_cell;
   CLASS receiver_cell;
   NODE n0   { NODETYPE = source; NODE_CLASS = driver_cell; }
   NODE n2   { NODETYPE = driver; NODE_CLASS = driver_cell; }
   NODE n4   { NODETYPE = receiver; NODE_CLASS = receiver_cell; }
   NODE n5   { NODETYPE = sink;     NODE_CLASS = receiver_cell; }
   NODE vdd1 { NODETYPE = power;  NODE_CLASS = driver_cell; }
   NODE vss1 { NODETYPE = ground; NODE_CLASS = driver_cell; }
   NODE vdd2 { NODETYPE = power;  NODE_CLASS = receiver_cell; }
   NODE vss2 { NODETYPE = ground; NODE_CLASS = receiver_cell; }
}
```

If NODE_CLASS is not specified, the nodes with NODETYPE=power|ground are supposed to be global. The DC-connected nodes with NODETYPE=driver|source are supposed to belong to the same device. The DC-connected nodes with NODETYPE=receiver|sink are supposed to belong to the same device.

8.17.5 Interconnect delay and noise calculation

In the context of a VECTOR inside a WIRE, calculation models for DELAY and SLEWRATE may be described. The PIN assignments in these models shall refer to predeclared NODEs inside the WIRE.

Example:

```
WIRE my_interconnect_model {
   /* node declarations */
   /* electrical component declarations */
   VECTOR ( (01 n0 ~> 01 n5) | (10 n0 ~> 10 n5) ) {
      /* DELAY model */
      /* SLEWRATE model */
   }
}
```

The predeclared electrical components which are part of the network can be used within an EQUATION without being redeclared in the HEADER of the model.

Example:

```
DELAY {
   FROM { PIN = n0; } TO { PIN = n5; }
   EQUATION {
      R1*(C1+C2+C3+C4+C5) + R2*(C2+C3+C4+C5)
      + R3*(C3+C4+C5) + R4*(C4+C5) + R5*C5
   }
}
```

External components or stimuli which are not part of the network must be declared in the HEADER. Also, all arguments for TABLE-based models must be in the HEADER. To avoid redeclaration of predeclared components, an EQUATION shall also be used for those arguments in the HEADER which refer to predeclared components.

Example:

```
SLEWRATE {
   PIN = n5;
   HEADER {
      SLEWRATE { PIN = n0; TABLE {/* numbers */} }
      RESISTANCE  { EQUATION { R1+R2+R3+R4+R5 } TABLE {/* numbers */} }
      CAPACITANCE { EQUATION { C1+C2+C3+C4+C5 } TABLE {/* numbers */} }
   }
   TABLE { /* numbers */ }
}
```

In order to model crosstalk delay and noise, at least two driver and receiver nodes are required. The symbolic state "*" (see section ??) shall be used to indicate the signal subjected to noise.

Example:

```
WIRE interconnect_model_with_coupling {
   NODE aggressor_source { NODETYPE = driver; }
   NODE victim_source    { NODETYPE = driver; }
   NODE aggressor_sink   { NODETYPE = receiver; }
   NODE victim_sink      { NODETYPE = receiver; }
   NODE vdd { NODETYPE = power;  }
   NODE gnd { NODETYPE = ground; }
   CAPACITANCE cc { NODE {aggressor_sink victim_sink}}
   CAPACITANCE cv { NODE {victim_sink gnd }}
   RESISTANCE  rv { NODE {victim_source victim_sink}}
   VECTOR ( 01 aggressor_sink -> ?* victim_sink -> *? victim_sink ) {
      /* xtalk noise model */
   }
   VECTOR (
      ( 01 aggressor_source <&> 01 victim_source )
       -> 01 aggressor_sink -> 01 victim_sink
   ) {
   /* xtalk DELAY model */
   }
}
```

Example for noise model:

```
VOLTAGE {
      PIN = victim_sink;
      MEASUREMENT = peak;
      CALCULATION = incremental;
      HEADER {
         SLEWRATE   tra { PIN = aggressor_sink; }
         VOLTAGE     va { NODE {vdd gnd} }
      }
      EQUATION { (1-EXP(-tra/(rv*cv)))*va*rv*cc/tra }
   }
}
```

Example for delay model:

```
DELAY {
   FROM { PIN = victim_source; } TO { PIN = victim_sink; }
   CALCULATION = incremental;
   HEADER {
      SLEWRATE tra { PIN = aggressor_sink; }
      SLEWRATE trv { PIN = victim_source; }
   }
   EQUATION { (1-EXP(-tra/(rv*cv)))*rv*cc*trv/tra }
}
```

The VOLTAGE model applies for rising aggressor signal while the victim signal is stable. The DELAY model applies for rising victim signal simultaneous with or followed by rising

aggressor signal at the coupling point. Note that the VECTOR implicitely defines the time window of interaction between aggresssor and victim: Interaction occurs only, if the aggressor signal at the coupling point intervenes during the propagation of the victim signal from its source to the coupling point. Both VOLTAGE and DELAY represent incremental numbers.

8.17.6 SELECT_CLASS annotation for WIRE statement

A sophisticated tool may support more than one interconnect model. Each calculation model may have its “netlist” with the appropriate validity range of the RC components. For instance, a lumped model may be used for short nets, a distributed model may be used for longer nets.Also, models with different accururacy for the same net may be defined. For instance, the lumped model may be used for estimation purpose, the distributed model for signoff.

For this purpose, classes may be defined to select a set of models. The selection must be defined by the user, in a similar way as a user would select wireload models for prelayout parasitic estimation. The selected class shall be indicated by the SELECT_CLASS annoation within the WIRE statement.

Example:

```
LIBRARY my_library {
   CLASS estimation;
   CLASS verification;
   WIRE rough_model_for_short_nets {
      SELECT_CLASS = estimation; /* etc.*/
   }
   WIRE detailed_model_for_short_nets {
      SELECT_CLASS = verification; /* etc.*/
   }
   WIRE rough_model_for_long_nets {
      SELECT_CLASS = estimation; /* etc.*/
   }
   WIRE detailed_model_for_long_nets {
      SELECT_CLASS = verification; /* etc.*/
   }
}
```