Library Harmonization for Timing

Version	Date
0.0	2003/11/17
0.1	2003/12/12
0.2	2004/01/19
0.3	2004/02/23
0.4	2004/03/22
0.5	2004/05/01
0.6	2004/10/10

Template for Liberty/ALF xref examples

```
/* Liberty */                          /* ALF */
```

Table of contents

1.0 Basic description of timing arcs

1.1 Timing measurement overview

Timing arcs are defined not only by standalone statements but also by the context in which the statements appear. In both Liberty and ALF, a timing arc is defined in the context of a CELL identified by a *Cell Name*. A declaration of each PIN involved in the timing arc is required, referred herein as the *Pin Name* and the *Related Pin Name*.

In Liberty, the timing model is further defined inside the declaration of the *Pin Name*. The occurring edge combinations are defined by *Timing Type* and *Timing Sense*.

Some timing data in Liberty appear as *timing model*, others appear as a *timing attribute*. A timing attribute supports only a scalar value, whereas a timing model supports a mathematical calculation model. A timing arc description in Liberty is shown in Figure 1 on page 3.

FIGURE 1. Timing arc description in Liberty

```
/* Liberty */
cell (CellName) {
  pin(RelatedPinName) {
    direction : RelatedPinDirection;
  }
  pin(PinName) {
    direction : PinDirection;
    timing() {
      timing_type  : TimingType;
      timing_sense : TimingSense;
      related_pin  : "RelatedPinName";
      /* lib_TimingModel */
      ModelKeyword (CalculationType) { values ( /* lib_Data */ ); }
    }
    /* lib_TimingAttribute */
  AttributeKeyword : AttributeValue ;
  }
}
```

In ALF, pins and timing arcs are declared separately. A timing arc is established by the declaration of a VECTOR, separate from the declaration of each PIN involved in the timing arc. The edge combinations are defined by a *Vector Expression*. A timing arc description in ALF is shown on Figure 2 on page 4.

FIGURE 2. Timing arc description in ALF

```
/* ALF */
CELL CellName {
  PIN RelatedPinName {
    DIRECTION = RelatedPinDirection;
  }
  PIN PinName {
    DIRECTION = PinDirection;
  }
  VECTOR (VectorExpression) {
    /* ALF_TimingModel */
    // see Figure 4 on page 6 through Figure 23 on page 19
  }
}
```

The ALF description of a timing model and its mapping to a Liberty construct depends on the nature of the timing measurement. The following table shows an overview of measurement and the pointer to the corresponding ALF description and the Liberty to ALF mapping table.

TABLE 1. Overview of timing measurements

Measurement	Comment
delay, slew	see Section 1.2 on page 4
delay, retain, slew	see Section 1.4 on page 8
independent setup, hold	see Section 1.5 on page 10
independent recovery, removal	see Section 1.6 on page 12
co-dependent setup, hold	see Section 1.7 on page 13
co-dependent recovery, removal	see Section 1.8 on page 14
setup, hold with nochange constraint	see Section 1.9 on page 15
maximum skew constraint	see Section 1.10 on page 17
minimum period and minimum pulsewidth constraint	see Section 1.11 on page 18

1.2 Delay and slew

FIGURE 3. Delay and slew measurements

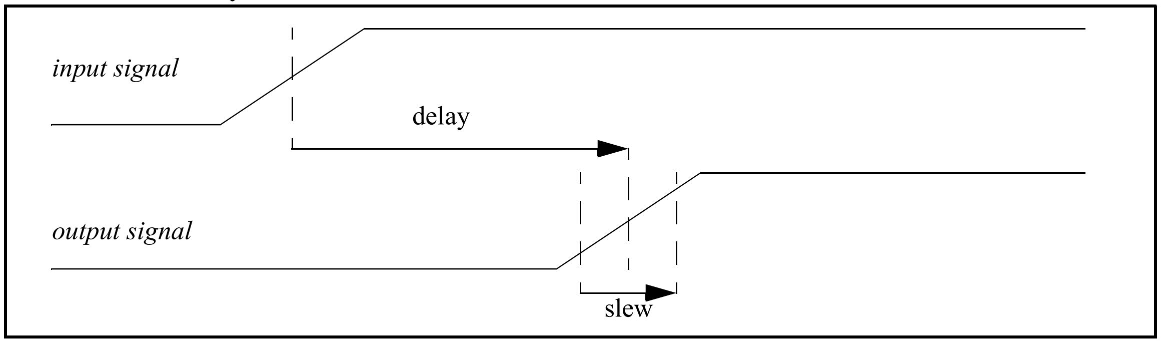


TABLE 2. Mapping of Liberty and ALF constructs for delay and slew measurements

Liberty construct			ALF construct PN = Pin Name, RPN = Related Pin Name	
Timing Type	***Timing Sense***	***Model Keyword***	***Keyword***	***Vector Expression***
combinational	positive_unate	cell_rise	DELAY	01 RPN -> 01 PN
		rise_transition	SLEWRATE	
		cell_fall	DELAY	10 RPN -> 10 PN
		fall_transition	SLEWRATE	
	negative_unate	cell_rise	DELAY	10 RPN -> 01 PN
		rise_transition	SLEWRATE	
		cell_fall	DELAY	01 RPN -> 10 PN
		fall_transition	SLEWRATE	
	non_unate	cell_rise	DELAY	?! RPN -> 01 PN
		rise_transition	SLEWRATE	
		cell_fall	DELAY	?! RPN -> 10 PN
		fall_transition	SLEWRATE	
three_state_enable	positive_unate	cell_rise ?	DELAY	01 RPN -> Z1 PN
		cell_fall ?		01 RPN -> Z0 PN
	negative_unate	cell_rise ?		10 RPN -> Z1 PN
		cell_fall ?		10 RPN -> Z0 PN
three_state_disable	positive_unate	cell_rise ?	DELAY	01 RPN -> 0Z PN
		cell_fall ?		01 RPN -> 1Z PN
	negative_unate	cell_rise ?		10 RPN -> 0Z PN
		cell_fall ?		10 RPN -> 1Z PN
rising_edge	N/A	cell_rise	DELAY	01 RPN -> 01 PN
		rise_transition	SLEWRATE	
		cell_fall	DELAY	01 RPN -> 10 PN
		fall_transition	SLEWRATE	

TABLE 2. Mapping of Liberty and ALF constructs for delay and slew measurements

Liberty construct			ALF construct PN = Pin Name, RPN = Related Pin Name	
Timing Type	***Timing Sense***	***Model Keyword***	***Keyword***	***Vector Expression***
falling_edge	N/A	cell_rise	DELAY	10 RPN -> 01 PN
		rise_transition	SLEWRATE	
		cell_fall	DELAY	10 RPN -> 10 PN
		fall_transition	SLEWRATE	
preset	positive_unate	cell_rise	DELAY	01 RPN -> 01 PN
		rise_transition	SLEWRATE	
	negative_unate	cell_rise	DELAY	10 RPN -> 01 PN
		rise_transition	SLEWRATE	
clear	positive_unate	cell_fall	DELAY	10 RPN -> 10 PN
		fall_transition	SLEWRATE	
	negative_unate	cell_fall	DELAY	01 RPN -> 10 PN
		fall_transition	SLEWRATE	

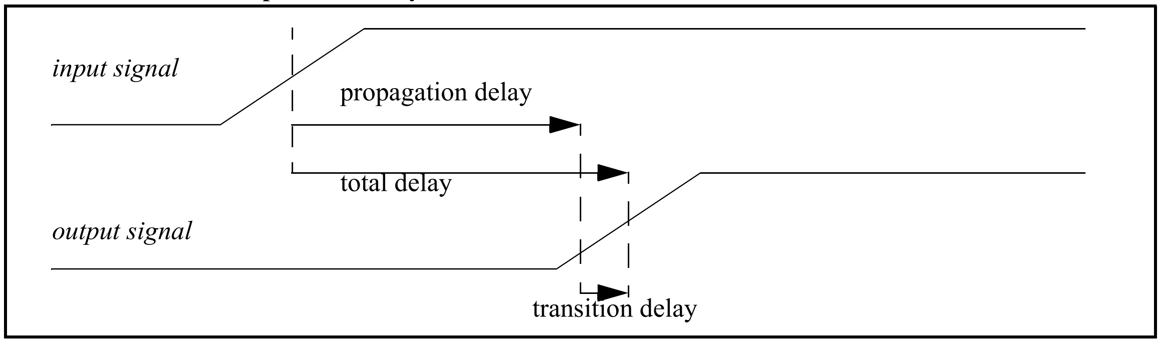
FIGURE 4. Description of delay and slew measurements in ALF

```
VECTOR (VectorExpression) {
/* ALF_TimingModel */
  DELAY {
    FROM { PIN = RelatedPinName ; }
    TO { PIN = PinName ; }
    /* ALF_data */
  }
  SLEWRATE {
    PIN = PinName ;
    /* ALF_data */
  }
}
```

1.3 Propagation and transition delay

In Section 1.2 on page 4, delay and slew are defined as mutually independent measurements. In this section, however, delay is broken into two pieces, propagation delay and transition delay, as shown in Figure 3 on page 5.

FIGURE 5. Interdependent delay and slew measurements



Liberty contains either a total delay (using the keywords cell_rise and cell_fall) or a propagation delay (using the keywords rise_propagation and fall_propagation).

TABLE 3. Interpretation of delay components in liberty

Liberty keyword	Liberty delay calculation concept
rise_propagation	cell_rise = rise_propagation + rise_transition
fall_propagation	cell_fall = fall_propagation + fall_transition

The start point of the transition delay measurement must be identical to the end point of the propagation delay measurement. Without this constraint, a propagation delay measurement amounts to the same as a previously defined delay easurement (i.e., a measurement involving threshold crossings of an input transition and an output transition), and a transition delay measurement amounts to the same as a slew measurement (i.e., a measurement involving threshold crossings of an output transition).

ALF does not have special keywords to express the propagation/transition delay concept. The standard keywords DELAY and SLEWRATE are used. To express the concept of adding two delay components together, the following construct can be used.

FIGURE 6. Description of the propagation/transition delay concept in ALF

```
VECTOR (VectorExpression) {
/* ALF_TimingModel */
  DELAY propagationID {
    FROM { PIN = RelatedPinName ; }
    TO { PIN = PinName ; }
    /* ALF_data */
  }
  SLEWRATE slewID {
    PIN = PinName ;
    /* ALF_data */
  }
  DELAY transitionID {
    TO { PIN = PinName ; }
    CALCULATION = incremental ;
    MODEL = slewID ;
  }
}
```

The delay named “transitionID” is to be added to the delay named “propagationID”, due to the CALCULATION annotation. The data of the delay named “transitionID” are refered from the slew named “slewID”, due to the MODEL annotation.

1.4 Delay and slew with retain

FIGURE 7. Retain, delay, and slew measurements

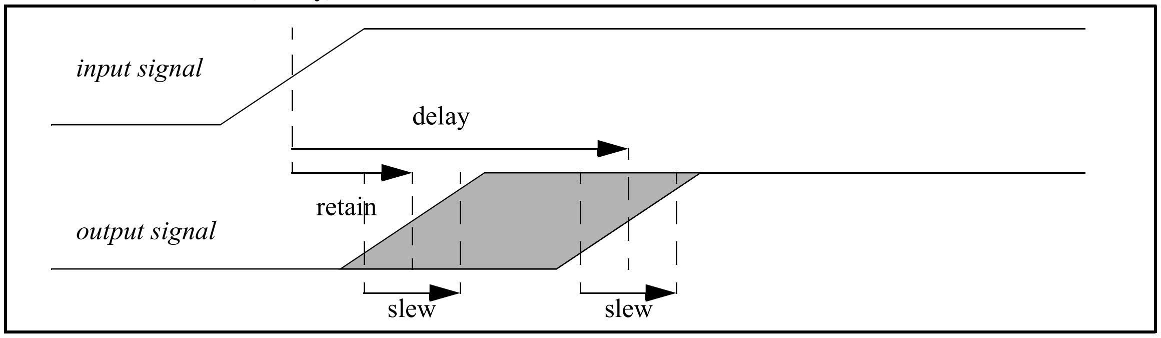


TABLE 4. Mapping of Liberty and ALF constructs for retain, delay, and slew measurements

Liberty construct			**ALF construct** PN = Pin Name, RPN = Related Pin Name	
Timing Type	***Timing Sense***	***Model Keyword***	***Keyword***	***Vector Expression***
combinational	positive_unate	retaining_rise	RETAIN	01 RPN -> 0* PN -> *1 PN
		retain_rise_slew	SLEWRATE	
		cell_rise	DELAY	
		rise_transition	SLEWRATE	
		retaining_fall	RETAIN	10 RPN -> 1* PN -> *0 PN
		retain_fall_slew	SLEWRATE	
		cell_fall	RETAIN	
		fall_transition	SLEWRATE	
	negative_unate	retaining_rise	RETAIN	10 RPN -> 0* PN -> *1 PN
		retain_rise_slew	SLEWRATE	
		cell_rise	DELAY	
		rise_transition	SLEWRATE	
		retaining_fall	RETAIN	01 RPN -> 1* PN -> *0 PN
		retain_fall_slew	SLEWRATE	
		cell_fall	DELAY	
		fall_transition	SLEWRATE	
	non_unate	retaining_rise	RETAIN	?! RPN -> 0* PN -> *1 PN
		retain_rise_slew	SLEWRATE	
		cell_rise	DELAY	
		rise_transition	SLEWRATE	
		retaining_fall	RETAIN	?! RPN -> 1* PN -> *0 PN
		retain_fall_slew	SLEWRATE	
		cell_fall	DELAY	
		fall_transition	SLEWRATE	

FIGURE 8. Description of retain, delay, and slew measurements in ALF

```
VECTOR (VectorExpression) {
/* ALF_TimingModel */
  RETAIN {
    FROM { PIN = RelatedPinName ; }
    TO { PIN = PinName ; EDGE_NUMBER = 0 ; }
    /* ALF_data */
  }
  SLEWRATE SlewForEdgeNumber0 {
    PIN = PinName ; EDGE_NUMBER = 0 ;
    /* ALF_data */
  }
  DELAY {
    FROM { PIN = RelatedPinName ; }
    TO { PIN = PinName ; EDGE_NUMBER = 1 ; }
    /* ALF_data */
  }
  SLEWRATE SlewForEdgeNumber1 {
    PIN = PinName ; EDGE_NUMBER = 1 ;
    /* ALF_data */
  }
}
```

1.5 Setup and hold

FIGURE 9. Setup and hold measurements

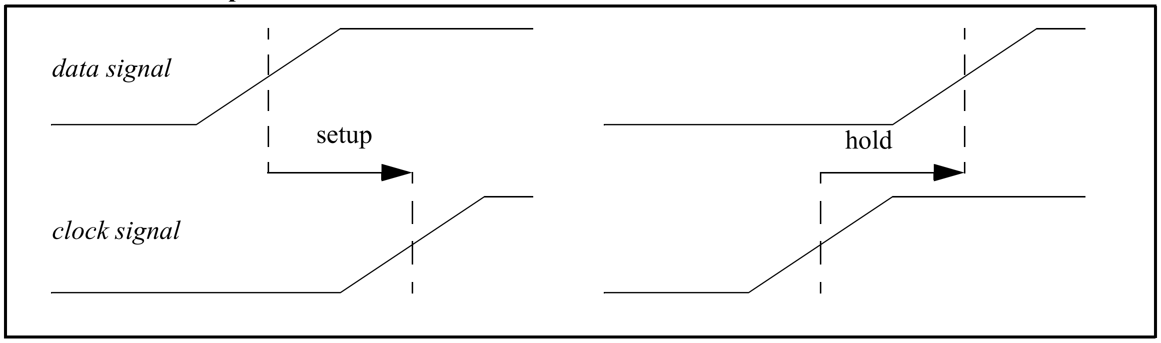


TABLE 5. Mapping of Liberty and ALF constructs for independent setup, hold

Liberty construct			**ALF construct** PN = Pin Name, RPN = Related Pin Name	
Timing Type	***Timing Sense***	***Model Keyword***	***Keyword***	***Vector Expression***
setup_rising	N/A	rise_constraint	SETUP	01 PN -> 01 RPN
		fall_constraint		10 PN -> 01 RPN
setup_falling	N/A	rise_constraint		01 PN -> 10 RPN
		fall_constraint		10 PN -> 10 RPN
hold_rising	N/A	rise_constraint	HOLD	01 RPN -> 01 PN
		fall_constraint		01 RPN -> 10 PN
hold_falling	N/A	rise_constraint		10 RPN -> 01 PN
		fall_constraint		10 RPN -> 10 PN
non_seq_setup_rising		rise_constraint	SETUP	01 PN -> 01 RPN
		fall_constraint		10 PN -> 01 RPN
non_seq_setup_falling		rise_constraint		01 PN -> 10 RPN
		fall_constraint		10 PN -> 10 RPN
non_seq_hold_rising		rise_constraint	HOLD	01 RPN -> 01 PN
		fall_constraint		01 RPN -> 10 PN
non_seq_hold_falling		rise_constraint		10 RPN -> 01 PN
		fall_constraint		10 RPN -> 10 PN

FIGURE 10. Description of independent setup and hold in ALF

```
VECTOR (VectorExpression) {
/* ALF_TimingModel */
  SETUP {
    FROM { PIN = PinName ; }
    TO { PIN = RelatedPinName ; }
    /* ALF_data */
  }
}
VECTOR (VectorExpression) {
/* ALF_TimingModel */
  HOLD {
    FROM { PIN = RelatedPinName ; }
    TO { PIN = PinName ; }
    /* ALF_data */
  }
}
```

1.6 Recovery and removal

FIGURE 11. Recovery and removal measurements

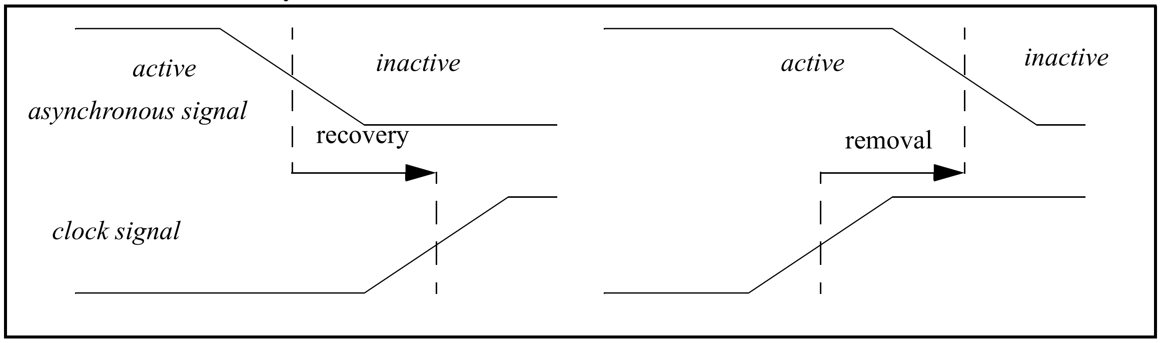


TABLE 6. Mapping of Liberty and ALF constructs for independent recovery, removal

Liberty construct			ALF construct PN = Pin Name, RPN = Related Pin Name	
Timing Type	***Timing Sense***	***Model Keyword***	***Keyword***	***Vector Expression***
recovery_rising	N/A	rise_constraint	RECOVERY	01 PN -> 01 RPN
		fall_constraint		10 PN -> 01 RPN
recovery_falling	N/A	rise_constraint		01 PN -> 10 RPN
		fall_constraint		10 PN -> 10 RPN
removal_rising	N/A	rise_constraint	REMOVAL	01 RPN -> 01 PN
		fall_constraint		01 RPN -> 10 PN
removal_falling	N/A	rise_constraint		10 RPN -> 01 PN
		fall_constraint		10 RPN -> 10 PN

FIGURE 12. Description of independent recovery and removal in ALF

```
VECTOR (VectorExpression) {
/* ALF_TimingModel */
  RECOVERY {
    FROM { PIN = PinName ; }
    TO { PIN = RelatedPinName ; }
    /* ALF_data */
  }
}
VECTOR (VectorExpression) {
/* ALF_TimingModel */
  REMOVAL {
    FROM { PIN = RelatedPinName ; }
    TO { PIN = PinName ; }
    /* ALF_data */
  }
}
```

1.7 Co-dependent setup and hold

FIGURE 13. Co-dependent setup and hold measurements

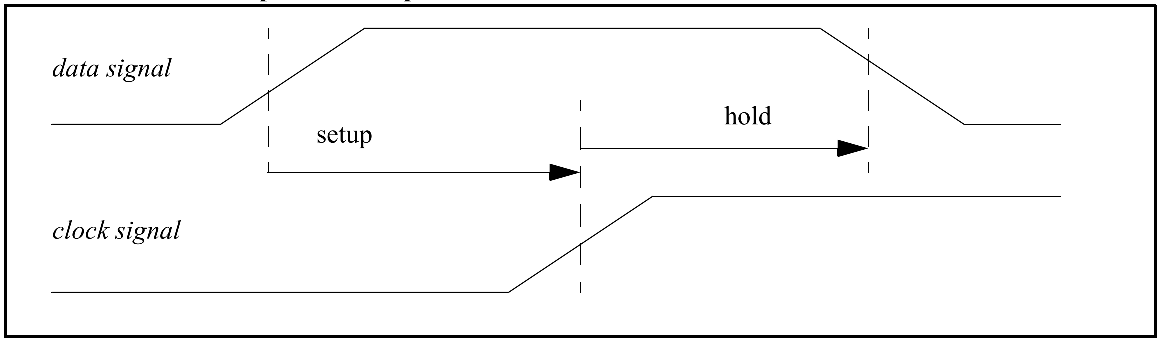


TABLE 7. Mapping of Liberty and ALF for co-dependent setup, hold

Liberty construct			ALF construct PN = Pin Name, RPN = Related Pin Name	
Timing Type	***Timing Sense***	***Model Keyword***	***Keyword***	***Vector Expression***
setup_rising	N/A	rise_constraint	SETUP	01 PN -> 01 RPN -> 10 PN
hold_rising	N/A	fall_constraint	HOLD	
setup_rising	N/A	fall_constraint	SETUP	10 PN -> 01 RPN -> 01 PN
hold_rising	N/A	rise_constraint	HOLD	

TABLE 7. Mapping of Liberty and ALF for co-dependent setup, hold

Liberty construct			**ALF construct** PN = Pin Name, RPN = Related Pin Name	
Timing Type	***Timing Sense***	***Model Keyword***	***Keyword***	***Vector Expression***
setup_falling	N/A	rise_constraint	SETUP	01 PN -> 10 RPN -> 10 PN
hold_falling	N/A	fall_constraint	HOLD	
setup_falling	N/A	fall_constraint	SETUP	10 PN -> 10 RPN -> 01 PN
hold_falling	N/A	rise_constraint	HOLD	

FIGURE 14. Description of co-dependent setup and hold in ALF

```
VECTOR (VectorExpression) {
/* ALF_TimingModel */
  SETUP {
    FROM { PIN = PinName ; EDGE_NUMBER = 0 ; }
    TO { PIN = RelatedPinName ; }
    /* ALF_data */
  }
  HOLD {
    FROM { PIN = RelatedPinName ; }
    TO { PIN = PinName ; EDGE_NUMBER = 1 ; }
    /* ALF_data */
  }
}
```

1.8 Co-dependent recovery and removal

FIGURE 15. Co-dependent recovery and removal measurements

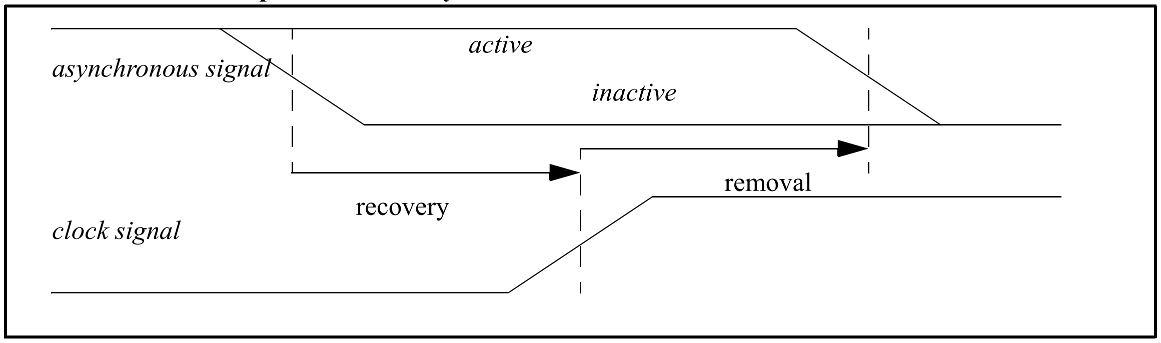


TABLE 8. Mapping of Liberty and ALF for co-dependent recovery, and removal

Liberty construct			ALF construct PN = Pin Name, RPN = Related Pin Name	
Timing Type	***Timing Sense***	***Model Keyword***	***Keyword***	***Vector Expression***
recovery_rising	N/A	rise_constraint	RECOVERY	01 PN <&> 01 RPN
removal_rising	N/A	rise_constraint	REMOVAL	
recovery_rising	N/A	fall_constraint	RECOVERY	10 PN <&> 01 RPN
removal_rising	N/A	fall_constraint	REMOVAL	
recovery_falling	N/A	rise_constraint	RECOVERY	01PN <&> 10 RPN
removal_falling	N/A	rise_constraint	REMOVAL	
recovery_falling	N/A	fall_constraint	RECOVERY	10 PN <&> 10 RPN
removal_falling	N/A	fall_constraint	REMOVAL	

FIGURE 16. Description of co-dependent recovery and removal in ALF

```
VECTOR (VectorExpression) {
/* ALF_TimingModel */
  RECOVERY {
    FROM { PIN = PinName ; }
    TO { PIN = RelatedPinName ; }
    /* ALF_data */
  }
  REMOVAL {
    FROM { PIN = RelatedPinName ; }
    TO { PIN = PinName ; }
    /* ALF_data */
  }
}
```

1.9 Setup and hold with nochange

FIGURE 17. Setup and hold measurements with nochange constraint

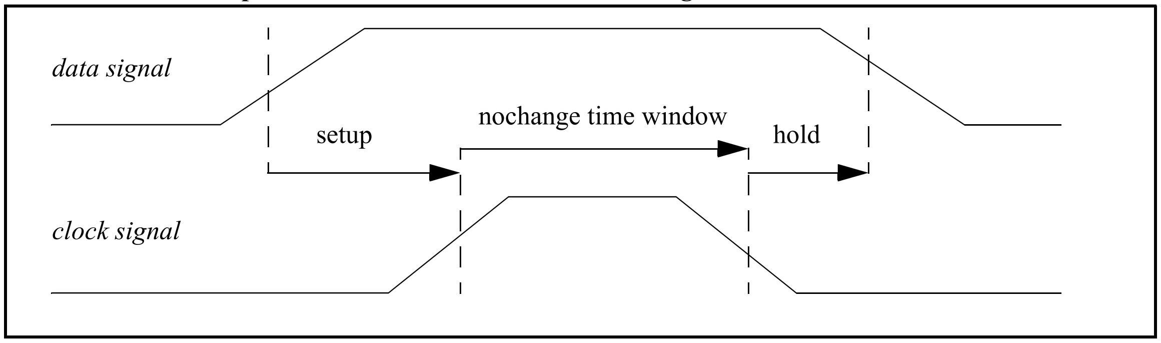


TABLE 9. Mapping of Liberty and ALF for setup and hold with nochange constraint

Liberty construct			ALF construct PN = Pin Name, RPN = Related Pin Name	
Timing Type	***Timing Sense***	***Model Keyword***	***Keyword***	***Vector Expression***
nochange_high_high		rise_constraint	SETUP	01 PN -> 01 RPN -> 10 RPN -> 10 PN
		fall_constraint	HOLD	
nochange_high_low		rise_constraint	SETUP	01 PN -> 10 RPN -> 01 RPN -> 10 PN
		fall_constraint	HOLD	
nochange_low_high		fall_constraint	SETUP	10 PN -> 01 RPN -> 10 RPN -> 01 PN
		rise_constraint	HOLD	
nochange_low_low		fall_constraint	SETUP	10 PN ->10 RPN -> 01 RPN -> 01 PN
		rise_constraint	HOLD	

FIGURE 18. Description of setup and hold with nochange constraint in ALF

```
VECTOR (VectorExpression) {
/* ALF_TimingModel */
  SETUP {
    FROM { PIN = PinName ; EDGE_NUMBER = 0 ; }
    TO { PIN = RelatedPinName ; EDGE_NUMBER = 0 ; }
    /* ALF_data */
  }
  HOLD {
    FROM { PIN = RelatedPinName ; EDGE_NUMBER = 1 ; }
    TO { PIN = PinName ; EDGE_NUMBER = 1 ; }
    /* ALF_data */
  }
  NOCHANGE {
    FROM { PIN = RelatedPinName ; EDGE_NUMBER = 0 ; }
    TO { PIN = RelatedPinName ; EDGE_NUMBER = 1 ; }
  }
}
```

1.10 Maximum skew constraint

FIGURE 19. Maximum skew constraint

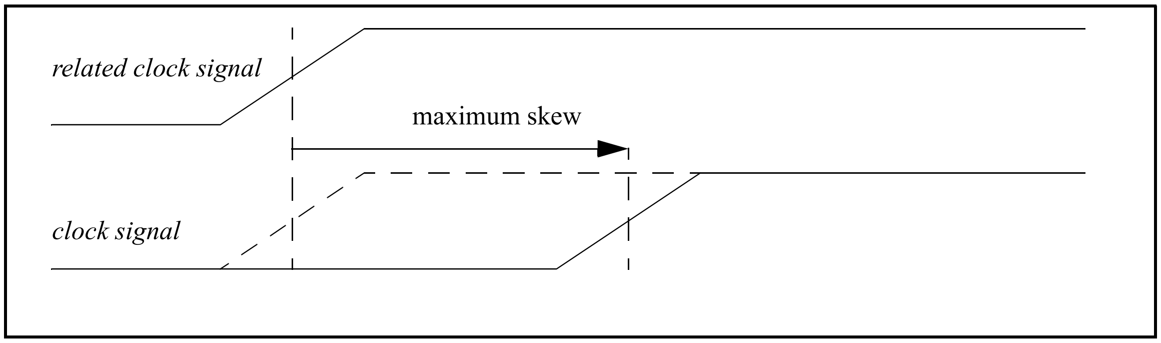


TABLE 10. Mapping of Liberty and ALF constructs for maximum skew constraint

Liberty construct			ALF construct PN = Pin Name, RPN = Related Pin Name	
Timing Type	***Timing Sense***	***Model Keyword***	***Keyword***	***Vector Expression***
skew_rising	N/A	rise_constraint	SKEW	01 RPN -> 01 PN
		fall_constraint		01 RPN -> 10 PN
skew_falling	N/A	rise_constraint		10 RPN -> 01 PN
		fall_constraint		10 RPN -> 10 PN

FIGURE 20. Description of maximum skew constraint in ALF

```
VECTOR (VectorExpression) {
/* ALF_TimingModel */
  LIMIT {
    SKEW {
      PIN { PinName RelatedPinName } // pin order is irrelevant here
      MAX {
        /* ALF_data */
      }
    }
  }
}
```

1.11 Minimum period and minimum pulsewidth constraints

FIGURE 21. Minimum period and minimum pulsewidth constraints

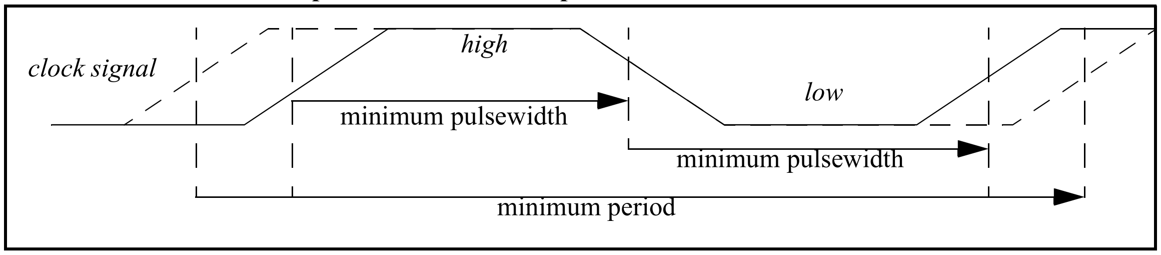


TABLE 11. Mapping of Liberty and ALF constructs for minimum period and minimum pulsewidth constraints

Liberty construct (pin-based)	**alternative Liberty construct (arc-based)**		**ALF construct** PN = Pin Name, RPN = Related Pin Name	
Attribute Keyword	***Timing Type***	***Model Keyword***	***Keyword***	***Vector Expression***
min_period	minimum_period	constraint	PERIOD	01 PN[a]
				10 PN[b]
min_pulse_width_high	min_pulse_width	constraint_high	PULSEWIDTH	01 PN -> 10 PN
min_pulse_width_low	min_pulse_width	constraint_low		10 PN -> 01 PN

a. for positive-edge triggered clock

b. for negative-edge triggered clock

FIGURE 22. Description of minimum period constraint in ALF

```
VECTOR (VectorExpression) {
/* ALF_TimingModel */
  LIMIT {
    PERIOD {
      MIN { /* ALF_data */ }
    }
  }
}
```

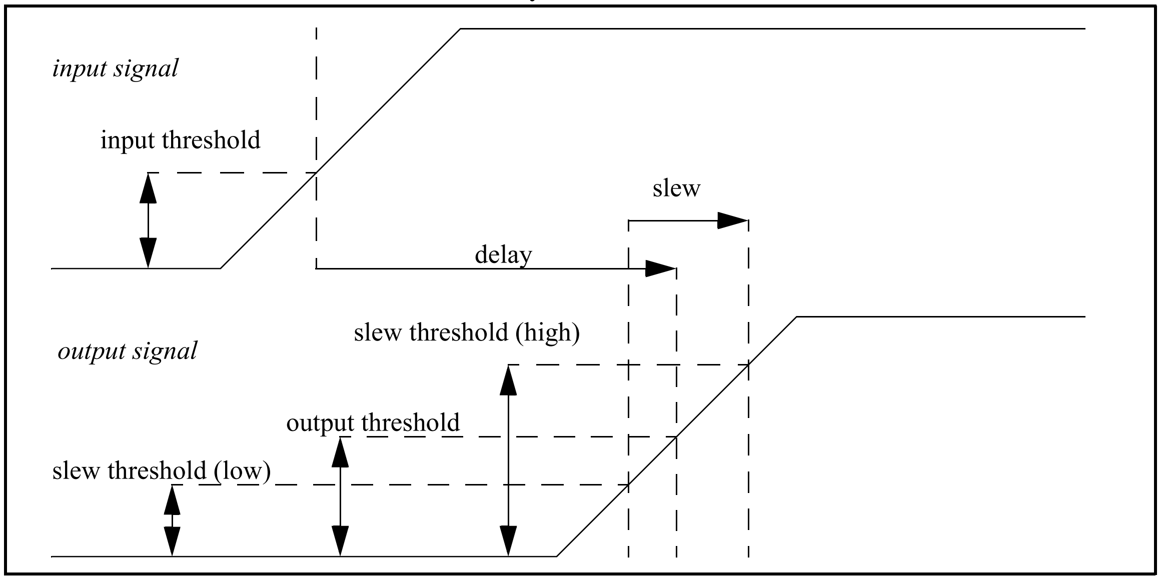
FIGURE 23. Description of minimum pulsewidth constraint in ALF

```
VECTOR (VectorExpression) {
/* ALF_TimingModel */
  LIMIT {
    PULSEWIDTH {
      PIN = PinName ;
      MIN { /* ALF_data */ }
    }
  }
}
```

1.12 Threshold definitions

The purpose of threshold definitions is to preserve the reference measurement points for delay and slew measurements in the presence of non-linear waveforms. Especially in long interconnect, a relatively linear shape of a waveform at a driver output degrades to an almost exponential shae at a receiving input.

FIGURE 24. Threshold definitions for delay and slew



The applicable threshold values (input or output) depend on the direction of a pin.

The thresholds for delay and slew measurements in Liberty are normalized values between 0 and 100, to be interpreted as percentage values. The corresponding thresholds in ALF are normalized values between 0 and 1. Therefore, the conversion involves either dividing Liberty data by 100 or multiplying ALF data by 100.

Per default, the slew data in the library are understood to be the measured values according to the slew threshold definitions. However, the slew data might be represented in a normalized way, for example scaled from rail-to-rail. In order to allow for such a normalized representation, a scaling factor can be defined. The slew data multiplied with the scaling factor is then understood to be the measured values according to the slew threshold definitions. In Liberty, the keyword slew_derate_from_library defines the scaling factor. The scaling factor multiplied with the base unit defines the absolute slew data. In ALF, the UNIT annotation defines the multiplier, i.e., the product of scaling factor and base unit.

To make the differences between Liberty and ALF clearer, numerical values are shown in the following Figure 25 on page 21 and Figure 26 on page 22.

FIGURE 25. Liberty description of library threshold definitions

```
/* Liberty */
library (LibraryName) {
  time_unit : "1ns" ;
  input_threshold_pct_rise : 45 ;
  input_threshold_pct_fall : 55 ;
  output_threshold_pct_rise : 35 ;
  output_threshold_pct_fall : 65 ;
  slew_lower_threshold_pct_rise : 30 ;
  slew_upper_threshold_pct_rise : 50 ;
  slew_upper_threshold_pct_fall : 70 ;
  slew_lower_threshold_pct_fall : 50 ;
  slew_derate_from_library : 0.2 ;
}
```

The following restriction applies for slew thresholds:

```
   slew_upper_threshold_pct_rise - slew_lower_threshold_pct_rise
=  slew_upper_threshold_pct_fall - slew_lower_threshold_pct_fall
```

In this example, 50 - 30 = 70 - 50 = 20.

FIGURE 26. ALF description of library threshold definitions

```
/* ALF */
LIBRARY LibraryName {
  TIME { UNIT = 1e-9 ; }
  DELAY {
    FROM {
      THRESHOLD {
        RISE = 0.45 ;
        FALL = 0.55 ;
      }
    }
    TO {
      THRESHOLD {
        RISE = 0.35 ;
        FALL = 0.65 ;
      }
    }
  }
  SLEWRATE {
    UNIT = 0.2e-9 ;
    FROM {
      THRESHOLD {
        RISE = 0.3 ;
        FALL = 0.7 ;
      }
    }
    TO {
      THRESHOLD {
        RISE = 0.5 ;
        FALL = 0.5 ;
      }
    }
  }
}
```

According to this example, a numerical slew value of “1” really means 0.2ns, measured from 30% to 50% for rising transition and from 70% to 50% for falling transition, respectively.

1.13 Conditional timing arcs

The *existence condition* for a timing arc is the necessary and sufficient condition for a timing arc to be activated. A *value condition* is a sufficient condition.

Mathematically, the existence condition can be expressed as a boolean expression in a sum-of-product form.

For example, a timing arc from input `A` to output `Y` can be activated, if the existence condition (`E1|E2`) is satisfied, where `E1` and `E2` are side inputs. The sum-of-product form of the existence condition reads as follows:

```
E1 | E2 = E1 & E2 | E1 & !E2 | !E1 & E2
```

The delay from `A` to `Y` depends possibly on the state of `E1` and `E2`. The value condition is a particular state for which a particular value applies. It can be either (`E1&E2`) or (`E1&!E2`) or (`!E1&E2`).

In Liberty, the *value condition* is expressed in a “when” statement. In ALF, the value condition is expressed as a co-factor within the vector expression.

In Liberty, the *existence condition* can not be described explicitly. However, the existence condition can be inferred either by evaluation of the “function” statement or by combining all the “when” statements of all timing groups with same pin, same related pin, same timing_type and same timing_sense. The same inference can be applied to ALF. However, ALF supports also an explicit statement for existence condition.

FIGURE 27. Conditional timing and existence condition example in Liberty and ALF

```
/* Liberty */                         /* ALF */
pin(Y) {
  timing() {                          VECTOR ((01 A -> 01 Y)&(E1&E2)) {
    timing_type : combinational;        EXISTENCE_CONDITION
    timing_sense : positive_unate;      = E1&E2 | E1&!E2 | !E1&E2 ;
    related_pin : "A";
    when : "E1&E2";                     DELAY ...
    cell_rise ...                       SLEWRATE ...
    rise_transition ...               }
  }
  timing() {                          VECTOR ((01 A -> 01 Y)&(E1&!E2)) {
    timing_type : combinational;        EXISTENCE_CONDITION
    timing_sense : positive_unate;      = E1&E2 | E1&!E2 | !E1&E2 ;
    related_pin : "A";
    when : "E1&!E2";                    DELAY ...
    cell_rise ...                       SLEWRATE ...
    rise_transition ...               }
  }
  timing() {                          VECTOR ((01 A -> 01 Y)&(!E1&E2)) {
    timing_type : combinational;        EXISTENCE_CONDITION
    timing_sense : positive_unate;      = E1&E2 | E1&!E2 | !E1&E2 ;
    related_pin : "A";
    when : "!E1&E2";                    DELAY ...
    cell_rise ...                       SLEWRATE ...
    rise_transition ...               }
  }
}
/* inferred existence condition:
   E1&E2 | E1&!E2 | !E1&E2 */
```

A “when_start” and a “when_end” statement in Liberty means that the condition is checked at the time of the FromPin event and the ToPin event, respectively.

In ALF, these conditions are described as co-factors in the vector expression.

FIGURE 28. Timing with start and end condition in Liberty and ALF

```
/* Liberty */                        /* ALF */
pin(Y) {
  timing() {                         VECTOR
    timing_type : combinational;     ((01 A) & E1 ~> (01 Y) & E2) {
    timing_sense : positive_unate;
    related_pin : "A";                 DELAY ...
    when_start : "E1";                 SLEWRATE ...
    when_end : "E2";                 }
    cell_rise ...
    rise_transition ...
  }
}
```

A default value condition in ALF is specified by a timing model with **`CALCULATION`** annotation value **`absolute`** in the context of an ALF vector without boolean co-factor[1].

The default condition shall apply in the following situations:

- None of the specified value conditions evaluates true
- The application context (e.g. a set of timing constraints) does not provide enough information to decide whether one, and only one, specified value condition evaluates true
- The application tool does not support evaluation of a value condition

1.14 Timing arcs involving bus pins

Timing arcs involving a bus can be described in a compact way without enumerating the timing arc for each bit.

Timing arcs between two buses where there is a one-to-one correspondence between bits of each bus, shall be extended bitwise, i.e., a timing arc exists between every bit of one bus and the corresponding bit of the related bus.

The usage of the *type* statement in Liberty is a prerequisite for defining a bus. If a timing statement within a bus defines a related pin of the same *bus type*, the timing arc shall be expanded bitwise.

1. If the value of the **`CALCULATION`** annotation is **`absolute`**, then the timing data shall not be combined with any other timing data. If the value of the **`CALCULATION`** annotation is **`incremental`**, then the timing data can be combined with other timing data.

FIGURE 29. Timing arc on a bus with bit-to-bit extension in Liberty

```
cell(CellName) {
  type(DataBit) {
    base_type : array ;
    data_type : bit ;
    bit_width : 8 ;
    bit_from : 1 ;
    bit_to : 8 ;
    downto : false ;
  }
  bus(DataBusIn) {
    direction : input ;
    bus_type : DataBit ;
  }
  bus(DataBusOut) {
    direction : output ;
    bus_type : DataBit ;
    timing() {
      timing_type  : combinational;
      timing_sense : positive_unate;
      related_pin : "DataBusIn";
      cell_rise (scalar) { values (”1.0”); }
    }
  }
}
```

The usage of a GROUP statement in ALF is a prerequisite for defining an expandable timing arc. A timing VECTOR containing the same GROUP identifier for the FROM and the TO pin shall be expanded bitwise.

FIGURE 30. Timing arc on a bus with bit-to-bit extension in ALF

```
CELL CellName {
  GROUP DataBit { 1 : 8 }
  PIN [1:8] DataBusIn  { DIRECTION = input ; }
  PIN [1:8] DataBusOut { DIRECTION = output ; }
  VECTOR ( 01 DataBusIn[DataBit] -> 01 DataBusOut[DataBit] ) {
    DELAY = 1.0 {
      FROM { PIN = DataBusIn[DataBit] ; }
      TO   { PIN = DataBusOut[DataBit] ; }
    }
  }
}
```

Assumption: The following rule is supported.

If a Liberty timing statement within a bus defines a related pin [of a different *bus type*,] using the keyword *related_bus_pins* rather than *related_pin*, the timing arc shall be expanded by permutation from every bit to every bit.

FIGURE 31. Timing arc on a bus with all-to-all extension in Liberty

```
cell(CellName) {
  type(AddressBit) {
    base_type : array ;
    data_type : bit ;
    bit_width : 4 ;
    bit_from : 3 ;
    bit_to : 0 ;
    downto : true ;
  }
  type(DataBit) {
    base_type : array ;
    data_type : bit ;
    bit_width : 8 ;
    bit_from : 1 ;
    bit_to : 8 ;
    downto : false ;
  }
  bus(AddressBus) {
    direction : input ;
    bus_type : AddressBit ;
  }
  bus(DataBusOut) {
    direction : input ;
    bus_type : DataBit ;
    timing() {
      timing_type  : combinational;
      timing_sense : positive_unate;
      related_bus_pins : "AddressBus";
      cell_rise (scalar) { values (”1.0”); }
    }
  }
}
```

If the assumption was false, the following style should work:

```
cell(CellName) {
  type(AddressBit) {
    base_type : array ;
    data_type : bit ;
    bit_width : 4 ;
    bit_from : 3 ;
    bit_to : 0 ;
    downto : true ;
  }
  type(DataBit) {
    base_type : array ;
    data_type : bit ;
    bit_width : 8 ;
    bit_from : 1 ;
    bit_to : 8 ;
    downto : false ;
  }
  bus(AddressBus) {
    direction : input ;
    bus_type : AddressBit ;
    pin(Addr[3:0]) {
    }
  }
  bus(DataBusOut) {
    direction : input ;
    bus_type : DataBit ;
    timing() {
      timing_type  : combinational;
      timing_sense : positive_unate;
      related_pin : "Addr[0] Addr[1] Addr[2] Addr[3]";
      cell_rise (scalar) { values (”1.0”); }
    }
```

Questions:

When declaring a vector pin within a bus (e.g. Addr[3:0] within AddressBus), can one chose another name for the pin (e.g. Addr) or must one use the same name as for the bus (e.g. AddressBus)?

Is it necessary to declare a vector pin also in the reference bus (e.g. DataBusOut)?

Are relationships involving bundles (see power relationships) also supported for timing?

A timing VECTOR containing two different GROUP identifiers for the FROM and the TO pin shall be expanded by permutation from every bit to every bit.

FIGURE 32. Timing arc on a bus with all-to-all extension in ALF

```
CELL CellName {
  GROUP AddressBit { 0 : 3 }
  GROUP DataBit { 1 : 8 }
  PIN [3:0] AddressBus { DIRECTION = input ; }
  PIN [1:8] DataBusOut { DIRECTION = output ; }
  VECTOR ( 01 AddressBus[AddressBit] -> 01 DataBusOut[DataBit] ) {
    DELAY = 1.0 {
      FROM { PIN = AddressBus[AddressBit] ; }
      TO   { PIN = DataBusOut[DataBit] ; }
    }
  }
}
```

2.0 Interoperability with SDF

2.1 SDF cross-reference overview

TABLE 12. Cross-reference between library constructs and SDF constructs

SDF construct	Comment
PATHPULSE	N/A
PATHPULSEPERCENT	N/A
ABSOLUTE	N/A
INCREMENT	N/A
IOPATH	delay measurement, see Table 2 on page 5, Figure 4 on page 6
RETAIN	see Table 4 on page 9, Figure 8 on page 10
COND	see Section 1.13 on page 22, Section 2.2 on page 30
CONDELSE	N/A
PORT	N/A
INTERCONNECT	see note [a]
NETDELAY	N/A
DEVICE	N/A
SETUP	see Table 5 on page 11, Figure 12 on page 13
HOLD	see Table 5 on page 11, Figure 12 on page 13
SETUPHOLD	see Table 7 on page 13, Figure 14 on page 14
RECOVERY	see Table 5 on page 11, Figure 12 on page 13
REMOVAL	see Table 5 on page 11, Figure 12 on page 13
RECREM	see Table 7 on page 13, Figure 14 on page 14
SKEW	see Table 10 on page 18, Figure 20 on page 18
BIDIRECTSKEW	see Section 1.10 on page 17
WIDTH	see Table 11 on page 19, Figure 23 on page 19
PERIOD	see Table 11 on page 19, Figure 22 on page 19
NOCHANGE	see Table 9 on page 16, Figure 18 on page 17
SCOND	see Section 2.2 on page 30
CCOND	see Section 2.2 on page 30
LABEL	see Section 2.3 on page 31

a. No library construct for interconnect, but other library constructs, such as *threshold* (see “Threshold definitions” on page 20) influence the result of interconnect delay calculation.

2.2 Conditions in SDF

Conditions in SDF are expressed in Verilog syntax, which is different from Liberty syntax. Therefore, Liberty provides “SDF_cond”, “SDF_cond_start”, “SDF_cond_end” statements, which are basically “when”, “when_start”, “when_end” statements translated into Verilog syntax.

TABLE 13. Cross reference between SDF and Liberty keywords related to conditions

SDF keyword	Liberty keyword	Liberty native keyword
COND	SDF_cond	when
SCOND	SDF_cond_start	when_start
CCOND	SDF_cond_end	when_end

The ALF syntax for conditions closely matches the Verilog syntax. Therefore, “SDF_cond”, “SDF_cond_start”, “SDF_cond_end” are not provided as standard annotations in ALF. However, if desired, they can be defined as library-specific annotations in the following way:

```
KEYWORD SDF_cond = single_value_annotation {
   VALUETYPE = quoted_string;
   CONTEXT = VECTOR;
}
KEYWORD SDF_cond_start = single_value_annotation {
   VALUETYPE = quoted_string;
   CONTEXT = VECTOR;
}
KEYWORD SDF_cond_end = single_value_annotation {
   VALUETYPE = quoted_string;
   CONTEXT = VECTOR;
}
```

FIGURE 33. SDF condition example in Liberty and ALF

```
/* Liberty */                         /* ALF */
pin(Y) {
  timing() {                          VECTOR ((01 A -> 01 Y)&(E1&E2)) {
    timing_type : combinational;        SDF_cond = "E1==1’b1&& E2==1’b1";
    timing_sense : positive_unate;      DELAY ...
    related_pin : "A";                  SLEWRATE ...
    when : "E1&E2";                   }
    SDF_cond : "E1==1’b1&&E2==1’b1";
    cell_rise ...
    rise_transition ...
  }
}
```

FIGURE 34. SDF start and end condition in Liberty and ALF

```
/* Liberty */                       /* ALF */
pin(Y) {
  timing() {                        VECTOR
    timing_type : combinational;    ((01 A) & E1 ~> (01 Y) & E2) {
    timing_sense : positive_unate;    SDF_cond_start = "E1==1'b1" ;
    related_pin : "A";                SDF_cond_end = "E2==1'b1" ;
    when_start : "E1";                DELAY ...
    when_end : "E2";                  SLEWRATE ...
    SDF_cond_start : "E1==1'b1";    }
    SDF_cond_end : "E2==1'b1";
    cell_rise ...
    rise_transition ...
  }
}
```

2.3 Usage of a label in SDF

A label in SDF is used to establish a semantic-free correspondence between a model parameter described in a format other than SDF (e.g. Verilog, Liberty, or ALF) and a value definition for this parameter in the SDF file.

In particular, a LABEL statement can be used instead of a IOPATH statement. The IOPATH statement implies the semantics of a delay measurement, whereas the LABEL statement does not imply any semantics.

A named timing statement in Liberty can be used to define the name of a label. However, this timing statement can only contain one single timing model (e.g. cell_rise), otherwise it would be unclear which model corresponds to the data in the SDF file.

FIGURE 35. A named timing statement in Liberty

```
pin(PinName) {
  timing(myLabel) {
    /* put one timing model here */
  }
}
```

A LABEL annotation within a VECTOR in ALF can be used to define the name of a label. However, this VECTOR can only contain one single timing model (e.g. DELAY), otherwise it would be unclear which model corresponds to the data in the SDF file.

FIGURE 36. A vector with LABEL annotation in ALF

```
VECTOR (VectorExpression) {
  LABEL = myLabel ;
  /* put timing model here */
}
```

There can be cases where multiple models (e.g. delay and slew) within the same context (i.e., pin in Liberty, VECTOR in ALF) need to be referenced in the SDF file.

In Liberty, multiple named timing statements can be used.

FIGURE 37. Multiple named timing statements in Liberty

```
pin(PinName) {
  timing(myLabel) {
    cell_rise ...
  }
  timing(myLabel2) {
    rise_transition ...
  }
}
```

In ALF, it is illegal to duplicate a VECTOR with the same vector expression and different labels. Instead, multiple named arithmetic models within the same VECTOR can be used. If a LABEL annotation were also present in this case, it would only establish a reference to a single unnamed arithmetic model within the VECTOR.

FIGURE 38. A vector with multiple named arithmetic models in ALF

```
VECTOR (VectorExpression) {
  DELAY myLabel ...
  SLEWRATE myLabel2 ...
}
```