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

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 21 on page 17
  }
}
```

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 1.3 on page 6
independent setup, hold	see Section 1.4 on page 8
independent recovery, removal	see Section 1.5 on page 10
co-dependent setup, hold	see Section 1.6 on page 11
co-dependent recovery, removal	see Section 1.7 on page 12
setup, hold with nochange constraint	see Section 1.8 on page 13
maximum skew constraint	see Section 1.9 on page 15
minimum period and minimum pulsewidth constraint	see Section 1.10 on page 16

1.2 Delay and slew

FIGURE 3. Delay and slew measurements

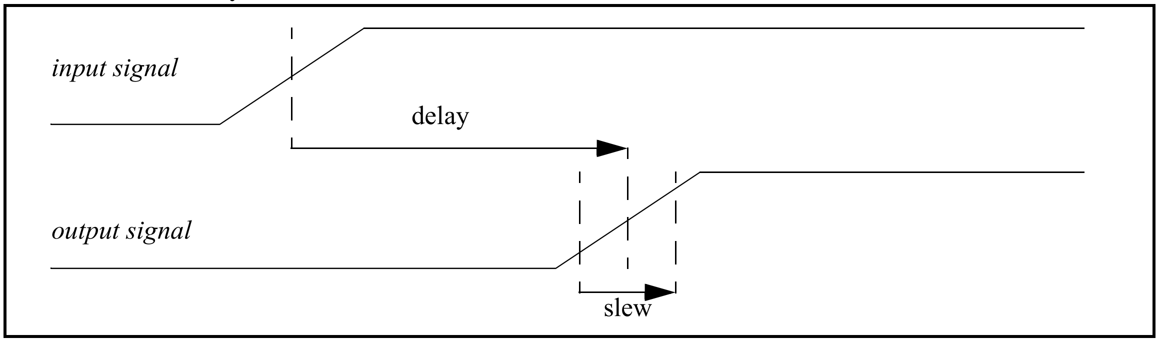


TABLE 2. Mapping of liberty and ALF constructs for delay and slew measurements

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	?	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	?	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 Delay and slew with retain

FIGURE 5. Retain, delay, and slew measurements

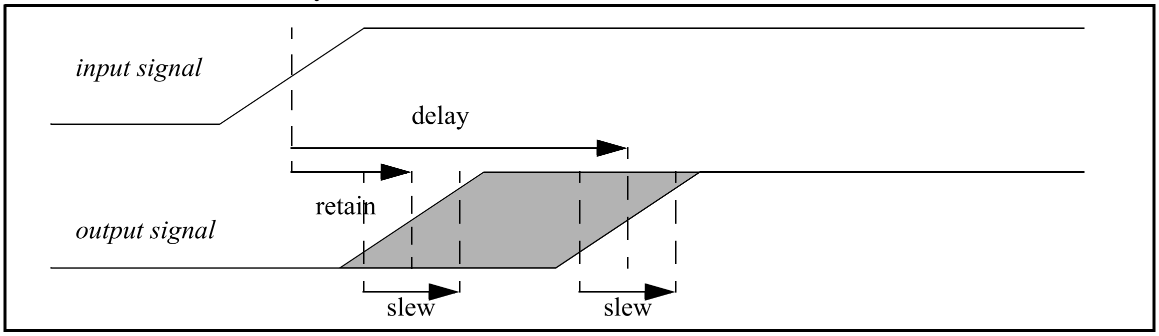


TABLE 3. Mapping of liberty and ALF constructs for retain, delay, and slew measurements

liberty construct			ALF construct PN = Pin Name, RPN = Related Pin Name	
Timing Type	***Timing Sense***	***Model Keyword***	***Keyword***	***Vector Expression***
?	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 6. 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 */
  }
}
```

1.4 Setup and hold

FIGURE 7. Setup and hold measurements

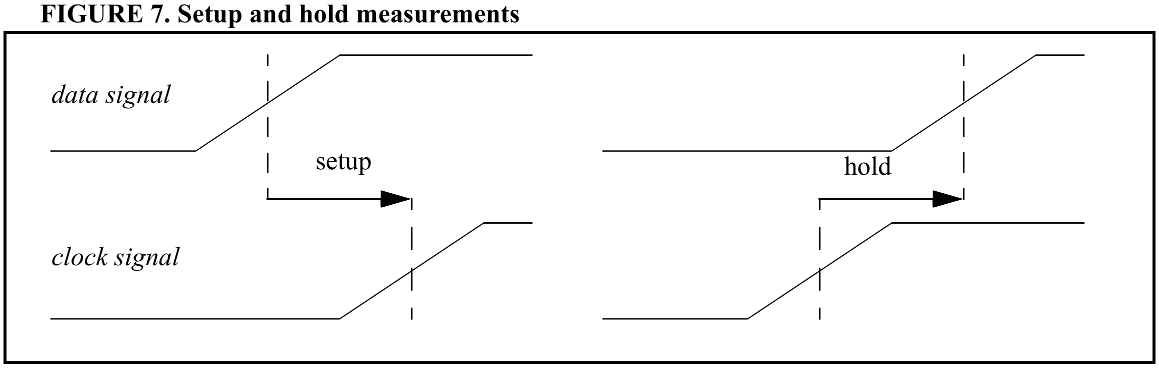


TABLE 4. Mapping of liberty and ALF constructs for independent setup, hold

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

FIGURE 8. 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 */
  }
}
```

1.5 Recovery and removal

FIGURE 9. Recovery and removal measurements

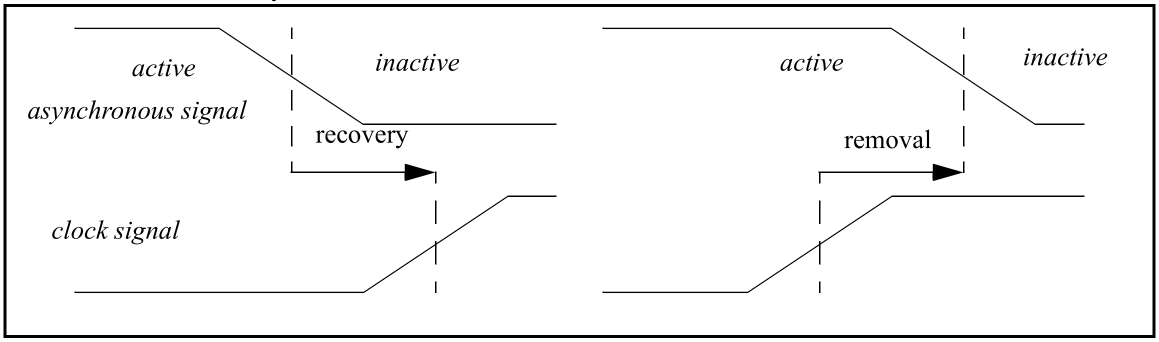


TABLE 5. Mapping of liberty and ALF constructs for independent recovery, removal

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

FIGURE 10. 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 */
  }
}
```

1.6 Co-dependent setup and hold

FIGURE 11. Co-dependent setup and hold measurements

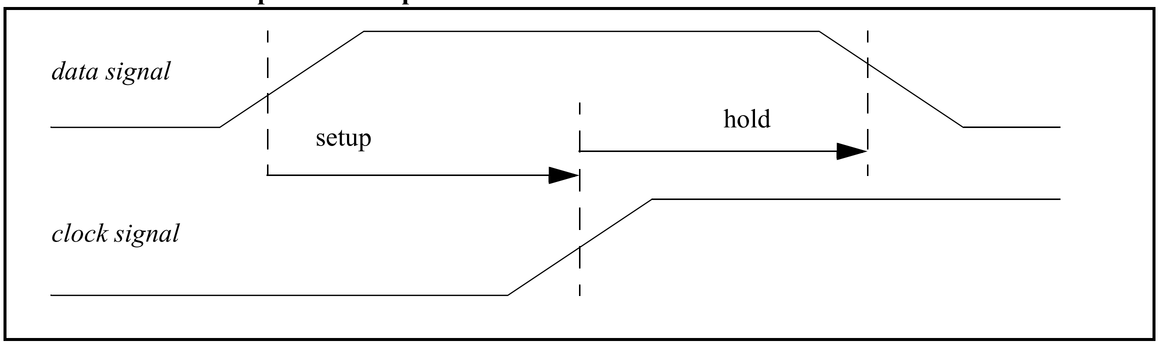


TABLE 6. Mapping of liberty and ALF for co-dependent setup, hold

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

TABLE 6. 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	?	rise_constraint	SETUP	01 PN -> 10 RPN -> 10 PN
hold_falling	?	fall_constraint	HOLD	
setup_falling	?	fall_constraint	SETUP	10 PN -> 10 RPN -> 01 PN
hold_falling	?	rise_constraint	HOLD	

FIGURE 12. 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 */
  }
}
```

1.7 Co-dependent recovery and removal

FIGURE 13. Co-dependent recovery and removal measurements

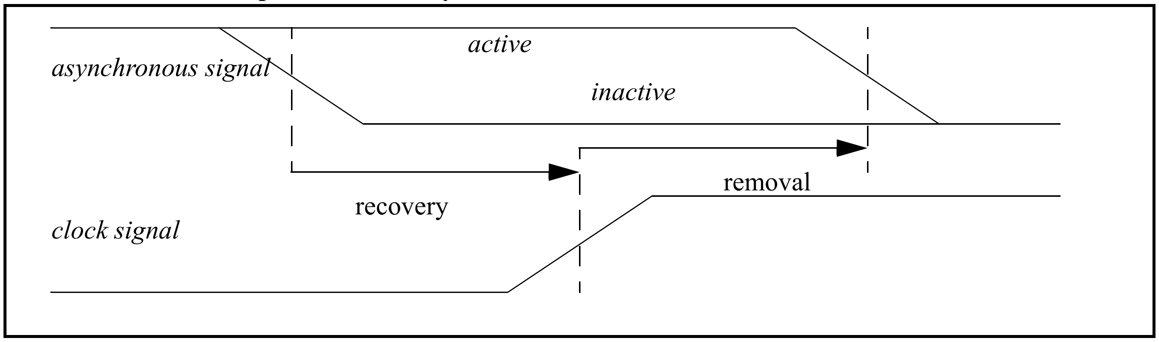


TABLE 7. Mapping of liberty and ALF for co-dependent recovery, and removal

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

FIGURE 14. 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 */
  }
}
```

1.8 Setup and hold with nochange

FIGURE 15. Setup and hold measurements with nochange constraint

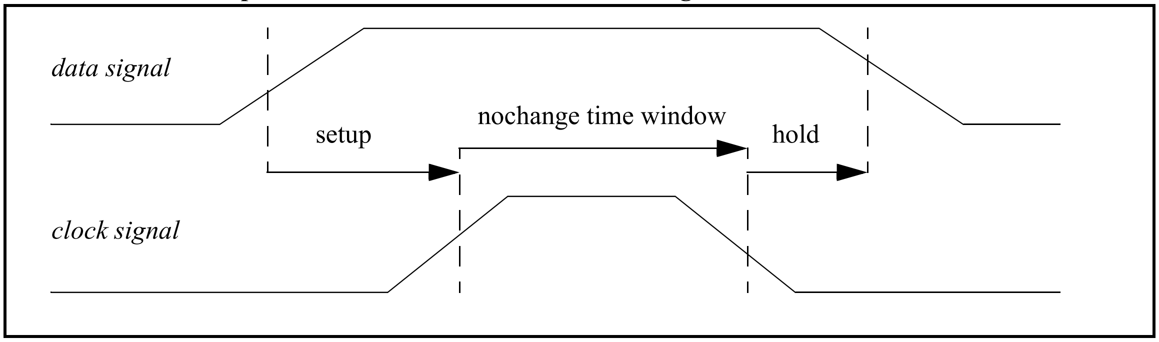


TABLE 8. Mapping of liberty and ALF for setup and hold with nochange constraint

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 16. 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 ; }
  }
}
```

1.9 Maximum skew constraint

FIGURE 17. Maximum skew constraint

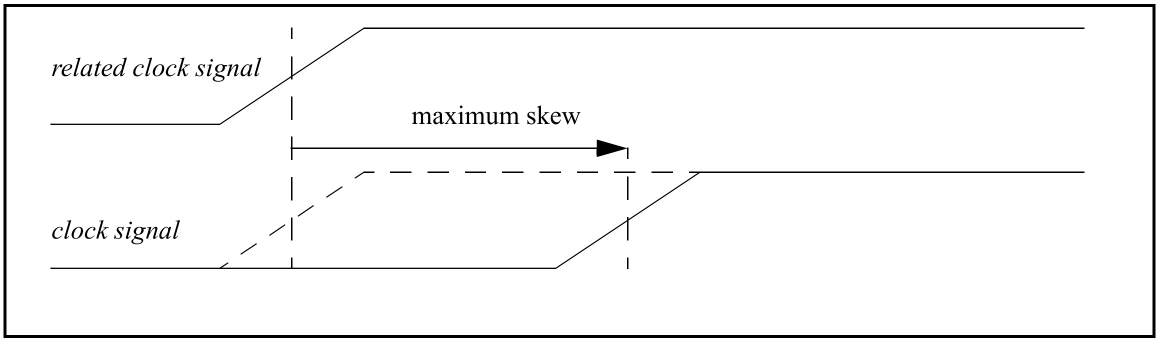


TABLE 9. Mapping of liberty and ALF constructs for maximum skew constraint

liberty construct			ALF construct PN = Pin Name, RPN = Related Pin Name	
Timing Type	***Timing Sense***	***Model Keyword***	***Keyword***	***Vector Expression***
skew_rising	?	intrinsic_rise ?	SKEW	01 RPN -> 01 PN
		intrinsic_fall ?		01 RPN -> 10 PN
skew_falling	?	intrinsic_rise ?		10 RPN -> 01 PN
		intrinsic_fall ?		10 RPN -> 10 PN

FIGURE 18. Description of maximum skew constraint in ALF

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

1.10 Minimum period and minimum pulsewidth constraints

FIGURE 19. Minimum period and minimum pulsewidth constraints

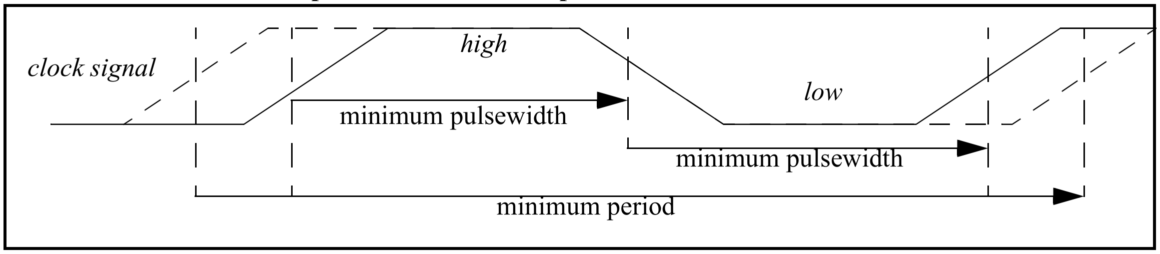


TABLE 10. Mapping of liberty and ALF constructs for minimum period and minimum pulsewidth constraints

liberty construct	ALF construct PN = Pin Name, RPN = Related Pin Name		
Attribute Keyword	*Keyword*	*Vector Expression*	**Comment**
min_period	PERIOD	01 PN	for positive edge triggered clock
		10 PN	for negative edge triggered clock
min_pulse_width_high	PULSEWIDTH	01 PN -> 10 PN	01 PN -> 10 PN
min_pulse_width_low		10 PN -> 01 PN	10 PN -> 01 PN

min_period or minimum_period (inconsistency in liberty ref manual)?

FIGURE 20. Description of minimum period constraint in ALF

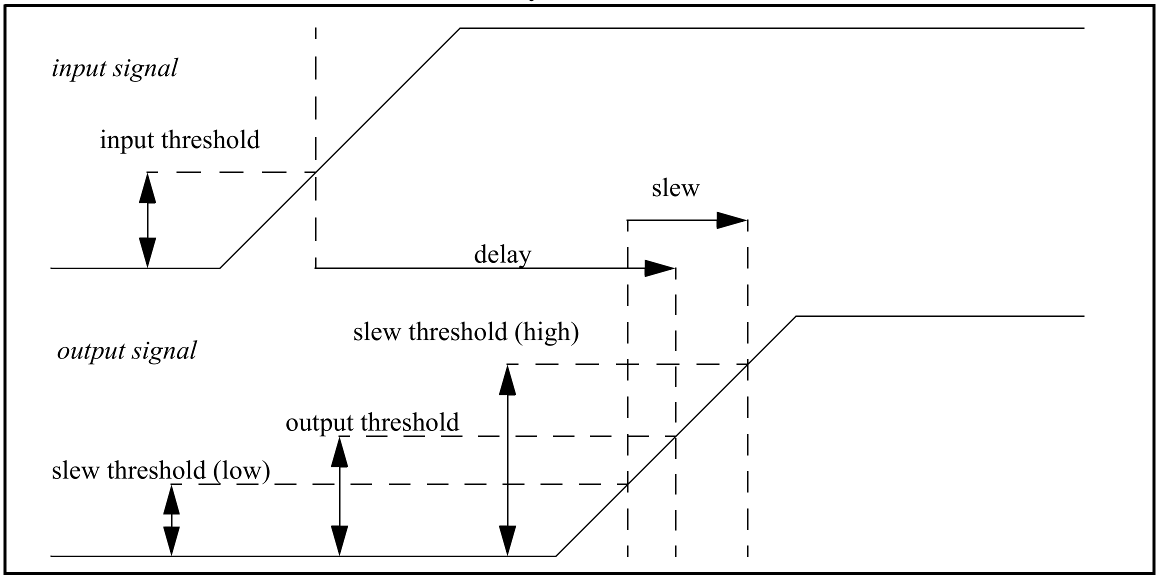
```
VECTOR (VectorExpression) {
/* ALF_TimingModel */
  LIMIT {
    PERIOD {
      MIN { /* ALF_data */ }
    }
  }
}
```

FIGURE 21. Description of minimum pulsewidth constraint in ALF

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

1.11 Threshold definitions

FIGURE 22. Threshold definitions for delay and slew



Do the ì inputî thresholds always apply for ì related pinî and the ì outputî thresholds for the parent pin or do the applicable thresholds depend on the ì directionî of the pin (not clear in case of setup, hold, recovery, removal)?

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.

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 23 on page 19 and Figure 24 on page 19.

FIGURE 23. 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 ;
}
```

FIGURE 24. 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 ì 1î really means 0.2ns, measured from 30% to 50% for rising transition and from 70% to 50% for falling transition, respectively.

1.12 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 ì whenî 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 ì functionî statement or by combining all the ì whenî 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 25. 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 ìwhen_startî and a ìwhen_endî 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 26. 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 ...
  }
}
```

1.13 Timing arcs involving bus pins

Fill in liberty examples

FIGURE 27. 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] ; }
    }
  }
}
```

FIGURE 28. 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

TABLE 11.

SDF construct	**Comment**
PATHPULSE	
PATHPULSEPERCENT	
ABSOLUTE	
INCREMENT	
IOPATH	delay measurement, see Table 2 on page 5, Figure 4 on page 6
RETAIN	see Table 3 on page 7, Figure 6 on page 8
COND	
CONDELSE	
PORT	
INTERCONNECT	
NETDELAY	
DEVICE	
SETUP	see Table 4 on page 9, Figure 10 on page 11
HOLD	see Table 4 on page 9, Figure 10 on page 11
SETUPHOLD	see Table 6 on page 11, Figure 12 on page 12
RECOVERY	see Table 4 on page 9, Figure 10 on page 11
REMOVAL	see Table 4 on page 9, Figure 10 on page 11
RECREM	see Table 6 on page 11, Figure 12 on page 12
SKEW	see Table 9 on page 16, Figure 18 on page 16
BIDIRECTSKEW	
WIDTH	see Table 10 on page 17, Figure 21 on page 17
PERIOD	see Table 10 on page 17, Figure 20 on page 17
NOCHANGE	see Table 8 on page 14, Figure 16 on page 15
SCOND	
CCOND	
LABEL	

Conditions in SDF are expressed in Verilog syntax, which is different from Liberty syntax. Therefore, liberty provides ìSDF_condî, ìSDF_cond_startî, ìSDF_cond_endî statements, which are basically ìwhenî, ìwhen_startî, ìwhen_endî statements translated into Verilog syntax.

The ALF syntax for conditions closely matches the Verilog syntax. Therefore, ìSDF_condî, ìSDF_cond_startî, ìSDF_cond_endî 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;
}
```