# Library Harmonization for Timing

| Version | Date |
|---|---|
| 0.0 | 11/17/03 |
| 0.1 | 12/12/03 |

**Template for liberty/ALF xref examples**

```
/* liberty */                    /* ALF */
```

## 1.0 Basic description of timing arcs

### 1.1 Overview

Timing arcs are defined not only by standalone statements but also by the context in which the statements appear. This is shown in Figure 1 on page 2.

**FIGURE 1. Basic timing arc description in liberty and ALF**

```
/* liberty */                          /* ALF */
cell (CellName) {                      CELL CellName {
  pin(FromPin) {                         PIN FromPin {
    direction : input;                     DIRECTION = input;
  }                                      }
  pin(ToPin) {                           PIN ToPin {
    direction : output;                    DIRECTION = output;
    timing() {                           }
      timing_type  : timing_type;        VECTOR (vector_expression) {
      timing_sense : timing_sense;         ALF_TimingModel
      related_pin  : "FromPin";          }
      lib_TimingModel                  }
    }
  }
```

In both liberty and ALF, a timing arc is defined in the context of a CELL identified by a **CellName**. A declaration of each PIN involved in the timing arc is required, refered herein as the **FromPin** and the **ToPin**.

In liberty, the timing model is further defined inside the declaration of the **ToPin**. In ALF, the timing model is defined by the declaration of a VECTOR, separate from the declaration of each PIN.

The occuring edge combinations are defined in liberty by ***timing_type*** and ***timing_sense***. In ALF, the edge combinations are defined by a ***vector_expression***.

In ALF, there is no dependency between the ***vector_expression*** and the ***ALF_TimingModel***. In liberty, there is a dependency between the ***timing_type*** and the ***lib_TimingModel***, as shown below.

**TABLE 1. Mapping between timing keywords in liberty and ALF**

| **Liberty keyword** | | **ALF keyword** |
|---|---|---|
| ***timing_type*** | ***lib_TimingModel*** | ***ALF_TimingModel*** |
| combinational | cell_rise, cell_fall | DELAY |
| | rise_transition, fall_transition | SLEWRATE |
| three_state_enable | cell_rise, cell_fall ? | DELAY |
| three_state_disable | cell_rise, cell_fall ? | DELAY |
| rising_edge | cell_rise, cell_fall | DELAY |
| | rise_transition, fall_transition | SLEWRATE |
| falling_edge | cell_rise, cell_fall | DELAY |
| | rise_transition, fall_transition | SLEWRATE |
| preset | cell_rise | DELAY |
| | rise_transition | SLEWRATE |

**TABLE 1. Mapping between timing keywords in liberty and ALF**

| **Liberty keyword** | | **ALF keyword** |
|---|---|---|
| ***timing_type*** | ***lib_TimingModel*** | ***ALF_TimingModel*** |
| clear | cell_fall | DELAY |
| | fall_transition | SLEWRATE |
| setup_rising | rise_constraint, fall_constraint | SETUP |
| setup_falling | rise_constraint, fall_constraint | SETUP |
| hold_rising | rise_constraint, fall_constraint | HOLD |
| hold_falling | rise_constraint, fall_constraint | HOLD |
| recovery_rising | intrinsic_rise, intrinsic_fall | RECOVERY |
| recovery_falling | intrinsic_rise, intrinsic_fall | RECOVERY |
| removal_rising | intrinsic_rise, intrinsic_fall | REMOVAL |
| removal_falling | intrinsic_rise, intrinsic_fall | REMOVAL |
| skew_rising | intrinsic_rise, intrinsic_fall | LIMIT.SKEW.MAX |
| skew_falling | intrinsic_rise, intrinsic_fall | LIMIT.SKEW.MAX |
| non_seq_setup_rising | intrinsic_rise, intrinsic_fall | SETUP |
| non_seq_setup_falling | intrinsic_rise, intrinsic_fall | SETUP |
| non_seq_hold_rising | intrinsic_rise, intrinsic_fall | HOLD |
| non_seq_hold_falling | intrinsic_rise, intrinsic_fall | HOLD |
| nochange_high_high | rise_constraint | SETUP |
| | fall_constraint | HOLD |
| nochange_high_low | rise_constraint | SETUP |
| | fall_constraint | HOLD |
| nochange_low_high | rise_constraint | SETUP |
| | fall_constraint | HOLD |
| nochange_low_low | rise_constraint | SETUP |
| | fall_constraint | HOLD |

In liberty, the timing model declaration can be qualified by a ***timing_type***. The combination of edges is defined by the combination of ***timing_sense***, ***DelayKeyword*** and ***SlewKeyword***. The mapping of these liberty constructs into a ***vector_expression*** in ALF is shown in Table 2 on page 3.

**TABLE 2. Mapping of liberty and ALF constructs for timing**

| **liberty construct** | | | | **ALF construct** |
|---|---|---|---|---|
| ***timing_type*** | ***timing_sense*** | ***Delay*** | ***Slew*** | ***vector_expression*** |
| combinational | positive_unate | cell_rise | rise_transition | 01 FromPin -> 01 ToPin |
| | | cell_fall | fall_transition | 10 FromPin -> 10 ToPin |

**TABLE 2. Mapping of liberty and ALF constructs for timing**

| **liberty construct** | | | | **ALF construct** |
|---|---|---|---|---|
| ***timing_type*** | ***timing_sense*** | ***Delay*** | ***Slew*** | ***vector_expression*** |
| | negative_unate | cell_rise | rise_transition | 10 FromPin -> 01 ToPin |
| | | cell_fall | fall_transition | 01 FromPin -> 10 ToPin |
| | non_unate | cell_rise | rise_transition | ?! FromPin -> 01 ToPin |
| | | cell_fall | fall_transition | ?! FromPin -> 10 ToPin |
| three_state_enable | positive_unate? | cell_rise ? | | 01 FromPin -> Z1 ToPin |
| | | cell_fall ? | | 01 FromPin -> Z0 ToPin |
| | negative_unate? | cell_rise ? | | 10 FromPin -> Z1 ToPin |
| | | cell_fall ? | | 10 FromPin -> Z0 ToPin |
| three_state_disable | positive_unate? | cell_rise ? | | 01 FromPin -> 0Z ToPin |
| | | cell_fall ? | | 01 FromPin -> 1Z ToPin |
| | negative_unate? | cell_rise ? | | 10 FromPin -> 0Z ToPin |
| | | cell_fall ? | | 10 FromPin -> 1Z ToPin |
| rising_edge | ? | cell_rise | rise_transition | 01 FromPin -> 01 ToPin |
| | ? | cell_fall | fall_transition | 01 FromPin -> 10 ToPin |
| falling_edge | ? | cell_rise | rise_transition | 10 FromPin -> 01 ToPin |
| | ? | cell_fall | fall_transition | 10 FromPin -> 10 ToPin |
| preset | positive_unate | cell_rise | rise_transition | 01 CtrlPin -> 01 DataPin |
| | negative_unate | cell_rise | rise_transition | 10 CtrlPin -> 01 DataPin |
| clear | positive_unate | cell_fall | fall_transition | 01 CtrlPin -> 10 DataPin |
| | negative_unate | cell_fall | fall_transition | 10 CtrlPin -> 10 DataPin |
| setup_rising | ? | rise_constraint | | 01 DataPin -> 01 ClkPin |
| | | fall_constraint | | 10 DataPin -> 01 ClkPin |
| setup_falling | ? | rise_constraint | | 01 DataPin -> 10 ClkPin |
| | | fall_constraint | | 10 DataPin -> 10 ClkPin |
| hold_rising | ? | rise_constraint | | 01 ClkPin -> 01 DataPin |
| | | fall_constraint | | 01 ClkPin -> 10 DataPin |
| hold_falling | ? | rise_constraint | | 10 ClkPin -> 01 DataPin |
| | | fall_constraint | | 10 ClkPin -> 10 DataPin |
| recovery_rising | ? | intrinsic_rise | | 01 CtrlPin -> 01 ClkPin |
| | | intrinsic_fall | | 10 CtrlPin -> 01 ClkPin |
| recovery_falling | ? | intrinsic_rise | | 01 CtrlPin -> 10 ClkPin |
| | | intrinsic_fall | | 10 CtrlPin -> 10 ClkPin |
| removal_rising | ? | intrinsic_rise | | 01 ClkPin -> 01 CtrlPin |
| | | intrinsic_fall | | 01 ClkPin -> 10 CtrlPin |
| removal_falling | ? | intrinsic_rise | | 10 ClkPin -> 01 CtrlPin |
| | | intrinsic_fall | | 10 ClkPin -> 10 CtrlPin |

**TABLE 2. Mapping of liberty and ALF constructs for timing**

| **liberty construct** | | | | **ALF construct** |
|---|---|---|---|---|
| ***timing_type*** | ***timing_sense*** | ***Delay*** | ***Slew*** | ***vector_expression*** |
| skew_rising | ? | intrinsic_rise | | 01 FromPin -> 01 ToPin |
| | | intrinsic_fall | | 01 FromPin -> 10 ToPin |
| skew_falling | ? | intrinsic_rise | | 10 FromPin -> 01 ToPin |
| | | intrinsic_fall | | 10 FromPin -> 10 ToPin |
| non_seq_setup_rising | | intrinsic_rise | | 01 DataPin -> 01 EnbPin |
| | | intrinsic_fall | | 10 DataPin -> 01 EnbPin |
| non_seq_setup_falling | | intrinsic_rise | | 01 DataPin -> 10 EnbPin |
| | | intrinsic_fall | | 10 DataPin -> 10 EnbPin |
| non_seq_hold_rising | | intrinsic_rise | | 01 EnbPin -> 01 DataPin |
| | | intrinsic_fall | | 01 EnbPin -> 10 DataPin |
| non_seq_hold_falling | | intrinsic_rise | | 10 EnbPin -> 01 DataPin |
| | | intrinsic_fall | | 10 EnbPin -> 10 DataPin |
| nochange_high_high | | rise_constraint | | 01 CtrlPin -> 01 ClkPin -> 10 ClkPin -> 10 CtrlPin |
| | | fall_constraint | | 10 DataPin -> 01 ClkPin |

Note: The representation of the actual calculation data in liberty (*lib_CalcType*, *lib_CalcData*) and ALF (*ALF_CalcData*) is independent of the physical nature of the data, i.e., timing data or power data or other data. The mapping between those liberty and ALF constructs is shown [insert reference].

## 1.2 Threshold definitions

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.

**FIGURE 2. ALF template for liberty threshold definitions**

```
DELAY {
  FROM {
    THRESHOLD {
      RISE = input_threshold_pct_rise ;
      FALL = input_threshold_pct_fall ;
    }
  }
  TO {
    THRESHOLD {
      RISE = output_threshold_pct_rise ;
      FALL = output_threshold_pct_fall ;
    }
  }
}
SLEWRATE {
  FROM {
    THRESHOLD {
      RISE = slew_lower_threshold_pct_rise ;
      FALL = slew_upper_threshold_pct_fall ;
    }
  }
  TO {
    THRESHOLD {
      RISE = slew_upper_threshold_pct_rise ;
      FALL = slew_lower_threshold_pct_fall ;
    }
  }
}
```

## 1.3 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 explicitely. However, the existence condition can be infered 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 3. 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 ...                }
  }
}
/* infered 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 4. 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 ...
  }
}
```

# 2.0 Interoperability with SDF

**TABLE 3. Mapping between SDF, liberty, and ALF constructs**

| SDF keyword | set of liberty keywords | | ALF keyword |
|---|---|---|---|
| IOPATH | combinational | cell_rise, cell_fall | DELAY in context of CELL statement |
| INTERCONNECT | N/A | N/A | DELAY in context of WIRE statement |
| PATHPULSE | ? | ? | PULSEWIDTH.MIN in context of two VECTOR statements |
| RETAIN | | | RETAIN |
| PORT | | | DELAY without FROM statement |
| SETUP | setup_rising, setup_falling | rise_constraint, fall_constraint | SETUP |
| HOLD | hold_rising, hold_falling | rise_constraint, fall_constraint | HOLD |
| SETUPHOLD | N/A | N/A | SETUP and HOLD in context of same VECTOR statement |
| RECOVERY | recovery_rising, recovery_falling | | RECOVERY |
| REMOVAL | removal_rising, removal_falling | | REMOVAL |

**TABLE 3. Mapping between SDF, liberty, and ALF constructs**

| SDF keyword | set of liberty keywords | | ALF keyword |
|---|---|---|---|
| RECREM | N/A | N/A | RECOVERY and REMOVAL in context of same VECTOR statement |
| SKEW | skew_rising,<br>skew_falling | | LIMIT.SKEW.MAX |
| WIDTH | | | LIMIT.PULSEWIDTH.MIN |
| PERIOD | | | LIMIT.PERIOD.MIN |
| NOCHANGE | nochange_high_high,<br>nochange_high_low,<br>nochange_low_high,<br>nochange_low_low | | SETUP and HOLD in context of same VECTOR statement, possibly in conjunction with LIMIT.PULSEWIDTH.MIN |

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.

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;
}
```