Work document for tracking the development of the IEEE 1603 std

This document contains suggested enhancements to the Advanced Library Format, using ALF 2.0 as baseline. The document serves as a worksheet rather than a formal proposal. The suggested enhancements are collected in no particular order. The idea is to keep track of evolving proposals here and then agree formally whether or not they should be part of the IEEE spec.

The following template is used throughout this document:

X.0 Item

relation to ALF 2.0	reference to ALF 2.0 chapter
relation to IEEE P1603	reference to IEEE P1603 chapter
History	date of initial draft, date of revisions
Status	open or closed, accepted or rejected

X.1 Motivation

Explain reason for new feature

X.2 Proposal

Describe new feature

Table of contents

Part 1: Language features for library modeling

1.0 Level definition for Vector Expression Language

relation to ALF 2.0	5.3, 5.4, 11.3
relation to IEEE P1603	N/A
History	initial draft April 16, 2001 by Wolfgang reviewed and rejected by Study Group April 16 rejection confirmed by Tim Ehrler May 1 changed title and closed May 4 by Wolfgang
Status	closed, rejected

1.1 Motivation

The vector expression language is a new concept which has almost no equivalent in legacy library model description languages. Currently there are EDA tools which support a subset of the vector expression language. Purpose of this proposal is to re-write the definitions in such a way that it is easy to identify subsets for different levels of support. For example: level0=basic subset, level1=intermediate subset, level2=full set in ALF 2.0, level3=full set in ALF 2.0 plus new proposed extensions.

1.2 Proposal

Level 0: single event, single event & boolean condition, two-event sequence

Level 1: N-event sequence, N-event sequence & boolean condition, alternative event sequence

Level 2: everything in ALF 2.0 (except if we decide to drop something fundamentally unpractical or un-implementable)

Level 3: new operators for repetition of sub-sequences

2.0 Metal Density

relation to ALF 2.0 9.2, 9.5

relation to IEEE P1603 11.13

History initial draft April 16, 2001 by Wolfgang
reviewed and retained by Study Group April 16
o.k. as is by Tim Ehrler May 1
supplementary proposal by Wolfgang Oct. 5
reviewed Oct. 9, supplementary proposal o.k.

Status closed, accepted

2.1 Motivation

Manufacturability in 130 nm technology and below requires so-called metal density rules. For a given routing layer, metal must cover a certain percentage of the total area within a lower and upper bound in order to ensure planarity. This percentage also depends on the total area under consideration, i.e., there are “local” and “global” metal density rules.

Manufacturing rules also specify, how density should be calculated. For example, only structures wider than a certain minimum width should be taken into account.

Also, for local rules, the shape of the region to be checked can be specified. For example, check the rule on a square of x*x mm^2, check the density on a region of x mm width in X or Y direction etc.

2.2 Proposal

Introduce new keyword DENSITY (or other word) for arithmetic model. Shall be non-negative number normalized between 0 and 1 (1 means 100%). Usable in context of LAYER (see ALF 2.0, chapter 9.5.1) with PURPOSE=routing (see ALF 2.0, chapter 9.5.2). Legal argument (i.e. HEADER) includes AREA, meaning the die area subjected to manufacturing of this layer.

Example:

```
LAYER metal1 {
   PURPOSE = routing;
   LIMIT {
      DENSITY {
         MIN {
            HEADER {
               AREA {
                  INTERPOLATION = floor;
                  TABLE { 0 100 1000 }
               }
```

```
            }
            TABLE { 0.2 0.3 0.4 }
         }
         MAX {
            HEADER {
               AREA {
                  INTERPOLATION = floor;
                  TABLE { 0 100 1000 }
               }
            }
            TABLE { 0.8 0.7 0.6 }
         }
      }
   }
}
```

Within an area of less than 100 units, the metal density must be between 20% and 80%. Within an area of 100 up to less than 1000 units, the metal density must be between 30% and 70%. Within an are of 1000 units or more, the metal density must be between 40% and 60%. The annotation INTERPOLATION=floor indicates that no interpolation is made for areas in-between, but the next lower value is used (see ALF 2.0, chapter 7.4.4).

To allow for particularities in density calculation, the DENSITY statement must be in context of a RULE (see ALF 2.0, chapter 9.11). The applicable layer is given as annotation. Both a model for calculation of DENSITY and a model for the limit of DENSITY must be given in context of the RULE.

Example:

```
RULE min_density {
   DENSITY {
      LAYER = metal1;
      CALCULATION = incremental;
      HEADER {
         WIDTH
         LENGTH
         AREA
      }
      EQUATION { WIDTH * LENGTH / AREA }
   }
   LIMIT { DENSITY { LAYER = metal1; MIN = 0.2; } }
}
RULE max_density {
   DENSITY {
      LAYER = metal1;
      CALCULATION = incremental;
      HEADER {
         WIDTH
         LENGTH
         AREA
      }
      EQUATION { (WIDTH<0.1)? 0 : WIDTH * LENGTH / AREA }
   }
```

```
   LIMIT { DENSITY { LAYER = metal1; MAX = 0.8; } }
}
```

Note: WIDTH (see ALF 2.0, chapter 9.2, table 9-4) and LENGTH (see ALF 2.0, chapter 9.2, table 9-6) are the dimensions of a routable object in the layer. AREA (see ALF 2.0, chapter 9.2, table 9-7) should be defined as the area of the environment in this context.

The example specifies, that objects smaller than 0.1 units of WIDTH are to be disregarded for DENSITY calculation in context of the RULE max_density.

3.0 Types of electrical CURRENT

relation to ALF 2.0 8.1, 8.7, 8.15

relation to IEEE P1603 11.12.5, 11.12.11

History initial draft April 16, 2001 by Wolfgang
reviewed and retained by Study Group April 16
also reviewed by Tim Ehrler May 1
add text to clarify purpose by Wolfgang May 4
proposal reviewed May 8, added supplementary proposal
reviewed, amended and accepted Oct. 9

Status closed, accepted

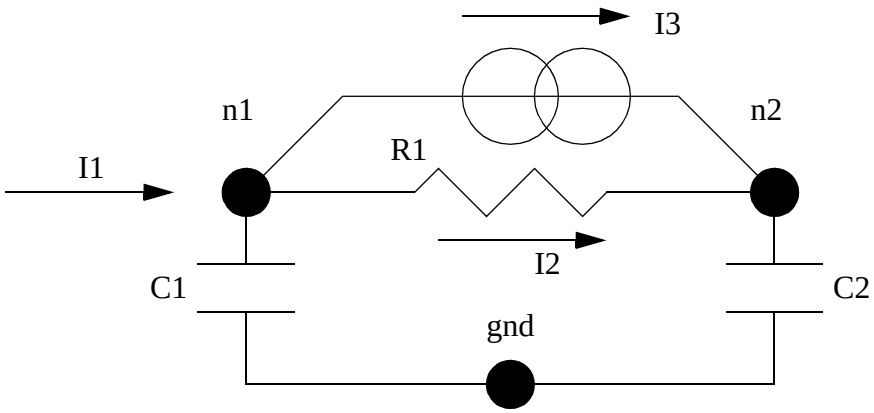
3.1 Motivation

CURRENT needs PIN annotation indicating the target point where the current is flowing into. Cannot define a branch of an electrical network where the current flows through.

Therefore there will be 3 types of CURRENT specification:

I1 = current into PIN from unspecified source (already supported in ALF 2.0)
I2 = current through a COMPONENT with two terminal nodes
I3 = current through an independent current source connected between two NODEs

see I1, I2, I3 in illustration



3.2 Proposal

In the context of WIRE, the following mutually exclusive annotations for CURRENT shall be legal:

```
PIN = pin_identifier ;
```

Current flows from unknown source into the pin (already supported).

COMPONENT = *component*_identifier **;**

Current flows through the component. The component must be a declared two-terminal electrical component in the context of the WIRE, i.e. a RESISTANCE, CAPACITANCE, VOLTAGE or INDUCTANCE (excluding mutual inductance, which has 4 terminals). The direction of the current flow is given by the order of node identifiers in the NODE annotation for that component (see ALF 2.0, chapter 8.15.3, 8.15.4).

NODE { *1st_node*_identifier *2nd_node*_identifier **}**

Current flows through a current source connected between the nodes. The direction of the current flow is given by the order of node identifiers in this NODE annotation.

Example:

```
WIRE interconnect_analysis_model_1 {
   CAPACITANCE C1 { NODE { n1 gnd } }
   CAPACITANCE C2 { NODE { n2 gnd } }
   RESISTANCE  R1 { NODE { n1 n2  } }
   CURRENT I1 { PIN = n1; }
   CURRENT I2 { COMPONENT = R1; }
   CURRENT I3 { NODE { n1 n2 } }
}
```

This example corresponds exactly to the illustration shown above.

3.3 Supplementary proposal

According to ALF 2.0, chapter 8.7.3, the sense of measurement for current associated with a pin shall be *into* the node. However, in some cases, the natural sense of measurement is *out of* the node. In order to allow explicit specification of the sense of measurement, the following feature is proposed:

FLOW annotation for current shall specify the sense of measurement of current. Default value shall be “in”, which is backward compatible with ALF 2.0.

FLOW = in | out;

For example, the following two statements are equivalent:

```
CURRENT I1 = 3.0 { PIN = n1; FLOW = in; }

CURRENT I1 = -3.0 { PIN = n1; FLOW = out; }
```

This is illustrated in the picture below.

4.0 NOISE modeling

relation to ALF 2.0 8.1, 8.14

relation to IEEE P1603 11.12.10

History initial draft April 16, 2001 by Wolfgang
o.k by Tim Ehrler May 1
updated by Wolfgang May 4
reviewed and updated (see minutes) May 8
reviewed and accepted Oct. 9

Status closed, accepted

4.1 Motivation

NOISE_MARGIN defines a normalized voltage difference between nominal signal level and tolerated signal level. If violated, the correct signal level can not be determined. In order to check against noise margin, actual noise must be calculated. Currently VOLTAGE is used for noise calculations. However, since noise margin is normalized to signal voltage swing, it would be convenient, if the actual noise could also be represented in a normalized way. In CMOS, actual noise and noise margin tend to scale with supply voltage. A non-normalized model requires supply voltage as a parameter, if the supply voltage is subject to variation. A normalized model would to a 1st order degree approximate the voltage scaling effect already and therefore eliminate the supply voltage as a model parameter.

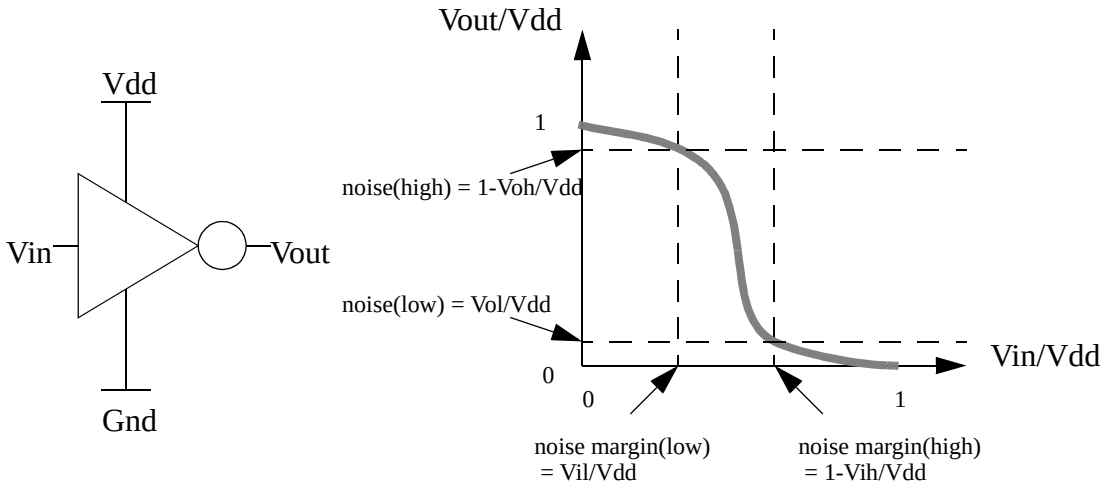
4.2 Proposal

Introduce new keyword NOISE, representing a normalized voltage difference between nominal signal level and actual signal level. Same measurement definition as for noise margin (see ALF 2.0, chapter 8.14). Noise margin is violated, if noise is larger than noise margin.

Context-specific meaning of NOISE

1. Context is output or bidirectional PIN

NOISE specifies maximum amount of noise at output pin, when any input pin is subjected to the amount of noise specified by NOISE_MARGIN. NOISE may have submodel HIGH and LOW. The relation between noise at output pin and noise margin at input pin is illustrated in the following picture.



Example:

```
PIN my_input_pin {
   DIRECTION = input;
   NOISE_MARGIN { HIGH = 0.3; LOW = 0.2; }
}
PIN my_output_pin {
   DIRECTION = output;
   NOISE { HIGH = 0.02; LOW = 0.01; }
}
```

2. Context is VECTOR with `vector_expression`

NOISE needs PIN annotation. NOISE specifies peak noise while pin is in “*” state. NOISE may only have submodel HIGH and LOW, if “?” state as opposed to “0” or “1” state is specified in vector_expression.

Example:

```
VECTOR ( 0* my_pin -> *0 my_pin) {
   NOISE = 0.2 { PIN = my_pin; }
}
```

3. Context is CELL, SUBLIBRARY, or LIBRARY

no PIN annotation. NOISE specifies maximum amount of noise at any output or bidirectional pin within scope, unless this specification is overwritten locally.

Example:

```
LIBRARY my_library {
   NOISE { HIGH = 0.02; LOW = 0.01; }
}
```

5.0 Simplification of NON_SCAN_CELL statement

relation to ALF 2.0 6.2, 11.2

relation to IEEE P1603 9.2.2

History initial draft April 16, 2001 by Wolfgang
o.k. by Tim Ehrler May 1
accepted and closed per default Oct. 9

Status closed, accepted

5.1 Motivation

Non-scan cell defines the mapping between the pins of a non-scan cell (left-hand side) and the pins of a scan cell (right-hand side). The scan cells has always certain pins which do not exist in the non-scan cell. In some cases, the non-scan cell might have certain pins which do not exist in the scan cell (In such a case, the scan replacement can only be done, if the pin in question was tied to an inactive level in the non-scan cell in the first place).

Currently, the non-scan cell statement supports definition of LHS or RHS constants which specify the logic level to which the non-corresponding pins should be tied to. However, this definition is redundant, because every relevant pin in a cell model must have annotations for SIGNALTYPE and POLARITY in order to be usable for DFT tools. These annotations specify already the logic level to which non-corresponding pins must be tied.

5.2 Proposal

Reduce syntax for pin_assignment (see ALF 2.0, chapter 11.2) to the following:

```
pin_assignment ::=
   pin_identifier [ index ] = pin_identifier [ index ] ;
|  pin_identifier [ index ] = logic_constant ;
```

Only “ *pin*_identifier [index] = *pin*_identifier [index] ; “ will actually be used for non-scan cell. Since POLARITY defines the active signal level, the pin should be tied to the opposite level. For pins without POLARITY, the level does not matter (e.g. scan input for scan flip-flop in non-scan mode).

Example (taken from ALF 2.0, chapter 6.2):

```
CELL my_flipflop {
   PIN q   { DIRECTION=output; } // SIGNALTYPE defaults to “data”
   PIN d   { DIRECTION=input; } // SIGNALTYPE defaults to “data”
   PIN clk { DIRECTION=input; SIGNALTYPE=clock; POLARITY=rising_edge; }
   PIN clear { DIRECTION=input; SIGNALTYPE=clear; POLARITY=low; }
}
CELL my_scan_flipflop {
   PIN data_out { DIRECTION=output; } // SIGNALTYPE defaults to “data”
```

```
   PIN data_in  { DIRECTION=input; } // SIGNALTYPE defaults to “data”
   PIN scan_in  { DIRECTION=input; SIGNALTYPE=scan_data; }
   PIN scan_sel { DIRECTION=input; SIGNALTYPE=scan_control;
      POLARITY { SCAN=high; } } // scan mode when 1, non-scan mode when 0
   PIN clock {DIRECTION=input; SIGNALTYPE=clock; POLARITY=rising_edge;}
   NON_SCAN_CELL {
      my_flipflop {
         clk = clock;
         d   = data_in;
         q   = data_out;
      }
   }
}
```

The scan replacement works only, if the **`clear`** pin of **`my_flipflop`** is tied high (active level is low). Note: This is an exceptional case and only shown because it might happen eventually. Normally, the pins of the scan cell represent a superset of the pins of the non-scan cell.

In order to simulate the non-scan mode, when the non-scan cell is replaced by the scan cell, the **`scan_sel`** pin of **`my_scan_flipflop`** must be tied low (scan mode level is high). The **`scan_in`** pin can be tied to either high or low.

This example shows that the constant logic levels need not be defined in the non-scan cell statements, because they can be completely inferred from the POLARITY statements. The POLARITY statements are mandatory for DFT tools anyway.

6.0 VIOLATION in context of LIMIT

relation to ALF 2.0 7.5, 7.6, 8.4

relation to IEEE P1603 9.10.5, 11.6.4

History Proposal May 1, 2001 by Tim Ehrler
written in doc May 4 by Wolfgang
reviewed and updated (see minutes) May8
reviewed, accepted and closed Oct. 9

Status closed, accepted

6.1 Motivation

Want to specify level of severity, if a LIMIT is violated. Target is appropriate error report from tool.

6.2 Proposal

The VIOLATION statement may appear within the context of an arithmetic model within LIMIT or an arithmetic submodel within LIMIT.

In this context, a MESSAGE_TYPE annotation or a MESSAGE annotation or both shall be legal within VIOLATION. A BEHAVIOR statement within VIOLATION shall only be legal if the LIMIT is within the context of a VECTOR. In the latter case, the `vector_expression` or `boolean_expression` which identifies the VECTOR shall define the triggering condition for the behavior described in the BEHAVIOR statement.

7.0 New value for MEASUREMENT annotation

relation to ALF 2.0 8.9.1

relation to IEEE P1603 11.12.11

History Proposal by Wolfgang, May 22, 2001
reviewed June 27, o.k. July 10 (see minutes)
accepted and closed Oct. 9

Status closed, accepted

7.1 Motivation

Currently, measurements of analog quantities can be specified as “average”, “rms”, “peak”, “transient”, “static”. Another commonly used measurement is the average over absolute values, which cannot be specified.

7.2 Proposal

The MEASUREMENT annotation shall support the following values:

```
MEASUREMENT =
   transient
|  static
|  average
|  rms
|  peak
|  absolute_average[1]
```

The mathematical definition of **absolute_average** is the following[2]:

$$\frac{\int_{(t=0)}^{(t=T)} |E(t)|\,dt}{T}$$

1. everything except **absolute_average** is already supported in ALF 2.0
2. Note: The parentheses around (t = 0) and (t = T) are an artefact of the framemaker equation editor.

8.0 MONITOR statement for VECTOR

relation to ALF 2.0 5.3.7, 5.4, 6.4.16

relation to IEEE P1603 9.5.3

History Proposal by Wolfgang, May 22, 2001
reviewed July 10 (see minutes)
reviewed Oct. 9, added comments based on discussion
reviewed Nov. 12, Alex requested to keep it open
reviewed Jan. 14, 2002, agreed to accept by majority

Status closed, accepted

8.1 Motivation

Any `vector_expression` in the context of a VECTOR has an associated set of variables, which are monitored for the purpose of evaluating the `vector_expression`. The set of variables is given by the set of declared PINs, featuring a SCOPE annotation.

```
SCOPE = behavior | measure | both | none ; // see ALF 2.0, chapter 6.4.16
```

In the context of a VECTOR, all PINs with **`SCOPE = measure | both`** are monitored. Sometimes it would be practical to reduce the set of monitored pins within the scope of a particular vector. For example, in a multiport RAM, only the pins associated with a particular logical port should be monitored, if the `vector_expression` describes a transaction involving only this port. Currently, this can only be achieved by applying the “?*” operator to all unmonitored pins. Therefore the `vector_expression` can become quite lengthy for complex cells.

8.2 Proposal

Note: To understand and appreciate the proposal, it is mandatory that the reader be familiar with ALF 2.0, chapter 5.4, pp. 55-80.

A VECTOR identified by a `vector_expression` may have the following MONITOR annotation:

```
monitor_multivalue_annotation :==
   MONITOR { pin_identifiers }
```

The set of *`pin_identifiers`* shall be a subset of pins with **`SCOPE = measure | both`**.

If the MONITOR annotation is present, all pins appearing within this annotation shall be monitored. Any pin appearing in the `vector_expression` must also appear in the MONITOR annotation. However, all pins appearing in the MONITOR annotation need not appear in the `vector_expression`.

If the MONITOR annotation is not present, all pins with **`SCOPE = measure | both`** shall be monitored (backward compatible with ALF 2.0).

Example:

```
CELL my_4_bit_register_file {
   PIN clk { DIRECTION=input; }
   PIN [4:1] din  { DIRECTION=input; }
   PIN [4:1] dout { DIRECTION=output; }
   VECTOR ( 01 clk -> ?! dout[1] ) {
      MONITOR { din[1] dout[1] clk } // put in delay, power etc.
   }
   VECTOR ( 01 clk -> ?! dout[2] ) {
      MONITOR { din[2] dout[2] clk } // put in delay, power etc.
   }
   VECTOR ( 01 clk -> ?! dout[3] ) {
      MONITOR { din[3] dout[3] clk } // put in delay, power etc.
   }
   VECTOR ( 01 clk -> ?! dout[4] ) {
      MONITOR { din[4] dout[4] clk } // put in delay, power etc.
   }
}
```

It has been suggested that the MONITOR statement should only contain the variables which are not already present in the vector_expression. This has the following drawback: A vector_expression with all monitored variables present would need an empty MONITOR statement in order to be compatible with ALF 2.0 semantics. Also, identification of the full set of monitored variables would not be possible without analysis of the vector expression. It was argued that specifying all variables is redundant and inconvenient. However, the latter applies only if both the vector_expression and the MONITOR statement are specified by hand. Eventually, a user may specify only a set of MONITOR statements and leave the generation of appropriate vector_expressions to an intelligent characterization tool. The redundancy between MONITOR statement and vector_expression could also serve as a validity check especially for automatically generated vector_expressions. Discussion to be continued ...

Alex accepted to close the discussion, but his dissens and concerns are noted.

9.0 Features for creating a standard ALF header file

relation to ALF 2.0 3.2.4, 3.2.6, 3.2.8, 3.2.9, 11.x

relation to IEEE P1603 8.6, 8/7, 8.8, 8.9, new Annex (normative or not TBD)

History Proposal by Wolfgang, May 22, 2001
review pending as of July 10
supplementary proposal by Wolfgang Oct. 7
left open for review by ALF parser developpers
agreed on Jan. 14, 2002 to create a header file

Status closed

9.1 Motivation

The idea is to define pertinent features of ALF using the ALF language itself. Such a definition could be used as a standard “header” file for ALF. Eventually, certain extensions of the language could then be defined by changing the header file instead of changing the language. This can be used for pure documentation purpose as well as for development of self-adapting ALF parsers.

The remainder of this chapter is for illustration purpose only. The contents of the header file shall be in a separate document.

9.2 Proposal

Use the KEYWORD statement to define standard arithmetic models.

Use the `definition_for_arithmetic_model` construct to define legal statements in the context of arithmetic models.

Use the CLASS statement for shared definitions.

Example (just to show the idea):

```
KEYWORD PROCESS = arithmetic_model ;
KEYWORD SLEWRATE = arithmetic_model ;
KEYWORD CURRENT = arithmetic_model ;

PROCESS {
   TABLE { nom spsn spwn wpsn wpwn }
}
CLASS all_models {
   KEYWORD UNIT = single_value_annotation ;
}
CLASS timing_models {
   CLASS { all_models }
   UNIT = 1e-9 ;
```

```
   KEYWORD RISE = arithmetic_model ;
   KEYWORD FALL = arithmetic_model ;
}
CLASS analog_models {
   CLASS { all_models }
   KEYWORD MEASUREMENT = single_value_annotation ;
}
SLEWRATE {
   CLASS { timing_models }
}
CURRENT {
   CLASS { analog_models }
   UNIT = 1e-3 ;
}
```

It may be worthwhile to explore how far we can get in describing ALF features in this language.

9.3 Supplementary proposal

Current definition for `keyword_declaration` (see ALF 2.0, chapter 3.2.9):

```
keyword_declaration ::=
   KEYWORD context_sensitive_keyword = syntax_item_identifier ;
```

Introduce the following extension:

```
keyword_declaration ::=
   KEYWORD context_sensitive_keyword = syntax_item_identifier ;
|  KEYWORD context_sensitive_keyword = syntax_item_identifier {
      VALUE_TYPE = value_type_identifier ;
   }

value_type_identifier ::=
   number
|  positive_number
|  non_negative_number
|  integer
|  unsigned
|  bit_literal
|  quoted_string
|  identifier
```

Note: need to add which value_type is compatible with which syntax_item_identifier (see grammar definition).

10.0 Amended semantics of LIMIT

relation to ALF 2.0 7.5

relation to IEEE P1603 11.6.4

History Wolfgang, July 2, 2001, o.k. on July 10
refined and incorporated in this doc on July 19
reviewed, amended, accepted and closed October 9

Status closed, accepted

10.1 Motivation

ALF 2.0 misses a specification on how a design tool should handle a LIMIT.

10.2 Proposal

Existing text:

A `LIMIT` container shall contain arithmetic models. The arithmetic models shall contain submodels identified by `MIN` and/or `MAX`.
Proposed modification:
A `LIMIT` container shall contain arithmetic models. The arithmetic models shall contain submodels. These submodels shall either be exclusively identified by `MIN` and/or `MAX` or contain other submodels which shall be exclusively identified by `MIN` and/or `MAX`.
Example:

```
LIMIT { SLEWRATE {
      PIN = my_pin ; MAX = 5.4;
}  }
```

Alternative example:

```
LIMIT { SLEWRATE {
      PIN = my_pin ; RISE { MAX = 6.3; } FALL { MAX = 5.4; }
}  }
```

Proposed addition:

The values specified within LIMIT shall be considered as design limits. That means, design tools must create a design in such ways that the limits are respected. If the calculated actual values are found to be equal to the specified limit values, they shall be considered within the design limits. The `MAX` shall specify an upper limit. The `MIN` value shall specify a lower limit. Therefore, if both `MIN` and `MAX` values are specified for the same quantity under the same operating conditions, the `MAX` value must be greater or equal to the `MIN` value.

11.0 Semantics of SUPPLYTYPE and SUPPLY_CLASS for multi-rail support

relation to ALF 2.0 6.4.11, 6.4.13

relation to IEEE P1603 9.3.4

History email discussion on reflector initiated by Sergei Sokolov
captured in minutes July 10, 2001
incorporated in this document by Wolfgang, July 19
reviewed Oct. 9, pending comments wrt VHDL-AMS
reviewed and accepted Nov. 12

Status closed, accepted

11.1 Motivation

Semantics of SUPPLYTYPE are missing in ALF 2.0. Semantics of SUPPLY_CLASS for support of multiple power/ground rails are not well-defined.

11.2 Proposal for SUPPLYTYPE semantics

Syntax and set of values for SUPPLYTYPE are already defined in ALF 2.0, chapter 6.4.11. Following table contains proposed semantics.

TABLE 1. SUPPLYTYPE annotation for PIN object

Annotation value	description
power (default)	The PIN is the interface between a CELL and a power supply device, designed to source or sink a significant part of the CURRENT affecting the POWER consumption of the CELL. The VOLTAGE measured at this PIN is with respect to ground.
ground	The PIN is the interface between a CELL and the environmental common ground. Therefore, the nominal VOLTAGE measured at this PIN is zero. However, spurious non-zero VOLTAGE may occur and LIMITs for such VOLTAGE may be specified. The PIN is designed to serve as return path for a significant part of the CURRENT affecting the POWER consumption of the CELL.
reference	The PIN is the interface between a CELL and a device which supplies either a well-defined VOLTAGE or a well-defined CURRENT without being a significant contributor to the POWER consumption of the CELL. From an electrical standpoint, a reference is similar to a signal. However, from an information-theoretical standpoint, a reference is similar to a supply, because it does not contain information.

Note: ALF 2.0, chapter 6.4.3 defines the semantic implication of DIRECTION on a PIN with PINTYPE= SUPPLY. If the DIRECTION is input, then the CELL must be connected to a supply device in order to operate. If the DIRECTION is output, then the CELL itself is the supply device.

Note: A CELL needs not have exactly one PIN with SUPPLYTYPE=power and another PIN with SUPPLYTYPE=ground. Passive devices (e.g. capacitor, resistor, diode) do not have any supply pins. Semi-passive devices (e.g. clamp cells) have only supply pins corresponding to the voltage level of the clamp. For example, a clamp cell to zero would have a pin with SUPPLYTYPE=ground and DIRECTION=input and a pin with SIGNALTYPE=TIE, POLARITY=low, and DIRECTION=output. Active devices have, at least, either one pin with SUPPLYTYPE=power and another pin with SUPPLYTYPE=ground or two pins with SUPPLYTYPE=power and different supply voltages, usually one positive and one negative. In general, a cell may have zero to multiple pins with SUPPLYTYPE=power or ground or reference.

11.3 Proposal for SUPPLY_CLASS semantics

Note: This section is proposed to supersede ALF 2.0, chapter 6.4.13.

The purpose of SUPPLY_CLASS is to define a relation between a power supply system and a circuit utilizing the power supply system. The power supply system herein is understood to be a set of nets (also called “rails”) capable to maintain a well-defined electrical potential with respect to each other.

The power supply system itself shall be declared using a CLASS statement for global use in the context of a LIBRARY or a SUBLIBRARY or for local use in the context of a CELL or a WIRE.

The characteristics of the power supply system shall be defined in the context of the objects which refer to the system using the SUPPLY_CLASS annotation. The value of the annotation shall be the name of the CLASS declaring the power supply system. Multi-value annotation shall be legal. Multi-value annotation shall indicate that the object can be used within either power supply system appearing in the set of values, but not necessarily within all of them at the same time.

The object, in the context of which the SUPPLY_CLASS annotation and the optional characteristics of the power supply system appear, shall be one of the following:

- A PIN within a CELL
- A NODE within a WIRE
- A CLASS for global usage within a LIBRARY or a SUBLIBRARY or for local usage within a CELL or a WIRE

The characteristics of the power supply system, i.e., the characteristics of each net within the power supply system, shall optionally include the following items:

- An arithmetic model for VOLTAGE, eventually containing arithmetic submodels for MIN, TYP, MAX, and/or DEFAULT. In the context of a PIN with SUPPLYTYPE=power or a NODE with NODETYPE=power, the arithmetic model shall specify the value of the supply voltage itself. In the context of a PIN with SUPPLY-

TYPE=ground or a NODE with NODETYPE=ground, the value of the supply voltage shall be presumed zero. In the context of another PIN or NODE, an arithmetic model for VOLTAGE may appear, but no relationship to supply voltage shall be implied.

- A LIMIT statement, containing an arithmetic model for VOLTAGE with arithmetic submodels for MIN and/or MAX. In the context of a PIN with any SUPPLYTYPE, including “ground”, this model shall specify the tolerable limit for spurious supply voltage change, which may occur due to resistive, capacitive or inductive noise. In the context of another PIN, a LIMIT for VOLTAGE may appear, but no relationship to supply voltage shall be implied.
- A SUPPLYTYPE may appear in the context of a CLASS for the purpose to be inherited by a PIN. Similarly, a NODETYPE may appear in the context of a CLASS for the purpose to be inherited by a NODE.

The CONNECT_CLASS annotation (see ALF 2.0, chapter 9.17) within a PIN shall be used to establish connectivity between terminals of a power supply net. The annotation value shall be the name of a CLASS. The PIN shall inherit the statements appearing in the context of that CLASS, including, but not restricted to, the SUPPLY_CLASS annotation, the arithmetic model for VOLTAGE, the LIMIT for VOLTAGE, and eventually the SUPPLYTYPE annotation.

The SUPPLY_CLASS annotation shall also be legal within an arithmetic model for ENERGY or POWER. It shall indicate, which power supply system provides the energy or power described by the arithmetic model.

Example:

```
LIBRARY my_library {
   CLASS io ;
   CLASS core ;
   CLASS Vdd_io { SUPPLY_CLASS=io; SUPPLYTYPE=power; VOLTAGE=2.5; }
   CLASS Vss_io { SUPPLY_CLASS=io; SUPPLYTYPE=ground; }
   CLASS Vdd_core { SUPPLY_CLASS=core; SUPPLYTYPE=power; VOLTAGE=1.8; }
   CLASS Vss_core { SUPPLY_CLASS=core; SUPPLYTYPE=ground; }
   CELL core2io_interface {
      PIN Vdd1 { PINTYPE=supply; CONNECT_CLASS=Vdd_io; }
      PIN Vdd2 { PINTYPE=supply; CONNECT_CLASS=Vdd_core; }
      PIN Vss1 { PINTYPE=supply; CONNECT_CLASS=Vss_io; }
      PIN Vss2 { PINTYPE=supply; CONNECT_CLASS=Vss_core; }
      PIN in  { PINTYPE=digital; DIRECTION=input; SUPPLY_CLASS=core; }
      PIN out { PINTYPE=digital; DIRECTION=output; SUPPLY_CLASS=io; }
      VECTOR (?! in -> ?! out) {
         ENERGY e1 = 15.8 { SUPPLY_CLASS=io; }
         ENERGY e2 = 3.42 { SUPPLY_CLASS=core; }
      }
   }
}
```

12.0 Amended semantics of RESTRICT_CLASS and SWAP_CLASS

relation to ALF 2.0	6.1.3, 6.1.4, 6.1.5, 6.1.6
relation to IEEE P1603	9.2.3
History	extensive email discussion involving Kevin Grotjohn, Tim Ehrler, Sean Huang proposal formulated by Wolfgang, July 31, 2001 reviewed Nov. 12, made modifications Kevin requested to trace history of discussion By request from Kevin, Jan. 14, 2002, two proposals from the email discussion are included here by reference.
Status	open

12.1 Motivation

The semantics of RESTRICT_CLASS and SWAP_CLASS, as described in ALF 2.0, do not fit the intended usage models.

12.2 Proposal for RESTRICT_CLASS

Note: This section is proposed to supersede ALF 2.0, chapter 6.1.4.

The purpose of the optional RESTRICT_CLASS annotation shall be to identify characteristics of a CELL which allow or disallow usage of the CELL for certain application tools. Single-value or multi-value annotation shall be legal.

If the usage of the CELL is allowed, the application tool may add, remove, or substitute instances of such a cell in the design. If the usage of the CELL is not allowed, the application tool may not add, remove, or substitute instances of such a cell in the design.

The condition for usage is not governed by the library. The library provides only a set of RESTRICT_CLASS values upon which a condition for usage can be specified for an application tool. The usage specification consists of two parts:

1. A set of RESTRICT_CLASS values “known” to the application tool
2. A condition for usage, involving the set of “known” values

This standard does not specify the mechanism by which the usage specification is established. The possibilities range from hardcoded usage specification to programmable usage specification.

Example:

Supposed, the following usage specification has been established for an application tool:

1. RESTRICT_CLASS values known by the tool = **(A, B, C, D, E)**
2. Condition for usage = **A and not B or C**

Supposed, the following cells X, Y, and Z are in the library:

RESTRICT_CLASS values of CELL X = **(A, B)**
Condition is false, therefore usage of CELL X is not allowed

RESTRICT_CLASS values of CELL Y = **(A, C)**
Condition is true, therefore usage of CELL Y is allowed

RESTRICT_CLASS values of CELL Z = **(A, C, F)**
Condition is true, but usage of CELL Z is not allowed due to unknown value **F**

End of example

This standard proposes a set of RESTRICT_CLASS values with predefined semantics in order to facilitate the cell usage specification for a wide range of applications with well-understood functionality.

In addition, the standard permits customized RESTRICT_CLASS values in order to control the cell usage in special applications.

12.3 Alternative proposal #1

see email from Tim.Ehrler@philips.com, Wed Jul 11, recorded in following document:

http://www.eda.org/alf/homepage/restrict_class_history.txt

12.4 Alternative proposal #2

see email from wroethig@el.nec.com, Thu Jul 12, recorded in following document:

http://www.eda.org/alf/homepage/restrict_class_history.txt

12.5 Semantics of predefined RESTRICT_CLASS values

Note: The following table is proposed to replace table 6-6 in ALF 2.0, which contains some circular definitions.

TABLE 2. RESTRICT_CLASS annotation for CELL object

Annotation value	description
synthesis	Cell is suitable for usage by a tool performing transformations from a RTL design representation to a structural gate-level design representation or between functionally equivalent structural gate-level design representations
scan	Cell is suitable for usage by a tool creating or modifying a structural design representation by inserting circuitry for testability enabling serial shift of data through storage elements
datapath	Cell is suitable for usage by a tool creating or modifying a structural implementation of a dataflow graph within a design
clock	Cell is suitable for usage by a tool creating or modifying circuitry for the distribution of synchronization signals (also called clock signals) within a design
layout	Cell is suitable for usage by a tool creating or modifying physical locations (placement) and physical interconnects (routes) of components within a design

The usage of RESTRICT_CLASS values other than these predefined values shall be legal. It shall not be implied that these predefined RESTRICT_CLASS values are automatically “known” by every application tool.

12.6 Proposal for SWAP_CLASS

Note: This section is proposed to supersede ALF 2.0, chapter 6.1.3, 6.1.5, 6.1.6.

The purpose of SWAP_CLASS shall be to identify sets of CELLS, wherein each CELL in the set can be substituted for each other by a particular application tool. Multi-value annotation shall be legal.

If the usage of two CELLs is authorized for a particular application tool according to RESTRICT_CLASS (see Section 12.2 on page 24) and the intersection of SWAP_CLASS values of the two CELLs is not empty, then the two CELLs shall be considered equivalent for the particular application tool, and the application tool is free to substitute one cell for the other.

Any SWAP_CLASS value shall make reference to a declared CLASS within a LIBRARY or SUBLIBRARY.

The CLASS statement may contain a RESTRICT_CLASS statement. In this case, the set of RESTRICT_CLASS values shall be inherited by the CELL containing the SWAP_CLASS statement. If the intersection of SWAP_CLASS values of the two CELLs is not empty and the usage of two CELLs is authorized according to the inherited RESTRICT_CLASS values, then the two CELLs shall be considered equivalent for the

particular application tool, and the application tool is free to substitute one cell for the other.

Example with RESTRICT_CLASS and SWAP_CLASS (from ALF 2.0, chapter 6.1.5):

```
CLASS foo;
CLASS bar;
CLASS whatever;
CLASS my_tool;
CELL cell1 {
   SWAP_CLASS { foo bar }
   RESTRICT_CLASS { synthesis datapath }
}
CELL cell2 {
   SWAP_CLASS { foo whatever }
   RESTRICT_CLASS { synthesis scan my_tool }
}
```

In order to swap cell1 and cell2, application tool must know all RESTRICT_CLASS values mentioned in this example. Usage condition may be (synthesis) or (datapath or my_tool) or (synthesis and datapath or scan and my_tool) etc.

[modify figure 6-1 from ALF 2.0: non-empty intersection applies only to SWAP_CLASS]

Example with SWAP_CLASS and inherited RESTRICT_CLASS (from ALF 2.0, chapter6.1.6):

```
CLASS all_nand2 { RESTRICT_CLASS { synthesis } }
CLASS all_high_power_nand2 { RESTRICT_CLASS { layout } }
CLASS all_low_power_nand2 { RESTRICT_CLASS { layout } }
CELL cell1 {
   SWAP_CLASS { all_nand2 all_low_power_nand2 }
}
CELL cell2 {
   SWAP_CLASS { all_nand2 all_high_power_nand2 }
}
CELL cell3 {
   SWAP_CLASS { all_low_power_nand2 }
}
CELL cell4 {
   SWAP_CLASS { all_high_power_nand2 }
}
```

A tool must know synthesis in order to utilize and swap cell1 and cell2. Another tool must know layout in order to utilize cell1, cell2, cell3, cell4 and swap cell1 with cell3 or cell2 with cell4. A tool that knows both synthesis and layout may utilize and swap all four cells.

[modify figure 6-1 from ALF 2.0: non-empty intersection applies only to SWAP_CLASS]

13.0 Amended semantics of CONNECTIVITY

relation to ALF 2.0 9.15

relation to IEEE P1603 11.13.1

History initial draft by Wolfgang, Oct. 7, 2001
initial review Nov. 12
reviewed Jan. 14, 2002
tentatively closed if no objection raised by e/o Jan. 2002.

Status tentatively closed, accepted

13.1 Motivation

CONNECTIVITY has been formulated as arithmetic_model in ALF 1.1, but not in ALF 2.0. In ALF 2.0, CONNECTIVITY is an exceptional statement different from arithmetic_model, albeit it features HEADER and TABLE like an arithmetic_model. The advantage of re-formulating CONNECTIVITY as arithmetic_model is to get rid of the exception and to utilize CONNECTIVITY also as argument in arithmetic_model. For example, other arithmetic models, for example minimum spacing, antenna rule etc., may depend on CONNECTIVITY. Another proposed enhancement is to utilize CONNECTIVITY not only as a requirement for connections but also as actual connection.

13.2 Proposal

The CONNECTIVITY statement shall be an arithmetic_model with value_type bit_literal. It may contain the optional CONNECT_RULE annotation, which shall specify a requirement for connections (see ALF 2.0, chapter 9.15.2). Without the CONNECT_RULE annotation, the CONNECTIVITY statement shall specify actual connectivity. The value “1” shall specify existing connection, the value “0” shall specify non-existing connection.

Example:

The following example describes pins on POLY layer with and without connection to diffusion.

```
PIN pin1 { // this pin has a POLY feature connected to NDIFF
   AREA A1 = 0.01 { LAYER=POLY; }
   CONNECTIVITY = 1 { BETWEEN { POLY NDIFF } }
}
PIN pin2 { // this pin has a POLY feature not connected to NDIFF
   AREA A1 = 0.01 { LAYER=POLY; }
   CONNECTIVITY = 0 { BETWEEN { POLY NDIFF } }
}
```

The following example describes a spacing rule between wires on the same layer, dependent whether they are on the same net or not.

```
// min distance between two wires, depending whether same net or not
RULE min_distance {
   PATTERN p1 { SHAPE = line; LAYER = metal1; }
   PATTERN p2 { SHAPE = line; LAYER = metal1; }
   LIMIT {
      DISTANCE {
         BETWEEN { p1 p2 }
         MIN {
            HEADER { CONNECTIVITY { BETWEEN { p1 p2 } } }
            EQUATION { CONNECTIVITY ? 0.1 : 0.2 }
         }
      }
   }
}
```

13.3 Supplementary proposal for CONNECT_TYPE

The CONNECT_TYPE annotation within the CONNECTIVITY statement shall specify the nature of the connection.

```
connect_type_single_value_annotation ::=
   CONNECT_TYPE = connect_type_identifier ;

connect_type_identifier ::=
   electrical
|  physical
```

The value “electrical” shall indicate that the objects are subjected to electrical connection, i.e., a permanent direct current path does or does not exist between the objects. The value “physical” shall indicate that the objects do or do not share common physical boundaries with each other. The value “electrical” shall be the default.

Supplementary explanation: A driver pin and a receiver pin of a routed wire have CONNECT_TYPE electrical. A via cut and the adjacent metal segments have CONNECT_TYPE physical. CONNECT_TYPE physical does not always imply electrical connection. For example, objects of electrically insulating material may be physically connected to each other.

14.0 Amended semantics of PULSEWIDTH, PERIOD

relation to ALF 2.0 8.3.17, 8/3/18

relation to IEEE P1603 11.9.9, 11.9.10

History initial draft by Wolfgang, Oct. 7, 2001
reviewed and accepted Nov. 12

Status closed, accepted

14.1 Motivation

PULSEWIDTH and PERIOD are introduced in ALF 1.1 and ALF 2.0 for the purpose of defining minimum pulse width and minimum period requirements in the context of a VECTOR. The keywords PULSEWIDTH and PEERIOD are used in the same way as SETUP, HOLD, RECOVERY, REMOVAL, which also define minimum timing requirements, without using the LIMIT or MIN statement. However, while SETUP, HOLD, RECOVERY, REMOVAL always represent minimum timing requirements, PULSEWIDTH and PERIOD could represent actual measurements or maximum requirements. Therefore we propose to amend the definitions of PULSEWIDTH and PERIOD to specify actual measurements.

14.2 Proposal

The keywords PULSEWIDTH and PERIOD shall specify arithmetic models in the context of a VECTOR.

PULSEWIDTH shall specify a measured time between two subsequent transitions on a pin, where the state of the pin after the second transition shall be equal to the state of the pin before the first transition. The PIN annotation shall be mandatory. The EDGE_NUMBER annotation shall be optional and specify the first transition of the two subsequent transitions. To specify a minimum or maximum constraint, use PULSEWIDTH in the context of LIMIT with submodel MIN or MAX, respectively.

PERIOD shall specify the measured time between two subsequent occurrences of the VECTOR. PIN annotation and EDGE_NUMBER annotation do not apply. To specify a minimum or maximum constraint, use PERIOD in the context of LIMIT with submodel MIN or MAX, respectively.

Example:

The following example specifies pulse width degradation through a buffer.

```
CELL my_buffer {
   PIN in { DIRECTION=input; }
   PIN out { DIRECTION=output }
   VECTOR ( 01 in -> 10 in <&> 01 out -> 10 out ) {
```

```
      // output pulse width = f(input pulse width)
      PULSEWIDTH { PIN = out;
         HEADER {
            PULSEWIDTH { PIN = in;
               TABLE { 0.1 0.2 0.3 0.4 0.5 }
            }
         }
         // short pulses are shortened, long pulses keep their width
         TABLE { 0.05 0.18 0.29 0.4 0.5 }
      }
   }
}
```

The following example specifies cycle time (minimum period) and refresh time (maximum period) of a DRAM.

```
CELL my_DRAM {
   PIN CE { DIRECTION = input; SIGNALTYPE = enable; }
   // fill in other pins etc.
   VECTOR ( 01 ce ) {
      // for simplicity, presume that CE controls all operations
      LIMIT {
         PERIOD {
            MIN = 10;
            MAX = 100000;
         }
      }
   }
}
```

15.0 Amended definition of TIME and FREQUENCY statement in context of arithmetic model

relation to ALF 2.0	8.3.6, 8.9
relation to IEEE P1603	11.9.1
History	initial draft by Wolfgang, Oct. 7 reviewed and accepted in principle Nov. 12 no objection raised in Jan. 14, 2002 meeting
Status	closed, accepted

15.1 Motivation

TIME and FREQUENCY are defined as arithmetic models. In addition, they are defined as annotations for arithmetic models featuring the MEASUREMENT annotation. The reason is the necessity to know either the time or the repetition frequency of a measurement. To get rid of the double-usage of the keywords, we propose to specify TIME and FREQUENCY as auxiliary “arithmetic model” within another arithmetic model rather than as “annotation”.

15.2 Proposal

TIME and FREQUENCY shall be usable as auxiliary arithmetic model within the context of another arithmetic model featuring the MEASUREMENT annotation with value “average”, “absolute_average”, “transient”, “RMS”, or “peak”. The evaluation of the auxiliary TIME or FREQUENCY models must be independent from the evaluation of the main model. Otherwise, TIME or FREQUENCY would have to appear within the HEADER of the main model.

In the context of a VECTOR, the auxiliary TIME model may feature a FROM or a TO statement. In the case of “peak”, this statement relates the occurrence time of the peak measurement to a transition appearing in the VECTOR (see ALF 2.0, chapter 8.9.3). In case of “average”, “absolute_average”, “transient”, “RMS”, the FROM and TO statement define the occurrence time of a transition appearing in the VECTOR as the start or end time, respectively, of the measurement.

Example:

The following example specifies multiple average power measurements within a single vector.

```
VECTOR ( 01 in -> 01 out ) {
   POWER p1 = 0.3 {
      MEASUREMENT = average;
      TIME { FROM { PIN = in; } TO { PIN = out; } }
```

```
   }
// average power is 0.3 measured between the transition at ”in”
// and the transition at ”out”
   POWER p2 = 0.4 {
      MEASUREMENT = average;
      TIME = 0.2 { FROM { PIN = out; } }
   }
// average power is 0.4 measured during 0.2 time units
// after transition at ”out”
}
```

The following example specifies time-window-sensitive noise margin.

```
VECTOR ( *? data -> 01 clock -> ?* data ) {
   NOISE_MARGIN = 0.45 {
      PIN = data;
      MEASUREMENT = transient;
      TIME {
         FROM { PIN=data; EDGE_NUMBER=0; }
         TO { PIN=data; EDGE_NUMBER=1; }
      }
   }
// pin ”data” is noise-sensitive only around transition at pin ”clock”
   SETUP = 0.2 {
         FROM { PIN=data; EDGE_NUMBER=0; } TO { PIN=clock; }
   }
// sensitivity window starts 0.2 time units before ”clock” transition
   HOLD = 0.3 {
         FROM { PIN=clock; } TO { PIN=data; EDGE_NUMBER=1; }
   }
// sensitivity window ends 0.3 time units after ”clock” transition
}
```

16.0 Reference to models in other format than ALF

relation to ALF 2.0 3.2.3, 7, others?

relation to IEEE P1603 TBD

History proposed by Alex, Oct. 9, 2001
incorporated in this document by Wolfgang, Oct. 16
initial review and discussion, Nov. 12
Alex should provide more comments
Decision on Jan. 14, 2002:
Alex need to refine the proposal by Apr. 16,
otherwise it will be closed and rejected

Status open

16.1 Motivation

VHDL and Verilog 2000 provide features to reference models written in other languages than VHDL and Verilog, respectively. The trend is multi-language support, and the capability to reference models, written for instance in C or C++ eliminates the need for translation and makes re-use of existing models more efficient.

16.2 Proposal

Note: This proposal would represent a major enhancement of ALF. It should be driven by the need and the feasibility of an implementation proving the concept. To get started, only rough ideas are given here.

The INCLUDE statement (see ALF 2.0, chapter 3.2.3) could be enhanced to specify the format of included files.

Example:

```
INCLUDE ”model1.vhd” { FORMAT = VHDL ; }
INCLUDE ”model2.v” { FORMAT = Verilog ; }
INCLUDE ”model3.c” { FORMAT = ”C++” ; }
INCLUDE ”model4.alf” { FORMAT = ALF ; } //default
```

The arithmetic_model statement (see ALF 2.0, chapter 7) could be enhanced to specify a reference to an external subroutine for evaluation of a model, instead of a TABLE or EQUATION. Such an external subroutine must be found in an included file. The arguments of the subroutine could be specified in the HEADER as long as they can be semantically interpreted as artihmetic_models. The complete set of arguments, including arguments which are alien to the ALF semantics, such as pointers to file handles etc., should be specified within the body of the subroutine statement.

Example:

```
DELAY Tdelay { FROM { PIN=X; } TO { PIN=Y; }
   HEADER {
      SLEWRATE Tslew { PIN=X; }
      CAPACITANCE Cload { PIN=Y; }
   }
   SUBROUTINE {
      Tdelay = double;
      Tslew = double ;
      Cload = double ;
   }
}
```

Corresponding C code:

```
double Tdelay (Tslew, Cload)
double Tslew, Cload ;
{
   /* calculate return_value */
   return (return_value) ;
}
```

17.0 ROUTE annotation for PATTERN

relation to ALF 2.0 9.7

relation to IEEE P1603 9.9.3

History proposed by Wolfgang, Oct. 16, 2001
initial review Nov. 12
reviewed, amended and accepted Jan. 14, 2002

Status closed, accepted

17.1 Motivation

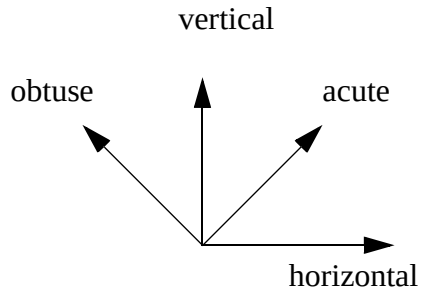
Rules involving layout patterns may be anisotrop, i.e., depending on he routing direction. For example, the minimum distance between parallel lines on a given routing layer may depend on whether they are routed in horizontal or vertical direction (assuming that either tounting direction is allowed).

17.2 Proposal

The PATTERN statement shall have an optional ROUTE annotation with the legal values “horizontal”, “vertical”, “acute”, and “obtuse”. In absence of the ROUTE annotation, the prefered routing direction (see PREFERENCE statement, ALF 2.0, chapter 9.5.4) shall be presumed.

The ROUTE annotation shall define the angle between the routing direction and a horizontal line, as specified in the following illustration.

annotation value	angle [degrees]
horizontal	0
acute	45
vertical	90
obtuse	135



Example:

```
RULE min_distance_horizontal {
   PATTERN p1 { LAYER=metal1; SHAPE=line; ROUTE=horizontal; }
   PATTERN p2 { LAYER=metal1; SHAPE=line; ROUTE=horizontal; }
   LIMIT { DISTANCE { BETWEEN { p1 p2 } MIN=0.5; } }
```

```
}
RULE min_distance_vertical {
   PATTERN p1 { LAYER=metal1; SHAPE=line; ROUTE=vertical; }
   PATTERN p2 { LAYER=metal1; SHAPE=line; ROUTE=vertical; }
   LIMIT { DISTANCE { BETWEEN { p1 p2 } MIN=0.4; } }
}
```

Note: Does this make the arithmetic submodels HORIZONTAL and VERTICAL (ALF 2.0, table 7-8) obsolete? Or, if they are kept, should aditional arithemtic submodels ACUTE and OBTUSE be defined?

This question is addressed in Section 30.0, "Arithmetic submodels for physical library," on page 64.

18.0 REGION statement

relation to ALF 2.0 9

relation to IEEE P1603 9.9

History proposed by Wolfgang, Oct. 16, 2001
initial review Nov. 12
reviewed Jan. 14, 2002, accepted in principle

Status tentatively closed, accepted

18.1 Motivation

The definition of abstract regions (as opposed to concrete layout patterns) has many applications: wire load models with obstructions, definition of transistors as intersection of poly and diffusion, scope of metal density check etc. Boolean operations on regions (and, or, exor) are also useful.

18.2 Proposal

The REGION statement shall be defined as follows:

```
region ::=
REGION region_identifier { region_items }

region_items ::= region_item { region_item }

region_item ::=
   all_purpose_item
   geometric_model
   geometric_transformation
   BOOLEAN_single_value_annotation

// all_purpose_item, geometric_model, geometric_transformation
// see existing grammar

BOOLEAN_single_value_annotation ::=
   BOOLEAN = boolean_expression ;
```

The operands *BOOLEAN*_single_value_annotation in the shall be *region*_identifiers of already defined regions or *pattern*_identifiers of already defined patterns or *layer*_identifiers of already defined layers.

The REGION statement shall be legal in the context of LIBRARY, SUBLIBRARY, CELL, WIRE, RULE, ANTENNA.

Reference to a REGION statement is made by a single-value annotation of the form

```
REGION = region_name_identifier ;
```

Example:

```
/* This antenna rule relates “gate” area, i.e. intersection of poly and
diffusion with total area of poly including routing */
LAYER poly { PURPOSE = reserved ; }
Layer diff { PURPOSE = reserved ; }
ANTENNA for_poly {
   REGION gate { BOOLEAN = POLY && DIFF; }
   SIZE {
      HEADER {
         AREA Atotal { LAYER = poly; }
         AREA Agate { REGION = gate; }
      }
      EQUATION { Atotal / Agate }
   }
   LIMIT { SIZE { MAX = 100; } }
}

/* This rule defines local metal density in a 300um*300um region */
RULE local_metal_density {
   REGION local { WIDTH = 300; HEIGHT = 300; }
   LIMIT { DENSITY { REGION = local; MIN = 0.2; } }
}
```

19.0 WIRE instantiation within arithmetic model

relation to ALF 2.0 8.15

relation to IEEE P1603 9.4

History proposed by Wolfgang, Oct. 16, 2001
initial review Nov. 12, made modifications
reviewed and accepted Jan. 14, 2002

Status closed, accepted

19.1 Motivation

Cells may be characterized with more complex load models than just a lumped capacitance, e.g. pi-model, lumped RLC, transmission line etc. Such complex load models can be described using the WIRE statement. However, there must be a statement connecting such models to a pin of a cell subjected to characterization.

19.2 Proposal

An arithmetic model describing electrical cell characterization data may contain a wire-instantiation statement defined as follows:

```
wire_instantiation ::=
   wire_identifier wire_instance_identifier { pin_assignments }
// pin_assignments see existing grammar
```

The *wire*_identifier shall be the name of an already defined WIRE. The *wire_instance*_identifier shall provide means to reference a named arithmetic model inside the WIRE using a hierarchical identifier. The pin_assignments shall define connectivity between a node within the WIRE (LHS) and a pin within the CELL (RHS).

19.3 Supplementary proposal

To enable referencing of the components of the WIRE by the HEADER of the arithmetic model, the MODEL annotation (see Section 19.0, “WIRE instantiation within arithmetic model,” on page 40) shall be used, in conjunction with an hierarchical identifier.

Example:

```
CELL my_cell {
   PIN in { DIRECTION=input; }
   PIN out { DIRECTION=output; }
   WIRE pi_model {
      NODE n1 { NODETYPE=driver; }
      NODE n2 { NODETYPE=receiver; }
      NODE n3 { NODETYPE=gnd; }
```

```
      CAPACITANCE C1 { NODE { n1 n3 } }
      CAPACITANCE C2 { NODE { n2 n3 } }
      RESISTANCE R1 { NODE { n1 n2 } }
   }
   DELAY {
      FROM { PIN=in; } TO { PIN=out; }
      pi_model load { n1 = out; }
      HEADER {
         CAPACITANCE C_near { MODEL = load.C1; TABLE { x x x } }
         CAPACITANCE C_far  { MODEL = load.C2; TABLE { x x x } }
         RESISTANCE { MODEL = load.R1; TABLE { x x } }
      }
      TABLE { x x x  x x x  x x x   x x x  x x x  x x x }
   }
}
```

20.0 Amendments and simplifications for arithmetic model

relation to ALF 2.0 7.1, 7.3 through 7.6

relation to IEEE P1603 11.2 through 11.6

History Jan. 14, 2002 by Wolfgang
reviewed and accepted

Status closed, accepted

20.1 Motivation

There are many ways to construct arithmetic models in ALF. Some of them are predominantly used, others are seldom used and eventually redundant. The goal here is to simplify the rules for arithmetic models and eventually remove redundant features or replace them by new features which are easier to understand and implement.

20.2 MODEL annotation

The optional MODEL annotation within a *partial arithmetic model* or a *partial arithmetic submodel* shall be a single-value annotation, consisting of the name of another arithmetic model or the hierarchical name of an arithmetic submodel.

The arithmetic model or submodel referenced by the MODEL annotation shall be used for evaluation of the arithmetic model or submodel containing the annotation. Both arithmetic models must have the same type. In the case of arithmetic submodels, the parental arithmetic models must have the same type.

Example:

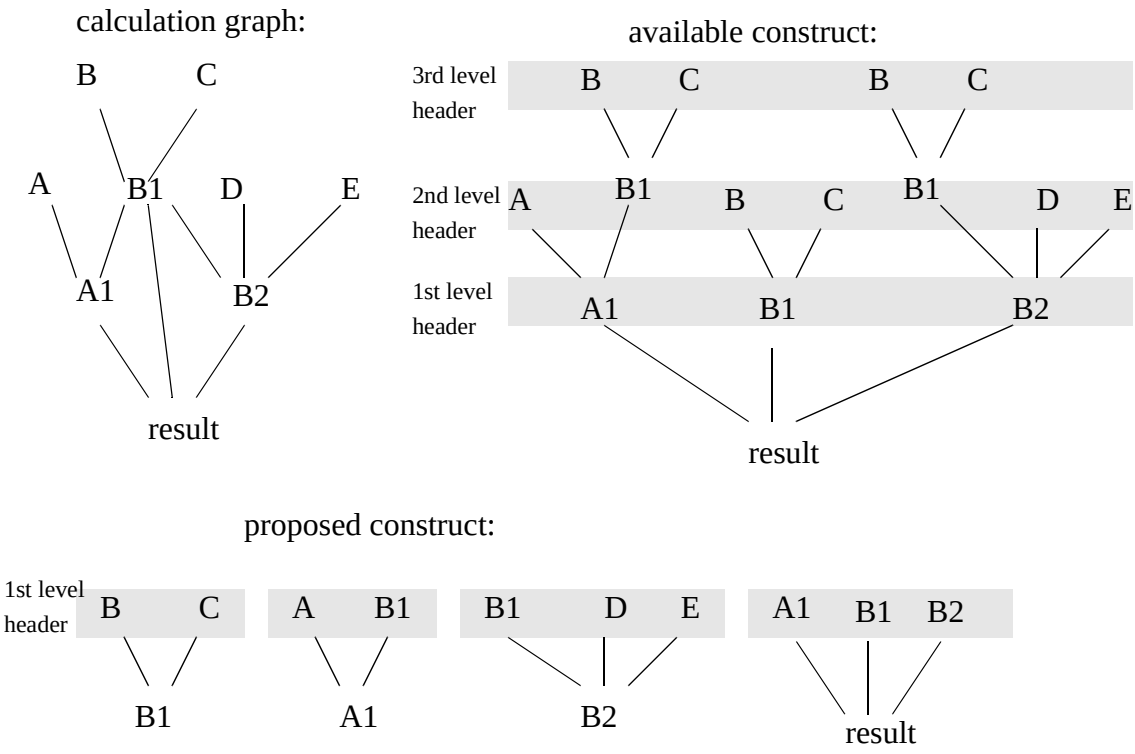
```
LIBRARY my_library {
   KEYWORD DERATE_FACTOR = arithmetic_model;
   DERATE_FACTOR library_default_for_timing {
      HEADER { DERATE_CASE { TABLE {wccom nom bccom } } }
      TABLE { 1.3 1.0 0.8 }
   }
   CELL my_cell {
      PIN A { DIRECTION = input; }
      PIN Y { DIRECTION = output; }
      DELAY cell_default { RISE = 4.5 ; FALL = 3.8 ; }
      VECTOR ( 01 A -> 01 Y) {
         DELAY a_to_y_rise { FROM { PIN = A; } TO { PIN = Y; }
            HEADER {
               DELAY { MODEL = cell_default.RISE ; }
               DERATE_FACTOR { MODEL = library_default_for_timing ; }
            }
            EQUATION { DELAY * DERATE_FACTOR }
         }
}  }  }
```

20.3 Language simplification enabled by MODEL annotation

In ALF 2.0, an arithmetic model for `a_to_y_rise` with the same mathematical result can be described as follows:

```
DELAY a_to_y_rise { FROM { PIN = A; } TO { PIN = Y; }
   HEADER {
      DERATE_FACTOR {
         HEADER { DERATE_CASE { TABLE {wccom nom bccom } } }
         TABLE { 1.3 1.0 0.8 }
      }
   }
   EQUATION { 4.5 * DERATE_FACTOR }
}
```

The ALF 2.0 construct uses a nested arithmetic model (see ALF 2.0, chapter 7.1.5). A nested arithmetic model can be very cumbersome to describe, if multiple levels of intermediate arithmetic models are needed. Also, the nested arithmetic model construct can only describe calculation trees, therefore arguments in a reconvergent calculation graphs must be re-described multiple times. This is illustrated in the following picture:



Therefore, the MODEL annotation capability makes the nested model construct obsolete.

Note: It could be argued to use the TEMPLATE construct instead to simplify nested models. However, the description with template would still be more complex than with MODEL annotation, and it would be more expensive to parse. A parser would have to

evaluate the template instantiations all the way to the branches(i.e., highest level of HEADER) before the arithmetic model could be validated. In contrast, arithmetic models with only 1 level of HEADER could be all validated independently.

ALF 2.0 provides a “shortcut” for mapping non-numerical values of PROCESS or DERATE_CASE into numerical values (see ALF 2.0, chapter 8.6.4). The arithmetic model for `a_to_y_rise` could use the shortcut as follows:

```
DELAY a_to_y_rise { FROM { PIN = A; } TO { PIN = Y; }
   HEADER {
      DERATE_CASE {
         HEADER { wccom nom bccom }
         TABLE { 1.3 1.0 0.8 }
      }
   }
   EQUATION { DELAY * DERATE_CASE }
}
```

If the nested model construct is obsoleted, this “shortcut” is also obsoleted. The parser is further simplified, because the specific rules for this “shortcut” go away.

The syntax cnstruct to support the “shortcut” was

```
HEADER { identifier { identifier } }
```

where the identifier could not only be a set of non-numerical values of PROCESS or DERATE_CASE, but also a set of keywords of self-contained partial arithmetic models, that is, arithmetic models without any annotation.

Example:

```
LAYER via_1_2 { // this is a cut layer
   LIMIT {
      CURRENT {
         HEADER { HEIGHT WIDTH }
         EQUATION { 0.3 * HEIGHT * WIDTH }
         // 0.3 is the current density limit, i.e., current per area
      }
   }
}
```

To support self-contained partial arithmetic models with the proposed language simplification, we propose to support “empty” arithmetic models.

Old rule:

```
model_keyword_identifier [ model_name_identifier ]
   { arithmetic_model_items }

arithmetic_model_items ::=
   arithmetic_model_item { arithmetic_model_item }
```

New rule:

```
   model_keyword_identifier [ model_name_identifier ]
      { { arithmetic_model_item } }
```

With the new rule, the example will read:

```
LAYER via_1_2 { // this is a cut layer
   LIMIT {
      CURRENT {
         HEADER { HEIGHT { } WIDTH { } }
         EQUATION { 0.3 * HEIGHT * WIDTH }
         // 0.3 is the current density limit, i.e., current per area
      }
   }
}
```

The old rule supports only uniformly self-contained or uniformely non-self contained arithmetic models within a HEADER. The new rule supports a HEADER where some arithmetic models are self-contained and others are not.

Note: Should the rule for "empty" constructs be generalized for all ALF statements?

20.4 Obsolete construct with both TABLE and EQUATION

An arithmetic model containing both TABLE and EQUATION statement is supposed to be supported in the following way:

If the values of all arguments lie within the range of their respective validity, the table applies. If the value of some argument lies outside the range, the equation is applied instead of the table.

The drawback of this construct is, that the case where the equation has also a limited range of validity can not be described. Also, the case of using different equations beyond the upper bound and beyond the lower bound cannot be described with this construct. Moreover, the construct is redundant, since the intent of this construct can be described using the following generally supported features within arithmetic model:

- TABLE with optional range defined by MIN and MAX
- EQUATION with optional range defined by MIN and MAX
- Describe a model using TABLE, another model using EQUATION, a target model referencing the other models using MODEL annotation and an EQUATION describing the usage condition of each referenced model

We propose to obsolete the construct featuring both TABLE and EQUATION because of its redundancy and limited applicability.

Example:

Old construct:

```
DELAY my_target {
   HEADER {
      CAPACITANCE my_arg {
         TABLE { 1 2 4 8 } MIN = 0.1; MAX = 10.5;
   }  }
   TABLE { 0.5 0.9 1.9 4.0 }
   EQUATION { 0.5 * my_arg }
}
```

New construct:

```
DELAY intermediate_model {
   HEADER {
      CAPACITANCE { TABLE { 1 2 4 8 } }
   }
   TABLE { 0.5 0.9 1.9 4.0 }
}
DELAY my_target {
   HEADER {
      DELAY my_table { MODEL = intermediate_model; }
      CAPACITANCE my_arg { /* put extended range of validity here */ }
   }
   EQUATION {
      (my_arg >= 0.1 && my_arg <= 10.5)? my_table : 0.5 * my_arg
   }
}
```

21.0 Amendments for hierarchical antenna support

relation to ALF 2.0 9.7, 9.13

relation to IEEE P1603 9.9.2, 9.10.3, 11.13.12

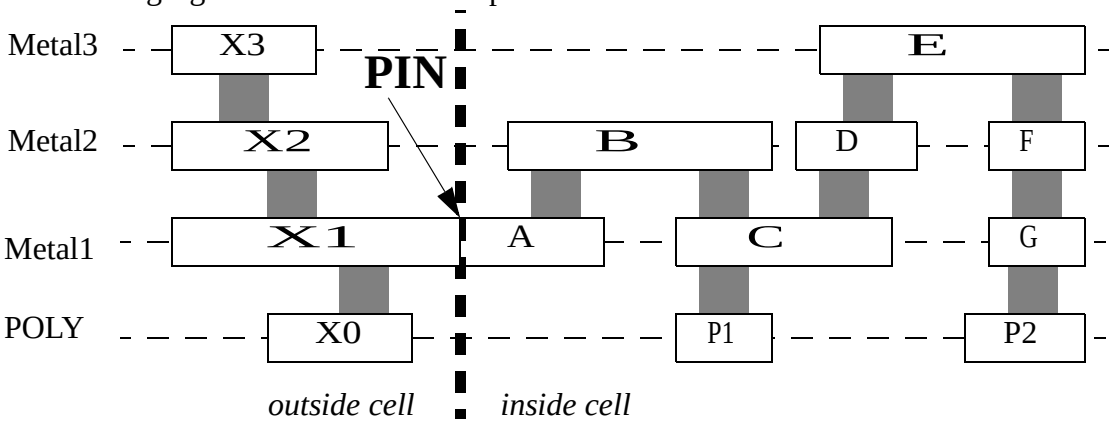
History Jan. 14, 2002 by Wolfgang
reviewed Jan. 14 and Feb. 13, accepted

Status closed, accepted

21.1 Motivation

Antenna rules are established to prevent transistors to be damaged during manufacturing of upper metal layers. Evaluation of antenna rules requires visibility of all layers, starting from polysilicon and diffusion which constitute the transistors. In cell-based design, the artwork of a transistor connected to a pin of a cell is not visible to a layout tool. ALF supports the annotation of transistor AREA to the pin as a pertinent parameter for antenna rule evaluation. For complex cells or blocks, this abstraction is not suffcient.

The following figure describes an example:



When the Metal2 segment “X2” is manufactured, the transistor area “P1” is exposed to antenna. When the Metal3 segment “X3” is manufactured, the transistor areas “P1” and “P2” are exposed to antenna. The smaller the exposed transistor area, the greater the damage. Therefore, using the area “P1” is pessimistic, using the area “P1+P2” is not safe.

21.2 PATTERN annotation in context of PIN

Allow the PATTERN statement in context of PIN. Currently, the PATTERN statement is allowed in context of PORT, BLOCKAGE and RULE. A PORT is allowed in context of a PIN.

In conjunction with the amended CONNECTIVITY statement (see Section 13.0, “Amended semantics of CONNECTIVITY,” on page 28), the pertinent information for hierarchical antenna rule checking can be described.

Example:

```
// Note: items in italic are not verbatim, they should be numbers
PIN my_pin {
   PATTERN A  { LAYER = M1; RECTANGLE { left bottom right top } }
   PATTERN B  { LAYER = M2; AREA = B; }
   PATTERN C  { LAYER = M1; AREA = C; }
   PATTERN D  { LAYER = M2; AREA = D; }
   PATTERN E  { LAYER = M3; AREA = E; }
   PATTERN F  { LAYER = M2; AREA = F; }
   PATTERN G  { LAYER = M1; AREA = G; }
   PATTERN P1 { LAYER = POLY; AREA = P1; }
   PATTERN P2 { LAYER = POLY; AREA = P2; }
   PORT my_port { PATTERN = A; }
   CONNECTIVITY = 1 { BETWEEN { A B } CONNECT_TYPE=physical; }
   CONNECTIVITY = 1 { BETWEEN { B C P1 } CONNECT_TYPE=physical; }
   CONNECTIVITY = 1 { BETWEEN { C D E  } CONNECT_TYPE=physical; }
   CONNECTIVITY = 1 { BETWEEN { E F G P2 } CONNECT_TYPE=physical; }
}
```

This example corresponds to the figure above. Note that the information about the various patterns is restricted to area. Only the pattern A, which is must be visible as a physical port, is described as rectangle. Reference to pattern A is made by the port.

Note: For similar reasons, a PATTERN statement could be allowed in the context of a CELL. This would, for example, facilitate hierarchical parasitic extraction.

21.3 TARGET annotation within precalculated antenna SIZE

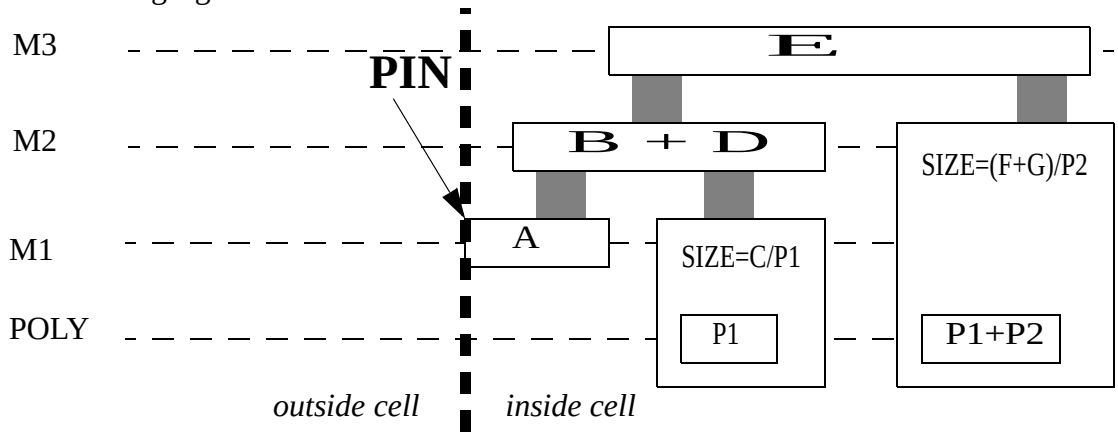
The pertinent data for antenna calculation can abstracted further, if the arithmetic model describing antenna SIZE is partially pre-calculated considering the patterns within the cell. The following calculation refers still to the same example

- Antenna with target X0:
  SIZE = AREA(**A**+X1) / AREA (X0)
  + AREA(**B**+**D**+X2) / AREA(**P1**+X0)
  + AREA (**E**+X3) / AREA(**P1**+**P2**+X0)
- Antenna with target P1:
  SIZE = **AREA(C) / AREA(P1)**
  + AREA(**B**+**D**+X2) / AREA(**P1**+X0)
  + AREA (**E**+X3) / AREA(**P1**+**P2**+X0)
- Antenna with target P2:
  SIZE = **AREA(F) / AREA(P2) + AREA(G) / AREA(P2)**
  + AREA(**E**+X3) / AREA(**P1**+**P2**+X0)

The segments A, E, and P1 are pertinent. The individual segments B and D can be combined, since only the sum B+D is pertinent. The partial SIZE=AREA(C)/AREA(P1) can be precalculated. The partial SIZE=AREA(F+G)/AREA(P2) can also be precalculated.

Thus the individual segment P2 is not exposed and can be replaced by a combination of P1 and P2.

The following figure shows the abstraction.



In order to describe this abstraction in ALF, the TARGET annotation within the context of SIZE is proposed. The value of the TARGET annotation is a PATTERN, corresponding to the transistor area exposed to antenna.

Example:

```
// Note: items in italic are not verbatim, they should be numbers
PIN my_pin {
   PATTERN A     { LAYER = M1; RECTANGLE { left bottom right top } }
   PATTERN B_D   { LAYER = M2; AREA = B+D; }
   PATTERN E     { LAYER = M3; AREA = E; }
   PATTERN P1    { LAYER = POLY; AREA = P1; }
   PATTERN P1_P2 { LAYER = POLY; AREA = P1+P2; }
   PORT my_port  { PATTERN = A; }
   CONNECTIVITY forX0 = 1 {
      BETWEEN { A B_D } CONNECT_TYPE = physical;
   }
   CONNECTIVITY forP1 = 1 {
      BETWEEN { P1 B_D E }  CONNECT_TYPE = physical;
   }
   CONNECTIVITY forP2 = 1 {
      BETWEEN { P1_P2 E }  CONNECT_TYPE = physical;
   }
   SIZE targetP1 = C/P1 {
      TARGET=P1; CALCULATION=incremental;
   }
   SIZE targetP2 = (F+G)/P2 {
      TARGET=P1_P2; CALCULATION=incremental;
   }
}
```

An eventual antenna rule violation will be assocated with the PATTERN referenced by the TARGET annotation.

22.0 Amendments for REFERENCE related to DISTANCE

relation to ALF 2.0 9.18

relation to IEEE P1603 11.13.12

History Jan.14, 2002 by Wolfgang
reviewed Jan. 14 and Feb. 13, accepted

Status closed, accepted

22.1 Motivation

A DISTANCE between two physical patterns can be described in ALF. The reference point for distance measurements is indicated by the REFERENCE annotation which is assocated with the DISTANCE statement. This annotation specifies whether the distance is measured between the centers or the edges or the origins of the patterns. However, a distance measured between the center of one pattern and the edge of another pattern cannot be specified in this way. Therefore, we propose to replace the annotation by an annotation container which can specify the point of reference for each object.

22.2 Proposal

Existing definition:

```
reference_annotation ::=
REFERENCE = center | origin | edge ;
```

Proposed definition:

```
reference_annotation_container ::=
REFERENCE {
   1st_pattern_identifier = center | origin | edge ;
   2nd_pattern_identifier = center | origin | edge ;
}
```

Example:

```
DISTANCE d1 {
   BETWEEN { pattern1 pattern2 }
   REFERENCE { pattern1 = origin; pattern2 = origin; }
   // instead of “REFERENCE = origin;” in ALF 2.0
}
DISTANCE d2 {
   BETWEEN { pattern3 pattern4 }
   REFERENCE { pattern3 = origin; pattern4 = edge; }
   // not possible to describe in ALF 2.0
}
```

23.0 Amendments for PIN_GROUP

relation to ALF 2.0 6.5

relation to IEEE P1603 9.4.2

History Jan. 14, 2002 by Wolfgang
reviewed Feb. 13

Status tentatively closed, accepted

23.1 Motivation

ALF 2.0 features a pin_instantiation statement and a PIN_GROUP statement. The PIN_GROUP statement is more general than the pin_instantiation statement. Therefore the pin_instantiation statement is redundant. Since both statements have been only introduced in ALF 2.0, chances are that they have not been adopted in practical applications yet. Therefore we propose to obsolete the pin_instantiation statement.

23.2 Proposal: make pin_instantiation obsolete

The purpose of the pin_instantiation statement is to specify information releveant for a scalar pin or a subarray of a 1-dimensional array pin. The statement allows for recursivity, i.e., a pin_instantiation statement inside another pin_instantiation statement.

Example:

```
PIN [1:100] my_array {
   // put information pertaining to the entire array [1:100] here
   my_array[1:50] { // this is a pin_instantiation
      // put information pertaining to the sub-array [1:50] here
   }
   my_array[51:100] { // this is another pin_instantiation
      // put information pertaining to the sub-array [51:100] here
   }
}
```

The equivalent construct using PIN_GROUP looks as follows:

```
PIN [1:100] my_array {
   // put information pertaining to the entire array [1:100] here
}
PIN_GROUP [1:50 ] subarray_low {
   MEMBERS { my_array [1:50] }
      // put information pertaining to the sub-array [1:50] here
}
PIN_GROUP [1:50 ] subarray_high {
   MEMBERS { my_array [51:100] }
      // put information pertaining to the sub-array [51:100] here
}
```

The PIN_GROUP statement is more general, because it allows for concatenation of arbitary pins, whereas the pin_instantiation allows only for subarrays within one parent pin.

The following example can only described with PIN_GROUP, not with pin_instance:

```
PIN pin1 { /* put information pertaining to pin1 here */ }
PIN [1:8] pin2 { /* put information pertaining to pin2 here */ }
PIN_GROUP [0:3] my_pingroup {
   MEMBERS { pin2[7:8] pin1 pin2[5] }
   // pin mapping is as follows:
   //     my_pingroup[0] <= pin2[7]
   //     my_pingroup[1] <= pin2[8]
   //     my_pingroup[2] <= pin1
   //     my_pingroup[3] <= pin2[5]
   /* put information pertaining to my_pingroup here */
}
```

Therefore the pin_instance statement can be obsoleted. The effect is langauge simplification without loss of description capability.

23.3 Proposal: rename PIN_GROUP to PINGROUP

A cosmetic change of the keyword PIN_GROUP to PINGROUP is proposed. The rationale is as follows:

In ALF convention, composite keywords building on a basic keyword use “_” as separator. For example, NON_SCAN_CELL builds on CELL, RESTRICT_CLASS builds on CLASS. However, PINGROUP does not build on GROUP. GROUP has in fact quite the opposite effect of PINGROUP. GROUP allows to generate multiple statements from a single statement. Maybe “EXPAND” would have been a better choice for a keyword than “GROUP”, but “GROUP” has been around since ALF 1.0 ...

Compatibility between changing keywords can be achieved by standardizing on an ALIAS.

Example:

```
ALIAS PIN_GROUP = PINGROUP ;
// backward compatibility:
// PINGROUP is new primary keyword
// PIN_GROUP can still be used as alternative keyword

ALIAS EXPAND = GROUP ;
// forward compatibility:
// GROUP remains primary keyword
// EXPAND becomes alternative keyword
```

Subsequent revisions of the ALF standard could introduce a new keyword as alternative keyword, then swap alternative keyword and primary keyword, finally obsolete the alternative keyword.

24.0 Extended definition of PURPOSE annotation

relation to ALF 2.0	6.6, 8.15
relation to IEEE P1603	8.6, 9.5, 9.6.3
History	Jan.14, 2002 by Wolfgang reviewed Feb. 13
Status	open

24.1 Motivation

As ALF is a very general language and many application tools need to build datamodels only for specific subsets of the ALF statements in a library, we propose to use the PURPOSE annotation. Currently, the PURPOSE annotation is supported only in the context of a few ALF statements.

24.2 Proposal: generalized PURPOSE for CLASS

The CLASS statement has many purposes, given by those other statements making reference to the CLASS statement, for instance as RESTRICT_CLASS, SWAP_CLASS, CONNECT_CLASS, EXISTENCE_CLASS, SUPPLY_CLASS. The PURPOSE annotation for CLASS statement is supported for the target usage EXISTENCE_CLASS. We propose to define a PURPOSE multivalue annotation indicating each target usage of a CLASS statement.

24.3 Proposal: PURPOSE for WIRE

A WIRE statement can describe a statistical wireload model, a model for boundary parasitics within a cell, a model for an electrical network to be connected to a cell, a model for interconnect delay and noise calculation, a model for parasitic reduction. We propose to define a PURPOSE (singlevalue or multivalue?) annotation to indicate what is described.

24.4 Proposal: PURPOSE for REGION

A REGION statement has multiple potential usages: description of electrical components such as transistors or diodes for antenna rule check, descriptions of region of interest for metal density check, description of an all-layer blockage in context of a cell, description of a bounding box of a cell, description of a blockage in context of a wireload model, description of allowed and disallowed regions for over-block routing, description of a subfloorplan within a block ...

We propose to define a PURPOSE annotation dependent on whether the REGION statement feature will be accepted and what semantics will be supported.

25.0 Amended semantics of ILLEGAL statement

relation to ALF 2.0 6.7

relation to IEEE P1603 9.6.2

History Jan.14, 2002 by Wolfgang
reviewed Feb. 13, 2002

Status open

25.1 Motivation

The ILLEGAL statement is similar to SETUP or HOLD or other timing constraint statement insofar as it appears in context of a VECTOR and can contain a VIOLATION statement. However, timing constraint statements are arithmetic models whereas ILLEGAL statement is a statement on its own. It indicates whether the VECTOR constitutes an illegal state. An illegal state may be tolerable for a short amount of time. Therefore the ILLEGAL statement could be interpreted as arithmetic model indicating the tolerable duration of the illegal state. This provides more modeling capability. As a side effect, the language is simplified, since the ILLEGAL statement now falls in the category of arithmetic models.

25.2 Proposal

The ILLEGAL statement shall describe an arithmetic model in the context of a VECTOR. If the related control expression is a boolean expression, the arithmetic model shall describe the tolerable duration of the state defined by the boolean expression. If the related control expression is a vector expression, the arithmetic model shall describe the tolerable duration of time measured between specified events, using the FROM and TO statements.

If no data is associated with the arithmetic model, the state or sequence of events defined by the control expression shall be considered illegal independent of time. The occurence of the VECTOR alone constitutes a VIOLATION.

If data is associated with the arithmetic model, an actual duration of the state equal or greater than the duration calculated by evaluation of the arithmetic model shall constitute a VIOLATION.

Example:

```
VECTOR ( A && ! A_bar ) {
   ILLEGAL = 0.5 {
      VIOLATION { /* put consequence of violation here */ }
   }
}
```

A violation occurs, if the state `A && ! A_bar` lasts 0.5 units of time or longer.

```
VECTOR ( 01 B -> 01 B_bar -> 10 B_bar ) {
   ILLEGAL = 0.8 {
      FROM { PIN = B_bar; EDGE_NUMBER = 0; }
      TO { PIN = B_bar; EDGE_NUMBER = 0; }
      VIOLATION { /* put consequence of violation here */ }
}
```

A violation occurs, if the elapsed time between `01 B_bar` and `10 B_bar` is 0.8 units of time or longer.

26.0 CONTROL_POLARITY statement

relation to ALF 2.0 6.4.6

relation to IEEE P1603 9.4.5

History Jan.14, 2002 by Wolfgang
reviewed Feb. 13

Status open

26.1 Motivation

ALF 2.0 features the POLARITY statement, which is either a single-value annotation or an annotation container, depending on the value of the SIGNALTYPE annotation. POLARITY as annotation container applies only for a composite signaltype value with the fundamendal signaltype “control” or “clock”. The composite consists of the names of operation modes which are controlled by the signal. The POLARITY statement states those modes again and asssociates each of them with a value. Currently, the names of operation modes are predefined, e.g. “read”, “write”, “scan”. But for many applications, a customized name space would be preferable. Also, priority of control signals can not be described within the POLARITY statement.

We propose to introduce the CONTROL_POLARITY statement as an amendment.

26.2 Proposal

The CONTROL_POLARITY statement shall be defined as one-level annotation container in the context of a PIN as follows:

```
control_polarity_one_level_annotation_container ::=
   CONTROL_POLARITY {
      mode_identifier = polarity_value_identifier ;
      { mode_identifier = polarity_value_identifier ; }
   }
```

where *polarity_value*_identifier supports the set of values already defined for the POLARITY annotation (i.e. “high”, “low”, “rising_edge”, “falling_edge”, “double_edge”) and *mode*_identifier supports an arbitrary set of values on top of already predefined values (i.e. “read”, “write”, “test”, “scan”, “bist”).

This statement eliminates the necessity for composite SIGNALTYPE values building on fundamental SIGNALTYPE values “control” or “clock” (for example “read_write_control”, “read_write_clock”).

The POLARITY statement shall now be only a single-value annotation. Its semantics for usage as single-value annotation shall remain unchanged. In particular, POLARITY shall

be supported for “clock”, but not for “control”. As a consequence, “clock” can be associated with both POLARITY and CONTROL_POLARITY.

Example for “control”:

```
PIN mode_sel_1 { SIGNALTYPE = control ;
   CONTROL_POLARITY {
      normal = high ;
      scan = low ;
      hold = low ;
}  }
PIN mode_sel_2 { SIGNALTYPE = control ;
   CONTROL_POLARITY {
      scan = high ;
      hold = low ;
}  }
// corresponding truth table:
// mode_sel_1 mode_sel_2 mode of operation
// 0          0          hold
// 0          1          scan
// 1          0          normal
// 1          1          normal
```

This construct provides priority information for control signals.

Example for “clock”:

old construct:

```
PIN rw_clock {
   SIGNALTYPE = read_write_clock ;
   POLARITY {
      read = rising_edge ;
      write = falling_edge ;
   }
}
```

new construct:

```
PIN rw_clock {
   SIGNALTYPE = clock ;
   POLARITY = double_edge ;
   CONTROL_POLARITY {
      read = rising_edge ;
      write = falling_edge ;
   }
```

In the old construct, there is a dependency between the value of SIGNALTYPE and the contents of the POLARITY statement. The new construct is more orthogonal. The modes are only found within the contents of the CONTROL_POLARITY statements.

From an electrical standpoint, the information “double_edge” indicates a requirement for clock distribution with tighter requirements on duty cycle and slewrates than “rising_edge” or “falling_edge”. In the old construct, “double_edge” was only supported if both edges control the same operation. In the case of different operations, “double_edge” had to be infered from the mode-specific polarity values. In the new construct, POLARITY is now used in a uniform way. The syntax and semantics rules for POLARITY are independent of its context.

27.0 Review of units for arithmetic models

relation to ALF 2.0	8.1, 9.2, 9.6
relation to IEEE P1603	9.10.5, 11.8
History	Jan.14, 2002 by Wolfgang, with input from Peter and Tak reviewed Feb. 13, 2002
Status	tentatively closed, accepted

27.1 Motivation

We should revisit the systems of base units and default units in order to comply with scientific standards. An error has been detected for base unit of FLUX and FLUENCE. Also, the system of default units should be revisited, so that mathematical calculations can eventually be done without unit conversion. Also, there is no definition of units for coordinates in geometric models. The assumption is that DISTANCE unit is used. Also, we can ponder whether arithmetic models for timing such as DELAY, SETUP etc. need their own units or they should uniformly use the unit of TIME.

27.2 Proposal

FLUX and FLUENCE: change base unit to [1/(m^2 s)] and [1/(m^2)], respectively.

DELAY, RETAIN, SLEWRATE, SETUP, HOLD, RECOVERY, REMOVAL, PULSEWIDTH, PERIOD, NOCHANGE, JITTER, ILLEGAL: use unit of TIME, unless unit for specific model is explicitely defined.

LENGTH, HEIGHT, WIDTH, THICKNESS, OVERHANG: use unit of DISTANCE, unless unit for specific model is explicitely defined.

Introduce following rule: A local definition of units for TIME or DISTANCE overrules a global definition of units for specific model. A local definition of units for specific model overrules definition of TIME or DISTANCE at same level or more global level.

Use the unit of DISTANCE for COORDINATE values within geometric models.

Define default units such that the following mathemetical relationships can be satisfied without unit conversion:

$f = 1 / t$

$P = dE / dt$

$V = R * I$

$I = C * dV / dt$

$V = L * dI / dt$

$P = C * V^2 * f$

flux = d fluence / dt

fluence = 1 / A

$A = D^2$

etc.

with following mathematical symbols

f	frequency
t	time
P	power
E	energy
V	voltage
I	current
R	resistance
C	capacitance
L	inductance
A	area
D	distance
?	flux
?	fluence

One possibility is to make all default units 1 and make sure that all base units are compatible with SI-units.

28.0 Eliminate redundant driver CELL and PIN annotation

relation to ALF 2.0 6.3, 6.4.14

relation to IEEE P1603 N/A

History Wolfgang, 16 April, 2002

Status new

28.1 Motivation

The driver cell and pin annotation is redundant, because the equivalent information can be described more concisely using the STRUCTURE statement. Therefore it is proposed to remove this annotation. This eliminates not only a redundant item, but also the rarely used syntax rule “two_level_annotation”.

28.2 Proposal

The driver cell and pin specification in ALF 2.0, chapter 6.4.14 is defined as follows:

```
CELL myCell {
   PIN myPin {
      CELL = cell1 { PIN = pin1; }
   }
}
```

where “myCell” is a complex block which has at least a boundary gate-level netlist. The pin “myPin” of that complex block is connected to the pin “pin1” which belongs to an instance of the cell “cell1”.

The STRUCTURE statement in ALF 2.0, chapter 6.3, can describe the equivalent information as follows.

```
CELL myCell {
   PIN myPin { ... }
   FUNCTION {
      STRUCTURE {
         cell1 instance1 { pin1 = myPin; }
      }
   }
}
```

Moreover, the STRUCTURE statement supports specification of an instance name (here “instance1”). Also it allows to specify connectivity between physical ports, using hierarchical pin identifiers.

Therefore the driver cell and pin specification is redundant and can be eliminated.

29.0 Substitution for VIA reference

relation to ALF 2.0 3.9.1, 9.8.4, 9.10.2, 11.4, 11.23

relation to IEEE P1603 9.8

History Wolfgang, 16 April, 2002

Status new

29.1 Motivation

The syntax rule for via reference is incorrectly formulated in ALF 2.0. We propose to reformulate the rule not only correctly, but to make the rule similar to other rules with similar semantics.

29.2 Proposal

The VIA reference rule in ALF 2.0, chapter 11.23 was defined as follows:

```
via_reference ::=
   VIA { via_instantiations }
```

where

```
via_instantiation ::=
   via_identifier { geometric_transformations }
```

However, the intent was:

```
via_instantiation ::=
   via_identifier { { geometric_transformations } }
```

Note: the outer brackets indicate “optional”, whereas the inner, bold brackets indicate that brackets shall be used.

Example:

```
VIA via1 { ... } // declaration of a via named “via1”

//somewhere else in the library:
VIA { via1 // reference to “via1” with geometric transformation
   SHIFT { HORIZONTAL = 1; VERTICAL = -2; }
   ROTATE = 90;
}

//somewhere else in the library:
VIA { via1 } // reference to “via1” without geometric transformation
```

End of example

We take issue even with the corrected rule.

The case with geometric-transformations has no precedence. No other instantiation statement (for example cell_instantiation, template_instantiation) is enclosed by the keyword of the ALF type. The case without geometric-transformations could be simply covered under multi-value annotation.

To make the rule similar to other rules, we propose the following amendments:

- To cover the case with geetric-transformation:

```
via_instantiation ::=
   via_identifier [ instance_identifier ] { geometric_transformations }
```

This supports an optional name and makes the via-instantiation similar to cell-instantiation, wire-instantiation etc.

- To cover the case without geometric transformation

A keyword for any library-specific object (not only CELL, PIN, PRIMITIVE as in ALF 2.0, chapter 3.9.1) shall be legal for use as annotation (not only single-value annotation, as in ALF 2.0, chapter 3.9.1).

The rule for via_reference itself can be eliminated, and the usage of via-instantiation or multi-value annotation can be allowed instead.

30.0 Arithmetic submodels for physical library

relation to ALF 2.0 7.6

relation to IEEE P1603 11.15

History Wolfgang, 16 April, 2002

Status new

30.1 Motivation

ALF 2.0 supports arithmetic submodels HORIZONTAL and VERTICAL. In this document, Section 17.0, “ROUTE annotation for PATTERN,” on page 36, new routing directions ACUTE and OBTUSE are proposed. Consequently, the corresponding arithmetic submodels should be introduced.

30.2 Proposal

Allow the arithmetic submodels ACUTE and OBTUSE in the same context as the arithmetic submodels HORIZONTAL and VERTICAL.

The allowed context shall be an arithmetic model of type WIDTH or LENGTH, if this arithmetic model appears in context of a routing layer.

For arithmetic models in context of RULE, the annotations according to Section 17.0 on page 36 shall be used rather than arithmetic submodels.

Part 2: Grammar-related items

31.0 Make grammar more compact by removing redundancies

relation to ALF 2.0 3.2, 11.x

relation to IEEE P1603 Annex A (normative)

History Proposal by Wolfgang, May 22, 2001
review pending as of July 10
Comments from Tim Ehrler per email:
no issue with proposed changes
left open for review by other ALF parser developpers

Status closed here, to be tracked within IEEE P1603

31.1 Motivation

Simplify the grammar by getting rid of redundant definitions. Definitions which are used in a particular context should be presented in that context. This will also simplify to introduce grammar “snippets” in the semantic sections, where they are needed.

31.2 Proposal

ALF 2.0 chapter 11.2

Get rid of chapter 11.2 and introduce the pertinent statements locally, where they are needed.

`unnamed_assignment_base`
remove

`unnamed_assignment`
rename to `single_value_annotation`, move to 11.7

`named_assignment_base`
remove

`named_assignment`
remove

```
single_value_assignment ::=
   identifier = value ;
```

`multi_value_assignment`
rename to `multi_value_annotation`, move to 11.7

`assignment`
remove

`pin_assignment`
modify according to Section 5.0 on page 12, move to 11.7

`arithmetic_assignment`
move to 11.7

ALF 2.0 chapter 11.3

split into 3 separate chapters:

- Boolean expressions and operators
  put `boolean_expression`
- Arithmetic expressions and operators
  put `arithmetic_expression`
- Vector expressions and operators
  put everything else

ALF 2.0 chapter 11.4

Get rid of chapter 11.4 and introduce the pertinent statements locally, where they are needed.

`cell_instantiation`
remove

`unnamed_cell_instantiation`
only used for NON_SCAN_CELL, move to 11.9

`named_cell_instantiation`
only used for STRUCTURE, move to 11.17

`pin_instantiation`
only used for PIN, move to 11.11

Error to be corrected:
incorrect use of `pin_instantiation` in chapter 6.3, should be `pin_assignments`

Error to be corrected:
`pin_instantiation` is not mentioned as `pin_item` in chapter 11.11

`primitive_instantiation`
only used for FUNCTION, move to 11.17

`template_instantiation`
move to 11.7

`dynamic_instantiation_item`
move to 11.7

`via_instantiation`
move to 11.23

ALF 2.0 chapter 11.5

move to “lexical rules” section (chapter 10)

ALF 2.0 chapter 11.6

Get rid of chapter 11.6, associate operators with the corresponding expressions.

- Boolean expressions and operators
  put all operators with prefix `boolean_`
- Arithmetic expressions and operators
  put all operators with prefix `arithmetic_`
- Vector expressions and operators
  put all operators with prefix `vector_`

move `sequential_if`, `sequential_else_if` to 11.17.

ALF 2.0 chapter 11.7

rename `logic_assignment` (see 11.17) into `boolean_assignment` and move into 11.7. Move `vector_assignment` into 11.7.

rewrite grammar involving `all_purpose_item`, `annotation`, `annotation_container`, the other items remain unchanged.

```
annotation ::=
   one_level_annotation
|  two_level_annotation
|  multi_level_annotation

one_level_annotation ::=
   single_value_annotation
|  multi_value_annotation

one_level_annotations ::=
   one_level_annotation { one_level_annotation }

two_level_annotation ::=
   one_level_annotation
|  identifier [ = value ] { one_level_annotations }
```

```
two_level_annotations ::=
   two_level_annotation { two_level_annotation }

multi_level_annotation ::=
   one_level_annotation
|  identifier [ = value ] { multi_level_annotations }

multi_level_annotations ::=
   multi_level_annotation { multi_level_annotation }

annotation_container ::=
   identifier { one_level_annotations }
```

Since `all_purpose_item` allows `generic_object` and `generic_object` includes `keyword_declaration` statement, consequently all *`syntax_item`*`_identifiers` that can be used by `keyword_declaration` (see chapter 3.2.9) must be covered by `all_purpose_item`.

```
all_purpose_item ::=
   generic_object
|  template_instantiation
|  annotation
|  arithmetic_model
|  arithmetic_model_container
|  boolean_assignment
|  vector_assignment
```

Error to be corrected:
`boolean_assignment` is not mentioned as *`syntax_item`*`_identifier` in chapter 3.2.9.

Note: `arithmetic_submodel` is also a *`syntax_item`*`_identifier`, but it is not included in `all_purpose_item`, because `arithmetic_submodel` is always in the context of `arithmetic_model`.

32.0 Rewrite grammar for more specific syntax and less semantic restriction

relation to ALF 2.0 3.2, 11.x

relation to IEEE P1603 Annex A (normative)

History Proposal by Wolfgang, May 22, 2001
review pending as of July 10
Comments from Tim Ehrler per email:
no issue with proposed changes
left open for review by other ALF parser developpers

Status closed here, to be tracked within IEEE P1603

32.1 Motivation

Certain syntax definitions of ALF are written in a very generic way. As a consequence, a lot of semantic restrictions apply. The idea is to rewrite the grammar so that the syntax section becomes more specific and as a consequence the semantic sections become less “heavy”. However, the changes to the existing grammar should be limited to modifications which specifically serve that purpose rather than re-writing the whole grammar from scratch. Also, eventual redundancy in the grammar can be eliminated.

32.2 Proposal

Use `all_purpose_item` only for statements with custom keywords, introduced by `keyword_declaration` statements and put the statements using standard keywords explicitly in the grammar.

ALF 2.0 Chapter 11.9

```
cell_item ::=
   all_purpose_item
|  CELLTYPE_single_value_annotation
|  SWAP_CLASS_one_level_annotation
|  RESTRICT_CLASS_one_level_annotation
|  SCANTYPE_single_value_annotation
|  SCAN_USAGE_single_value_annotation
|  BUFFERTYPE_single_value_annotation
|  DRIVERTYPE_single_value_annotation
|  PARALLEL_DRIVE_single_value_annotation
|  pin
|  pin_group
|  primitive
|  function
|  non_scan_cell
|  test
|  vector
```

```
|  wire
|  blockage
|  artwork
|  connectivity
```

ALF 2.0 Chapter 11.10

```
library_item ::=
   all_purpose_item
```

ALF 2.0 Chapter 11.11

```
pin_item ::=
   all_purpose_item
|  range
|  VIEW_single_value_annotation
|  PINTYPE_single_value_annotation
|  DIRECTION_single_value_annotation
|  SIGNALTYPE_single_value_annotation
|  ACTION_single_value_annotation
|  POLARITY_two_level_annotation
|  DATATYPE_single_value_annotation
|  INITIAL_VALUE_single_value_annotation
|  SCAN_POSITION_single_value_annotation
|  STUCK_single_value_annotation
|  SUPPLYTYPE_single_value_annotation
|  SIGNAL_CLASS_one_level_annotation
|  SUPPLY_CLASS_one_level_annotation
|  cell_pin_reference_two_level_annotation
|  DRIVETYPE_single_value_annotation
|  SCOPE_single_value_annotation
|  PULL_single_value_annotation
|  port
|  connectivity
|  pin_instantiation // this one is missing in chapter 11.11
```

ALF 2.0 Chapter 11.14

```
vector_item ::=
   all_purpose_item
|  PURPOSE_one_level_annotation
|  OPERATION_single_value_annotation
|  LABEL_single_value_annotation
|  EXISTENCE_CLASS_one_level_annotation
|  EXISTENCE_CONDITION_boolean_assignment
|  CHARACTERIZATION_CLASS_one_level_annotation
|  CHARACTERIZATION_CONDITION_boolean_assignment
|  CHARACTERIZATION_VECTOR_vector_assignment
|  MONITOR_one_level_annotation // proposed in this doc chapter 8
|  illegal_statement
```

ALF 2.0 Chapter 11.15

```
wire_item ::=
   all_purpose_item
|  SELECT_CLASS_one_level_annotation
|  node

node ::=
   NODE name_identifier { node_items }

node_items ::=
   node_item { node_item }

node_item ::=
   all_purpose_item
|  NODETYPE_single_value_annotation
|  NODE_CLASS_one_level_annotation
```

ALF 2.0 Chapter 11.16

```
arithmetic_models ::=
   arithmetic_model { arithmetic_model }

arithmetic_model ::=
   partial_arithmetic_model
|  full_arithmetic_model
```

Partial arithmetic model contains only definitions, no data. Can appear outside the semantically valid context of the model, as long as a semantically valid context exists within scope. (Example: semantically valid context of arithmetic model X is VECTOR, VECTOR exists within scope of LIBRARY, therefore partial arithmetic model X is legal within LIBRARY.) Definitions inside partial arithmetic model without *name*_identifier are inherited by each arithmetic model with *arithmetic_model*_identifier within scope. (Note: up to 2 levels of submodel are supported)

```
partial_arithmetic_model ::=
   arithmetic_model_identifier [ name_identifier ] {
      { all_purpose_item }
      { arithmetic_model_qualifier }
      { partial_arithmetic_submodel }
   }

partial_arithmetic_submodel ::=
   arithmetic_submodel_identifier [ name_identifier ] {
      { all_purpose_item }
      { partial_arithmetic_leaf_submodel }
   }

partial_arithmetic_leaf_submodel ::=
   arithmetic_submodel_identifier [ name_identifier ] {
      { all_purpose_item }
   }
```

Full arithmetic model contains both definitions and data. Can only appear in the semantically valid context of the model. Enables evaluation of arithmetic model in design context (e.g. delay calculation, power calculation). A trivial arithmetic model contains directly the

evaluation value. A non-trivial arithmetic model requires calculation of the value, based on evaluation conditions. (Note: up to 2 levels of submodel are supported)

```
full_arithmetic_model ::=
   trivial_arithmetic_model
|  non_trivial_arithmetic_model

trivial_arithmetic_model ::=
   arithmetic_model_identifier [ name_identifier ] = value ;
|  arithmetic_model_identifier [ name_identifier ] = value {
      { all_purpose_item }
      { arithmetic_model_qualifier }
   }

non_trivial_arithmetic_model ::=
   arithmetic_model_identifier [ name_identifier ] {
      { all_purpose_item }
      { arithmetic_model_qualifier }
      arithmetic_model_body
      { arithmetic_model_datarange }
   }
|  arithmetic_model_identifier [ name_identifier ] {
      { all_purpose_item }
      [ violation ]
      { arithmetic_model_qualifier }
      full_arithmetic_submodels
   }

full_arithmetic_submodels ::=
   full_arithmetic_submodel { full_arithmetic_submodel }

full_arithmetic_submodel ::=
   full_arithmetic_leaf_submodel
|  arithmetic_submodel_identifier [ name_identifier ] {
      { all_purpose_item }
      full_arithmetic_leaf_submodels
   }

full_arithmetic_leaf_submodels ::=
   full_arithmetic_leaf_submodel { full_arithmetic_leaf_submodel }

full_arithmetic_leaf_submodel ::=
   trivial_arithmetic_leaf_submodel
|  non_trivial_arithmetic_leaf_submodel

trivial_arithmetic_leaf_submodel ::=
   arithmetic_submodel_identifier [ name_identifier ] = value ;
|  arithmetic_submodel_identifier [ name_identifier ] = value {
      { all_purpose_item }
   }

non_trivial_arithmetic_leaf_submodel ::=
   arithmetic_submodel_identifier [ name_identifier ] {
      { all_purpose_item }
      arithmetic_model_body
```

```
      { arithmetic_model_datarange }
   }
```

Auxiliary definitions for arithmetic model. Semantic restrictions apply. (Note: the new grammar allows non-ambiguous distinction between usage of MIN/TYP/MAX/ DEFAULT either as `arithmetic_leaf_submodel` or as `single_value_annotation`.)

```
arithmetic_model_qualifier ::=
   general_arithmetic_model_qualifier
|  connected_arithmetic_model_qualifier
|  analog_arithmetic_model_qualifier
|  timing_arithmetic_model_qualifier
|  layout_arithmetic_model_qualifier

general_arithmetic_model_qualifier ::=
   UNIT_single_value_annotation
|  CALCULATION_single_value_annotation
|  INTERPOLATION_single_value_annotation

connected_arithmetic_model_qualifier ::=
   PIN_one_level_annotation
|  NODE_one_level_annotation

analog_arithmetic_model_qualifier ::=
   analog_MEASUREMENT_single_value_annotation
|  COMPONENT_single_value_annotation
|  TIME_arithmetic_model
|  FREQUENCY_arithmetic_model

timing_arithmetic_model_qualifier ::=
   EDGE_NUMBER_single_value_annotation
|  violation
|  from
|  to

layout_arithmetic_model_qualifier ::=
   distance_MEASUREMENT_single_value_annotation
|  BETWEEN_multi_value_annotation
|  REFERENCE_single_value_annotation
|  ANTENNA_one_level_annotation
|  PATTERN_single_value_annotation
|  VIA_single_value_annotation

arithmetic_model_datarange ::=
   MIN_single_value_annotation
|  TYP_single_value_annotation
|  MAX_single_value_annotation
|  DEFAULT_single_value_annotation

arithmetic_model_body ::=
   [ header ] table [ equation ]
|  [ header ] equation [ table ]

equation ::=
   EQUATION { arithmetic_expression }
```

```
from ::=
   FROM {
      [ PIN_single_value_annotation ]
      [ EDGE_NUMBER_single_value_annotation ]
      [ THRESHOLD_arithmetic_model ]
   }

to ::=
   TO {
      [ PIN_single_value_annotation ]
      [ EDGE_NUMBER_single_value_annotation ]
      [ THRESHOLD_arithmetic_model ]
   }
```

Auxiliary definitions for arithmetic model, also applicable elsewhere (separate chapter?).

VIOLATION is also applicable for ILLEGAL

```
violation ::=
   VIOLATION {
      [ MESSAGE_TYPE_single_value_annotation ]
      [ MESSAGE_single_value_annotation ]
      [ behavior ]
   }
```

TABLE and HEADER are also applicable for CONNECTIVITY.

```
table ::=
   TABLE { values }

header ::=
   HEADER { arithmetic_model_identifiers }
   HEADER { header_arithmetic_models }
```

Arithmetic model in context of HEADER (Note: no submodels allowed).

```
header_arithmetic_models ::=
   header_arithmetic_model { header_arithmetic_model }

header_arithmetic_model ::=
   arithmetic_model_identifier [ name_identifier ] {
      { all_purpose_item }
      { arithmetic_model_qualifier }
      { arithmetic_model_body }
      { arithmetic_model_datarange }
   }
```

Container of arithmetic model (Note: LIMIT is special).

```
arithmetic_model_containers ::=
   arithmetic_model_container { arithmetic_model_container }

arithmetic_model_container ::=
   limit
|  arithmetic_model_container_identifier { arithmetic_models }
```

```
limit ::=
   LIMIT { limit_arithmetic_models }
```

Arithmetic model in context of LIMIT (Note: must contain leaf submodels MIN and/or MAX).

```
limit_arithmetic_models ::=
   limit_arithmetic_model { limit_arithmetic_model }

limit_arithmetic_model ::=
   arithmetic_model_identifier [ name_identifier ] {
      { all_purpose_item }
      [ violation ]
      { arithmetic_model_qualifier }
      limit_arithmetic_submodels
   }

limit_arithmetic_submodels ::=
   limit_arithmetic_submodel { limit_arithmetic_submodel }

limit_arithmetic_submodel ::=
   limit_leaf_arithmetic_submodel
|  arithmetic_submodel_identifier [ name_identifier ]  {
      { all_purpose_item }
      [ violation ]
      limit_arithmetic_leaf_submodels
   }

limit_arithmetic_leaf_submodels ::=
   limit_arithmetic_leaf_submodel { limit_arithmetic_leaf_submodel }

limit_arithmetic_leaf_submodel ::=
   min_or_max = number ;
|  min_or_max {
      { all_purpose_item }
      [ violation ]
      [ arithmetic_model_body ]
   }

min_or_max ::=
   MIN
|  MAX
```

33.0 Miscellaneous Grammar enhancements

relation to ALF 2.0 3.2, 11.x

relation to IEEE P1603 6.x, Annex A

History initial draft Oct. 7, 2001 by Wolfgang
to be reviewed Nov. 12
to be reviewed Apr. 16, 2002

Status open

33.1 Motivation

The grammar serves not only the purpose of defining syntax, but also terminology. A parser does not care what terminology is used in grammar. However, if the grammar is written in a meaningful and concise way for human understanding, the terminology introduced therein can be used throughout the document for semantic explanation purpose. Since human understanding is always subjective, it may take some iterations, before the most meaningful and concise terminology is found.

33.2 Boolean_value literal

Current definition of pin_value in IEEE P1603, chapter 7.2.3:

```
pin_value ::=
   pin_variable
|  bit_literal
|  based_literal
|  unsigned
```

Issue: pin_value is referred to in IEEE 1603 chapter 6.6.1, which defines lexical rules. However, pin_value is not a lexical token. The following change provides a remedy:

```
pin_value ::=
   pin_variable
|  boolean_value

boolean_value ::=
   bit_literal
|  based_literal
|  unsigned
```

Instead of referring to pin_value in 6.6.1, refer to boolean_value. All the items in boolean_value are lexical tokens.

33.3 PULL statement

In ALF 2.0, chapter 6.4.17, PULL is defined as annotation. Chapter A.15.7 suggests to provide VOLTAGE and RESISTANCE annotation inside PULL statement. This would make PULL technically a two_level_annotation. However, RESISTANCE and VOLTAGE are arithmetic models rather than annotations. Therefore, the grammar for the PULL statement should be reformulated as follows:

```
pull ::=
   PULL = pull_value_identifier ;
|  PULL = pull_value_identifier { pull_items }
|  pull_template_instantiation

pull_items ::= pull_item { pull_item }

pull_item ::=
   voltage_arithmetic_model
|  resistance_arithmetic_model
```

Since PULL is used inside PIN, redefine pin_item (IEEE 1603, chapter 9.3.1) as follows:

```
pin_item ::=
   all_purpose_item
|  range
|  port
|  pull
|  pin_instantiation
```

Note:

```
pull_value_identifier ::=
   up
|  down
|  both
|  none
```

The *pull_value*_identifier eventually requires specification of both pull-up and pull-down resistance and voltage. Arithmetic submodels HIGH and LOW can be used for that purpose.

Example:

```
RESISTANCE { UNIT = 1ohm; }
VOLTAGE { UNIT = 1volt; }
PIN my_pin {
   PULL = both {
      RESISTANCE { HIGH = 500; LOW = 1000; }
      VOLTAGE { HIGH = 5; LOW = -5; }
   }
}
```

This pin features a pull up resistance of 500 ohm to be connected to 5 volt and a pull down resistance of 1000 ohm to be connected to -5 volt.

33.4 Annotation container

The annotation_container statement (see ALF 2.0, chapter 11.7) has been omitted in the new formulation of the grammar. Technically, annotation_container can be interpreted as a special case of two_level_annotation, but it may be advantageous to reintroduce annotation_container, because two_level_annotation features a value, whereas annotation_container does not. This distinction makes the data model more precise.

```
annotation_container ::=
   one_level_annotation_container
|  two_level_annotation_container
|  multi_level_annotation_container

one_level_annotation_container ::=
   annotation_container_identifier { one_level_annotations }

two_level_annotation_container ::=
   annotation_container_identifier { two_level_annotations }

multi_level_annotation_container ::=
   annotation_container_identifier { multi_level_annotations }
```

To do: identify all statements in the grammar which are actually annotation_container.

34.0 New item

relation to ALF 2.0	reference to ALF 2.0 chapter
relation to IEEE P1603	reference to IEEE P1603 chapter
History	date of initial draft, date of revisions
Status	open or closed, accepted or rejected

34.1 Motivation

Explain reason for new feature

34.2 Proposal

Describe new feature