Section 10

Physical Modeling

10.1 Overview

The following table summarizes the ALF statements for physical modeling.

Table 10-1 Statements in ALF describing physical objects

Statement	Scope	Comment
LAYER	LIBRARY, SUBLIBRARY	Description of a plane provided for physical objects consisting of electrically conducting material
VIA	LIBRARY, SUBLIBRARY	Description of a physical object for electrical connection between layers
SITE	LIBRARY, SUBLIBRARY	Placement grid for a class of physically placeable objects
BLOCKAGE	CELL	Physical object on a layer, forming an obstruction against placing or routing other objects
PORT	PIN	Physical object on a layer, providing electrical connections to a pin
PATTERN	VIA, RULE, BLOCKAGE, PORT	Physical object on a layer, described for the purpose of defining relationships with other physical objects
RULE	LIBRARY, SUBLIBRARY, CELL, PIN	Set of rules defining calculatable relationships between physical objects
ANTENNA	LIBRARY, SUBLIBRARY, CELL	Set of rules defining restrictions for physical size of electrically connected objects for the purpose of manufacturing
ARTWORK	VIA, CELL	Reference to an imported object from GDS2
ARRAY	LIBRARY, SUBLIBRARY	Description of a regular grid for placement, global and detailed routing
geometric model	PATTERN	Description of the geometric form of a physical object
REPEAT	physical object	Algorithm to replicate a physical object in a regular way
SHIFT	physical object	Specification to shift a physical object in x/y direction
FLIP	physical object	Specification to flip a physical object around an axis
ROTATE	physical object	Specification to rotate a physical object around an axis
BETWEEN	CONNECTIVITY, DISTANCE	Reference to objects with a relation to each other

10.2 Arithmetic models in the context of layout

The following table shows keywords for arithmetic models in the context of layout.

Table 10-2 Arithmetic models for layout data

Keyword	Value type	Base Units	Default Units	Description
`SIZE`	non-negative number	N/A	1	abstract, unitless measurement for the size of a physical object
`AREA`	non-negative number	Square Meter	p (pico)	area in square microns (pico = micro^2)
`DISTANCE`	number	Meter	u (micro)	distance between two points in microns
`HEIGHT`	non-negative number	Meter	u (micro)	y- dimension of a placeable object (e.g. cell, block) z- dimension of a routeable object (e.g. pattern on routing layer), representing the absolute height above substrate
`LENGTH`	non-negative number	Meter	u (micro)	x-, or y- dimension of a routeable object (e.g. pattern on routing layer) measured in routing direction
`WIDTH`	non-negative number	Meter	u (micro)	x-dimension of a placeable object (e.g. cell, block) x- or y- dimension of a routeable object (e.g. pattern on routing layer) measured in orthogonal direction to the route
`PERIMETER`	non-negative number	Meter	u (micro)	circumference of a physical object
`THICKNESS`	non-negative number	Meter	u (micro)	z- dimension of a manufactuable physical object, representing the distance between the bottom of the object above and the top of the object below
`OVERHANG`	non-negative number	Meter	u (micro)	distance between the edges of two overlapping physical objects
`EXTENSION`	non-negative number	Meter	u (micro)	distance between the center and the outer edge of a physical object

The following tables summarize the semantic meanings of arithmetic model keywords in the context of layout.

Table 10-3 Semantic meaning of SIZE

context	meaning
CELL	abstract measure for size of the cell, cost function for design implementation
WIRE	- as a model (TABLE or EQUATION): abstract measure for the size of the wire itself - as argument of a model (HEADER): abstract measure for size of the block for which the wireload model applies, can be calculated by combining the size of all cells and all wires in the block
ANTENNA	abstract measure for size of the antenna for which the antenna rule applies

Table 10-4 Semantic meaning of WIDTH

context	meaning
CELL, SITE	horizontal distance between cell or site boundaries, respectively
WIRE	- as argument of a model (HEADER): horizontal distance between block boundaries for which wireload model applies
LAYER, ANTENNA	width of a wire, orthogonal to routing direction

Table 10-5 Semantic meaning of HEIGHT

context	meaning
CELL, SITE	vertical distance between cell or site boundaries, respectively
WIRE	- as argument of a model (HEADER): vertical distance between block boundaries for which wireload model applies
LAYER	distance from top of ground plane to bottom of wire

Table 10-6 Semantic meaning of LENGTH

context	meaning
WIRE	estimated routing length of a wire in a wireload model
LAYER, ANTENNA	actual routing length of a wire in layout

Table 10-7 Semantic meaning of AREA

context	meaning
CELL	physical area of the cell, product of width and height of a rectangular cell
WIRE	- as a model (TABLE or EQUATION): physical area of the wire itself - as argument of a model (HEADER): physical area of the block for which wireload model applies, product of width and height of rectangular block
LAYER, VIA, ANTENNA	physical area of a placeable or routable object, measured in the x-y plane

Table 10-8 Semantic meaning of PERIMETER

context	meaning
CELL	perimeter of the cell, twice the sum of height and width for rectangular cell
WIRE	- as a model (TABLE or EQUATION): perimeter the wire itself - as argument of a model (HEADER): perimeter of the block for which wireload model applies, twice the sum of height and width for rectangular block
LAYER, VIA, ANTENNA	perimeter of a placeable or routable object, measured in the x-y plane

Table 10-9 Semantic meaning of DISTANCE

context	meaning
RULE	distance between objects for which the rule applies

Table 10-10 Semantic meaning of THICKNESS

context	meaning
LAYER, ANTENNA	distance between top and bottom of a physical object, orthogonal to the x-y plane

Table 10-11 Semantic meaning of OVERHANG

context	meaning
RULE	distance between the outer border of an object and the outer border of another object inside the first one

Table 10-12 Semantic meaning of EXTENSION

context	meaning
LAYER, VIA, RULE, geometric model	distance between the border of the original object and the border of the same object after enlargement

10.3 Statements for geometric transformation

Status: statements individually reviewed Dec. 7. SHIFT, ROTATE, FLIP, REPEAT are now regrouped in one chapter.

10.3.1 SHIFT statement

The SHIFT statement defines the horizontal and vertical offset measured between the coordinates of the geometric model and the actual placement of the object. Eventually, a layout tool may only support integer numbers.

```
shift_annotation_container ::=
   SHIFT { horizontal_or_vertical_annotations }

horizontal_or_vertical_annotations ::=
   horizontal_annotation
|  vertical_annotation
|  horizontal_annotation vertical_annotation

horizontal_annotation ::= HORIZONTAL = number ;

vertical_annotation ::= VERTICAL = number ;
```

If only one annotation is given, the default value for the other one is 0. If the SHIFT statement is not given, both values default to 0.

10.3.2 ROTATE statement

The `rotate_annotation` statement defines the angle of rotation in degrees measured between the orientation of the object described by the coordinates of the geometric model and the actual placement of the object in mathematical positive sense. Eventually, a layout tool may only support angles which are multiple of 90 degrees. Default is 0.

```
rotate_annotation ::=
   ROTATE = number ;
```

The object shall rotate around its origin.

10.3.3 FLIP statement

The `flip_annotation` specifies a transformation of the specified coordinates by flipping the object around an axis specified by a number between 0 and 90. The number indicates the flipping direction. The axis is orthogonal to the flipping direction. The axis shall go through the origin of the object.

```
flip_annotation ::=
   FLIP = number ;
```

Example:

FLIP = 0 means flip in horizontal direction, axis is vertical.
FLIP = 90 means flip in vertical direction, axis is horizontal.

10.3.4 REPEAT statement

The REPEAT statement shall be defined as follows:

```
repeat ::=
   REPEAT [ = unsigned ] {
         shift_annotation_container
         [ repeat ]
   }
```

The purpose of the REPEAT statement is to describe the replication of a physical object in a regular way, for example SITE (see chapter 3.8). The REPEAT statement may also appear within a `geometric_model`.

The `unsigned` number defines the total number of replications. The number 1 means, the object appears just once. If this number is not given, the REPEAT statement defines a rule for an arbitrary number of replications.

REPEAT statements can also be nested.

Examples:

The following example replicates an object 3 times along the horizontal axis in a distance of 7 units.

```
REPEAT = 3 {
   SHIFT { HORIZONTAL = 7; }
}
```

The following example replicates an object 5 times along a 45-degree axis.

```
REPEAT = 5 {
   SHIFT { HORIZONTAL = 4; VERTICAL = 4; }
}
```

The following example replicates an object 2 times along the horizontal axis and 4 times along the vertical axis.

```
REPEAT = 2 {
   SHIFT { HORIZONTAL = 5; }
   REPEAT = 4 {
         SHIFT { VERTICAL = 6; }
   }
}
```

Note: The order of nested REPEAT statements does not matter. The following example gives the same result as the previous example.

```
REPEAT = 4 {
   SHIFT { VERTICAL = 6; }
   REPEAT = 2 {
         SHIFT { HORIZONTAL = 5; }
   }
}
```

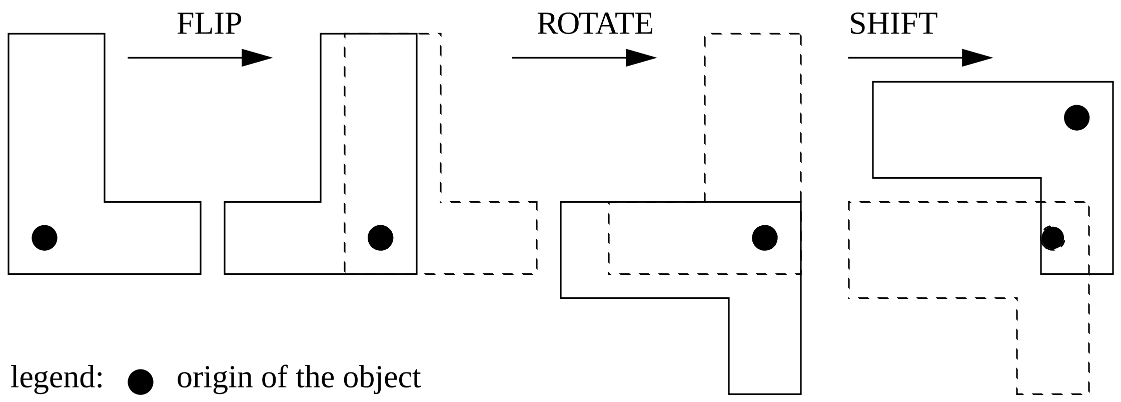
10.3.5 Summary of geometric transformations

```
geometric_transformations ::=
   geometric_transformation { geometric_transformation }

geometric_transformation ::=
   shift_annotation_container
|  rotate_annotation
|  flip_annotation
|  repeat
```

Rules and restrictions:

- A physical object may contain a `geometric_transformation` statement of any kind, but no more than one of a specific kind.
- The `geometric_transformation` statements shall apply to all `geometric_models` within the context of the object.
- The `geometric_transformation` statements shall refer to the origin of the object, i.e., the point with coordinates { 0 0 }. Therefore the result of a combined transformation will be independent of the order in which each individual transformation is applied.

FIGURE 4. Illustration of FLIP, ROTATE, SHIFT



10.4 ARTWORK Statement

The ARTWORK statement shall be defined as follows:

```
artwork ::=
   ARTWORK = artwork_identifier {
         [ shift_annotation_container ]
         [ flip_annotation ]
         [ rotate_annotation ]
         { pin_assignments }
   }
```

The ARTWORK statement creates a reference between the cell in the library and the original cell imported from a physical layout database (e.g. GDS2).

The *shift*_annotation_container statement (see previous section) defines the (x,y) offset measured between the origin of the original cell and the origin of the cell in this library. Eventually, a layout tool may only support integer numbers.

The *flip*_annotation statement (see previous section) defines the direction of a flip operation applied to the original cell before transformation into the model in this library.

The *rotate*_annotation statement (see previous section) defines the angle of rotation in degrees measured between the orientation of the original cell and the orientation of the cell in this library in mathematical positive sense. Eventually, a layout tool may only support angles which are multiple of 90 degrees.

The imported cell may have pins with different names. The LHS of the pin_assignments describes the pinnames of the original cell, the RHS describes the pinnames of the cell in this library. Syntax for pin_assignments see ALF1.1, chapter 3.4.3.

Example:

```
CELL my_cell {
   PIN A { /* fill in pin items */ }
   PIN Z { /* fill in pin items */ }
   ARTWORK = \GDS2$!@#$ {
         SHIFT { HORIZONTAL = 0; VERTICAL = 0; }
         ROTATE = 0;
         \GDS2$!@#$A = A;
         \GDS2$!@#$B = B;
   }
}
```

10.5 LAYER Statement

10.5.1 Definition

The LAYER statement shall be defined as follows:

```
layer ::= LAYER identifier { layer_items }

layer_items ::= layer_item { layer_item }

layer_item ::=
   all_purpose_item
|  arithmetic_model
|  arithmetic_model_container
```

The syntax and semantics of `all_purpose_item`, `arithmetic_model_container` and `arithmetic_model` are already defined in ALF1.1.

Specific items applicable for LAYER are listed in the following table.

Table 10-13 Items for LAYER description

item	applies for layer	usable ALF statement	comment
purpose	all	PURPOSE = <identifier> ;	see this doc., chapter 3.4
property	routing, cut, master	PROPERTY { ... }	see ALF 1.1, chapter 3.1.2.7
current density limit	routing, cut	LIMIT { CURRENT { ... MAX { ... } }	see ALF 1.1, chapters 3.6.7.1, 3.6.8.2, 3.6.9.1, 3.6.10.5 and example
resistance	routing, cut	RESISTANCE { ... }	see ALF 1.1, chapter 3.6.7.1 and example
capacitance	routing	CAPACITANCE {... }	see ALF 1.1, chapter 3.6.7.1 and example
default width or minimum width	routing	WIDTH { DEFAULT = <number>; }	see ALF 1.1, chapters 3.6.7.1, 3.6.10.1 and example

Table 10-13 Items for LAYER description

item	applies for layer	usable ALF statement	comment
manufacturing tolerance for width	routing	WIDTH { MIN = <number>; TYP = <number>; MAX = <number>; }	see ALF 1.1, chapters 3.6.7.1, 3.6.10.1 and example
default wire extension	routing	EXTENSION { DEFAULT = <number>; }	see this doc., chapter 3.0 and example
height	routing, cut, master	HEIGHT = <number>;	see this doc., chapter 3.0
thickness	routing, cut, master	THICKNESS = <number>;	see this doc., chapter 3.0
prefered routing direction	routing	PREFERENCE	see this doc., chapter 3.4

Note: Rules involving relationships between objects within one or several layers will be described in the RULE statement (chapter 3.6).

10.5.2 PURPOSE annotation

The purpose of each layer must be identified using the PURPOSE annotation.

```
layer_purpose_assignment ::=
   PURPOSE = layer_purpose_identifier ;

layer_purpose_identifier ::=
   routing
|  cut
|  substrate
|  dielectric
|  abstract
```

- routing: layer provides electrical connections within one plane
- cut: layer provides electrical connections between planes
- substrate: layer(s) at the bottom
- dielectric: provides electrical isolation between planes
- abstract: not a manufacturable layer, used for description of boundaries between objects

LAYER statements must be in sequential order defined by the manufacturing process, starting bottom-up in the following sequence: One or multiple substrate layers, followed by alternating cut and routing layers, dielectric layer. Abstract layers can appear at the end of the sequence.

10.5.3 PITCH annotation

The PITCH annotation identifies the routing pitch for a layer with PURPOSE=routing.

```
pitch_annotation ::=
   PITCH = non_negative_number ;
```

The pitch is measured between the center of two adjacent parallel wires routed on the layer.

10.5.4 PREFERENCE annotation

The PREFERENCE annotation for LAYER shall have the following form:

```
routing_preference_annotation ::=
   PREFERENCE = routing_preference_identifier ;

routing_preference_identifier ::=
   horizontal
|  vertical
```

The purpose is to indicate the preferered routing direction.

10.5.5 Example

This example contains default width (syntax is `all_purpose_item`), resistance, capacitance, current limits (syntax is `arithmetic_model`) for arbitrary wires in a routing layer. Since width and thickness are arguments of the models, special wires and fat wires are also taken into account.

```
LAYER metal1 {
   PURPOSE = routing;
   PREFERENCE { HORIZONTAL = 0.75; VERTICAL = 0.25; }
   WIDTH { DEFAULT = 0.4; MIN = 0.39; TYP = 0.40; MAX = 0.41; }
   THICKNESS { DEFAULT = 0.2; MIN = 0.19; TYP = 0.20; MAX = 0.21; }
   EXTENSION { DEFAULT = 0; }
   RESISTANCE {
         HEADER { LENGTH WIDTH THICKNESS TEMPERATURE }
         EQUATION {
               0.5*(LENGTH/(WIDTH*THICKNESS))
               *(1.0+0.01*(TEMPERATURE-25))
         }
   }
   CAPACITANCE {
         HEADER { AREA PERIMETER }
         EQUATION { 0.48*AREA + 0.13*PERIMETER*THICKNESS }
   }
   LIMIT {
         CURRENT ac_limit_for_avg {
               UNIT = mAmp ;
               MEASUREMENT = average ;
               HEADER {
                     WIDTH { UNIT = uM; TABLE { 0.4 0.8 } }
                     FREQUENCY { UNIT = megHz; { 1 100 } }
                     THICKNESS { UNIT = uM; TABLE { 0.2 0.4 } }
               }
               TABLE {
                     2.0e-6 4.0e-6 1.5e-6 3.0e-6
                     4.0e-6 8.0e-6 3.0e-6 6.0e-6
               }
         }
         CURRENT ac_limit_for_rms {
               UNIT = mAmp ;
               MEASUREMENT = rms ;
               HEADER {
                     WIDTH { UNIT = uM; TABLE { 0.4 0.8 } }
                     FREQUENCY { UNIT = megHz; { 1 100 } }
                     THICKNESS { UNIT = uM; TABLE { 0.2 0.4 } }
               }
               TABLE {
                     4.0e-6 7.0e-6 4.5e-6 7.5e-6
                     8.0e-6 14.0e-6 9.0e-6 15.0e-6
               }
         }
         CURRENT ac_limit_for_peak {
               UNIT = mAmp ;
               MEASUREMENT = peak ;
               HEADER {
                     WIDTH { UNIT = uM; TABLE { 0.4 0.8 } }
                     FREQUENCY { UNIT = megHz; { 1 100 } }
                     THICKNESS { UNIT = uM; TABLE { 0.2 0.4 } }
               }
               TABLE {
                     6.0e-6 10.0e-6 5.9e-6 9.9e-6
                     12.0e-6 20.0e-6 11.8e-6 19.8e-6
               }
         }
         CURRENT dc_limit {
               UNIT = mAmp ;
               MEASUREMENT = static ;
               HEADER {
                     WIDTH { UNIT = uM; TABLE { 0.4 0.8 } }
                     THICKNESS { UNIT = uM; TABLE { 0.2 0.4 } }
               }
               TABLE { 2.0e-6 4.0e-6 4.0e-6 8.0e-6 }
         }
   }
}
```

10.6 Geometric Model Statement

10.6.1 Definition

The geometric model statement shall be defined as follows:

```
geometric_model ::=
   geometric_model_identifier [ geometric_model_name_identifier ] {
         all_purpose_items
         coordinates
   }
|  geometric_model_template_instantiation

geometric_models ::= geometric_model { geometric_model }
```

```
geometric_model_identifier ::=
   DOT
|  POLYLINE
|  RING
|  POLYGON
```

```
coordinates ::=
   COORDINATES { x_number y_number { x_number y_number } }
```

A point is a pair of `x_number` and `y_number`.

A **DOT** is 1 point.

A **POLYLINE** is defined by N>1 connected points, forming an open object.

A **RING** is defined by N>1 connected points, forming a closed object, i.e. last point is connected with first point. The object occupies the edges of the enclosed space.

A **POLYGON** is defined by N>1 connected points, forming a closed object, i.e. last point is connected with first point. The object occupies the entire enclosed space.

See this document chapter 3.3 for the definition of the `repeat` statement.

The *point_to_point*_annotation applies for **POLYLINE**, **RING**, **POLYGON**.
It specifies how the connections between points is made. Default is straight. The value straight defines a straight connection. The value rectilinear specifies a connection by moving in x-direction first and then moving in y-direction. This enables a non-redundant specification of rectilinear objects using N/2 points instead of N points.

```
point_to_point_annotation ::=
   POINT_TO_POINT = point_to_point_identifier ;
```

```
point_to_point_identifier ::=
   straight
|  rectilinear
```

Example:

```
POLYGON {
   POINT_TO_POINT = straight;
   COORDINATES { -1 5 3 5 3 8 -1 8 }
}

POLYGON {
   POINT_TO_POINT = rectilinear;
   COORDINATES { -1 5 3 8 }
}
```

Both objects describe the same rectangle.

10.6.2 Predefined geometric models using TEMPLATE

The TEMPLATE contruct (see ALF 1.1, chapter 3.1.2.6) can be used to predefine some commonly used objects.

```
TEMPLATE RECTANGLE {
   POLYGON {
         POINT_TO_POINT = rectilinear;
         COORDINATES { <left> <bottom> <right> <top> }
   }
}

TEMPLATE LINE {
   POLYLINE {
         POINT_TO_POINT = straight;
         COORDINATES { <x_start> <y_start> <x_end> <y_end> }
   }
}

TEMPLATE HORIZONTAL_LINE {
   POLYLINE {
         POINT_TO_POINT = straight;
         COORDINATES { <left> <y> <right> <y> }
   }
}

TEMPLATE VERTICAL_LINE {
   POLYLINE {
         POINT_TO_POINT = straight;
         COORDINATES { <x> <bottom> <x> <top> }
   }
}

// same rectangle as in previous example
RECTANGLE {left = -1; bottom = 5; right = 3; top = 8; }
//or
RECTANGLE {-1 5 3 8 }

// diagonals through the rectangle
LINE {x_start = -1; y_start = 5; x_end = 3; y_end = 8; }
LINE {x_start = 3; y_start = 5; x_end = -1; y_end = 8; }
//or
LINE { -1 5 3 8 }
LINE { 3 5 -1 8 }

// lines bounding the rectangle
HORIZONTAL_LINE { y = 5; left = -1; right = 3; }
HORIZONTAL_LINE { y = 8; left = -1; right = 3; }
VERTICAL_LINE { x = -1; bottom = 5; top = 8; }
VERTICAL_LINE { x = 3; bottom = 5; top = 8; }
//or
HORIZONTAL_LINE { 5 -1 3 }
HORIZONTAL_LINE { 8 -1 3 }
VERTICAL_LINE { -1 5 8 }
VERTICAL_LINE { 3 5 8 }
```

10.7 PATTERN Statement

10.7.1 Definition

The PATTERN statement shall be defined as follows:

```
pattern ::=
PATTERN [ identifier ] {
   [ all purpose_items ]
   [ geometric_models ]
   [ geometric_transformations ]
}
```

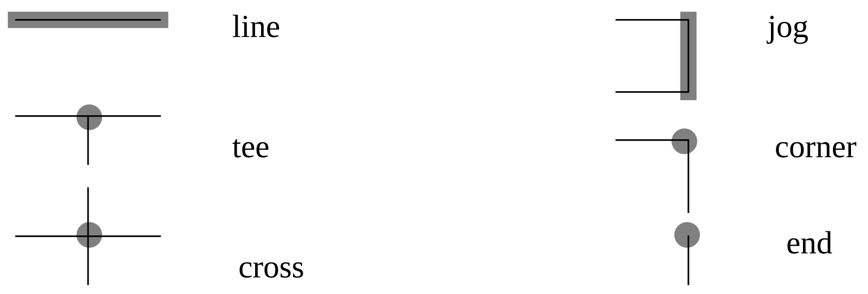
10.7.2 SHAPE annotation

The SHAPE annotation is defined as follows

```
shape_assignment ::=
   SHAPE = shape_identifier ;

shape_identifier ::=
   line
|  tee
|  cross
|  jog
|  corner
|  end
```

SHAPE applies only for PATTERN in a routing layer. Default is **line**.

See the following illustration:



Line and jog represent routing segments, which can have an individual LENGTH and WIDTH. LENGTH *between* routing segments is defined as common run length. DISTANCE *between* routing segments is measured orthogonal to routing direction.

Tee, cross, corner represent intersections between routing segments. End represents the end of a routing segment. Therefore they have points rather than lines as reference. The points can have an EXTENSION. DISTANCE between points can be measured either straight or HORIZONTAL and VERTICAL.

10.7.3 LAYER annotation

The *layer*_annotation defines the layer where the object resides. The layer must have been declared before.

```
layer_annotation ::=
   LAYER = layer_identifier ;
```

10.7.4 EXTENSION annotation

The *extension*_annotation specifies the value by which the drawn object is extended at all sides.

```
extension_annotation ::=
   EXTENSION = non_negative_number ;
```

Default value of *extension*_annotation is 0.

10.7.5 PATTERN with geometric model

A geometric_model describes the form of a physical object, it does not describe a physical object itself. The geometric_model shall be in the context of a PATTERN.

A pattern may contain geometric_model statements, geometric transformation statements (see chapter 3.3) as well as all_purpose_items (see ALF 1.1, chapter 3.4.6).

10.7.6 Example

```
PATTERN {
   LAYER = metal1;
   EXTENSION = 1;
   DOT { COORDINATES { 5 10 } }
}
```

This object is effectively a square with lower left corner (x=4,y=9) and upper right corner (x=6,y=11).

10.8 VIA statement

10.8.1 Definition

The VIA statement shall be defined as follows:

```
via ::=
VIA [ identifier ] { via_items }

via_items ::= via_item { via_item }

via_item ::=
   all_purpose_item
|  pattern
|  arithmetic_model
```

The VIA statement must contain at least 3 patterns, refering to the cut layer and two adjacent routing layers. Stacked vias may contain more than 3 patterns.

Specific `all_purpose_items` and `arithmetic_models` for VIA are listed in the following table.

TABLE 1. Items for VIA description

item	**usable ALF statement**	**comment**
property	PROPERTY	see ALF 1.1, chapter 3.1.2.7
resistance	RESISTANCE	see ALF 1.1, chapter 3.6.7.1
GDS2 reference	ARTWORK	see this document, chapter 3.10 and example
usage	USAGE	see this document, chapter 3.6 and example

10.8.2 USAGE annotation

The USAGE annotation for VIA shall have one of the following mutually exclusive values.

```
usage_annotation ::=
   USAGE = usage_identifier ;

usage_identifier ::=
   default
|  non_default
|  stack_only
```

- default: via can be used per default
- non_default: via can only beused if authorized by a RULE
- stack_only: via can only be used to build stacked vias. Bottom of stack can be default or non_default via.

10.8.3 Example

```
VIA via_with_two_contacts_in_x_direction {
   ARTWORK = GDS2_name_of_my_via {
         SHIFT { HORIZONTAL = -2; VERTICAL = -3; }
         ROTATE = 180;
   }
   PATTERN via_contacts {
         LAYER = cut_1_2 ;
         RECTANGLE { 1 1 3 3 }
         REPEAT = 2 {
               SHIFT{ HORIZONTAL = 4; }
               REPEAT = 1 {
                     SHIFT { VERTICAL = 4; }
   }     }     }
   PATTERN lower_metal {
         LAYER = metal_1 ;
         RECTANGLE { 0 0 8 4 }
   }
   PATTERN upper_metal {
         LAYER = metal_2 ;
         RECTANGLE { 0 0 8 4 }
   }
}
```

A template (see ALF 1.1, chapter 3.1.2.6) can be used to define a construction rule for a via.

```
TEMPLATE my_via_rule
   VIA <via_rule_name> {
         PATTERN via_contacts {
               LAYER = cut_1_2 ;
               RECTANGLE { 1 1 3 3 }
               REPEAT = <x_repeat> {
                     SHIFT{ HORIZONTAL = 4; }
                     REPEAT = <y_repeat> {
                           SHIFT { VERTICAL = 4; }
         }     }     }
         PATTERN lower_metal {
               LAYER = metal_1 ;
               RECTANGLE { 0 0 <x_cover> <y_cover> }
         }
         PATTERN upper_metal {
               LAYER = metal_2 ;
               RECTANGLE { 0 0 <x_cover> <y_cover> }
         }
   }
}
```

A static instance of the TEMPLATE can be used to create the same via as in the first example (except for the reference to GDS2):

```
my_via_rule {
   via_rule_name = via_with_two_contacts_in_x_direction;
   x_cover = 8;
   y_cover = 4;
   x_repeat = 2;
   y_repeat = 1;
}
```

A dynamic instance of the TEMPLATE (see ALF 1.1, chapter 3.11) can be used to create a via rule.

```
my_via_rule = dynamic {
   via_rule_name = via_with_NxM_contacts;
   x_cover = 8;
   y_cover = 4;
   x_repeat {
         HEADER { x_cover { TABLE { 4 8 12 16 } } }
         TABLE { 1 2 3 4 }
   }
   y_repeat {
         HEADER { y_cover { TABLE { 4 8 12 16 } } }
         TABLE { 1 2 3 4 }
   }
}
```

Instead of defining fixed values for the placeholders, mathematical relationships between the placeholders are defined which allow to generate a via rule for any set of values.

10.8.4 VIA reference

Certain physical objects may contain a reference to one or more vias, using the following statement.

```
   via_reference ::=
      VIA { via_instantiations }

   via_instantiations ::=
      via_instantiation { via_instantiation }

   via_instantiation ::=
      via_identifier { geometric_transformations }
```

The *via*`_identifier` must be the name of an already defined VIA.

10.9 BLOCKAGE Statement

10.9.1 Definition

The BLOCKAGE statement shall be defined as follows:

```
blockage ::=
BLOCKAGE [ identifier ] {
               [ all_purpose_items ]
               [ patterns ]
   }
```

See chapter 3.2 for applicable `all_purpose_items`.

10.9.2 Example

```
CELL my_cell {
   BLOCKAGE my_blockage {
         PATTERN p1 {
               LAYER = metal1;
               RECTANGLE { -1 5 3 8 }
               RECTANGLE { 6 12 3 8 }
         }
         PATTERN p2 {
               LAYER = metal2;
               RECTANGLE { -1 5 3 8 }
         }
   }
}
```

The BLOCKAGE consists of two rectangles covering metal1 and one rectangle covering metal2.

10.10 PORT Statement

10.10.1 Definition

A port is a collection of geometries within a pin, representing electrically equivalent points.

The PORT statement shall be defined as follows:

```
port ::=
   PORT port_identifier ;
|  PORT [ port_identifier ] {
            [ all_purpose_items ]
            [ patterns ]
            [ via_reference ]
   }
```

A numerical digit may be used as first character in `port_identifier`. In this case the number must be preceeded by the escape character (see ALF 1.1, chapter 3.2.12) in the declaration of the PORT.

The PORT statement is legal within the context of a PIN statement. For this purpose, the syntax for `pin_item` (see ALF1.1, chapter 3.4.10) shall be augmented as follows:

```
pin_item ::=
   all_purpose_item
|  arithmetic_model
|  port
```

A pin may have either no PORT statement or an arbitrary number of PORT statements with *`port_identifier`* or exactly one PORT statement without *`port_identifier`*.

10.10.2 VIA reference

A PORT may contain a reference to one or more vias, using the `via_reference` statement (see section 10.8.4).

10.10.3 CONNECTIVITY rules for PORT and PIN

Per default, all connections to a pin must be made to the same port. Also, it is not allowed to connect different ports of a pin externally. Those defaults can be overridden using connectivity rules for ports within a pin.

Also, pins of the same cell must not be shorted externally per default. This default can be overridden using connectivity rules for pins within a cell.

Example:

```
PIN A {
   PORT P1 { VIEW=physical; }
}
PIN B {
   PORT Q1 { VIEW=physical; }
   PORT Q2 { VIEW=physical; }
   PORT Q3 { VIEW=physical; }
   CONNECTIVITY {
         CONNECT_RULE = can_short;
         BETWEEN { Q1 Q3 }
   }
   CONNECTIVITY {
         CONNECT_RULE = cannot_short;
         BETWEEN { Q1 Q2 }
   }
   CONNECTIVITY {
         CONNECT_RULE = cannot_short;
         BETWEEN { Q2 Q3 }
   }
}
CONNECTIVITY {
   CONNECT_RULE = must_short;
   BETWEEN { A B }
}
```

The router can make external connections between Q1 and Q3, but not between Q1 and Q2 or between Q2 and Q3, respectively. The router must make an external connection between A.P1 and any port of B (B.Q1 or B.Q2 or B.Q3).

10.10.4 Reference of a declared PORT in a PIN annotation

In the context of timing modeling, a PORT may have the semantic meaning of a PIN. For examples, PORTs may be used as FROM and/or TO points of delay measurements. A reference by a `hierarchical_identifier` may be used:

Example:

```
CELL my_cell {
   PIN A {
         DIRECTION = input;
         PORT p1;
         PORT p2;
   }
   PIN Z {
         DIRECTION = output;
   }
   VECTOR ( 01 A -> 01 Z ) {
         DELAY {
               FROM { PIN = A.p1; }
               TO { PIN = Z; }
         }
         DELAY {
               FROM { PIN = A.p2; }
               TO { PIN = Z; }
         }
   }
}
```

10.10.5 VIEW annotation

A subset of values for the VIEW annotation inside a PIN (see ALF 1.1, chapter 3.6.3.1) shall be applicable for a PORT as well.

```
port_view_annotation ::=
   VIEW = port_view_identifier ;

port_view_identifier ::=
   physical
|  none
```

`VIEW=physical` shall qualify the PORT as a real port with the possibility to connect a routing wire to it.

`VIEW=none` shall qualify the PORT as a virtual port for modeling purpose only.

10.10.6 LAYER annotation

The `layer_annotation` may appear inside a PORT (see this document, chapter 3.2).

10.10.7 ROUTING_TYPE

A PORT may inherit the ROUTING_TYPE from its PIN, or it may have its own ROUTING_TYPE annotation.

10.11 RULE Statement

10.11.1 Definition

The RULE statement shall be defined as follows:

```
rule ::=
RULE [ identifier ] { rule_items }

rule_items ::= rule_item { rule_item }

rule_item ::=
   pattern
|  all_purpose_item
|  arithmetic_model
```

Specific `all_purpose_items` for RULE are listed in the following table.

TABLE 2. Items for RULE description

item	usable ALF statement	comment
rule is for same net or different nets	CONNECTIVITY	see ALF 1.1, chapter 3.6.10.3 and this chapter
spacing rule	LIMIT { DISTANCE ... }	see this document, chapter 3.0 and example
overhang rule	LIMIT { OVERHANG ... }	see this document, chapter 3.0 and example

Rules for spacing and overlap, respectively, shall be expressed using the LIMIT construct with DISTANCE and OVERHANG, respectively, as keyword for arithmetic models (see ALF spec. 1.1, chapter 3.6.8.2 and 3.6.9.1). The keywords HORIZONTAL and VERTICAL shall be introduced as qualifiers for arithmetic submodels (see ALF spec. 1.1, chapter 3.6.9) in order to distinguish rules for different routing directions. If these qualifiers are not used, the rule shall apply in any routing direction.

10.11.2 Width-dependent spacing

```
RULE width_and_length_dependent_spacing {
   PATTERN segment1 { LAYER = metal_1; SHAPE = line; }
   PATTERN segment2 { LAYER = metal_1; SHAPE = line; }
   CONNECTIVITY {
         CONNECT_RULE = cannot_short;
         BETWEEN { segment1 segment2 }
   }
   LIMIT {
         DISTANCE { BETWEEN { segment1 segment2 }
               MIN {
                     HEADER {
                           WIDTH w1 {
                                 PATTERN = segment1;
                                 /* TABLE, if applicable */
                           }
                           WIDTH w2 {
                                 PATTERN = segment2;
                                 /* TABLE, if applicable */
                           }
                           LENGTH common_run {
                                 BETWEEN { segment1 segment2 }
                                 /* TABLE, if applicable */
                           }
                     }
                     /* EQUATION or TABLE */
               }
               MAX { /* some technology have MAX spacing rules */ }
         }
   }
}
```

Spacing rules dependent on routing direction can be expressed as follows:

```
   LIMIT {
         DISTANCE { BETWEEN { segment1 segment2 }
               HORIZONTAL {
                     MIN { /* HEADER, EQUATION or TABLE */ }
               }
               VERTICAL {
                     MIN { /* HEADER, EQUATION or TABLE */ }
               }
         }
   }
```

10.11.3 End-of-line rule

End-of-line rules can be expressed as follows:

```
RULE lonely_via {
   PATTERN via_lower { LAYER = metal_1; SHAPE = line; }
   PATTERN via_cut   { LAYER = cut_1_2; }
   PATTERN via_upper { LAYER = metal_2; SHAPE = end;  }
   PATTERN adjacent  { LAYER = metal_2; SHAPE = line; }
   CONNECTIVITY {
         CONNECT_RULE = must_short;
         BETWEEN { via_lower via_cut via_upper }
   }
   CONNECTIVITY {
         CONNECT_RULE = cannot_short;
         BETWEEN { via_upper adjacent }
   }
   LIMIT {
         OVERHANG {
               BETWEEN { via_cut via_upper }
               MIN {
                     HEADER {
                           DISTANCE {
                                 BETWEEN { via_cut adjacent }
                                 /* TABLE, if applicable */
                           }
                     }
                     /* TABLE or EQUATION */
               }
         }
   }
}
```

Overhang dependent on routing direction can be expressed as follows:

```
   LIMIT {
         OVERHANG { BETWEEN { via_cut via_upper }
               HORIZONTAL {
                     MIN { /* HEADER, EQUATION or TABLE */ }
               }
               VERTICAL {
                     MIN { /* HEADER, EQUATION or TABLE */ }
               }
         }
   }
```

10.11.4 Redundant vias

Rules for redundant vias can be expressed as follows:

```
RULE constraint_for_redundant_vias {
   PATTERN via_lower { LAYER = metal_1; }
   PATTERN via_cut   { LAYER = cut_1_2; }
   PATTERN via_upper { LAYER = metal_2; }
   CONNECTIVITY {
         CONNECT_RULE = must_short;
         BETWEEN { via_lower via_cut via_upper }
   }
   LIMIT {
         WIDTH {
               PATTERN = via_cut;
               MIN = 3; MAX = 5;
         }
         DISTANCE {
               BETWEEN { via_cut }
               MIN = 1; MAX = 2;
         }
         OVERHANG {
               BETWEEN { via_lower via_cut }
               MIN = 2; MAX = 4;
         }
         OVERHANG {
               BETWEEN { via_upper via_cut }
               MIN = 2; MAX = 4;
         }
   }
}
```

10.11.5 Extraction rules

Extraction rules can be expressed as follows:

```
RULE parallel_lines_same_layer {
   PATTERN segment1 { LAYER = metal_1; SHAPE = line; }
   PATTERN segment2 { LAYER = metal_1; SHAPE = line; }
   CAPACITANCE {
         BETWEEN { segment1 segment2 }
         HEADER {
               DISTANCE {
                     BETWEEN { segment1 segment2 }
                     /* TABLE, if applicable */
               }
               LENGTH {
                     BETWEEN { segment1 segment2 }
                     /* TABLE, if applicable */
               }
         }
         /* EQUATION or TABLE */
   }
}
```

10.11.6 RULES within BLOCKAGE or PORT

General width-dependent spacing rules may not apply to blockages which are abstractions of smaller blockages collapsed together. The spacing rule between the constituents of the blockage and their neighboring objects should be applied instead.

For example, a blockage may consist of two parallel wires in vertical direction of width=1 and distance=1. They may be collapsed to form a blockage of width=3. Left and right of the blockage, the spacing rule should be based on the width of the constituent wires (i.e. 1) instead of the width of the blockage (i.e. 3).

Therefore, it shall be legal within a RULE statement to appear within the context of a BLOCKAGE or a PORT and to make reference to a PATTERN which has been defined within the context of the BLOCKAGE or a PORT.

Example:

```
   CELL my_cell {
      BLOCKAGE my_blockage {
         PATTERN my_pattern {
            LAYER = metal1;
            RECTANGLE { 5 0 8 10 }
         }
         RULE for_my_pattern {
            PATTERN my_metal1 { LAYER = metal1; }
            LIMIT {
               DISTANCE {
                  BETWEEN { my_metal1 my_pattern }
                  MIN = 1;
               }
            }
         }
      }
   }
```

It shall also be legal to define the spacing rule, which normally would be inside the RULE statement, directly within the context of a PATTERN using the LIMIT construct and the arithmetic model for DISTANCE. This arithmetic model shall not contain a BETWEEN statement. The spacing rule shall apply between the PATTERN and any external object on the same layer.

Example:

```
   CELL my_cell {
      BLOCKAGE my_blockage {
         PATTERN p1 {
            LAYER = metal1;
            RECTANGLE { 5 0 8 10 }
            LIMIT { DISTANCE { MIN = 1; } }
         }
      }
   }
```

10.11.7 VIA reference

A RULE may contain a reference to one or more vias, using the `via_reference` statement (see section 10.8.4).

10.12 SITE Statement

10.12.1 Definition

The SITE statement shall be defined as follows:

```
site ::=
SITE site_identifier { all_purpose_items }
```

The *`width`*`_annotation` and *`height`*`_annotation` (see this document, chapter 3.0) are mandatory.

10.12.2 ORIENTATION_CLASS and SYMMETRY_CLASS

A set of CLASS statements shall be used to define a set of legal orientations applicable to a SITE. Both the CLASS and the SITE statements shall be within the context of the same LIBRARY or SUBLIBRARY.

```
orientation_class ::=
    CLASS orientation_class_identifier {
        [ FLIP = number ; ]
        [ ROTATE = number ; ]
    }
```

Reference to predefined orientation class shall be made using the ORIENTATION_CLASS statement within a SITE and/or a CELL. ORIENTATION of a CELL means the orientation of the cell itself. ORIENTATION of a SITE means the orientation of rows that can be created using that site.

```
orientation_class_multivalue_annotation ::=
    ORIENTATION { orientation_class_identifiers }
```

The SYMMETRY_CLASS statement shall be used for a SITE to indicate symmetry between legal orientations. Multiple SYMMETRY statements shall be legal to enumerate all possible combinations in case they cannot be described within a single SYMMETRY statement.

```
symmetry_class_multivalue_annotation ::=
    SYMMETRY_CLASS { orientation_class_identifiers }
```

Legal orientation of a cell within a site shall be defined as the intersection of legal cell orientation and legal site orientation. If there is a set of common legal orientations for both cell and site without symmetry, the orientation of cell instance and site instance must match.

If there is a set of common legal orientations for both cell and site with symmetry, the cell may be placed on the side using any orientation within that set.

Case 1: no symmetry

Site has legal orientations "A" and "B". Cell has legal orientations "A" and "B". When the site is instantiated in "A" orientation, cell must be placed in "A" orientation.

Case 2: symmetry

Site has legal orientations "A" and "B" and symmetry between "A" and "B". Cell has legal orientations "A" and "B". When the site is instantiated in "A" orientation, cell can be placed in either "A" or "B" orientation.

10.12.3 Example

```
LIBRARY my_library {
   CLASS north { ROTATE = 0; }
   CLASS flip_north { ROTATE = 0; FLIP = 0; }
   CLASS south { ROTATE = 180; }
   CLASS flip_south { FLIP = 90; }

   SITE Site1 {
      ORIENTATION_CLASS { north flip_north }
   }

   SITE Site2 {
      ORIENTATION_CLASS { north flip_north south flip_south}
      SYMMETRY_CLASS { north flip_north }
      SYMMETRY_CLASS { south flip_south }
   }
   CELL Cell1 {
      SITE { Site1 Site2 }
      ORIENTATION_CLASS { north flip_north }
   }
   CELL Cell2 {
      SITE { Site2 }
      ORIENTATION_CLASS { north south }
   }
}
```

Cell1 may be placed on site1. Orientation of Site1 and Cell1 must match because of no symmetry between north and flip_north in Site1.

Cell1 may be placed on Site2, provided that site2 is instantiated in either north or flip_north orientation. Orientation of site2 and cell1 need not match because of the symmetry between north and flip_north in Site2.

Cell2 may be placed on Site2, provided that Site2 is instantiated in either north or south orientation. Orientation of Site2 and Cell2 must match because of no symmetry between north and south in Site2.

10.13 ANTENNA Statement

10.13.1 Definition

The ANTENNA statement shall be defined as follows:

```
antenna ::=
   ANTENNA [ antenna_identifier ] { antenna_items }

antenna_items ::= antenna_item { antenna_item }

antenna_item ::=
   all_purpose_item
|  arithmetic_model
|  arithmetic_model_container
```

The syntax and semantics of `all_purpose_item`, `arithmetic_model_container` and `arithmetic_model` are already defined in ALF1.1.

Specific items applicable for ANTENNA are in the following table.

Table 10-14 Items for ANTENNA description

item	**usable ALF statement**	**scope**	**comment**
maximum allowed antenna size	LIMIT { SIZE { MAX { ... } } }	LIBRARY, SUBLIBRARY CELL, PIN	see ALF 1.1, chapters 3.6.7.1, 3.6.8.2, 3.6.9.1, 3.6.10.5 and example
calculation method for antenna size	SIZE { HEADER { ... } TABLE { ...} or SIZE [id] { HEADER { ... } EQUATION { ...}	LIBRARY, SUBLIBRARY	see ALF 1.1, chapter 3.6.7.1 and example
argument values for antenna size calculation	*argument* = *value* ; or *argument* = *value* { ... }	CELL, PIN	see ALF 1.1, chapter 3.4.1 and example

The use of the keyword SIZE (see ALF 1.1, chapter 3.6.7.1) in the context of ANTENNA is proposed to represent an abstract, dimensionless model of the antenna size. It is related to the area of the net which forms the antenna, but it is not necessary a measure of area. It can be a measure of area ratio as well. However, the arguments of the calculation function for antenna SIZE must be measureable data, such as AREA, PERIMETER, LENGTH, THICKNESS, WIDTH, HEIGHT of metal segments connected to the net. The argument also need an annotation defining the applicable LAYER for the metal segments.

A process technology may have more than one antenna rule calculation method. In this case, the *`antenna_identifier`* is mandatory for each rule.

Antenna rules apply for routing and cut layers connected to polysilicon and eventually to diffusion. The CONNECT_RULE statement in conjunction with the BETWEEN statement shall be used to specify the connected layers. Connectivity shall only be checked up to the highest layer appearing in the CONNECT_RULE statement. Connectivity through higher layers shall not be taken into account, since such connectivity does not yet exist in the state of manufacturing process when the antenna effect occurs.

10.13.2 Layer-specific antenna rules

Antenna rules may be checked individually for each layer. In this case, the SIZE model contains only 2 or 3 arguments: AREA of the layer or perimeter (calculated from LENGTH and WIDTH) of the layer causing the antenna effect, area of polysilicon, eventually area of diffusion.

Example:

```
ANTENNA individual_m1 {
   LIMIT { SIZE { MAX = 1000; } }
   SIZE {
         CONNECTIVITY {
               CONNECT_RULE = must_short; BETWEEN { metal1 poly }
         }
         CONNECTIVITY {
               CONNECT_RULE = cannot_short; BETWEEN { metal1 diffusion }
         }
         HEADER {
               AREA a1 { LAYER = metal1; }
               AREA a0 { LAYER = poly; }
         }
         EQUATION { a1 / a0 }
   }
ANTENNA individual_m2 {
   LIMIT { SIZE { MAX = 1000; } }
   SIZE {
         CONNECTIVITY {
               CONNECT_RULE = must_short; BETWEEN { metal2 poly }
         }
         CONNECTIVITY {
               CONNECT_RULE = cannot_short; BETWEEN { metal2 diffusion }
         }
         HEADER {
               AREA a2 { LAYER = metal2; }
               AREA a0 { LAYER = poly; }
         }
         EQUATION { a2 / a0 }
   }
}
```

10.13.3 All-layer antenna rules

Antenna rules may also be checked globally for all layers. In that case, the SIZE model contains area or perimeter of all layers as additional arguments.

Example:

```
ANTENNA global_m2_m1 {
   LIMIT { SIZE { MAX = 2000; } }
   SIZE {
         CONNECTIVITY {
               CONNECT_RULE = must_short;
               BETWEEN { metal2 metal1 poly }
         }
         CONNECTIVITY {
               CONNECT_RULE = cannot_short;
               BETWEEN { metal2 diffusion }
         }
         HEADER {
               AREA a2 { LAYER = metal1; }
               AREA a1 { LAYER = metal1; }
               AREA a0 { LAYER = poly; }
         }
         EQUATION { (a2 + a1) / a0 }
   }
}
```

10.13.4 Cumulative antenna rules

Antenna rules may also be checked by accumulating the individual effect. In that case, the SIZE model can be represented as a nested arithmetic model, each of which contain the model of the individual effect.

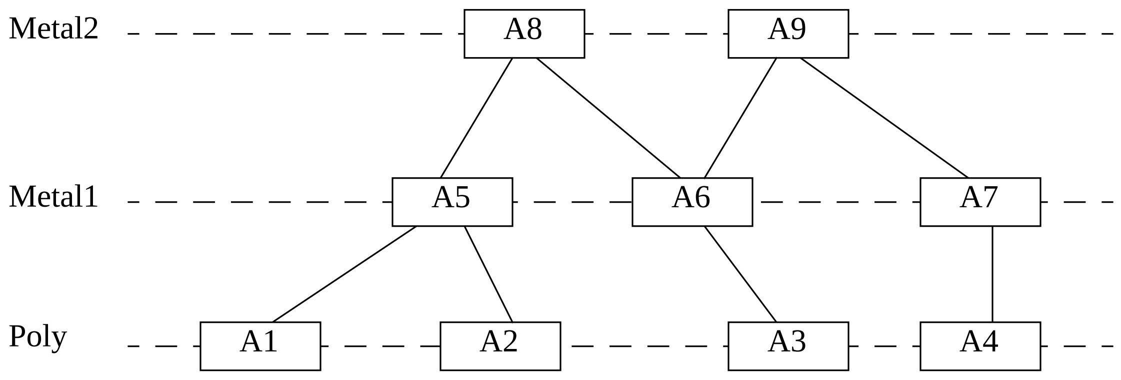
Example:

```
ANTENNA accumulate_m2_m1 {
   LIMIT { SIZE { MAX = 3000; } }
   SIZE {
         HEADER {
               SIZE ratio1 {
                     CONNECTIVITY {
                           CONNECT_RULE = must_short;
                           BETWEEN { metal1 poly }
                     }
                     CONNECTIVITY {
                           CONNECT_RULE = cannot_short;
                           BETWEEN { metal1 diffusion }
                     }
                     HEADER {
                           AREA a1 { LAYER = metal1; }
                           AREA a0 { LAYER = poly; }
                     }
                     EQUATION { a1 / a0 }
               }
               SIZE ratio2 {
                     CONNECTIVITY {
                           CONNECT_RULE = must_short;
                           BETWEEN { metal2 poly }
                     }
                     CONNECTIVITY {
                           CONNECT_RULE = cannot_short;
                           BETWEEN { metal2 diffusion }
                     }
                     HEADER {
                           AREA a2 { LAYER = metal2; }
                           AREA a0 { LAYER = poly; }
                     }
                     EQUATION { a2 / a0 }
               }
         }
         EQUATION { ratio1 + ratio2 }
   }
}
```

Note that the arguments a0 in ratio1 and ratio2 may are not the same. In ratio1, a0 represents the area of polysilicon connected to metal1 in a net. In ratio2, a0 represents the area of polysilicon connected to metal2 in a net, where the connection can be established through more than one subnet in metal1.

10.13.5 Illustration

Consider the following structure:



Checking this structure against the rules in the examples yields the following results:

```
individual_m1:
   1000 > A5 / (A1+A2)
   1000 > A6 / A3
   1000 > A7 / A4
individual_m2:
   1000 > (A8+A9) / (A1+A2+A3+A4)


global_m2_m1:
   2000 > (A8+A9+A5+A6+A7) / (A1+A2+A3+A4)


accumulate_m2_m1:
   3000 > (A8+A9) / (A1+A2+A3+A4) + A5 / (A1+A2)
   3000 > (A8+A9) / (A1+A2+A3+A4) + A6 / A3
   3000 > (A8+A9) / (A1+A2+A3+A4) + A7 / A4
```

10.14 ARRAY Statement

10.14.1 Definition

The ARRAY statement shall be defined as follows:

```
array ::=
   ARRAY identifier { all_purpose_items repeat }
```

10.14.2 PURPOSE annotation

Each array shall have a PURPOSE assignment.

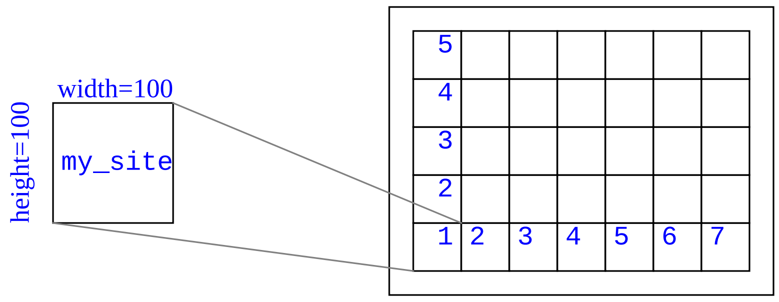
```
array_purpose_assignment ::=
   PURPOSE = array_purpose_identifier ;

array_purpose_identifier :: =
   floorplan
|  placement
|  global
|  routing
```

An array with purpose **floorplan** or **placement** shall have a reference to a SITE, a *shift*_annotation_container , *rotate*_annotation, *flip*_annotation to define the location and oritentation of the SITE in the context of the array.

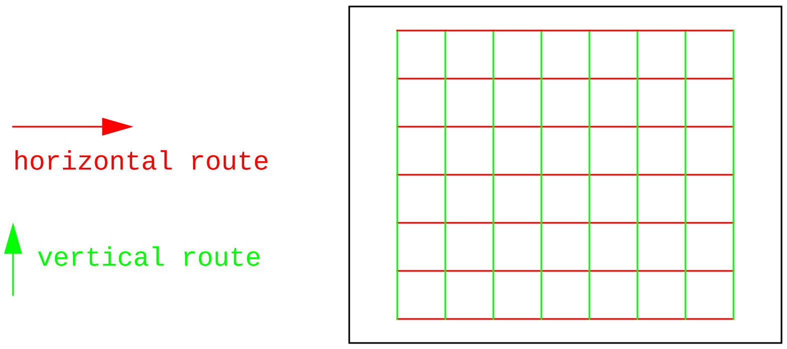
An array with purpose **routing** shall have a reference to one or more routing LAYERs and a *shift*_annotation_container to define the location of the starting point.

An array with purpose **global** shall have a *shift*_annotation_container to define the location of the starting point.

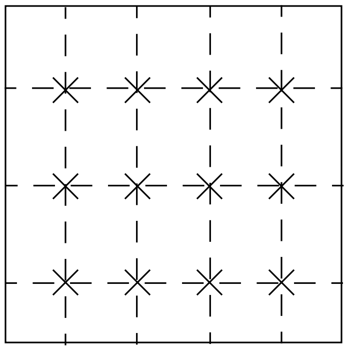
10.14.3 Examples



```
ARRAY grid_for_my_site {
   PURPOSE = placement;
   SITE = my_site;
   SHIFT { HORIZONTAL = 50; VERTICAL = 50; }
   REPEAT = 7 {
         SHIFT { HORIZONTAL = 100; }
         REPEAT = 5 {
               SHIFT { VERTICAL = 5; }
         }
   }
}
```



```
ARRAY grid_for_detailed_routing {
   PURPOSE = routing;
   LAYER { metal1 metal2 metal3 }
   SHIFT { HORIZONTAL = 100; VERTICAL = 50; }
   REPEAT = 7 {
         SHIFT { VERTICAL = 100; }
         REPEAT = 8 {
               SHIFT { HORIZONTAL = 100; }
         }
   }
}
```



```
ARRAY grid_for_global_routing {
   PURPOSE = global;
   SHIFT { HORIZONTAL = 100; VERTICAL = 100; }
   REPEAT = 3 {
         SHIFT { VERTICAL = 150; }
         REPEAT = 4 {
               SHIFT { HORIZONTAL = 100; }
         }
   }
}
```

10.15 CONNECTIVITY statement

10.15.1 Definition

A CONNECTIVITY statement shall have the following form:

```
connectivity ::=
   CONNECTIVITY [ identifier ] {
      connect_rule_annotation
      between_multi_value_assignment
   }
|  CONNECTIVITY [ identifier ] {
      connect_rule_annotation
      table_based_model
   }
```

10.15.2 CONNECT_RULE annotation

The *connect_rule annotation* may be only inside a CONNECTIVITY object. It specifies connectivity requirement.

```
CONNECT_RULE = string ;
```

which can take the following values:

Table 10-15 : CONNECT_RULE annotation

Annotation string	Description
`must_short`	electrical connection required
`can_short`	electrical connection allowed
`cannot_short`	electrical connection disallowed

It is not necessary to specify more than one rule between a given set of objects. If one rule is specified to be true, the logical value of the other rules can be implied as follows:

Table 10-16 Implications between connect rules

must_short	cannot_short	can_short
false	false	true
false	true	false
true	false	N/A

10.15.3 CONNECTIVITY modeled with BETWEEN statement

The BETWEEN statement specifies the objects for which the connectivity applies.

```
between_multi_value_assignment ::=
   BETWEEN { identifiers }
```

If the BETWEEN statement contains only one identifier, than the CONNECTIVITY shall apply between multiple instances of the same object.

Example:

```
CLASS analog_power;
CLASS analog_ground;
CLASS digital_power;
CLASS digital_ground;
CONNECTIVITY Aground { // connect all members of CLASS analog_ground
      CONNECT_RULE = must_short;
      BETWEEN { analog_ground }
}
CONNECTIVITY Dground { // connect all members of CLASS digital_ground
      CONNECT_RULE = must_short;
      BETWEEN { digital_ground }
}
CONNECTIVITY Apower { // connect all members of CLASS analog_power
      CONNECT_RULE = must_short;
      BETWEEN { analog_power }
}
CONNECTIVITY Dpower { // connect all members of CLASS digital_power
      CONNECT_RULE = must_short;
      BETWEEN { digital_power }
}
CONNECTIVITY Aground2Dground {
      CONNECT_RULE = must_short;
      BETWEEN { analog_ground digital_ground }
}
CONNECTIVITY Apower2Dpower {
      CONNECT_RULE = can_short;
      BETWEEN { analog_power digital_power }
}
CONNECTIVITY Apower2Aground {
      CONNECT_RULE = cannot_short;
      BETWEEN { analog_power analog_ground }
}
CONNECTIVITY Apower2Dground {
      CONNECT_RULE = cannot_short;
      BETWEEN { analog_power digital_ground }
}
CONNECTIVITY Dpower2Aground {
```

```
         CONNECT_RULE = cannot_short;
         BETWEEN { digital_power analog_ground }
   }
   CONNECTIVITY Dpower2Dground {
         CONNECT_RULE = cannot_short;
         BETWEEN { digital_power digital_ground }
   }
```

10.15.4 CONNECTIVITY modeled as lookup TABLE

The connectivity can also be described as a lookup table model. This description is usually more compact than the description using the BETWEEN statements.

The connectivity model may have the following arguments in the HEADER:

Table 10-17 : Arguments for Connectivity function

Argument	Value type	Description
DRIVER	string	argument of connectivity function
RECEIVER	string	argument of connectivity function

Each argument shall contain a TABLE.

The connectivity model specifies the allowed and disallowed connections amongst drivers or receivers in 1-dimensional tables, or between drivers and receivers in 2-dimensional tables.The boolean literals in the table refer to the `CONNECT_RULE` in the following way:

Table 10-18 : Boolean literals in non-interpolateable tables

Boolean literal	Description
1	`CONNECT_RULE` is true
0	`CONNECT_RULE` is false
?	`CONNECT_RULE` does not apply

Example:

```
   CLASS analog_power;
   CLASS analog_ground;
   CLASS digital_power;
   CLASS digital_ground;
   CONNECTIVITY all_must_short {
      CONNECT_RULE = must_short;
      HEADER {
         RECEIVER r1 {
            TABLE {analog_ground analog_power digital_ground digital_power}
         }
         RECEIVER r2 {
            TABLE {analog_ground analog_power digital_ground digital_power}
         }
      }
      TABLE {
         1 0 1 0
         0 1 0 0
         1 0 1 0
         0 0 0 1
      }
   /*
   The following table would apply, if the CONNECT_RULE was "cannot_short":
      TABLE {
         0 1 0 1
         1 0 1 0
         0 1 0 1
         1 0 1 0
      }
   The following table would apply, if the CONNECT_RULE was "can_short":
      TABLE {
         ? 0 ? 0
         0 ? 0 ?
         ? 0 ? 0
         0 ? 0 ?
      }
   */
   }
```

10.16 Physical annotations for CELL

10.16.1 PLACEMENT_TYPE annotation

A CELL may contain the following PLACEMENT_TYPE statement:

```
placement_type_assignment ::=
   PLACEMENT_TYPE = placement_type_identifier ;

placement_type_identifier ::=
   pad
|  core
|  ring
|  block
|  connector
```

- pad: I/O pad, to be placed in the I/O rows
- core: regular macro, to be placed in the core rows
- block: hierarchical block with regular power structure
- ring: macro with built-in power structure
- connector: macro at the end of core rows connecting with power or ground

10.16.2 Reference of a SITE by a CELL

A CELL may point to one or more legal placement SITEs.

Example:

```
CELL my_cell {
   SITE { my_site /* fill in other sites, if applicable */ }
   /* fill in contents of cell definition */
}
```

10.17 Physical annotations for PIN

10.17.1 CONNECT_CLASS annotation

```
   CONNECT_CLASS { class_identifiers }
```

annotates a declared class object for connectivity determination.

Connectivity rules involving those classes shall apply for the pin.

10.17.2 SIDE annotation

```
   SIDE = string ;
```

which can take the following values:

Table 10-19 : SIDE annotations for a PIN object

Annotation string	**Description**
`left`	pin is on the left side
`right`	pin is on the right side
`top`	pin is at the top
`bottom`	pin is at the bottom

10.17.3 ROW and COLUMN annotation

The following annotation shall be used for a pin in order to indicate the location of the pin within a placement row or column:

```
row_assignment ::=
   ROW = unsigned ;

column_assignment ::=
   COLUMN = unsigned ;
```

where *row*_assignment applies for pins with **SIDE = right | left** and *column*_assignment applies for pins with **SIDE = top | bottom**.

For bus pins, *row*_assignment and *column*_assignment shall have the form of multi_value_assignments.

```
row_multi_value_assignment ::=
   ROW { unsigned { unsigned } }
```

```
column_multi_value_assignment ::=
   COLUMN { unsigned { unsigned } }
```

10.17.4 ROUTING_TYPE annotation

A PIN may contain the following ROUTING_TYPE statement:

```
routing_type_assignment ::=
   ROUTING_TYPE = routing_type_identifier ;

routing_type_identifier ::=
   regular
|  abutment
|  ring
|  feedthrough
```

- regular: connection by regular routing
- abutment: connection by abutment, no routing
- ring: pin forms a ring around the block with connection allowed to any point of the ring
- feedthrough: both ends of the pin align and can be used for connection

10.18 Physical annotations for arithmetic models

10.18.1 BETWEEN statement within DISTANCE

The BETWEEN statement within DISTANCE shall identify the objects for which the distance measurement applies.

```
between_multi_value_assignment ::=
   BETWEEN { identifiers }
```

If the BETWEEN statement contains only one identifier, than the DISTANCE shall apply between multiple instances of the same object.

10.18.2 MEASUREMENT annotation for DISTANCE

The following statement shall specify, how the distance between objects is measured.

```
distance_measurement_assignment ::=
   MEASUREMENT = distance_measurement_identifier ;

distance_measurement_identifier ::=
   straight
|  horizontal
|  vertical
|  manhattan
```

Default is **straight**.

The mathematical definitions for distance measurements between two points with differential coordinates Δx and Δy are as follows:

- straight distance = $(\Delta x^2 + \Delta y^2)^{1/2}$
- horizontal distance = Δx
- vertical distance = Δy
- manhattan distance = Δx + Δy

10.18.3 Reference to ANTENNA rule within SIZE, AREA, PERIMETER

In hierarchical design, a PIN with physical PORTs may be abstracted. Therefore the arithmetic model for size, area, perimeter etc. relevant for certain antenna rules may be precalculated. The following statement within the arithmetic model allows to make reference to the set of antenna rules for which the arithmetic model applies.

```
antenna_reference_multi_value_assignment ::=
   ANTENNA { antenna_identifiers }
```

Example:

```
   CELL cell1 {
         PIN pin1 {
               AREA poly_area = 1.5 {
                     LAYER = poly;
                     ANTENNA { individual_m1 individual_via1 }
               }
               AREA m1_area = 1.0 {
                     LAYER = metal1;
                     ANTENNA { individual_m1 }
               }
               AREA via1_area = 0.5 {
                     LAYER = via1;
                     ANTENNA { individual_via1 }
               }
         }
   }
}
```

The area `poly_area` is used in the rules `individual_m1` and `individual_via1`.
The area `m1_area` is used in the rule`individual_m1` only.
The area `via1_area` is used in the rule `individual_via1` only.

The case with diffusion is illustrated in the following example:

```
CELL my_diode {
   CELLTYPE = special; ATTRIBUTE { DIODE }
   PIN my_diode_pin {
      AREA = 3.75 {
         LAYER = diffusion;
         ANTENNA { rule1_for_diffusion rule2_for_diffusion }
      }
   }
}
```