

**A standard for an
Advanced Library Format (ALF)
describing Integrated Circuit (IC)
technology, cells, and blocks**

**This is an unapproved draft for an IEEE standard
and subject to change**

IEEE 1596 Draft 0

August 19, 2001

Copyright© 2001, 2002, 2003 by IEEE. All rights reserved.

put in IEEE verbage

The following individuals contributed to the creation, editing, and review of this document

Wolfgang Roethig, Ph.D.

wroethig@eda.org

Official Reporter and WG Chair

Joe Daniels

chippewea@aol.org

Technical Editor

Revision history:

IEEE 1596 Draft 0

August 19, 2001

Table of Contents

1.	Introduction.....	1
1.1	Motivation.....	1
1.2	Goals	2
1.3	Target applications.....	2
1.4	Conventions	5
1.5	Contents of this standard.....	5
2.	References.....	7
3.	Definitions	9
4.	Acronyms and abbreviations	11
5.	Language construction principles	13
6.	Lexical Rules	15
6.1	Cross-reference of lexical tokens.....	15
6.2	Characters	15
6.2.1	Character set.....	15
6.2.2	Whitespace characters	16
6.2.3	Other characters.....	16
6.3	Lexical tokens	17
6.3.1	Delimiter.....	17
6.3.2	Comment	18
6.3.3	Number	18
6.3.4	Bit literals	18
6.3.5	Based literals	19
6.3.6	Edge literals	20
6.3.7	Quoted strings.....	20
6.3.8	Identifier	21
6.4	Keywords	23
6.4.1	Keywords for objects.....	23
6.4.2	Keywords for operators.....	23
6.4.3	Context-sensitive keywords	24
6.5	Rules against parser ambiguity	24
6.6	Values	24
6.6.1	Arithmetic value	24
6.6.2	String value.....	25
6.6.3	Edge value	25
6.6.4	Index value	25
7.	Auxiliary items	27
7.1	Index and related items	27
7.1.1	Index.....	27
7.1.2	Index range	27

7.2	Pin assignment and related items	27
7.2.1	Pin assignment	27
7.2.2	Pin variable	28
7.2.3	Pin value	28
7.3	Annotation and related items	28
7.3.1	Annotation	28
7.3.2	Annotation value	29
7.4	All purpose item	30
8.	Generic Objects	31
8.1	INCLUDE statement	31
8.2	ALIAS statement	31
8.3	CONSTANT statement	31
8.4	ATTRIBUTE statement	32
8.5	PROPERTY statement	32
8.6	CLASS statement	32
8.7	KEYWORD statement	33
8.8	GROUP statement	33
8.9	TEMPLATE statement	34
9.	Library-specific Objects	37
9.1	LIBRARY statement and related statements	37
9.1.1	LIBRARY statement	37
9.1.2	SUBLIBRARY statement	37
9.1.3	INFORMATION statement	38
9.2	CELL statement and related statements	38
9.2.1	CELL statement	38
9.2.2	NON_SCAN_CELL statement	39
9.2.3	Annotations and attributes for a CELL	39
9.3	PIN statement and related statements	40
9.3.1	PIN statement	40
9.3.2	RANGE statement	41
9.3.3	PIN_GROUP statement	41
9.3.4	Annotations and attributes for a PIN	42
9.4	WIRE statement and related statements	42
9.4.1	WIRE statement	43
9.4.2	NODE statement	43
9.5	VECTOR statement and related statements	44
9.5.1	VECTOR statement	44
9.5.2	ILLEGAL statement	44
9.5.3	Annotations and attributes for a VECTOR	44
9.6	LAYER statement and related statements	45
9.6.1	LAYER statement	45
9.6.2	Annotations for a LAYER	45
9.7	VIA statement and related statements	46
9.7.1	VIA statement	46
9.7.2	Annotations for a VIA	46
9.7.3	VIA reference statement	46
9.8	Statements related to physical design rules	47
9.8.1	RULE statement	47
9.8.2	ANTENNA statement	48
9.8.3	BLOCKAGE statement	48

9.8.4	PORT statement	49
9.9	Statements related to physical geometry	49
9.9.1	SITE statement	50
9.9.2	ARRAY statement.....	50
9.9.3	PATTERN statement.....	51
9.9.4	ARTWORK statement	51
9.9.5	Geometric model	52
9.9.6	Geometric transformation.....	53
9.10	Statements related to functional description	53
9.10.1	FUNCTION statement	53
9.10.2	TEST statement	54
9.10.3	BEHAVIOR statement	54
9.10.4	STRUCTURE statement	55
9.10.5	VIOLATION statement.....	56
9.10.6	STATETABLE statement	56
9.10.7	PRIMITIVE statement	57
10.	Constructs for modeling of digital behavior	59
10.1	Boolean expression language.....	59
10.2	Vector expression language	60
10.3	Control expression semantics	61
11.	Constructs for modeling of analog behavior.....	63
11.1	Arithmetic expression language.....	63
11.2	Arithmetic model and related statements.....	64
11.2.1	Arithmetic model statement	64
11.2.2	Partial arithmetic model	64
11.2.3	Non-trivial arithmetic model	65
11.2.4	Trivial arithmetic model	65
11.2.5	Assignment arithmetic model.....	66
11.2.6	Items for any arithmetic model	66
11.3	Arithmetic submodel and related statements	66
11.3.1	Arithmetic submodel statement.....	66
11.3.2	Non-trivial arithmetic submodel	66
11.3.3	Trivial arithmetic submodel	67
11.3.4	Items for any arithmetic submodel.....	67
11.4	Arithmetic body and related statements.....	67
11.4.1	Arithmetic body.....	67
11.4.2	HEADER statement	68
11.4.3	TABLE statement.....	68
11.4.4	EQUATION statement	68
11.5	Arithmetic model container	69
11.6	Statements related to arithmetic models for general purpose	69
11.6.1	MIN and MAX statement.....	69
11.6.2	TYP statement	69
11.6.3	DEFAULT statement	69
11.6.4	LIMIT statement.....	69
11.6.5	Annotations for arithmetic models for general purpose	70
11.7	Rules for evaluation of arithmetic models.....	70
11.7.1	Arithmetic model with arithmetic submodels	70
11.7.2	Arithmetic model with table arithmetic body.....	70
11.7.3	Arithmetic model with equation arithmetic body.....	70

11.8	Overview of arithmetic models.....	70
11.9	Arithmetic models for timing data	71
11.9.1	TIME statement	71
11.9.2	FREQUENCY statement.....	71
11.9.3	DELAY and RETAIN statement.....	71
11.9.4	SLEWRATE statement.....	71
11.9.5	SETUP and HOLD statement.....	72
11.9.6	NOCHANGE statement	72
11.9.7	RECOVERY and REMOVAL statement.....	72
11.9.8	SKEW statement.....	72
11.9.9	PULSEWIDTH statement	72
11.9.10	PERIOD statement	72
11.9.11	JITTER statement	73
11.9.12	THRESHOLD statement	73
11.10	Auxiliary statements related to timing data	73
11.10.1	FROM and TO statement	73
11.10.2	EARLY and LATE statement.....	73
11.10.3	Annotations for arithmetic models for timing data	74
11.11	Arithmetic models for environmental data	74
11.11.1	PROCESS and DERATE_CASE statement.....	74
11.11.2	TEMPERATURE statement.....	74
11.12	Arithmetic models for electrical data.....	75
11.12.1	CAPACITANCE statement.....	75
11.12.2	RESISTANCE statement.....	75
11.12.3	INDUCTANCE statement.....	75
11.12.4	VOLTAGE statement	75
11.12.5	CURRENT statement	75
11.12.6	POWER and ENERGY statement.....	75
11.12.7	FLUX and FLUENCE statement.....	75
11.12.8	DRIVE_STRENGTH statement.....	76
11.12.9	SWITCHING_BITS statement.....	76
11.12.10	NOISE and NOISE_MARGIN statement.....	76
11.12.11	Annotations for arithmetic models for electrical data	76
11.13	Arithmetic models for physical data	76
11.13.1	CONNECTIVITY statement	76
11.13.2	SIZE statement	77
11.13.3	AREA statement.....	77
11.13.4	WIDTH statement.....	77
11.13.5	HEIGHT statement.....	77
11.13.6	LENGTH statement.....	77
11.13.7	DISTANCE statement	77
11.13.8	OVERHANG statement	77
11.13.9	PERIMETER statement.....	77
11.13.10	EXTENSION statement.....	77
11.13.11	THICKNESS statement	77
11.13.12	Annotations for arithmetic models for physical data.....	77
11.14	Arithmetic submodels for timing and electrical data	78
11.14.1	RISE and FALL statement	78
11.14.2	HIGH and LOW statement.....	78
11.15	Arithmetic submodels for physical data.....	78
11.15.1	HORIZONTAL and VERTICAL statement.....	78
12.	Syntax Rule Summary	79

(informative)Bibliography	81
---------------------------------	----

List of Figures

Figure 1—ALF and its target applications	4
--	---

List of Tables

Table 1—Target applications and models supported by ALF	2
Table 2—Cross-reference of lexical tokens	15
Table 3—List of whitespace characters	16
Table 4—Single bit constants	19
Table 5—Special characters in quoted strings	21
Table 6—Object keywords.....	23
Table 7—Built-in arithmetic function keywords	23

IEEE Standard for an Advanced Library Format (ALF) describing Integrated Circuit (IC) technology, cells, and blocks

1. Introduction

I [**Add a lead-in OR change this to parallel an IEEE intro section**](#)

1.1 Motivation

Designing digital integrated circuits has become an increasingly complex process. More functions get integrated into a single chip, yet the cycle time of electronic products and technologies has become considerably shorter. It would be impossible to successfully design a chip of today's complexity within the time-to-market constraints without extensive use of EDA tools, which have become an integral part of the complex design flow. The efficiency of the tools and the reliability of the results for simulation, synthesis, timing and power analysis, layout and extraction rely significantly on the quality of available information about the cells in the technology library.

New challenges in the design flow, especially signal integrity, arise as the traditional tools and design flows hit their limits of capability in processing complex designs. As a result, new tools emerge, and libraries are needed in order to make them work properly. Library creation (generation) itself has become a very complex process and the choice or rejection of a particular application (tool) is often constrained or dictated by the availability of a library for that application. The library constraint can prevent designers from choosing an application program that is best suited for meeting specific design challenges. Similar considerations can inhibit the development and productization of such an application program altogether. As a result, competitiveness and innovation of the whole electronic industry can stagnate.

In order to remove these constraints, an industry-wide standard for library formats, the Advanced Library Format (ALF), is proposed. It enables the EDA industry to develop innovative products and ASIC designers to choose the best product without library format constraints. Since ASIC vendors have to support a multitude of libraries according to the preferences of their customers, a common standard library is expected to significantly reduce the library development cycle and facilitate the deployment of new technologies sooner.

1.2 Goals

The basic goals of the proposed library standard are

- *simplicity* - library creation process needs to be easy to understand and not become a cumbersome process only known by a few experts.
- *generality* - tools of any level of sophistication need to be able to retrieve necessary information from the library.
- *expandability* - this needs to be done for early adoption and future enhancement possibilities.
- *flexibility* - the choice of keeping information in one library or in separate libraries needs to be in the hand of the user not the standard.
- *efficiency* - the complexity of the design information requires the process of retrieving information from the library does not become a bottleneck. The right trade-off between compactness and verbosity needs to be established.
- *ease of implementation* - backward compatibility with existing libraries shall be provided and translation to the new library needs to be an easy task.
- *conciseness* - unambiguous description and accuracy of contents shall be detailed.
- *acceptance* - there needs to be a preference for the new standard library over existing libraries.

1.3 Target applications

The fundamental purpose of ALF is to serve as the primary database for all third-party applications of ASIC cells. In other words, it is an elaborate and formalized version of the *databook*.

In the early days, databooks provided all the information a designer needed for choosing a cell in a particular application: Logic symbols, schematics, and a truth table provided the functional specification for simple cells. For more complex blocks, the name of the cell (e.g., asynchronous ROM, synchronous 2-port RAM, or 4-bit synchronous up-down counters) and timing diagrams conveyed the functional information. The performance characteristics of each cell were provided by the loading characteristics, delay and timing constraints, and some information about DC and AC power consumption. The designers chose the cell type according to the functionality, estimated the performance of the design, and eventually re-implemented it in an optimized way as necessary to meet performance constraints.

Design automation enabled tremendous progress in efficiency, productivity, and the ability to deal with complexity, yet it did not change the fundamental requirements for ASIC design. Therefore, ALF needs to provide models with *functional* information and *performance* information, primarily including timing and power. Signal integrity characteristics, such as noise margin can also be included under performance category. Such information is typically found in any databook for analog cells. At deep sub-micron levels, digital cells behave similar to analog cells as electronic devices bound by physical laws and therefore are not infinitely robust against noise.

Table 1 shows a list of applications used in ASIC design flow and their relationship to ALF.

NOTE — ALF covers *library* data, whereas *design* data needs to be provided in other formats.

Table 1—Target applications and models supported by ALF

Application	Functional model	Performance model	Physical model
<i>Simulation</i>	Derived from ALF	N/A	N/A
<i>Synthesis</i>	Supported by ALF	Supported by ALF	Supported by ALF
<i>Design for test</i>	Supported by ALF	N/A	N/A

Table 1—Target applications and models supported by ALF (Continued)

Application	Functional model	Performance model	Physical model
<i>Design planning</i>	Supported by ALF	Supported by ALF	Supported by ALF
<i>Timing analysis</i>	N/A	Supported by ALF	N/A
<i>Power analysis</i>	N/A	Supported by ALF	N/A
<i>Signal integrity</i>	N/A	Supported by ALF	N/A
<i>Layout</i>	N/A	N/A	Supported by ALF

Historically, a functional model was virtually identical to a simulation model. A functional gate-level model was used by the proprietary simulator of the ASIC company and it was easy to lump it together with a rudimentary timing model. Timing analysis was done through dynamic functional simulation. However, with the advanced level of sophistication of both functional simulation and timing analysis, this is no longer the case. The capabilities of the functional simulators have evolved far beyond the gate-level and timing analysis has been decoupled from simulation.

RTL design planning is an emerging application type aiming to produce "virtual prototypes" of complex for system-on-chip (SOC) designs. RTL design planning is thought of as a combination of some or all of RTL floorplanning and global routing, timing budgeting, power estimation, and functional verification, as well as analysis of signal integrity, EMI, and thermal effects. The library components for RTL design planning range from simple logic gates to parameterizeable macro-functions, such as memories, logic building blocks, and cores.

From the point of view of library requirements, applications involved in RTL design planning need functional, performance, and physical data. The functional aspect of design planning includes RTL simulation and formal verification. The performance aspect covers timing and power as primary issues, while signal integrity, EMI, and thermal effects are emerging issues. The physical aspect is floorplanning. As stated previously, the functional and performance models of components can be described in ALF.

ALF also covers the requirements for physical data, including layout. This is important for the new generation of tools, where logical design merges with physical design. Also, all design steps involve optimization for timing, power, signal integrity, i.e. electrical correctness and physical correctness. EDA tools must be knowledgeable about an increasing number of design aspects. For example, a place and route tool must consider congestion as well as timing, crosstalk, electromigration, antenna rules etc. Therefore it is a logical step to combine the functional, electrical and physical models needed by such a tool in a unified library.

Figure 1 shows how ALF provides information to various design tools.

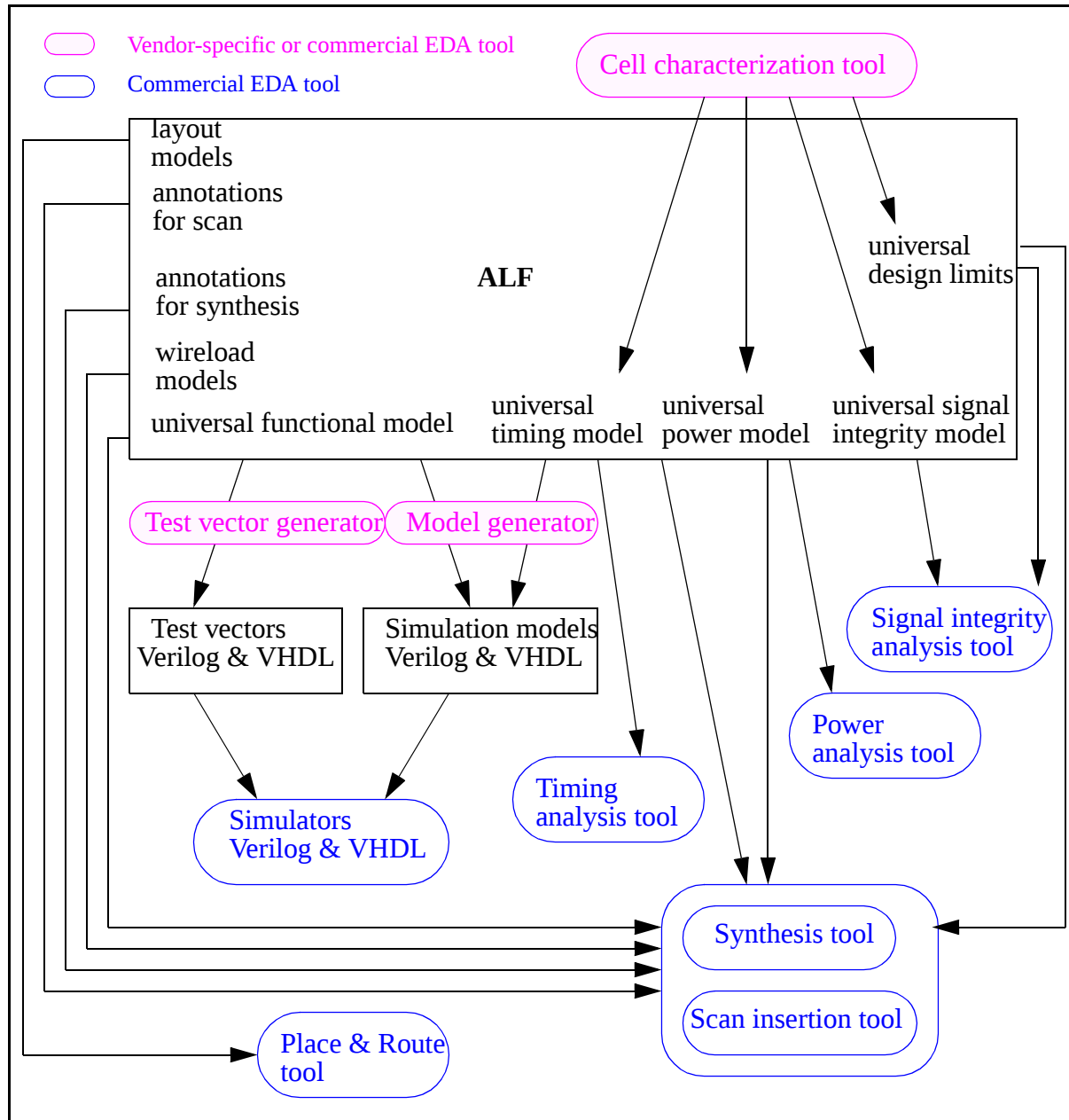


Figure 1—ALF and its target applications

The worldwide accepted standards for hardware description and simulation are VHDL and Verilog. Both languages have a wide scope of describing the design at various levels of abstraction: behavioral, functional, synthesizable RTL, and gate level. There are many ways to describe gate-level functions. The existing simulators are implemented in such a way that some constructs are more efficient for simulation run time than others. Also, how the simulation model handles timing constraints is a trade-off between efficiency and accuracy. Developing efficient simulation models which are functionally reliable (i.e., pessimistic for detecting timing constraint violation) is a major development effort for ASIC companies.

Hence, the use of a particular VHDL or Verilog simulation model as primary source of functional description of a cell is not very practical. Moreover, the existence of two simulation standards makes it difficult to pick one as a

reference with respect to the other. The purpose of a generic functional model is to serve as an absolute reference for all applications that require functional information. Applications such as synthesis, which need functional information merely for recognizing and choosing cell types, can use the generic functional model directly. For other applications, such as simulation and test, the generic functional model enables automated simulation model and test vector generation and verification, which has a tremendous benefit for the ASIC industry.

With progress of technology, the set of physical constraints under which the design functions have increased dramatically, along with the cost constraints. Therefore, the requirements for detailed characterization and analysis of those constraints, especially timing and power in deep submicron design, are now much more sophisticated. Only a subset of the increasing amount of characterization data appears in today's databooks.

ALF provides a generic format for all type of characterization data, without restriction to state-of-the art timing models. Power models are the most immediate extension and they have been the starter and primary driver for ALF.

Detailed timing and power characterization needs to take into account the *mode of operation* of the ASIC cell, which is related to the functionality. ALF introduces the concept of *vector-based modeling*, which is a generalization and a superset of today's timing and power modeling approaches. All existing timing and power analysis applications can retrieve the necessary model information from ALF.

1.4 Conventions

The syntax for description of lexical and syntax rules uses the following conventions.

****Consider using the BNF nomenclature from IEEE 1481****

```
 ::=      definition of a syntax rule
 |        alternative definition
[item]    an optional item
[item1 | item2 | ... ] optional item with alternatives
{item}    optional item that can be repeated
{item1 | item2 | ... } optional items with alternatives
                        which can be repeated
item     item in boldface font is taken verbatim
item     item in italic is for explanation purpose only
```

The syntax for explanation of semantics of expressions uses the following conventions.

```
===       left side and right side expressions are equivalent
<item>    a placeholder for an item in regular syntax
```

1.5 Contents of this standard

The organization of the remainder of this standard is

- Clause 2 (References) provides references to other applicable standards that are assumed or required for ALF.
- Clause 3 (Definitions) defines terms used throughout the different specifications contained in this standard.
- Clause 4 (Acronyms and abbreviations) defines the acronyms used in this standard.
- Clause 6 (Lexical Rules) specifies the lexical rules.
- Clause 5 (Language construction principles) defines the language construction principles.
- Clause 7 (Auxiliary items) defines syntax and semantics of auxiliary items used in this standard.
- Clause 8 (Generic Objects) defines syntax and semantics of generic objects used in this standard.

- Clause 9 (Library-specific Objects) defines syntax and semantics of library-specific objects used in this standard.
- Clause 10 (Constructs for modeling of digital behavior) defines syntax and semantics of the control expression language used in this standard
- Clause 11 (Constructs for modeling of analog behavior) defines syntax and semantics of arithmetic models used in this standard.
- Clause 12 (Syntax Rule Summary) gives a summary of syntax rules used in this standard.
- Annexes. Following Clause 12 are a series of normative and informative annexes.

2. References

****Fill in applicable references, i.e. standards on which the herein proposed standard depends.**

This standard shall be used in conjunction with the following publication. When the following standard is superseded by an approved revision, the revision shall apply.

****The following is only an example. ALF does not depend on C.**

ISO/IEC 9899:1990, Programming Languages—C.¹

[ISO 8859-1 : 1987(E)] ASCII character set

¹ISO publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembé, CH-1211, Genève 20, Switzerland/Suisse (<http://www.iso.ch/>). IEC publications are available from the Sales Department of the International Electrotechnical Commission, Case Postale 131, 3, rue de Varembé, CH-1211, Genève 20, Switzerland/Suisse (<http://www.iec.ch/>). ISO/IEC publications are also available in the United States from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA.

3. Definitions

For the purposes of this standard, the following terms and definitions apply. The *IEEE Standard Dictionary of Electrical and Electronics Terms* [B4] should be consulted for terms not defined in this standard.

[**Fill in definitions of terms which are used in the herein proposed standard.](#)

3.1 **advanced library format:** The format of any file that can be parsed according to the syntax and semantics defined within this standard.

3.2 **application, electric design automation (EDA) application:** Any software program that uses data represented in the Advanced Library Format (ALF). Examples include RTL (Register Transfer Level) synthesis tools, static timing analyzers, etc. *See also:* **advanced library format; register transfer level.**

3.3 **arc:** *See:* **timing arc.**

3.4 **argument:** A data item required for the mathematical evaluation of an arithmetic model. *See also:* **arithmetic model.**

3.5 **arithmetic model:** A representation of a library quantity that can be mathematically evaluated.

3.6 ...

3.7 **register transfer level:** A behavioral representation of a digital electronic design allowing inference of sequential and combinational logic components.

3.8 ...

3.9 **timing arc:** An abstract representation of a measurement between two points in time during operation of a library component.

3.10 ...

4. Acronyms and abbreviations

This clause lists the acronyms and abbreviations used in this standard.

ALF	advanced library format, title of the herein proposed standard
ASIC	application specific integrated circuit
AWE	asymptotic waveform evaluation
BIST	built-in self test
CAE	computer-aided engineering [the term electronic design automation (EDA) is preferred]
CAM	content-addressable memory
CLF	Common Library Format from Avant! Corporation
CPU	central processing unit
DCL	Delay Calculation Language from IEEE 1481 std
DEF	Design Exchange Format from Cadence Design Systems Inc.
DLL	delay-locked loop
DPCM	Delay and Power Calculation Module from IEEE 1481 std
DPCS	Delay and Power Calculation System from IEEE 1481 std
DSP	digital signal processor
EDA	electronic design automation
EDIF	Electronic Design Interchange Format
HDL	hardware description language
IC	integrated circuit
IP	intellectual property
ILM	Interface Logic Model from Synopsys Design Systems Inc.
LEF	Library Exchange Format from Cadence Design Systems Inc.
LIB	Library Format from Synopsys Inc.
LSSD	level-sensitive scan design
MPU	micro processor unit
OLA	Open Library Architecture from Silicon Integration Initiative Inc.
PDEF	Physical Design Exchange Format from IEEE 1481 std
PLL	Phase-locked loop
PVT	process/voltage/temperature
QTM	quick timing model
RAM	random access memory
RC	resistance times capacitance
RICE	rapid interconnect circuit evaluator
ROM	read-only memory
RSPF	Reduced Standard Parasitic Format
RTL	Register Transfer Level
SDF	Standard Delay Format from IEEE 1497 std
SLC	System Level Constraint format from Synopsys Inc.
SPEF	Standard Parasitic Exchange Format from IEEE 1481 std
SPF	Standard Parasitic Format
SPICE	Simulation Program with Integrated Circuit Emphasis
STA	Static Timing Analysis
STAMP	(STA Model Parameter ?) format from Synopsys Inc.
TCL	Tool Command Language (supported by multiple vendors)

TLF	Timing Library Format from Cadence Design Systems Inc.
VCD	Value Change Dump format (from IEEE 1364 std ?)
VHDL	VHSIC Hardware Description Language
VHSIC	very-high-speed integrated circuit
VITAL	VHDL Initiative Towards ASIC Libraries
VLSI	very-large-scale integration

5. Language construction principles

To fill in from ALF 2.0:

3.	Object model.....	15
3.1	Syntax conventions.....	15
3.7	Relationships between objects.....	23
3.9	Relations between objects.....	27
3.9.1	Keywords for referencing objects used as annotation.....	28
3.9.3	Other incremental definitions.....	29
4.	Library organization	31
4.1	Scoping rules.....	31

6. Lexical Rules

This section discusses the lexical rules.

6.1 Cross-reference of lexical tokens

****Table needs update****

Table 2 cross-references the lexical tokens used in ALF.

Table 2—Cross-reference of lexical tokens

Lexical token	Section	Lexical token	Section
alphanumeric_bit_literal	6.3.4	integer	6.3.3
any_character	6.2.3	nonescaped_identifier	6.3.8
based_literal	6.3.5	non_negative_number	6.3.3
binary_base	6.3.5	nonreserved_character	6.2.3
binary_digit	6.3.5	number	6.3.3
bit_edge_literal	6.3.6	numeric_bit_literal	6.3.4
bit_literal	6.3.4	octal_base	6.3.5
block_comment	6.3.2	octal_digit	6.3.5
comment	6.3.2	placeholder_identifier	6.3.8
decimal_base	6.3.5	quoted_string	6.3.7
delimiter	6.3.1	reserved_character	6.2.3
digit	6.2.3	sign	6.3.3
dont_care_literal	6.3.4	single_line_comment	6.3.2
edge_literal	6.3.6	symbolic_edge_literal	6.3.6
escape_character	6.2.3	unsigned	6.3.3
escaped_identifier	6.3.8	whitespace	6.2.2
hex_base	6.3.5	word_edge_literal	6.3.6
hex_digit	6.3.5		

6.2 Characters

This section defines the use of characters in ALF.

6.2.1 Character set

Each graphic character corresponds to a unique code of the ISO eight-bit coded character set [ISO 8859-1 : 1987(E)] and is represented (visually) by a graphical symbol.

6.2.2 Whitespace characters

The characters shown in Table 3 shall be considered *whitespace characters*.

Table 3—List of whitespace characters

Character	ASCII code (hex)
space	20
vertical tab	0B
horizontal tab	09
line feed (new line)	0A
carriage return	0D
form feed	0C

Comments are also considered white space (see 6.3.2).

A whitespace character shall be ignored except when it separates other lexical tokens or when it appears in a quoted string.

6.2.3 Other characters

The ASCII character set shall be divided in four categories: reserved characters, non-reserved characters, escape character, and whitespace (see 6.2.2), as shown in Syntax 1.

```
any_character ::=
    reserved_character
    | nonreserved_character
    | escape_character
    | whitespace
```

Syntax 1—ASCII character

6.2.3.1 Reserved character

The reserved characters are symbols that make up punctuation marks and operators, as shown in Syntax 2.

```
reserved_character ::=
    & | | ^ | ~ | + | - | * | / | % | ? | ! | = | < | > | : | ( | ) | [ | ] | { | } | @
    | ; | , | . | ' | "
```

Syntax 2—Reserved character

6.2.3.2 Non-reserved character

The non-reserved characters shall be used for creating identifiers and numbers, as shown in Syntax 3 — Syntax 5.

```

nonreserved_character ::=
    letter | digit | _ | $ | #

```

Syntax 3—Non-reserved character

```

letter ::=
    a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z
    | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W
    | X | Y | Z

```

Syntax 4—Letter

```

digit ::=
    0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

```

Syntax 5—Digit

6.2.3.3 Escape character

The escape character `...`, as shown in Syntax 6.

```

escape_character ::=
    \

```

Syntax 6—Escape character

ALF treats uppercase and lowercase characters as the same characters. In other words, ALF is a *case-insensitive language*.

Note — The characters \$ and # can be reserved in other languages, such as VERILOG. Therefore, if translation from ALF into VERILOG is required, these characters shall not be used for items which need to be translated, e.g., the names of cells and pins. Other languages can be case-sensitive, such as VERILOG. Therefore, if translation from ALF into VERILOG is required, the case of the name used in the declaration of the object, e.g., the name of a cell or a pin, shall always be preserved as a reference. For example, if the name of a cell is declared as MyCell1, reference to the cell can be made as MYCELL or mycell. However, it shall always be translated into VERILOG as MyCell1.

6.3 Lexical tokens

The ALF source text files shall be a stream of lexical tokens. Each lexical token is either a *delimiter*, a *comment*, a *number*, a *bit literal*, a *based literal*, an *edge literal*, a *quoted string*, or an *identifier*.

6.3.1 Delimiter

A *delimiter* is either a reserved character or a compound operator. A compound operator is composed of two or three adjacent reserved characters, as shown in Syntax 7.

```

delimiter ::=
    reserved_character
    | && | ~& | || | ~| | ~^ | == | != | ** | >= | <=
    | ?! | ?~ | ?- | ?? | ?* | *? | -> | <-> | &> | <& | >> | <<

```

Syntax 7—Delimiter

****need to refer to operators rather than enumerating them here, otherwise we may miss some of them****

Each special character in a single character delimiter list shall be a single delimiter, unless this character is used as a character in a compound operator or as a character in a quoted string.

6.3.2 Comment

ALF has two forms to introduce comments, as shown in Syntax 8.

```
comment ::=
    single_line_comment
    | block_comment
```

Syntax 8—Comment

A *single-line comment* shall start with the two characters `//` and end with a new line.

A *block comment* shall start with `/*` and end with `*/`. Comments shall not be nested. The single-line comment token `//` shall not have any special meaning in a block comment.

6.3.3 Number

****make subsections for “unsigned” and “integer” and “real”****

Constant *numbers* can be specified as integer or real, as shown in Syntax 9.

```
integer ::=
    [ sign ] unsigned
sign ::=
    + | -
unsigned ::=
    digit { _ | digit }
non_negative_number ::=
    unsigned [ . unsigned ]
    | unsigned [ . unsigned ] E [ sign ] unsigned
number ::=
    [ sign ] non_negative_number
```

Syntax 9—Integer and real numbers

An *integer* is a decimal integer constant.

6.3.4 Bit literals

Bit literals can be specified as numeric or alphabetic bit, don't care, or random, as shown in Syntax 10.

```

bit_literal ::=
    numeric_bit_literal
    | alphabetic_bit_literal
    | dont_care_literal
    | random_literal
numeric_bit_literal ::=
    0 | 1
alphabetic_bit_literal ::=
    X | Z | L | H | U | W
    | x | z | l | h | u | w
dont_care_literal ::=
    ?
random_literal ::=
    *

```

Syntax 10—Bit literal

A *bit* literal shall represent a single bit constant, as shown in Table 4.

Table 4—Single bit constants

Literal	Description
0	Value is logic zero.
1	Value is logic one.
X or x	Value is unknown.
L or l	Value is logic zero with weak drive strength.
H or h	Value is logic one with weak drive strength.
W or w	Value is unknown with weak drive strength.
Z or Z	Value is high-impedance.
U or u	Value is uninitialized.
?	Value is any of the above, yet stable.
*	Value may randomly change.

6.3.5 Based literals

A *based literal* is a constant expressed in a form that specifies the base explicitly. The base can be specified in *binary*, *octal*, *decimal* or *hexadecimal* format, as shown in Syntax 11.

The underscore (_) shall be legal anywhere in the number, except as the first character and this character is ignored. This feature can be used to break up long numbers for readability purposes. No white space shall be allowed between base and digit token in a based literal.

When an alphabetic bit literal is used as an octal digit, it shall represent three repeated bits with the same literal. When an alphabetic bit literal is used as a hex digit, it shall represent four repeated bits with the same literal.

```

based_literal ::=
    binary_base { _ | binary_digit }
    | octal_base { _ | octal_digit }
    | decimal_base { _ | digit }
    | hex_base { _ | hex_digit }
binary_base ::=
    'B' | 'b'
binary_digit ::=
    bit_literal
octal_base ::=
    'O' | 'o'
octal_digit ::=
    binary_digit | 2 | 3 | 4 | 5 | 6 | 7
decimal_base ::=
    'D' | 'd'
hex_base ::=
    'H' | 'h'
hex_digit ::=
    octal_digit | 8 | 9 | A | B | C | D | E | F | a | b | c | d | e | f

```

Syntax 11—Based literal

Example

'o2xw0u is the same as 'b010_xxx_www_000_uuu
'hLux is the same as 'bLLLL_uuuu_xxxx

6.3.6 Edge literals

An *edge literal* shall be constructed by two bit literals or two based literals, as shown in Syntax 12. It shall describe the transition of a signal from one discrete value to another. No white space shall be allowed within (between) the two literals. An underscore can be used.

```

edge_literal ::=
    bit_edge_literal
    | word_edge_literal
    | symbolic_edge_literal
bit_edge_literal ::=
    bit_literal bit_literal
word_edge_literal ::=
    based_literal based_literal
symbolic_edge_literal ::=
    ?? | ?~ | ?! | ?-

```

Syntax 12—Edge literal

6.3.7 Quoted strings

A *quoted string* shall be a sequence of zero or more characters enclosed between two quotation marks (") and contained on a single line, as shown in Syntax 13.

```

quoted_string ::=
    " { any_character } "

```

Syntax 13—Quoted string

Character *escape codes* are used inside the string literal to represent some common special characters. The characters which can follow the backslash (\) and their meanings are listed in Table 5.

Table 5—Special characters in quoted strings

Symbol	ASCII Code (octal)	Meaning
\g	007	Alert/bell.
\h	010	Backspace.
\t	011	Horizontal tab.
\n	012	New line.
\v	013	Vertical tab.
\f	014	Form feed.
\r	015	Carriage return.
\"	042	Double quotation mark.
\\	134	Backslash.
\ddd		Octal value of ASCII character (three digits).

A non-quoted string can not contain any reserved character. Therefore, use of a quoted string is necessary when referencing file names (which typically contain a dot (.) character).

6.3.8 Identifier

Identifiers are used in ALF as names of objects, reserved words, and context-sensitive keywords. An identifier shall be any sequence of letters, digits, underscore (_), and dollar sign (\$) character. Identifiers are treated in a case-insensitive way. They can be used in the definition of objects and in reference to already defined objects. A parser should preserve the case of an identifier in the definition of an object, since a downstream application could be case-sensitive.

Syntax:

```

identifiers ::= identifier { identifier }

identifier ::=
    nonescaped_identifier
    | escaped_identifier
    | placeholder_identifier
    | hierarchical_identifier

```

Purpose: Create a name for an object, create a predefined value for an object

6.3.8.1 Non-escaped identifier

If an identifier is constructed from one or more non-reserved characters, it is called a *non-escaped identifier*, as shown in Syntax 14.

```
nonescaped_identifier ::=  
    nonreserved_character { nonreserved_character }
```

Syntax 14—Non-escaped identifier

A `digit` shall not be allowed as first character of a non-escaped identifier.

6.3.8.2 Escaped identifier

A sequence of characters starting with an `escape_character` is called an *escaped identifier*. The escaped identifier legalizes the use of a `digit` as first character of an identifier and the use of `reserved_character` anywhere in an identifier. Or it can be used to prevent the misinterpretation of an identifier as a keyword. The escape character shall be followed by at least one non-white space character to form an escaped identifier. The escaped identifier shall contain all characters up to first white space character, as shown in Syntax 15.

```
escaped_identifier ::=  
    escape_character escaped_characters  
escaped_characters ::=  
    escaped_character { escaped_character }  
escaped_character ::=  
    nonreserved_character  
    | reserved_character  
    | escape_character
```

Syntax 15—Escaped identifier

6.3.8.3 Placeholder identifier

A *placeholder identifier* shall be a non-escaped identifier between the less-than character (<) and the greater-than character (>). No whitespace or delimiters are allowed between the non-escaped identifier and the placeholder characters (< and >). The placeholder identifier is used in template objects as a formal parameter, which is replaced by the actual parameter in template instantiation, as shown in Syntax 16.

```
placeholder_identifier ::=  
    < nonescaped_identifier >
```

Syntax 16—Placeholder identifier

6.3.8.4 Hierarchical identifier

A *hierarchical identifier* shall be defined as shown in Syntax 17, with no whitespace in-between the characters.

```
hierarchical_identifier ::=  
    identifier . { identifier . } identifier
```

Syntax 17—Hierarchical identifier

A dot (.) shall take precedence over an `escape_character`. To escape a dot, the `escape_character` shall be placed directly in front of it.

Examples

```
\id1.id2 //Only id1 is escaped.
```

```
id1\.id2 //Only the dot is escaped.
id1.\id2 //Only id2 is escaped.
```

6.4 Keywords

Keywords are case-insensitive non-escaped identifiers. For clarity, this document uses uppercase letters for keywords and lowercase letters elsewhere, unless otherwise mentioned.

Keywords are reserved for use as object identifiers, not for general symbols. To use an identifier that conflicts with the list of keywords, use the escape character, e.g., to declare a pin that is called PIN, use the form

```
PIN \PIN { . . }
```

A keyword can either be a *reserved keyword* (also called a *hard keyword*) or a *context-sensitive keyword* (also called a *soft keyword*). The hard keywords have fixed meanings and shall be understood by any parser of ALF. The soft keywords might be understood only by specific applications. For example, a parser for a timing analysis application can ignore objects that contain power related information described using soft keywords.

6.4.1 Keywords for objects

[**table not up to date, maybe should be omitted**](#)

The keywords shown in Table 6 are used to identify object types.

Table 6—Object keywords

ALIAS	ATTRIBUTE	BEHAVIOR	CELL
CLASS	CONSTANT	EQUATION	FUNCTION
GROUP	HEADER	INCLUDE	LIBRARY
PIN	PRIMITIVE	PROPERTY	STATETABLE
SUBLIBRARY	TABLE	TEMPLATE	VECTOR
WIRE			

6.4.2 Keywords for operators

[**table not up to date, refer to “arithmetic expression language” **](#)

The keywords shown in Table 7 are used for built-in arithmetic functions.

Table 7—Built-in arithmetic function keywords

Term	Definition
ABS	Absolute value.
EXP	Natural exponential function.

Table 7—Built-in arithmetic function keywords (Continued)

Term	Definition
LOG	Natural logarithm.
MIN	Minimum.
MAX	Maximum.

6.4.3 Context-sensitive keywords

In order to address the need of extensible modeling, ALF provides a predefined set of *public* context-sensitive keywords. Additional private context-sensitive keywords can be introduced as long as they do not have the same name as any existing public keyword.

6.5 Rules against parser ambiguity

The following rules shall apply when resolving ambiguity in parsing ALF source

- In a context where both `bit_literal` and `identifier` are legal syntax items, a `nonescaped_identifier` shall take priority over an `alphabetic_bit_literal`.
- In a context where both `bit_literal` and `number` are legal syntax items, a `number` shall take priority over a `numeric_bit_literal`.
- In a context where both `edge_literal` and `identifier` are legal syntax items, an `identifier` shall take priority over a `bit_edge_literal`.
- In a context where both `edge_literal` and `number` are legal syntax items, a `number` shall take priority over a `bit_edge_literal`.

In such contexts, a `based_literal` shall be used instead of a `bit_literal`.

6.6 Values

Note: A lexical token is semantically interpreted as a value, once its lower-level lexical components (i.e., literals) have been identified.

6.6.1 Arithmetic value

Syntax:

```

arithmetic_values ::= arithmetic_value { arithmetic_value }

arithmetic_value ::=
    number
    | identifier
    | pin_value

```

Purpose: data for calculation described in `arithmetic_model` or in `arithmetic_assignment` for `dynamic_template_instantiation`

Semantic restriction: `arithmetic_value` must resolve to a valid value for the particular `arithmetic_model`, where it is used. Some `arithmetic_models` allow only unsigned (e.g. `SWITCHING_BITS`, `FANOUT`), others allow only

non_negative_number (e.g. WIDTH, LENGTH). Non-interpolatable arithmetic_models (e.g. PROCESS, DERATE_CASE) allow only symbolic identifiers rather than numbers.

6.6.2 String value

Syntax:

```
string_value ::=
    quoted_string
    | identifier
```

Purpose: textual data

6.6.3 Edge value

Syntax:

```
edge_values ::=
    edge_value { edge_value }

edge_value ::= ( edge_literal )
```

Purpose: use edge_literal as a standalone value. For that purpose the edge_literal is enclosed by parentheses, to avoid parser ambiguity. Normally, an edge_literal would appear only within a vector_expression. In that context, the enclosing parentheses are not necessary.

6.6.4 Index value

Syntax:

```
index_value ::=
    unsigned
    | identifier
```

Index_value must resolve to unsigned, i.e., identifier must be the name of a CONSTANT with unsigned value or a placeholder in TEMPLATE that gets replaced with unsigned.

7. Auxiliary items

7.1 Index and related items

7.1.1 Index

Syntax:

```
index ::=
    [ index_range ]
    | [ index_value ]
```

7.1.2 Index range

Syntax:

```
index_range ::= index_value : index_value
```

Index_range shall define consecutive unsigned numbers, bound by index_value left and right of “:”. In the context of a PIN statement, left index_value shall be considered as MSB, right index_value shall be considered as MSB. Also used in RANGE statement and in GROUP statement.

7.2 Pin assignment and related items

7.2.1 Pin assignment

Syntax:

```
pin_assignments ::= pin_assignment { pin_assignment }

pin_assignment ::=
    pin_variable = pin_value ;
```

Purpose: Associates a pin_value with a pin_variable for the purpose of pin mapping. Used in NON_SCAN_CELL statement, in ARTWORK statement, and in STRUCTURE statement.

Semantic restrictions:

The pin_value must be compatible with the pin_variable. A scalar pin_variable can be assigned to another scalar pin_variable, or to a scalar pin_value, i.e. bit_literal or one-bit binary based_literal. A one-dimensional pin_variable or a one-dimensional slice of a two-dimensional pin_variable can be assigned to another one-dimensional pin_variable, or to another one-dimensional slice of a two-dimensional pin_variable of same bitwidth, or to a based_literal of the same bitwidth, or to a unsigned that can be converted into a binary number of the same bitwidth.

If the bitwidth of the pin_value is smaller than the bitwidth of the pin_variable, the LSBs shall be aligned. Excessive leading bits of the pin_variable shall be filled with zeros.

To be discussed: If the bitwidth of the pin_value is greater than the bitwidth of the pin_variable, the LSBs shall be aligned. Excessive leading bits of the pin_value shall be cut off.

7.2.2 Pin variable

Syntax:

```
pin_variables ::= pin_variable { pin_variable }  
  
pin_variable ::= pin_variable_identifier [ index ]
```

Purpose: A `pin_variable` shall be used to represent the information accessible through a PIN. A PIN (see section xxx) is the interface between a library component (i.e., a CELL or a PRIMITIVE) and its environment.

Semantics: A legal `pin_variable_identifier` shall make reference to a previously declared PIN, or to a previously declared PIN_GROUP, or to a previously declared NODE, or to a previously declared PORT (i.e., `pin_identifier.port_identifier`). A legal index shall be bound by the MSB and by the LSB of the `index_range` in the referred PIN.

7.2.3 Pin value

Syntax:

```
pin_values ::= pin_value { pin_value }  
  
pin_value ::=  
    pin_variable  
    | bit_literal  
    | based_literal  
    | unsigned
```

Purpose: `pin_value` defines the set of values that can be assigned to a `pin_variable`. Assigning a `pin_variable` to another `pin_variable` shall be legal.

Can be used context of NON_SCAN_CELL, STRUCTURE, primitive_instantiation as a short form of `pin_assignment`, i.e., pin mapping by order instead of pin mapping by name.

7.3 Annotation and related items

7.3.1 Annotation

Note: An annotation is an auxiliary statement within the context of a `library_specific_object`, a `library_specific_singular_object`, or an `arithmetic_model`. It serves as a qualifier of its context.

```
annotation ::=  
    one_level_annotation
```

```

    | two_level_annotation
    | multi_level_annotation

one_level_annotations ::=
    one_level_annotation { one_level_annotation }

one_level_annotation ::=
    single_value_annotation
    | multi_value_annotation

single_value_annotation ::=
    identifier = annotation_value ;

multi_value_annotation ::=
    identifier { annotation_values }

two_level_annotations ::=
    two_level_annotation { two_level_annotation }

two_level_annotation ::=
    one_level_annotation
    | identifier [ = annotation_value ]
      { one_level_annotations }

multi_level_annotations ::=
    multi_level_annotation { multi_level_annotation }

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

```

7.3.2 Annotation value

Syntax:

```

annotation_values ::= annotation_value { annotation_value }

annotation_value ::=
    index_value
    | string_value
    | edge_value
    | pin_value
    | arithmetic_value
    | boolean_expression
    | control_expression

```

Note: there is lexical overlap, but semantic distinction between the possible annotation values.

7.4 All purpose item

Syntax:

```
all_purpose_items ::=
    all_purpose_item { all_purpose_item }

all_purpose_item ::=
    include
    | alias
    | constant
    | attribute
    | property
    | class_declaration
    | keyword_declaration
    | group_declaration
    | template_declaration
    | template_instantiation
    | annotation
    | arithmetic_model
    | arithmetic_model_container
```

Purpose: Provide flexibility and generality of the ALF syntax. The ALF semantics shall define whether a particular `all_purpose_item` is legal within a specific context.

8. Generic Objects

To fill in from ALF 2.0:

3.2 Generic objects	16
---------------------------	----

8.1 INCLUDE statement

Syntax snippet:

```
include ::=  
    INCLUDE quoted_string ;
```

To fill in from ALF 2.0:

3.2.3 INCLUDE statement	17
4.2 Use of multiple files.....	32

8.2 ALIAS statement

Syntax snippet:

```
alias ::=  
    ALIAS identifier = identifier ;
```

To fill in from ALF 2.0:

3.2.2 ALIAS statement.....	17
----------------------------	----

8.3 CONSTANT statement

Syntax snippet:

```

constant ::=
    CONSTANT identifier = arithmetic_value ;

```

To fill in from ALF 2.0:

3.2.1	CONSTANT statement.....	17
-------	-------------------------	----

8.4 ATTRIBUTE statement

Syntax snippet:

```

attribute ::=
    ATTRIBUTE { identifiers }

```

To fill in from ALF 2.0:

3.2.5	ATTRIBUTE statement.....	18
-------	--------------------------	----

8.5 PROPERTY statement

Syntax snippet:

```

property ::=
    PROPERTY [ identifier ] { one_level_annotations }

```

To fill in from ALF 2.0:

3.2.7	PROPERTY statement	19
-------	--------------------------	----

8.6 CLASS statement

Syntax snippet:

```

class_declaration ::=
    CLASS identifier ;
    | CLASS identifier { all_purpose_items }

```

To fill in from ALF 2.0:

3.2.4 CLASS statement17

8.7 KEYWORD statement

Syntax sniplet:

```

keyword_declaration ::=
    KEYWORD context_sensitive_keyword =
        syntax_item_identifier ;

```

To fill in from ALF 2.0.2:

3.2.9 KEYWORD statement19

8.8 GROUP statement

Syntax sniplet:

```

group_declaration ::=
    GROUP group_identifier { annotation_values }
    | GROUP group_identifier { index_value : index_value }

```

To fill in from ALF 2.0:

3.2.8 GROUP statement19

Semantics: When the group identifier is used within an ALF statement within the scope of the GROUP declaration, that ALF statement shall be replaced by several statements, substituting the annotation values or the index values, respectively, for the group identifier. The replacing statements shall appear at the same scope as the GROUP declaration.

8.9 TEMPLATE statement

Syntax snippet:

```
template_declaration ::=
    TEMPLATE template_identifier { template_items }

template_items ::=
    template_item { template_item }

template_item ::=
    all_purpose_item
    | cell
    | library
    | node
    | pin
    | pin_group
    | primitive
    | sublibrary
    | vector
    | wire
    | antenna
    | array
    | blockage
    | layer
    | pattern
    | port
    | rule
    | site
    | via
    | function
    | non_scan_cell
    | test
    | range
    | artwork
    | from
    | to
    | illegal
    | violation
    | header
    | table
    | equation
    | arithmetic_submodel
```

```

| behavior_item
| geometric_model

template_instantiation ::=
    static_template_instantiation
| dynamic_template_instantiation

static_template_instantiation ::=
    template_identifier [ = static ] ;
| template_identifier [ = static ] { annotation_values }
| template_identifier [ = static ] { one_level_annotations }

dynamic_template_instantiation ::=
    template_identifier = dynamic
    { dynamic_template_instantiation_items }

dynamic_template_instantiation_items ::=
    dynamic_template_instantiation_item
    { dynamic_template_instantiation_item }

dynamic_template_instantiation_item ::=
    one_level_annotation
| arithmetic_model

```

To fill in from ALF 2.0:

3.2.6	TEMPLATE statement.....	18
5.6.8	Parameterizeable cells	97

9. Library-specific Objects

To fill in from ALF 2.0:

3.3 Library-specific objects	20
3.6 Library-specific singular objects	22
6. Modeling for synthesis and test	101

9.1 LIBRARY statement and related statements

9.1.1 LIBRARY statement

Syntax snippet:

```
library ::=
    LIBRARY library_identifier { library_items }
    | LIBRARY library_identifier ;
    | library_template_instantiation

library_items ::=
    library_item { library_item }

library_item ::=
    sublibrary
    | sublibrary_item
```

9.1.2 SUBLIBRARY statement

Syntax snippet:

```
library ::=
    SUBLIBRARY sublibrary_identifier { sublibrary_items }
    | SUBLIBRARY sublibrary_identifier ;
    | sublibrary_template_instantiation

sublibrary_items ::=
    sublibrary_item { sublibrary_item }

sublibrary_item ::=
    all_purpose_item
    | cell
```

```

| primitive
| wire
| layer
| via
| rule
| antenna
| array
| site

```

9.1.3 INFORMATION statement

Syntax snippet:

```

INFORMATION_two_level_annotation ::=
    INFORMATION { information_one_level_annotations }

information_one_level_annotations ::=
    information_one_level_annotation
    { information_one_level_annotation }

information_one_level_annotation ::=
    AUTHOR_one_level_annotation
    | VERSION_one_level_annotation
    | DATETIME_one_level_annotation
    | PROJECT_one_level_annotation

```

INFORMATION shall be used within LIBRARY, SUBLIBRARY, CELL, WIRE, PRIMITIVE, since these objects can be considered as standalone deliverables. Other objects, for example PIN, PORT, LAYER, VIA etc., cannot be considered as standalone deliverables.

To fill in from ALF 2.0:

3.8 INFORMATION container..... 26

9.2 CELL statement and related statements

9.2.1 CELL statement

Syntax snippet:

```

cell ::=
    CELL cell_identifier { cell_items }

```

```

| CELL cell_identifier ;
| cell_template_instantiation

cell_items ::=
    cell_item { cell_item }

cell_item ::=
    all_purpose_item
| pin
| pin_group
| primitive
| function
| non_scan_cell
| test
| vector
| wire
| blockage
| artwork

```

9.2.2 NON_SCAN_CELL statement

Syntax snippet:

```

non_scan_cell ::=
    NON_SCAN_CELL { unnamed_cell_instantiations }
| NON_SCAN_CELL = unnamed_cell_instantiation
| non_scan_cell_template_instantiation

unnamed_cell_instantiations ::=
    unnamed_cell_instantiation { unnamed_cell_instantiation }

unnamed_cell_instantiation ::=
    cell_identifier { pin_values }
| cell_identifier { pin_assignments }

```

To fill in from ALF 2.0:

6.2 NON_SCAN_CELL statement.....	108
----------------------------------	-----

9.2.3 Annotations and attributes for a CELL

Syntax snippet:

To fill in from ALF 2.0:

6.1	Annotations and attributes for a CELL	101
6.1.1	CELLTYPE annotation	101
6.1.2	ATTRIBUTE within a CELL object	101
6.1.3	SWAP_CLASS annotation.....	103
6.1.3	RESTRICT_CLASS annotation.....	103
6.1.3	Independent SWAP_CLASS and RESTRICT_CLASS.....	104
6.1.3	SWAP_CLASS with inherited RESTRICT_CLASS	105
6.1.4	SCAN_TYPE annotation.....	106
6.1.5	SCAN_USAGE annotation	106
6.1.6	BUFFERTYPE annotation	107
6.1.7	DRIVERTYPE annotation	107
6.1.8	PARALLEL_DRIVE annotation.....	107
9.16	Physical annotations for CELL	253
9.16.1	PLACEMENT_TYPE annotation	262
9.16.2	Reference of a SITE by a CELL.....	262

9.3 PIN statement and related statements

9.3.1 PIN statement

Syntax snippet:

```

pin ::=
    PIN [ [index_range ] ] pin_identifier [ [index_range ] ]
    { pin_items }
| PIN [ [index_range ] ] pin_identifier [ [index_range ] ] ;
| pin_template_instantiation

pin_item ::=
    all_purpose_item
| range

```

```

| port
| pin_instantiation

pin_items ::=
    pin_item { pin_item }

pin_instantiation ::=
    pin_variable { pin_items }

```

To fill in from ALF 2.0:

6.5	Definitions for bus pins.....	127
5.5.3	Multi-dimensional variables.....	83
6.5.2	Scalar pins inside a bus	128

9.3.2 RANGE statement

Syntax snippet:

```

range ::=
    RANGE { index_range }

```

To fill in from ALF 2.0:

6.5.1	RANGE for bus pins	127
-------	--------------------------	-----

9.3.3 PIN_GROUP statement

Syntax snippet:

```

pin_group ::=
    PIN_GROUP [ [ index_range ] ] pin_group_identifier

```

```

        { pin_group_items }
    | pin_group_template_instantiation

pin_group_items ::=
    pin_group_item { pin_group_item }

pin_group_item ::=
    all_purpose_item
    | range

```

To fill in from ALF 2.0:

6.5.3	PIN_GROUP statement.....	129
-------	--------------------------	-----

9.3.4 Annotations and attributes for a PIN

To fill in from ALF 2.0:

6.4	Annotations and attributes for a PIN.....	114
6.4.1	VIEW annotation.....	114
6.4.2	PINTYPE annotation.....	115
6.4.3	DIRECTION annotation.....	115
6.4.4	SIGNALTYPE annotation.....	116
6.4.5	ACTION annotation.....	120
6.4.6	POLARITY annotation.....	121
6.4.7	DATATYPE annotation.....	122
6.4.8	INITIAL_VALUE annotation.....	122
6.4.9	SCAN_POSITION annotation.....	122
6.4.10	STUCK annotation.....	122
6.4.11	SUPPLYTYPE.....	123
6.4.12	SIGNAL_CLASS.....	123
6.4.13	SUPPLY_CLASS.....	124
6.4.14	Driver CELL and PIN specification.....	125
6.4.15	DRIVETYPE annotation.....	125
6.4.16	SCOPE annotation.....	126
6.4.17	PULL annotation.....	126
6.4.18	ATTRIBUTE for PIN objects.....	126
6.9.2	Definitions of pin ATTRIBUTE values for memory BIST.....	138
9.17	Physical annotations for PIN.....	263
9.17.1	CONNECT_CLASS annotation.....	263
9.17.2	SIDE annotation.....	263
9.17.3	ROW and COLUMN annotation.....	263
9.17.4	ROUTING_TYPE annotation.....	264

9.4 WIRE statement and related statements

9.4.1 WIRE statement

Syntax snippet:

```
wire ::=
    WIRE wire_identifier { wire_items }
  | WIRE wire_identifier ;
  | wire_template_instantiation

wire_items ::=
    wire_item { wire_item }

wire_item ::=
    all_purpose_item
  | node
```

To fill in from ALF 2.0:

8.15 Interconnect parasitics and analysis.....	209
8.15.1 Principles of the WIRE statement	209
8.15.2 Statistical wireload models.....	210
8.15.3 Boundary parasitics	211
8.15.5 Interconnect delay and noise calculation.....	216
8.15.6 SELECT_CLASS annotation for WIRE statement.....	217

9.4.2 NODE statement

Syntax snippet:

```
node ::=
    NODE node_identifier { node_items }
  | NODE node_identifier ;
  | node_template_instantiation

node_items ::=
    node_item { node_item }

node_item ::=
    all_purpose_item
```

To fill in from ALF 2.0:

8.15.4 NODE declaration	213
-------------------------------	-----

9.5 VECTOR statement and related statements

9.5.1 VECTOR statement

Syntax snippet:

```
vector ::=
    VECTOR control_expression { vector_items }
    | VECTOR control_expression ;
    | vector_template_instantiation

vector_items ::=
    vector_item { vector_item }

vector_item ::=
    all_purpose_item
    | illegal
```

9.5.2 ILLEGAL statement

Syntax snippet:

```
illegal ::=
    ILLEGAL { illegal_items }
    | illegal_template_instantiation

illegal_items ::=
    illegal_item { illegal_item }

illegal_item ::=
    all_purpose_item
    | violation
```

To fill in from ALF 2.0:

6.7 ILLEGAL statement for VECTOR..... 135

9.5.3 Annotations and attributes for a VECTOR

To fill in from ALF 2.0:

6.6	Annotations for CLASS and VECTOR	130
6.6.1	PURPOSE annotation.....	130
6.6.2	OPERATION annotation	131
6.6.3	LABEL annotation	133
6.6.4	EXISTENCE_CONDITION annotation	133
6.6.5	EXISTENCE_CLASS annotation.....	133
6.6.6	CHARACTERIZATION_CONDITION annotation	134
6.6.7	CHARACTERIZATION_VECTOR annotation.....	134
6.6.8	CHARACTERIZATION_CLASS annotation	135
3.9.2	Incremental definitions for VECTOR	29

9.6 LAYER statement and related statements

To fill in from ALF 2.0:

9.	Physical modeling.....	219
9.1	Overview.....	219

9.6.1 LAYER statement

Syntax snippet:

```

layer ::=
    LAYER layer_identifier { layer_items }
  | LAYER layer_identifier ;
  | layer_template_instantiation

layer_items ::=
    layer_item { layer_item }

layer_item ::=
    all_purpose_item

```

9.6.2 Annotations for a LAYER

To fill in from ALF 2.0:

9.5	LAYER statement.....	227
9.5.1	Definition.....	227
9.5.2	PURPOSE annotation.....	228
9.5.3	PITCH annotation.....	229
9.5.4	PREFERENCE annotation.....	229

9.5.5	Example.....	229
-------	--------------	-----

9.7 VIA statement and related statements

9.7.1 VIA statement

Syntax snippet:

```

via ::=
    VIA via_identifier { via_items }
  | VIA via_identifier ;
  | via_template_instantiation

via_items ::=
    via_item { via_item }

via_item ::=
    all_purpose_item
  | pattern
  | artwork

```

9.7.2 Annotations for a VIA

To fill in from ALF 2.0:

9.8	VIA statement	237
9.8.1	Definition.....	237
9.8.2	USAGE annotation.....	238
9.8.3	Example	239

9.7.3 VIA reference statement

```

via_reference ::=
    VIA { via_instantiations }
    | VIA { via_identifiers }

via_instantiations ::=
    via_instantiation { via_instantiation }

via_instantiation ::=
    via_identifier { geometric_transformations }

```

To fill in from ALF 2.0:

9.8.4	VIA reference	240
9.11.7	VIA reference	249

9.8 Statements related to physical design rules

9.8.1 RULE statement

Syntax snippet:

```

rule ::=
    RULE rule_identifier { rule_items }
    | RULE rule_identifier ;
    | rule_template_instantiation

rule_items ::=
    rule_item { rule_item }

rule_item ::=
    all_purpose_item
    | pattern
    | via_reference

```

To fill in from ALF 2.0:

9.11	RULE statement.....	244
9.11.1	Definition.....	244
9.11.2	Width-dependent spacing	245
9.11.3	End-of-line rule	246
9.11.4	Redundant vias	247
9.11.5	Extraction rules.....	248

9.8.2 ANTENNA statement

Syntax snippet:

```

antenna ::=
    ANTENNA antenna_identifier { antenna_items }
  | ANTENNA antenna_identifier ;
  | antenna_template_instantiation

antenna_items ::=
    antenna_item { antenna_item }

antenna_item ::=
    all_purpose_item

```

To fill in from ALF 2.0:

9.13 ANTENNA statement.....	251
9.13.1 Definition.....	251
9.13.2 Layer-specific antenna rules.....	252
9.13.3 All-layer antenna rules.....	253
9.13.4 Cumulative antenna rules	254
9.13.5 Illustration.....	255

9.8.3 BLOCKAGE statement

Syntax snippet:

```

blockage ::=
    BLOCKAGE blockage_identifier { blockage_items }
  | BLOCKAGE blockage_identifier ;
  | blockage_template_instantiation

blockage_items ::=
    blockage_item { blockage_item }

blockage_item ::=
    all_purpose_item
  | pattern

```

```

| rule
| via_reference

```

To fill in from ALF 2.0:

9.9 BLOCKAGE statement	240
9.9.1 Definition.....	240
9.9.2 Example.....	241

9.8.4 PORT statement

Syntax snippet:

```

port ::=
    PORT port_identifier { port_items }
| PORT port_identifier ;
| port_template_instantiation

port_items ::=
    port_item { port_item }

port_item ::=
    all_purpose_item
| pattern
| rule
| via_reference

```

To fill in from ALF 2.0:

9.10 PORT statement.....	241
9.10.1 Definition.....	241
9.10.2 VIA reference	242
9.10.3 CONNECTIVITY rules for PORT and PIN	242
9.10.4 Reference of a declared PORT in a PIN annotation.....	243
9.10.5 VIEW annotation.....	244
9.10.6 LAYER annotation.....	244
9.10.7 ROUTING_TYPE	244

9.9 Statements related to physical geometry

9.9.1 SITE statement

Syntax snippet:

```
site ::=
    SITE site_identifier { site_items }
  | SITE site_identifier ;
  | site_template_instantiation

site_items ::=
    site_item { site_item }

site_item ::=
    all_purpose_item
  | ORIENTATION_CLASS_one_level_annotation
  | SYMMETRY_CLASS_one_level_annotation
```

To fill in from ALF 2.0.2:

9.12 SITE statement.....	249
9.12.1 Definition.....	249
9.12.2 ORIENTATION_CLASS and SYMMETRY_CLASS.....	249
9.12.3 Example.....	250

9.9.2 ARRAY statement

Syntax snippet:

```
array ::=
    ARRAY array_identifier { array_items }
  | ARRAY array_identifier ;
  | array_template_instantiation

array_items ::=
    array_item { array_item }

array_item ::=
    all_purpose_item
  | PURPOSE_single_value_annotation
  | geometric_transformation
```

To fill in from ALF 2.0:

9.14 ARRAY Statement	256
9.14.1 Definition.....	256
9.14.2 PURPOSE annotation.....	256
9.14.3 Examples	257

9.9.3 PATTERN statement

Syntax snippet:

```

pattern ::=
    PATTERN pattern_identifier { pattern_items }
    | PATTERN pattern_identifier ;
    | pattern_template_instantiation

pattern_items ::=
    pattern_item { pattern_item }

pattern_item ::=
    all_purpose_item
    | SHAPE_single_value_annotation
    | LAYER_single_value_annotation
    | EXTENSION_single_value_annotation
    | VERTEX_single_value_annotation
    | geometric_model
    | geometric_transformation

```

To fill in from ALF 2.0:

9.7 PATTERN statement	235
9.7.1 Definition.....	235
9.7.2 SHAPE annotation	235
9.7.3 LAYER annotation.....	236
9.7.4 EXTENSION annotation.....	236
9.7.5 VERTEX annotation	237
9.7.6 PATTERN with geometric model.....	237
9.7.7 Example.....	237

9.9.4 ARTWORK statement

Syntax snippet:

```

artwork ::=
    ARTWORK = artwork_identifier { artwork_items }

```

```

| ARTWORK = artwork_identifier ;
| artwork_template_instantiation

artwork_items ::=
    artwork_item { artwork_item }

artwork_item ::=
    geometric_transformation
| pin_assignment

```

To fill in from ALF 2.0:

9.4 ARTWORK statement	226
-----------------------------	-----

9.9.5 Geometric model

Syntax snippet:

```

geometric_model ::=
    nonescaped_identifier [ geometric_model_identifier ]
    { geometric_model_items }
| geometric_model_template_instantiation

geometric_model_items ::=
    geometric_model_item { geometric_model_item }

geometric_model_item ::=
    all_purpose_item
| POINT_TO_POINT_one_level_annotation
| coordinates

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

```

To fill in from ALF 2.0:

3.5 Geometric models	22
9.6 Geometric model statement	230
9.6.1 Definition.....	231
9.6.2 Predefined geometric models using TEMPLATE.....	233

9.9.6 Geometric transformation

```
geometric_transformations ::=
    geometric_transformation { geometric_transformation }

geometric_transformation ::=
    SHIFT_two_level_annotation
    | ROTATE_one_level_annotation
    | FLIP_one_level_annotation
    | repeat

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

To fill in from ALF 2.0.2:

9.3	Statements for geometric transformation.....	223
9.3.1	SHIFT statement.....	223
9.3.2	ROTATE statement.....	223
9.3.3	FLIP statement	224
9.3.4	REPEAT statement.....	224
9.3.5	Summary of geometric transformations	225

9.10 Statements related to functional description

To fill in from ALF 2.0.2:

5.	Functional modeling	33
----	---------------------------	----

9.10.1 FUNCTION statement

Syntax snippet:

```
function ::=
    FUNCTION { function_items }
    | function_template_instantiation

function_items ::=
    function_item { function_item }

function_item ::=
    all_purpose_item
```

```

| behavior
| structure
| statetable

```

9.10.2 TEST statement

Syntax snippet:

```

test ::=
    TEST { test_items }
    | test_template_instantiation

test_items ::=
    test_item { test_item }

test_item ::=
    all_purpose_item
    | behavior
    | statetable

```

To fill in from ALF 2.0:

6.8	TEST statement.....	136
6.9	Physical bitmap for memory BIST	136
6.9.1	Definition of concepts	136
6.9.3	Explanatory example	138

9.10.3 BEHAVIOR statement

Syntax snippet:

```

behavior ::=
    BEHAVIOR { behavior_items }
    | behavior_template_instantiation

behavior_items ::=
    behavior_item { behavior_item }

behavior_item ::=
    boolean_assignment
    | control_statement

```

```

| primitive_instantiation
| behavior_item_template_instantiation

boolean_assignments ::=
    boolean_assignment { boolean_assignment }

boolean_assignment ::=
    pin_variable = boolean_expression ;

primitive_instantiation ::=
    primitive_identifier [ identifier ] { pin_values }
| primitive_identifier [ identifier ]
    { boolean_assignments }

control_statement ::=
    @ control_expression { boolean_assignments }
    { : control_expression { boolean_assignments } }

```

To fill in from ALF 2.0:

5.5.1	BEHAVIOR	81
-------	----------------	----

9.10.4 STRUCTURE statement

Syntax snippet:

```

structure ::=
    STRUCTURE { named_cell_instantiations }
| structure_template_instantiation

named_cell_instantiations ::=
    named_cell_instantiation { named_cell_instantiation }

named_cell_instantiation ::=
    cell_identifier instance_identifier { pin_values }
| cell_identifier instance_identifier { pin_assignments }

```

To fill in from ALF 2.0:

6.3	STRUCTURE statement.....	109
-----	--------------------------	-----

9.10.5 VIOLATION statement

Syntax snippet:

```
violation ::=
    VIOLATION { violation_items }
    | violation_template_instantiation

violation_items ::=
    violation_item { violation_item }

violation_item ::=
    MESSAGE_TYPE_single_value_annotation
    | MESSAGE_single_value_annotation
    | behavior
```

To fill in from ALF 2.0.2:

8.4 VIOLATION container 180

9.10.6 STATETABLE statement

Syntax snippet:

```
statetable ::=
    STATETABLE [ identifier ]
    { statetable_header statetable_row { statetable_row } }
    | statetable_template_instantiation

statetable_header ::=
    input_pin_variables : output_pin_variables ;

statetable_row ::=
    statetable_control_values : statetable_data_values ;

statetable_control_values ::=
    statetable_control_value { statetable_control_value }

statetable_control_value ::=
    bit_literal
    | based_literal
```

```

| unsigned
| edge_value

statetable_data_values ::=
    statetable_data_value { statetable_data_value }

statetable_data_value ::=
    bit_literal
| based_literal
| unsigned
| ( [ ! ] pin_variable )
| ( [ ~ ] pin_variable )

```

To fill in from ALF 2.0:

5.5.2	STATETABLE.....	81
5.5.4	ROM initialization.....	84

9.10.7 PRIMITIVE statement

Syntax sniplet:

```

primitive ::=
    PRIMITIVE primitive_identifer { primitive_items }
| PRIMITIVE primitive_identifer ;
| primitive_template_instantiation

primitive_items ::=
    primitive_item { primitive_item }

primitive_item ::=
    all_purpose_item
| pin
| pin_group
| function
| test

```

To fill in from ALF 2.0:

5.6	Predefined models.....	85
5.6.1	Usage of PRIMITIVES.....	85
5.6.2	Concept of user-defined and predefined primitives	85
5.6.3	Predefined combinational primitives.....	87
5.6.4	Predefined tristate primitives.....	91
5.6.5	Predefined multiplexor	93

8.6.6 Predefined flip-flop..... 94

8.6.7 Predefined latch 95

10. Constructs for modeling of digital behavior

To fill in from ALF 2.0:

5.5	Variable declarations	80
5.1	Combinational functions	33
5.1.1	Combinational logic	33
5.1.2	Boolean operators on scalars	33
5.1.3	Boolean operators on words	34
5.1.4	Operator priorities	35
5.1.5	Datatype mapping	36
5.1.6	Rules for combinational functions	37
5.1.7	Concurrency in combinational functions	38
5.2	Sequential functions	39
5.2.1	Level-sensitive sequential logic	39
5.2.2	Edge-sensitive sequential logic	39
5.2.3	Unary operators for vector expressions	41
5.2.4	Basic rules for sequential functions	43
5.2.5	Concurrency in sequential functions	45
5.2.6	Initial values for logic variables	47
5.3	Higher-order sequential functions	48
5.3.1	Vector-sensitive sequential logic	48
5.3.2	Canonical binary operators for vector expressions	49
5.3.3	Complex binary operators for vector expressions	50
5.3.4	Operators for conditional vector expressions	53
5.3.5	Operators for sequential logic	54
5.3.6	Operator priorities	54
5.3.7	Using PINs in VECTORS	54
5.4	Modeling with vector expressions	55
5.4.1	Event reports	56
5.4.2	Event sequences	57
5.4.3	Scope and content of event sequences	58
5.4.4	Alternative event sequences	60
5.4.5	Symbolic edge operators	61
5.4.6	Non-events	62
5.4.7	Compact and verbose event sequences	63
5.4.8	Unspecified simultaneous events within scope	64
5.4.9	Simultaneous event sequences	66
5.4.10	Implicit local variables	68
5.4.11	Conditional event sequences	69
5.4.12	Alternative conditional event sequences	71
5.4.13	Change of scope within a vector expression	72
5.4.14	Sequences of conditional event sequences	76
5.4.15	Incompletely specified event sequences	78
5.4.16	How to determine well-specified vector expressions	79

10.1 Boolean expression language

```

boolean_expression ::=
    ( boolean_expression )
    | pin_value
    | boolean_unary boolean_expression
    | boolean_expression boolean_binary boolean_expression
    | boolean_expression ? boolean_expression :
      { boolean_expression ? boolean_expression : }
      boolean_expression

boolean_unary ::=
    !
    | ~
    | &
    | ~&
    | |
    | ~|
    | ^
    | ~^

boolean_binary ::=
    &
    | &&
    | |
    | ||
    | ^
    | ~^
    | !=
    | ==
    | >=
    | <=
    | >
    | <
    | +
    | -
    | *
    | /
    | %
    | >>
    | <<

```

10.2 Vector expression language

```

vector_expression ::=
    ( vector_expression )

```

```

| vector_unary boolean_expression
| vector_expression vector_binary vector_expression
| boolean_expression ? vector_expression :
    { boolean_expression ? vector_expression : }
    vector_expression
| boolean_expression control_and vector_expression
| vector_expression control_and boolean_expression

vector_unary ::=
    edge_literal

vector_binary ::=
    &
    | &&
    | |
    | ||
    | ->
    | ~>
    | <->
    | <~>
    | &>
    | <&>

control_and ::=
    & | &&

control_expression ::=
    ( vector_expression )
    | ( boolean_expression )

```

10.3 Control expression semantics

11. Constructs for modeling of analog behavior

11.1 Arithmetic expression language

Syntax snippet:

```
arithmetic_expression ::=
    ( arithmetic_expression )
  | arithmetic_value
  | [ arithmetic_unary ] arithmetic_expression
  | arithmetic_expression arithmetic_binary
    arithmetic_expression
  | boolean_expression ? arithmetic_expression :
    { boolean_expression ? arithmetic_expression : }
    arithmetic_expression
  | arithmetic_macro
    ( arithmetic_expression { , arithmetic_expression } )
```

```
arithmetic_unary ::= sign
```

```
arithmetic_binary ::=
    +
  | -
  | *
  | /
  | **
  | %
```

```
arithmetic_macro ::=
    abs
  | exp
  | log
  | min
  | max
```

To fill in from ALF 2.0:

7.2 Arithmetic expressions	146
----------------------------------	-----

7.2.1	Syntax of arithmetic expressions.....	146
7.2.2	Arithmetic operators	147
7.2.3	Operator priorities.....	148

11.2 Arithmetic model and related statements

To fill in from ALF 2.0:

3.4	Arithmetic models.....	22
7.	General rules for arithmetic models.....	143
7.1	Principles of arithmetic models.....	143
7.1.1	Global definitions for arithmetic models.....	143
7.1.2	Trivial arithmetic model	143
7.1.3	Arithmetic model using EQUATION.....	144
7.1.4	Arithmetic model using TABLE	144
7.1.5	Complex arithmetic model	145
7.3	Construction of arithmetic models.....	148
7.6	Arithmetic submodels	156

11.2.1 Arithmetic model statement

Syntax snippet:

```
arithmetic_models ::=
    arithmetic_model { arithmetic_model }

arithmetic_model ::=
    partial_arithmetic_model
    | non_trivial_arithmetic_model
    | trivial_arithmetic_model
    | assignment_arithmetic_model
    | arithmetic_model_template_instantiation
```

11.2.2 Partial arithmetic model

```

partial_arithmetic_model ::=
    nonescaped_identifier [ arithmetic_model_identifier ]
    { partial_arithmetic_model_items }

partial_arithmetic_model_items ::=
    partial_arithmetic_model_item
    { partial_arithmetic_model_item }

partial_arithmetic_model_item ::=
    any_arithmetic_model_item
    | table

```

Partial arithmetic model contains only definitions relevant for the model, but not sufficient data to evaluate the model.

Definitions within unnamed partial arithmetic model (i.e. partial arithmetic model without arithmetic model identifier) shall be inherited by all arithmetic models of the same type (i.e., using the same nonescaped identifier) within scope. However, these definitions can be locally overwritten.

Named partial arithmetic model (i.e. partial arithmetic model without arithmetic model identifier) can be used as argument of an EQUATION within another arithmetic model within scope, without appearing in the HEADER.

If a partial arithmetic model outside a HEADER contains a TABLE, the arithmetic values in the TABLE shall define a discrete set of valid values for the model.

If a partial arithmetic model within a HEADER contains a TABLE, the arithmetic values in the TABLE shall define the entries for table-lookup.

11.2.3 Non-trivial arithmetic model

```

non_trivial_arithmetic_model ::=
    nonescaped_identifier [ arithmetic_model_identifier ] {
        [ any_arithmetic_model_items ]
        arithmetic_body
        [ any_arithmetic_model_items ]
    }

```

Non-trivial arithmetic model contains sufficient data to evaluate the model.

11.2.4 Trivial arithmetic model

```

trivial_arithmetic_model ::=
    nonescaped_identifier [ arithmetic_model_identifier ]
    = arithmetic_value ;
    | nonescaped_identifier [ arithmetic_model_identifier ]
    = arithmetic_value { any_arithmetic_model_items }

```

Trivial arithmetic model is associated with a constant arithmetic value. Therefore the evaluation of the arithmetic model is trivial.

11.2.5 Assignment arithmetic model

```
assignment_arithmetic_model ::=  
    arithmetic_model_identifier = arithmetic_expression ;
```

This form of arithmetic model is valid only in the following cases:

A partial arithmetic model has been defined using the arithmetic model identifier

AND

arithmetic models for all arguments contained in the arithmetic expression have been defined.

OR

This construct is used in a dynamic template instantiation.

11.2.6 Items for any arithmetic model

```
any_arithmetic_model_items ::=  
    any_arithmetic_model_item { any_arithmetic_model_item }  
  
any_arithmetic_model_item ::=  
    all_purpose_item  
    | from  
    | to  
    | violation
```

Semantic restrictions apply, dependent on type and context of the arithmetic model.

11.3 Arithmetic submodel and related statements

11.3.1 Arithmetic submodel statement

```
arithmetic_submodels ::=  
    arithmetic_submodel { arithmetic_submodel }  
  
arithmetic_submodel ::=  
    non_trivial_arithmetic_submodel  
    | trivial_arithmetic_submodel  
    | arithmetic_submodel_template_instantiation
```

11.3.2 Non-trivial arithmetic submodel

```

non_trivial_arithmetic_submodel ::=
  nonescaped_identifier {
    [ any_arithmetic_submodel_items ]
    arithmetic_body
    [ any_arithmetic_submodel_items ]
  }

```

Non-trivial arithmetic submodel contains sufficient data to evaluate the arithmetic submodel.

11.3.3 Trivial arithmetic submodel

```

trivial_arithmetic_submodel ::=
  nonescaped_identifier = arithmetic_value ;
| nonescaped_identifier
  = arithmetic_value { any_arithmetic_submodel_items }

```

Trivial arithmetic submodel is associated with a constant arithmetic value. Therefore the evaluation of the arithmetic submodel is trivial.

11.3.4 Items for any arithmetic submodel

```

any_arithmetic_submodel_items ::=
  any_arithmetic_submodel_item
  { any_arithmetic_submodel_item }

any_arithmetic_submodel_item ::=
  all_purpose_item
| violation

```

Semantic restrictions apply, dependent on type and context of the arithmetic submodel.

11.4 Arithmetic body and related statements

11.4.1 Arithmetic body

```

arithmetic_body ::=
  arithmetic_submodels

```

```

| table_arithmetic_body
| equation_arithmetic_body

table_arithmetic_body ::=
    header table [ equation ]

equation_arithmetic_body ::=
    [ header ] equation [ table ]

```

Arithmetic model body shall supply the data necessary for evaluation of the arithmetic model.

11.4.2 HEADER statement

```

header ::=
    HEADER { identifiers }
| HEADER { header_arithmetic_models }
| header_template_instantiation

header_arithmetic_models ::=
    header_arithmetic_model { header_arithmetic_model }

header_arithmetic_model ::=
    non_trivial_arithmetic_model
| partial_arithmetic_model

```

HEADER shall contain arguments for evaluation of arithmetic model. The arithmetic values of those arguments must be supplied by application program.

Semantic restriction: No arithmetic submodel within arithmetic model body.

11.4.3 TABLE statement

```

table ::=
    TABLE { arithmetic_values }
| table_template_instantiation

```

TABLE shall provide means for evaluation using a look-up method. All arithmetic_values within TABLE must all be of the same type and compatible with the type of the arithmetic model under evaluation.

11.4.4 EQUATION statement

```

equation ::=
    EQUATION { arithmetic_expression }
| equation_template_instantiation

```

EQUATION shall provide means for evaluation using an analytical method.

11.5 Arithmetic model container

To fill in from ALF 2.0:

7.5	Containers for arithmetic models.....	155
7.1.6	Containers for arithmetic models and submodels	146

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

11.6 Statements related to arithmetic models for general purpose

11.6.1 MIN and MAX statement

To fill in from ALF 2.0:

7.6.1	Semantics of MIN / TYP / MAX	157
-------	------------------------------------	-----

11.6.2 TYP statement

11.6.3 DEFAULT statement

To fill in from ALF 2.0:

7.4.1	DEFAULT annotation.....	150
7.6.2	Semantics of DEFAULT	158

11.6.4 LIMIT statement

To fill in from ALF 2.0:

8.13	Reliability calculation	197
8.13.3	Global LIMIT specifications	199
8.13.4	LIMIT and model specification in the same context.....	199

8.13.5 Model and argument specification in the same context	201
---	-----

11.6.5 Annotations for arithmetic models for general purpose

To fill in from ALF 2.0:

7.4 Annotations for arithmetic models.....	150
7.4.2 UNIT annotation.....	150
7.4.3 CALCULATION annotation.....	151
7.4.4 INTERPOLATION annotation.....	152

11.7 Rules for evaluation of arithmetic models

11.7.1 Arithmetic model with arithmetic submodels

The application program shall decide which arithmetic submodel applies for evaluation in a particular situation. Per default, the arithmetic submodel identified by the DEFAULT keyword or the arithmetic submodel referenced by the DEFAULT annotation shall apply.

11.7.2 Arithmetic model with table arithmetic body

All arithmetic models in HEADER must contain TABLE.

Describe algorithm to identify correct table entry.

Refer to INTERPOLATION annotation.

Supplementary EQUATION is legal, shall be used for interpolation or extrapolation values outside range.

11.7.3 Arithmetic model with equation arithmetic body

Operands in arithmetic expression shall be defined as arithmetic models in HEADER or as partial arithmetic models outside HEADER, but within scope. It shall be legal to some arguments defined in the HEADER and some others outside HEADER.

For named arithmetic model, the name shall be used as operand. For unnamed arithmetic model, the keyword shall be used as operand.

Supplementary TABLE is legal, shall be used as lookup entry for downstream arithmetic models, when the arithmetic model itself is within HEADER.

11.8 Overview of arithmetic models

To fill in from ALF 2.0:

8. Electrical performance modeling	161
8.1 Overview of modeling keywords.....	161
8.1.1 Timing models.....	161

8.1.2	Analog models.....	163
8.1.3	Supplementary models	164
9.2	Arithmetic models in the context of layout	220

11.9 Arithmetic models for timing data

To fill in from ALF 2.0:

8.3	Specification of timing models	171
8.3.1	Template for timing measurements / constraints	171
8.3.2	Partially defined timing measurements and constraints	173
8.3.3	Template for same-pin timing measurements / constraints	173
8.3.4	Absolute and incremental evaluation of timing models	174
8.7.4	PIN-related timing models	184

11.9.1 TIME statement

To fill in from ALF 2.0:

8.3.6	TIME	176
8.13.1	TIME within the LIMIT construct	197
8.9.3	TIME to peak measurement	190
8.10	Waveform description.....	192
8.10.1	Principles	192
8.10.2	Annotations within a waveform	194

11.9.2 FREQUENCY statement

To fill in from ALF 2.0:

8.13.2	FREQUENCY within a LIMIT construct	198
8.9.2	TIME and FREQUENCY annotation.....	189

11.9.3 DELAY and RETAIN statement

To fill in from ALF 2.0:

8.3.7	DELAY	176
8.3.8	RETAIN	176

11.9.4 SLEWRATE statement

To fill in from ALF 2.0:

8.3.9	SLEWRATE.....	177
-------	---------------	-----

11.9.5 SETUP and HOLD statement

To fill in from ALF 2.0:

8.3.10	SETUP.....	177
8.3.11	HOLD	177

11.9.6 NOCHANGE statement

To fill in from ALF 2.0:

8.3.12	NOCHANGE.....	178
--------	---------------	-----

11.9.7 RECOVERY and REMOVAL statement

To fill in from ALF 2.0:

8.3.13	RECOVERY.....	178
8.3.14	REMOVAL	178

11.9.8 SKEW statement

To fill in from ALF 2.0:

8.3.15	SKEW between two signals	179
8.3.16	SKEW between multiple signals	179

11.9.9 PULSEWIDTH statement

To fill in from ALF 2.0:

8.3.17	PULSEWIDTH.....	180
--------	-----------------	-----

11.9.10 PERIOD statement

To fill in from ALF 2.0:

8.3.18	PERIOD.....	180
--------	-------------	-----

11.9.11 JITTER statement

To fill in from ALF 2.0:

8.3.19 JITTER.....	180
--------------------	-----

11.9.12 THRESHOLD statement

To fill in from ALF 2.0:

8.2.1 THRESHOLD definition.....	165
8.2.5 Context of THRESHOLD definitions	169

11.10 Auxiliary statements related to timing data

11.10.1 FROM and TO statement

To fill in from ALF 2.0:

8.2.2 FROM and TO container.....	166
----------------------------------	-----

```
from ::=  
    FROM { from_to_items }  
  
to ::=  
    TO { from_to_items }  
  
from_to_items ::= from_to_item { from_to_item }  
  
from_to_item ::=  
    PIN_single_value_annotation  
    | EDGE_single_value_annotation  
    | THRESHOLD_arithmetic_model
```

11.10.2 EARLY and LATE statement

To fill in from ALF 2.0:

8.5 EARLY and LATE container	181
------------------------------------	-----

```

EARLY_arithmetic_model_container ::=
    EARLY { early_late_arithmetic_models }

LATE_arithmetic_model_container ::=
    LATE { early_late_arithmetic_models }

early_late_arithmetic_models ::=
    early_late_arithmetic_model
    { early_late_arithmetic_model }

early_late_arithmetic_model ::=
    DELAY_arithmetic_model
    | RETAIN_arithmetic_model
    | SLEWRATE_arithmetic_model

```

11.10.3 Annotations for arithmetic models for timing data

To fill in from ALF 2.0:

8.2	Auxiliary statements for timing models.....	165
8.2.3	PIN annotation.....	166
8.2.4	EDGE_NUMBER annotation.....	166

11.11 Arithmetic models for environmental data

To fill in from ALF 2.0:

8.6	Environmental dependency for electrical data.....	181
-----	---	-----

11.11.1 PROCESS and DERATE_CASE statement

To fill in from ALF 2.0:

8.6.1	PROCESS.....	182
8.6.2	DERATE_CASE.....	182
8.6.3	Lookup table without interpolation.....	182
8.6.4	Lookup table for process- or derating-case coefficients.....	183

11.11.2 TEMPERATURE statement

To fill in from ALF 2.0:

8.6.5	TEMPERATURE.....	183
-------	------------------	-----

11.12 Arithmetic models for electrical data

To fill in from ALF 2.0:

- 8.7 PIN-related arithmetic models for electrical data183
 - 8.7.1 Principles183
 - 8.7.2 CAPACITANCE, RESISTANCE, and INDUCTANCE184
 - 8.7.3 VOLTAGE and CURRENT184
 - 8.7.6 Context-specific semantics.....185

11.12.1 CAPACITANCE statement

11.12.2 RESISTANCE statement

11.12.3 INDUCTANCE statement

11.12.4 VOLTAGE statement

11.12.5 CURRENT statement

11.12.6 POWER and ENERGY statement

To fill in from ALF 2.0:

- 8.11 Arithmetic models for power calculation194
 - 8.11.1 Principles194
 - 8.11.2 POWER and ENERGY195

11.12.7 FLUX and FLUENCE statement

To fill in from ALF 2.0:

- 8.12 Arithmetic models for hot electron calculation196
 - 8.12.1 Principles196
 - 8.12.2 FLUX and FLUENCE.....196

11.12.8 DRIVE_STRENGTH statement

To fill in from ALF 2.0.2:

8.8 Other PIN-related arithmetic models	187
8.8.1 DRIVE_STRENGTH	187

11.12.9 SWITCHING_BITS statement

To fill in from ALF 2.0:

8.8.2 SWITCHING_BITS	188
----------------------------	-----

11.12.10 NOISE and NOISE_MARGIN statement

To fill in from ALF 2.0:

8.14 Noise calculation	201
8.14.1 NOISE_MARGIN definition	202
8.14.2 Representation of noise in a VECTOR	203
8.14.3 Context of NOISE_MARGIN	204
8.14.4 Noise propagation	206
8.14.5 Noise rejection	208

11.12.11 Annotations for arithmetic models for electrical data

To fill in from ALF 2.0:

8.9 Annotations for arithmetic models	188
8.9.1 MEASUREMENT annotation	189
8.9.4 Rules for combinations of annotations	192

11.13 Arithmetic models for physical data

11.13.1 CONNECTIVITY statement

To fill in from ALF 2.0:

9.15 CONNECTIVITY statement	258
9.15.1 Definition	258
9.15.2 CONNECT_RULE annotation	259
9.15.3 CONNECTIVITY modeled with BETWEEN statement	259
9.15.4 CONNECTIVITY modeled as lookup TABLE	260

11.13.2 SIZE statement

11.13.3 AREA statement

11.13.4 WIDTH statement

11.13.5 HEIGHT statement

11.13.6 LENGTH statement

11.13.7 DISTANCE statement

11.13.8 OVERHANG statement

11.13.9 PERIMETER statement

11.13.10 EXTENSION statement

11.13.11 THICKNESS statement

11.13.12 Annotations for arithmetic models for physical data

To fill in from ALF 2.0:

9.18 Physical annotations for arithmetic models	264
9.18.1 BETWEEN statement within DISTANCE, LENGTH.....	264
9.18.2 MEASUREMENT annotation for DISTANCE	265
9.18.3 REFERENCE annotation for DISTANCE.....	265
9.18.4 Reference to ANTENNA	266
9.18.5 Reference to PATTERN.....	267

11.14 Arithmetic submodels for timing and electrical data

11.14.1 RISE and FALL statement

To fill in from ALF 2.0:

8.3.5	RISE and FALL submodels.....	175
-------	------------------------------	-----

11.14.2 HIGH and LOW statement

To fill in from ALF 2.0:

8.7.5	Submodels for RISE, FALL, HIGH, and LOW	184
-------	---	-----

11.15 Arithmetic submodels for physical data

11.15.1 HORIZONTAL and VERTICAL statement

12. Syntax Rule Summary

Suggestion: put all syntax items in alphabetical order, no subsections, since the syntax is already introduced for each item in a dedicated subchapter.

Annex A

(informative)

Bibliography

[B1] Ratzlaff, C. L., Gopal, N., and Pillage, L. T., “RICE: Rapid Interconnect Circuit Evaluator,” *Proceedings of 28th Design Automation Conference*, pp. 555–560, 1991.

[B2] SPICE 2G6 User’s Guide.

[B3] Standard Delay Format Specification, Version 3.0, Open Verilog International, May 1995.

[B4] The IEEE Standard Dictionary of Electrical and Electronics Terms, Sixth Edition.

Index

Symbols

?- 20

?! 20

?? 20

?~ 20

A

abs 63

alias 31

all_purpose_items 30

alphabetic_bit_literal 19

annotation

 CELL

 NON_SCAN_CELL 39

any_character 16

arithmetic_binary_operator 63

arithmetic_expression 63

arithmetic_function_operator 63

arithmetic_unary_operator 63

attribute 32

B

based literal 19

based_literal 20

behavior 54

behavior_body 54

binary 19

binary_base 20

binary_digit 20

bit 19

bit_edge_literal 20

bit_literal 19

block comment 18

boolean_binary_operator 60

boolean_expression 60

boolean_unary_operator 60

C

case-insensitive language 17

cell 38

cell_identifier 38, 39

cell_items 39

cell_template_instantiation 39

characterization 5

class 33

combinational_assignments 55

comment 17

 block 18

 long 18

 short 18

 single-line 18

comments

 nested 18

compound operators 17

constant 32

constant numbers 18

context-sensitive keyword 23

D

decimal 19

decimal_base 20

deep submicron 5

delimiter 17

digit 20

E

edge literal 20

edge_literal 20

edge_literals 25

equation 68

equation_template_instantiation 68

escape codes 21

escape_character 17

escaped identifier 22

escaped_identifier 22

exp 63

F

function 53, 54

function_template_instantiation 53, 54

functional model 5

G

group 33

group_identifier 33

H

- hard keyword 23
- header 68
- header_template_instantiation 68
- hex_base 20
- hex_digit 20
- hexadecimal 19

I

- identifier 17
- Identifiers 21
- identifiers 21
- include 31
- index 27
- integer 18

K

- Keywords
 - context-sensitive 24
 - generic objects 23
 - operators 23

L

- Library creation 1
- library_items 37
- library_template_instantiation 37
- literal 17
- log 63
- logic_values 56, 57
- logic_variables 28

M

- max 63
- min 63
- mode of operation 5

N

- nested comments 18
- non_negative_number 18
- non-escaped identifier 21
- nonescaped_identifier 22
- nonreserved_character 17
- Number 18
- number 18
- numeric_bit_literal 19

O

- objects 34
- octal 19
- octal_base 20
- octal_digit 20
- operation mode 5

P

- pin_assignments 27
- pin_identifier 40
- pin_items 41
- pin_template_instantiation 40
- placeholder identifier 22
- placeholder_identifier 21
- power constraint 5
- Power model 5
- primitive_identifier 55, 57
- primitive_instantiation 55
- primitive_items 57
- primitive_template_instantiation 57
- private keywords 24
- property 32
- public keywords 24

Q

- quoted string 16, 20
- quoted_string 20

R

- real 18
- reserved keyword 23
- reserved_character 16
- RTL 4

S

- sequential_assignment 55
- sign 18
- simulation model 5
- single-line comment 18
- soft keyword 23
- statetable 56
- statetable_body 56
- string 25
- symbolic_edge_literal 20

T

- table 68
- table_template_instantiation 68
- template 34
- template_identifier 34
- template_instantiation 35
- timing constraints 5
- timing models 5

U

- unnamed_assignment 29
- unsigned 18

V

- vector 44
- vector_expression 44, 60
- vector_items 44
- vector_template_instantiation 44
- vector_unary_operator 61
- vector-based modeling 5
- Verilog 4
- VHDL 4

W

- whitespace 16
- whitespace characters 16
- wildcard_literal 19
- wire 43, 45, 46, 47, 48, 49, 50, 51
- wire_identifier 43, 45, 46, 47, 48, 50
- wire_items 43
- wire_template_instantiation 43, 45, 46, 47, 48, 49, 50, 51, 52
- word_edge_literal 20

