IEC 62258-2

Edition 2.0 2011-05

INTERNATIONAL STANDARD

NORME INTERNATIONALE

Semiconductor die products – Part 2: Exchange data formats

Produits de puces de semiconducteurs – Partie 2: Formats d'échange de données

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland
Email: inmail@iec.ch
Web: www.iec.ch

About the IEC

The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications

The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

- Catalogue of IEC publications: www.iec.ch/searchpub

The IEC on-line Catalogue enables you to search by a variety of criteria (reference number, text, technical committee,…). It also gives information on projects, withdrawn and replaced publications.

- IEC Just Published: www.iec.ch/online_news/justpub

Stay up to date on all new IEC publications. Just Published details twice a month all new publications released. Available on-line and also by email.

- Electropedia: www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 20 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary online.

- Customer Service Centre: www.iec.ch/webstore/custserv

If you wish to give us your feedback on this publication or need further assistance, please visit the Customer Service Centre FAQ or contact us:
Email: csc@iec.ch
Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00

A propos de la CEI

La Commission Electrotechnique Internationale (CEI) est la première organisation mondiale qui élabore et publie des normes internationales pour tout ce qui a trait à l'électricité, à l'électronique et aux technologies apparentées.

A propos des publications CEI

Le contenu technique des publications de la CEI est constamment revu. Veuillez vous assurer que vous possédez l'édition la plus récente, un corrigendum ou amendement peut avoir été publié.

- Catalogue des publications de la CEI: www.iec.ch/searchpub/cur_fut-f.htm

Le Catalogue en-ligne de la CEI vous permet d'effectuer des recherches en utilisant différents critères (numéro de référence, texte, comité d'études,…). Il donne aussi des informations sur les projets et les publications retirées ou remplacées.

- Just Published CEI: www.iec.ch/online_news/justpub

Restez informé sur les nouvelles publications de la CEI. Just Published détaille deux fois par mois les nouvelles publications parues. Disponible en-ligne et aussi par email.

- Electropedia: www.electropedia.org

Le premier dictionnaire en ligne au monde de termes électroniques et électriques. Il contient plus de 20 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans les langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International en ligne.

- Service Clients: www.iec.ch/webstore/custserv/custserv_entry-f.htm

Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions, visitez le FAQ du Service clients ou contactez-nous:
Email: csc@iec.ch
Tél.: +41 22 919 02 11
Fax: +41 22 919 03 00

IEC 62258-2

Edition 2.0 2011-05

INTERNATIONAL STANDARD

NORME INTERNATIONALE

Semiconductor die products – Part 2: Exchange data formats

Produits de puces de semiconducteurs – Partie 2: Formats d'échange de données

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

PRICE CODE
CODE PRIX XB

ICS 31.080.99

ISBN 978-2-88912-496-1

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

SEMICONDUCTOR DIE PRODUCTS –

Part 2: Exchange data formats

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC 62258-2 has been prepared by IEC technical committee 47: Semiconductor devices.

This standard shall be read in conjunction with IEC 62258-1.

This second edition cancels and replaces the first edition published in 2005, and constitutes a technical revision.

With respect to the first edition, the following parameters have been updated for this edition:

Subclause	Parameter name
8.2.9	DEVICE_PICTURE_FILE
8.2.10	DEVICE_DATA_FILE
8.4.6	TERMINAL_GROUP
8.4.7	PERMUTABLE
8.5.1	TERMINAL_MATERIAL (was DIE_TERMINAL_MATERIAL)
8.5.2	TERMINAL_MATERIAL_STRUCTURE
8.6.2	MAX_TEMP_TIME
8.7.6	SIMULATOR_*simulator*_TERM_GROUP
8.8.3	ASSEMBLY
8.9.2	WAFER_THICKNESS
8.9.3	WAFER_THICKNESS_TOLERANCE
8.9.9	WAFER_INK
8.10.4	BUMP_SHAPE
8.10.5	BUMP_SIZE
8.10.6	BUMP_SPECIFICATION_DRAWING
8.10.7	BUMP_ATTACHMENT_METHOD
8.11.4	MPD_MSL_LEVEL
8.11.5	MPD_PACKAGE_DRAWING
8.12.1	QUALITY
8.12.2	TEST
8.13.1	TEXT
8.14.1	PARSE

The text of this standard is based on the following documents:

FDIS	Report on voting
47/2085/FDIS	47/2095/RVD

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

The committee has decided that the contents of this publication will remain unchanged until the stability date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

INTRODUCTION

This International Standard is based on the work carried out in the ESPRIT 4th Framework project GOODDIE which resulted in publication of the ES 59008 series of European specifications. Organisations that helped prepare this document include the ESPRIT ENCAST and ENCASIT projects, the Die Products Consortium, JEITA, JEDEC and ZVEI.

The structure of this International Standard as currently conceived is as follows:

Under main title: IEC 62258: Semiconductor die products

- Part 1: Procurement and use
- Part 2: Exchange data formats
- Part 3: Recommendations for good practice in handling, packing and storage (Technical report)
- Part 4: Questionnaire for die users and suppliers (Technical report)
- Part 5: Requirements for information concerning electrical simulation
- Part 6: Requirements for information concerning thermal simulation
- Part 7: XML schema for data exchange (Technical report)
- Part 8: EXPRESS model schema for data exchange (Technical report)

Further parts may be added as required.

SEMICONDUCTOR DIE PRODUCTS –

Part 2: Exchange data formats

1 Scope and object

This Part of IEC 62258 specifies the data formats that may be used for the exchange of data which is covered by other parts of the IEC 62258 series, as well as definitions of all parameters used according to the principles and methods of IEC 61360. It introduces a Device Data Exchange (DDX) format, with the prime goal of facilitating the transfer of adequate geometric data between die manufacturer and CAD/CAE user and formal information models that allow data exchange in other formats such as STEP physical file format, in accordance with ISO 10303-21, and XML. The data format has been kept intentionally flexible to permit usage beyond this initial scope.

It has been developed to facilitate the production, supply and use of semiconductor die products, including but not limited to:

- wafers,
- singulated bare die,
- die and wafers with attached connection structures,
- minimally or partially encapsulated die and wafers.

This standard reflects the DDX data format at version **1.3.0**

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 62258-1, *Semiconductor die products – Part 1: Procurement and use*

IEC 61360-4:2005, *Standard data element types with associated classification scheme for electric components – Part 4: IEC reference collection of standard data element types, component classes303-21*

ISO 8601:2004, *Data elements and interchange formats – Information interchange – Representation of dates and times*

ISO 6093:1985, *Information processing – Representation of numerical values in character strings for information interchange*

IPC/JEDEC J-STD-033B:2007, *Handling, Packing, Shipping and Use of Moisture/Reflow Sensitive Surface Mount Devices*

ISO 10303-21:2002, *Industrial automation systems and integration – Product data representation and exchange – Part 21: Implementation methods: Clear text encoding of the exchange structure*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC 62258-1 apply.

4 Requirements

Specific reference for parameter variables is made to the IEC 61360 data element type (DET) codes, which are defined in Part 4 of IEC 61360.

5 Device Data eXchange format (DDX) file goals and usage

5.1 To facilitate the transferral of data by electronic media from the device vendor to the end-user for use within a CAD or CAE system, a data file format, **D**evice **D**ata e**X**change, (**DDX**), shall be used. This data file format has been deliberately kept flexible, to permit further enhancements and additions for future use.

5.2 It is strongly recommended that **D**evice **D**ata e**X**change files have the three letter **DDX** file extension, and a **D**evice **D**ata e**X**change file shall hereon be referred to as a **DDX** file.

5.3 Data that are to be transferred from a device vendor to a user shall be contained in a single computer-readable DDX file, and the minimum contents of this file shall suffice a geometric CAD/CAE software design system. The file shall be textually readable, to permit simple manual verification.

5.4 The DDX file and its data contents shall be independent of both computer machine and operating system.

5.5 The DDX file contents shall include mechanical and interconnectivity information, but may additionally include electrical and functional data.

5.6 The DDX file may contain data for one or more devices, and shall be capable of being used as a library file by a CAD/CAE software design system. The file may contain one or more sets of data for the same device type, each having different delivery forms, such as bumped die, bare die, and Chip-Scale packaging.

5.7 The DDX file shall be capable of being simply or automatically generated, such as by an ASCII text editor or a spreadsheet.

5.8 The DDX file shall be capable of referencing additional external files, such as simulation and thermal model files.

5.9 All data shall be defined in such a way that conversion to or from other exchange formats is possible, such as GDSII and CIF for geometric data of die. As close compatibility to the existing DIE (Die Information Exchange) data as possible is desired, to facilitate simple translation of partial DIE data files.

5.10 Definitions of parameters shall be in conformity with IEC 61360 (refer to Clause 5 of IEC 62258-1).

6 DDX file format and file format rules

NOTE 1 Version 1.2.1 of DDX supersedes version 1.0.0 contained in ES 59008-6-1.

NOTE 2 Version 1.3.0 of DDX supersedes version 1.2.1 contained in IEC 62258-2:2005.

Refer to Clause 1 for the DDX version of this standard.

6.1 Data validity

6.1.1 All data not complying with the data syntax (refer to 7.3) shall be treated as a remark and, as such, ignored.

6.1.2 All mandatory data shall be present. Missing data shall be flagged as an error, rendering that data unusable.

6.1.3 Mathematical operations, calculations or formulae shall not be permitted within numeric data.

6.2 Character set

6.2.1 The DDX file shall be an ASCII compatible text file with suitable line termination. Line termination will depend upon the operating system. DOS/Windows ® generally uses a carriage/line-feed <CR/LF> terminator (ASCII 0Dh/0Ah), whereas UNIX® invariably relies solely upon a line-feed <LF> (ASCII 0x0A) terminator, the carriage return <CR> (ASCII 0x0D) being present by implication.

6.2.2 ASCII characters 0x00 to 0x7F are permitted, ASCII characters 0x80 to 0xFF shall be ignored.

6.2.3 All text data shall be case independent.

6.2.4 Space characters (ASCII 20h) and tab characters (ASCII 09h) shall both be treated as space separators, multiple space and tab characters will syntactically be treated as a single space separator.

6.3 SYNTAX RULES

6.3.1 All data lines shall be terminated with a semicolon: ";".

6.3.2 A comma "," shall be used as a data separator.

6.3.3 Lines beginning with a hash "#" shall be treated as an intentional comment. All data on that line shall be ignored.

6.3.4 Underscores "_" shall be ignored in a variable or property name, and may be used as intermediate name separators. Underscores are valid within textual string and name data.

6.3.5 Braces are used to open and close structures or BLOCKs. An open brace "{" shall be used to begin a structure or block, and a close brace "}" shall be used to terminate a structure or block.

6.3.6 Brackets "()" shall be permitted, then ignored, in numeric data for clarity (e.g. in co-ordinate pairs).

6.3.7 To accommodate typical spreadsheet CSV (Comma Separated Variable) format outputs, textual data may be inside double quotes "", and matching pairs of double quotes shall be ignored.

6.3.8 There is no specific line continuation character. A textual string opened with a double quote '"' shall close with a matching double quote '"', irrespective of the number of line breaks within that text. As all DDX commands terminate with a semicolon, the non-textual data will be

deemed to have ended at that semicolon. Textual data will be deemed to have ended at the semicolon following the closing double quote. Textual data not enclosed within double quotes may not include line break or control characters, and shall terminate at the first occurrence of a semicolon. Textual data following this semicolon will be treated as erroneous data and discarded.

6.3.9 For practicality, readability and ease of line parsing, is it recommended that the line length (between line termination characters) does not exceed 255 characters. It is further strongly recommended that a maximum limit of 1 023 characters per line be imposed to prevent other parsing software from having an input buffer overrun error.

7 DDX file content

7.1 DDX file content rules

7.1.1 Block structure

Data shall only exist within a block structure, referred to as a DEVICE block, and one or more DEVICE blocks, each containing data, may exist within a single file. Each DEVICE block is unique, and shall only contain data relevant to a single device, having a specific device form. All data within each DEVICE block shall be treated as being local and unique only to that block. (Refer to 6.3.4)

7.1.2 Parameter types

There are two types of parameters use for data, structures and variables, and these parameters shall only exist with a DEVICE block:

- a structure determines a set or multiple sets of data having different data types.
- a variable is equated to a single or multiple data of a single data type.

7.1.3 Data types

Data types are as follows.

7.1.3.1 Textual string data

All ASCII characters from ASCII 20h to ASCII 7Fh are permitted within textual data, characters including and above ASCII 80h shall be ignored. Consideration may be given to special print and display control characters to permit the printing of underscore or overscore characters. It is advised that textual string data is placed within pairs of double quotes, refer to 6.3.7.

7.1.3.2 Textual name data

All names shall be unique, and shall only consist of the following characters from the ASCII character set: -

A-Z a-z 0-9 $ - % & ! @ _ .

When textual name data are used to form a file name, it is advisable for the name to be limited to eight characters for the file name and to three characters for the file extension, with a point "**.**" used as the name/extension delimiter, in line with many common operating systems. It is advisable for textual name data to be placed within pairs of double quotes (refer to 6.3.7).

Note that all textual name data is case independent, and spaces are not permitted within a textual name.

7.1.3.3 Real numeric data

Real numeric data shall comply with ISO 6093:1985, and shall consist of the characters: -

0-9 + - . E e

The data values may be signed, and use engineering or scientific notation, but shall not include dimensional units, e.g.

90008, 9000.80, 9.0008E5, -5207, -5.207E3, 0.102, 102E-3

Note that a comma "**,**" is used as a data separator, and therefore shall not be used as a replacement for a decimal point "**.**".

7.1.3.4 Integer numeric data

Integer numeric data values shall comply to ISO 6093:1985, and only the characters **0** to **9** are permitted. Integers shall be unsigned, and shall not include dimensional units.

For practical purposes, an integer shall be limited to 16-bit resolution, i.e. integer values between and including 0 to 65536 only are acceptable.

7.1.3.5 Date data

Date data values shall comply with ISO 8601:2004 format, and may include time information as well, e.g.

"YYYY-MM-DD", "YYYYMMDD", "YYYY-MM-DDTHH:MM:SS".

7.1.4 Forward references

To permit single-pass parsing, no variable identifier or variable name shall be referenced prior to being defined.

7.1.5 Units

All units shall belong to the SI system, apart from the geometric unit of the micron (10^{-6}m), the inch and the mil (10^{-3} inch). Only one unit of dimension shall be permitted within a single **DEVICE** block. Note that the inch and the mil are non-preferred units, and are only present due to continued common usage.

7.1.6 Co-ordinate data

In all co-ordinate data, the **X** co-ordinate shall precede the **Y** co-ordinate and the **Y** co-ordinate shall precede the **Z** co-ordinate (i.e. **X,Y** or **X,Y,Z**).

The **X** co-ordinate shall be the horizontal axis (numerically left to right), the **Y** co-ordinate shall be the vertical axis (numerically bottom to top), and the **Z** co-ordinate shall be depth axis (numerically near to far).

7.1.7 Reserved words

All parameter names shall be considered as reserved and no variable identifier or variable name shall be permitted to have the same name. This restriction does not apply to free form textual data within quotes or double quotes (as 7.1.3.1 and 7.1.3.2).

7.2 DDX DEVICE block syntax

DEVICE device_name device_form {
relevant die data ……
}

The **DDX** file may contain one or more **DEVICE** blocks, all data pertaining to a particular device shall be embedded within the relevant block. (Refer to clause 6.1.1 and clause 7.1.1).

A **DEVICE** block is opened by the **DEVICE** keyword and opening brace "**{**", (as shown), and the **DEVICE** block is closed by the matching closing brace "**}**".

Data not within a **DEVICE** block structure shall be treated as a remark, permitting the future addition of checksum information, file creation date and historical data etc., within the **DDX** file, without affecting the actual device data.

The **device_name** is the given name by which the device shall be referred, and the **device_form** is the mechanical form of the device to which the block data pertains.

Valid data for the **device_form** variable are:

- **bare_die,**
- **bumped_die,**
- **lead_frame_die**
- **minimally_packaged_device (**or **MPD).**

Further **device_form** types may be added at a later stage, refer to IEC 61360-4:2005, AAD004-001, "die type code", for further details.

Only one **DEVICE** block having **device_name** of type **device_form** shall be present within the **DDX** file, but duplication of either **device_name** or **device_form** is permissible.

An example of a typical **DDX** file arrangement of **DEVICE** blocks:-

```
DEVICE name1 bare_die {
relevant data for device "name1" as a bare die..
}
DEVICE name1 bumped_die {
relevant data for device "name1"as a bumped die..
}
DEVICE name2 mpd {
relevant data for device "name2" as a minimally packaged device..
}
DEVICE name2 bare_die {
relevant data for device "name2" as a bare die..
}
DEVICE name1 mpd {
relevant data for device "name1" as a minimally packaged device..
}
DEVICE name3 bare_die {
relevant data for device "name3" as a bare die..
}
```

In the above example, there are three occurrences of a **DEVICE** block for device "name1", and two occurrences of a **DEVICE** block for device "name2", but each of these **DEVICE** blocks specify a different **device_form**. The order or sequencing of the **DEVICE** blocks has no relevance.

7.3 DDX data syntax

Property = value [, value];

<**property**>[*equate separator*]<value/variable {*separator* <value/variable> }>[*data terminator*][*line terminator*]

<**property**>	::=	Parameter name
[*space*]	::=	{space character (20h) or tab character (09h)}0+
[*equate separator*]	::=	[space]{equal **=**}[space]
[*separator*]	::=	[space]{comma **,**}[space]
[*data terminator*]	::=	[space]{semicolon;}
[*line terminator*]	::=	{CR or CR/LF}

For example:

```
thickness   =   100.0 ;
Thickness=470;
geometric_units=micron;
geometricunits     =     micron;
GeometricUnits= “millimetres”;
terminal_type = T1, Circle, 220;
Terminal_Type = T2, Rectangle,  200 ,  250;
TerminalType = T2, O, (200, 250);
TERMINALTYPE     =     T2, O,  200  ,  250;
```

Thus, **terminal_type, Terminal_Type**, **TerminalType** and **TERMINALTYPE** will all reference the same parameter name.

8 Definitions of DEVICE block parameters

8.0 General usage notes

8.0.1 Device form notes

Where a parameter is unique to the **device_form**, as defined in the **DEVICE** block, the parameter will be preceded with the following …

8.0.1.1 **DIE_** data parameter is unique to bare die or bumped die form

8.0.1.2 **BUMP_** data parameter is unique to only bumped die

8.0.1.3 **MPD_** data parameter is unique to a minimally packaged device, such as a CSP

8.0.1.4 **WAFER_** data parameter is unique to a die device delivered at wafer level

8.0.1.5 **LEAD_** data parameter is unique to a die device with attached lead frame.

8.0.2 Data Parameter Items

Within the following list of data parameters (see 8.1 onwards), the following items are shown:

8.0.2.1 the parameter name, as used syntactically within the **DDX** file,

8.0.2.2 the parameter type, indicating either a variable or structure data type,

8.0.2.3 the parameter function, determining its usage and meaning,

8.0.2.4 the parameter value, indicating the type of data expected,

8.0.2.5 any parameter limitation, indicating any limitation within the **DEVICE** block,

8.0.2.6 parameter dependencies, highlighting parameters that need to be declared prior to invocation,

8.0.2.7 one or more practical examples, and

8.0.2.8 any relevant notes.

A brief table of parameters is given in Annex F, and a working example of a full **DDX DEVICE** block is given in Annex A, with its expected graphical output in Annex C.

All parameters shall conform to the relevant IEC 61360 Data Element Type (DET) codes, as defined in IEC 61360-4:2005. Refer to Annex F for a cross-reference table.

8.0.3 Terms and conventions

A point of electrical connection is called a terminal. This may be a bond-pad for a bare die, and may equally refer to the landing or connection footprint area required by an interconnection medium. It is a common convention for die to have the initial terminal, numbered 1, in the upper left hand corner of the die, and for terminal or pin numbering to continue counter-clockwise in sequence.

The X co-ordinate dimensions are for the length in the horizontal plane with increasing positive values to the right. The Y co-ordinate dimensions are for the width in the horizontal plane with increasing positive values away from the user's view. The Z co-ordinate dimensions are for height in the vertical direction with increasing positive values upwards (towards the viewer).

As a point of reference, all die components, including bumped die, are generally viewed from above with the active side upwards.

8.0.4 Summary of general rules

8.0.4.1 All valid data shall be contained within a **DEVICE** block (refer to 7.1.1).

8.0.4.2 Any local or unique parameter, such as a name, shall be defined prior to its usage.

8.0.4.3 All parameters shall conform to the relevant IEC 61360 DET codes, as defined in Part 4 of IEC 61360 (refer to 5.8 and Annex F).

8.0.4.4 The units of measurement, **GEOMETRIC_UNITS**, shall be defined before any geometric variable is defined.

8.0.4.5 The geometric origin, **GEOMETRIC_ORIGIN**, and the geometric view, **GEOMETRIC_VIEW**, shall be defined before any geometric co-ordinates are defined.

8.0.4.6 The **TERMINAL_COUNT** parameter shall be defined before any **TERMINAL** parameters are referred to and the number of **TERMINAL** parameters shall not exceed the **TERMINAL_COUNT** value.

8.0.4.7 The **TERMINAL_TYPE_COUNT** parameter shall be defined before any **TERMINAL_TYPE** parameters are referred to and the number of **TERMINAL_TYPE** parameters shall not exceed the **TERMINAL_TYPE_COUNT** value.

8.1 BLOCK DATA

8.1.1 DEVICE_NAME Parameter

This is defined within the **DEVICE** block heading as the **device_name** parameter

Parameter Name	**device_name**
Parameter Type	Variable, refer to 7.2 on **DEVICE** blocks
Parameter Function	Defines the device manufacturer's type number or the reference name.
Parameter Values	Textual name data, as 7.1.3.2
Example	`SN74LS04`
Reference	See 7.1.7, 7.2 and Annex G.

8.1.2 DEVICE_FORM Parameter

This is defined within the **DEVICE** block heading by the **device_form** parameter.

Parameter Name	**device_form**
Parameter Type	Variable, refer to 7.2 on **DEVICE** blocks
Parameter Function	Defines the physical form of the device.
Parameter Values	Textual string data, as 7.1.3.1
Example	`bare_die, bumped_die, MPD`
Reference	Subclause 7.2 and Annex G.

8.1.3 BLOCK_VERSION Parameter

Parameter Name	**BLOCK_VERSION**
Parameter Type	Variable
Parameter Function	Specifies the version number and/or issue number of the DEVICE block.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.
Example	`BLOCK_VERSION = “1.0A”;`
Reference	Annex G.

8.1.4 BLOCK_CREATION_DATE Parameter

Parameter Name	**BLOCK_CREATION_DATE**
Parameter Type	Variable
Parameter Function	Specifies the date that the DEVICE block was created and last edited.
Parameter Values	Date, as 7.1.3.5
Limitations	Shall be declared only once within a single DEVICE block.
Example	`BLOCK_CREATION_DATE = “1997-12-25”;`
Reference	Annex G.

8.1.5 VERSION Parameter

Parameter Name	**VERSION**
Parameter Type	Variable
Parameter Function	Specifies the revision/version number of the DDX standard, (currently at version 1.3.0), to which this DEVICE block conforms.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.
Example	`VERSION = “1.3.0”;`
Notes	Refer to Clause 1 for the version of this standard document. Note that this document may not be the latest version, so refer to your Standards Authority if in doubt.
Reference	Annex G.

8.2 DEVICE DATA

8.2.1 DIE_NAME Parameter

Parameter Name	**DIE_NAME**
Parameter Type	Variable
Parameter Function	Specifies the name of the die or mask set from which the die was produced. This may be different to the DEVICE_NAME parameter.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.
Example	`DIE_NAME = “XXC345”;`
Reference	Annex G.

8.2.2 DIE_PACKAGED_PART_NAME Parameter

Parameter Name	**DIE_PACKAGED_PART_NAME**

Parameter Type	Variable
Parameter Function	Specifies the manufacturers part name for the equivalent packaged part, where available or applicable. This may be different to the DEVICE_NAME parameter.
Parameter Values	Textual string data, as 7.1.3.1
Example	`DIE_PACKAGED_PART_NAME = “SN5405JN”;`
Notes	Used to reference the identical packaged die part, not merely similar function, when supplied by the same manufacturer in packaged form.

8.2.3 DIE_MASK_REVISION Parameter

Parameter Name	**DIE_MASK_REVISION**
Parameter Type	Variable
Parameter Function	Specifies the mask revision details associated with that particular version on the die.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.
Example	`DIE_MASK_REVISION = “RTDAC1”;` `DIE_MASK_REVISION = “9033-2-101-M5/2”;`
Notes	Intended to ensure that the die data matches the geometric version of the actual die.
Reference	Annex G.

8.2.4 MANUFACTURER Parameter

Parameter Name	**MANUFACTURER**
Parameter Type	Variable
Parameter Function	Specifies the manufacturer or fabrication house of the device.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.
Example	`MANUFACTURER = “Fuzziwuz Logic Inc.”;`

8.2.5 DATA_SOURCE Parameter

Parameter Name	**DATA_SOURCE**
Parameter Type	Variable
Parameter Function	Specifies the source of the device data, essentially where this is different from the manufacturer or fabrication house.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.
Example	`DATA_SOURCE = “AnyChip Technology Ltd.”;` `DATA_SOURCE = “Good-Die database”;`

8.2.6 DATA_VERSION Parameter

Parameter Name	**DATA_VERSION**
Parameter Type	Variable
Parameter Function	Specifies the revision of the data source use to determine the parameters within the DEVICE block. This parameter is linked to 8.2.5, the DATA_SOURCE parameter.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.
Example	`DATA_VERSION = “Initial Issue 1.0”;`
Reference	Refer to Annex G.

8.2.7 FUNCTION Parameter

Parameter Name	**FUNCTION**
Parameter Type	Variable
Parameter Function	A brief description of the devices’ function.
Parameter Values	Textual string data, as 7.1.3.1

Limitations	Shall be declared only once within a single DEVICE block.
Example	`FUNCTION = “16 Bit Microprocessor”;`
Notes	IEC 61360-4:2005 specifies certain functional and application classes that may be used.

8.2.8 IC_TECHNOLOGY Parameter

Parameter Name	**IC_TECHNOLOGY**
Parameter Type	Variable
Parameter Function	Specifies the fabrication technology of the device.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.

Example	`IC_TECHNOLOGY = “CMOS”;`
	`IC_TECHNOLOGY = “bipolar”;`
	`IC_TECHNOLOGY = “bicmos”;`
	`IC_TECHNOLOGY = “GaAs”;`

8.2.9 DEVICE_PICTURE_FILE Parameter

Parameter Name	**DEVICE_PICTURE_FILE**
Parameter Type	Variable
Parameter Function	Specifies the name(s) of the graphics or picture file(s) representing the device.
Parameter Values	Textual name data, as 7.1.3.2
Example	`DEVICE_PICTURE_FILE = “DIE001.JPG”;` `DEVICE_PICTURE_FILE = “AA0B0C.SF”, “FRED.GIF”;`
Notes	The picture file may be an actual photograph, or a pictorial representation of the device. The file extent should indicate the file format used. Only file names, without relative or absolute path names, shall be used. Multiple parameters (picture file names) may be introduced within a single declaration, or multiple declarations maybe be employed to the same effect. There is no scaling or positional data requirement.
Reference	Annex I, Notes 3 and 4.

8.2.10 DEVICE_DATA_FILE Parameter

Parameter Name	**DEVICE_DATA_FILE**
Parameter Type	Variable
Parameter Function	Specifies the name(s) of a file(s) containing pertinent data, such as technical specifications, procurement documents or general data-sheets.
Parameter Values	Textual name data, as 7.1.3.2
Example	`DEVICE_DATA_FILE = “SpecSheet.PDF”;` `DEVICE_DATA_FILE = “AA0B0C.DOC”, “AAOBOC.TXT”;`
Notes	The data file may be of any recognised format. The file extent should indicate the file format used. Only file names, without relative or absolute path names, shall be used. Multiple parameters (data file names) may be introduced within a single declaration, or multiple declarations maybe be employed to the same effect.
Reference	Annex I, Notes 3 and 4.

8.3 GEOMETRIC DATA

8.3.1 GEOMETRIC_UNITS Parameter

Parameter Name	**GEOMETRIC_UNITS**
Parameter Type	Variable
Parameter Function	Specifies the geometric units that shall apply to all geometric values within the DEVICE block.
Parameter Values	Textual string data, as 7.1.3.1. One of the following values: ▪ micrometre, or micron, ▪ metre, ▪ millimetre, ▪ inch ▪ mil (1.0E-3 inch)
Limitations	Shall be declared only once within a single DEVICE block.
Example	`GEOMETRIC_UNITS = microns;` `GEOMETRIC_UNITS = mil;`
Notes	This GEOMETRIC_UNITS parameter shall be declared before any geometric units are used. The micron is generally the default dimensional unit for die dimensions, and the inch and mil are non-preferred units.
Reference	Subclause 7.1.5 and Annex E.

8.3.2 GEOMETRIC_VIEW Parameter

Parameter Name	**GEOMETRIC_VIEW**
Parameter Type	Variable
Parameter Function	Specifies the geometric view that shall apply to all geometric shapes within the DEVICE block.
Parameter Values	Textual string data, as 7.1.3.1. One of the following values: • **TOP** meaning active side upwards, and • **BOTTOM** meaning active side downwards.
Limitations	Shall be declared only once within a single DEVICE block.
Example	`GEOMETRIC_VIEW = top;` `GEOMETRIC_VIEW = “bottom”;`
Notes	The GEOMETRIC_VIEW parameter shall be declared before any geometric shapes are created. It would be common for a bare die and packaged part to be viewed in the “TOP” view, whereas bumped die may well be viewed from the “BOTTOM” (i.e. through the substrate).
Reference	Annex E.

8.3.3 GEOMETRIC_ORIGIN Parameter

Parameter Name	**GEOMETRIC_ORIGIN**
Parameter Type	Variable
Parameter Function	Determines the X- and Y-geometric origin from which all other co-ordinate pairs are referenced. The origin is given with respect to the geometric centre of the die in the units specified by the GEOMETRIC_UNITS parameter.
Parameter Values	Real X co-ordinate origin, Real Y co-ordinate origin, as 7.1.3.3
Dependencies	GEOMETRIC_UNITS, SIZE
Limitations	Shall be declared only once within a single DEVICE block.
Example	`GEOMETRIC_ORIGIN = -6000,-7500;`
Notes	Dimension values are in GEOMETRIC_UNITS. The origin co-ordinate pair values relate to the geometric die centre, refer to Figure 1. for further explanation.
Reference	Annex E.

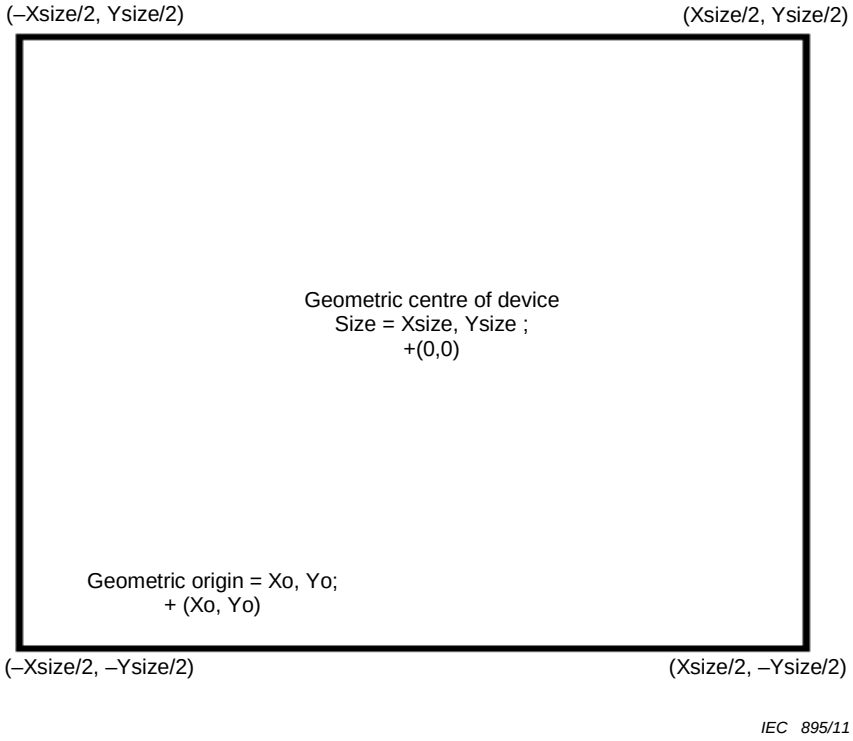

Figure 1 – Relationship between geometric centre and geometric origin

The GEOMETRIC_ORIGIN is treated as an offset for all co-ordinate data, so that the GEOMETRIC_ORIGIN values are ***added*** to ***all*** individual co-ordinate data pairs to give the X- and Y- position relative to the geometric centre of the device.

The primary use of the GEOMETRIC_ORIGIN parameter is to permit the centring of the origin on a terminal or other geometric feature, rather than an arbitrary geometric position, as, in practice, SIZE may be subject to an asymmetric tolerance so that all related references to this geometric centre could therefore also be subject to a tolerance error.

8.3.4 SIZE Parameter

Parameter Name	**SIZE**
Parameter Type	Variable
Parameter Function	Determines the X- and Y- dimensions of the device, and optionally specifies its shape as an ellipse.
Parameter Values	Real X-dimension, Real Y-dimension, (as 7.1.3.3), {Ellipse}
Dependencies	GEOMETRIC_UNITS, GEOMETRIC_VIEW
Limitations	Shall be declared only once within a single DEVICE block.
Example	`SIZE = 250, 500.5;` `SIZE = 350, 350, E;`
Notes	Dimension values are in GEOMETRIC_UNITS. In the latter example, the character “E” as the 3rd parameter defines an ellipse, in this instance a circular die of 350 GEOMETRIC_UNITS in diameter.
Reference	Annex E.

8.3.5 SIZE_TOLERANCE Parameter

Parameter Name	**SIZE_TOLERANCE**
Parameter Type	Variable
Parameter Function	Specify the geometric tolerance(s) of the SIZE parameter values.
Parameter Values	Real in GEOMETRIC_UNITS, as 7.1.3.3

Dependencies	GEOMETRIC_UNITS, SIZE, GEOMETRIC_VIEW
Limitations	Shall be declared only once within a single DEVICE block.

Example	`SIZE_TOLERANCE = 0.5;`
	`SIZE_TOLERANCE = -0.2,0.5;`
	`SIZE_TOLERANCE = -0.2,0.5,-0.1,0.4;`

Notes	One, two or four values may be given:- Where a single tolerance value is given, the tolerance shall be taken as an unsigned ± value for both the X- and Y-axis. Where two tolerance values are given:- the first value shall be taken as the minimum for both the X-axis and the Y-axis, the second value shall be taken as the maximum for both X-axis and the Y-axis. Where four tolerance values are given:- the first value shall be taken as the minimum for the X-axis, the second value shall be taken as the maximum for the X-axis, the third value shall be taken as the minimum for the Y-axis, and the fourth value shall be taken as the maximum for the Y-axis.

8.3.6 THICKNESS Parameter

Parameter Name	**THICKNESS**
Parameter Type	Variable
Parameter Function	Determines the thickness (Z-dimension) of the device.
Parameter Values	Real Z-dimension value, as 7.1.3.3
Dependencies	GEOMETRIC_UNITS
Limitations	Shall be declared only once within a single DEVICE block.
Example	`THICKNESS = 10.5;`
Notes	Dimension values are in GEOMETRIC_UNITS.

8.3.7 THICKNESS_TOLERANCE Parameter

Parameter Name	**THICKNESS_TOLERANCE**
Parameter Type	Variable
Parameter Function	Specifies the tolerance(s) of the THICKNESS parameter that may be expected due to normal process variations.
Parameter Values	Real in GEOMETRIC_UNITS, as 7.1.3.3
Dependencies	GEOMETRIC_UNITS, THICKNESS
Limitations	Shall be declared only once within a single DEVICE block.
Example	`THICKNESS_TOLERANCE = 2.5;` `THICKNESS_TOLERANCE = -1.2,1.5;`
Notes	One or two values may be given:- Where a single tolerance value is given, the tolerance shall be taken as an unsigned ± value. Where two tolerance values are given:- the first value shall be taken as the minimum, and the second value shall be taken as the maximum.

8.3.8 FIDUCIAL_TYPE Parameter

Parameter Name	**FIDUCIAL_TYPE**
Parameter Type	Structure
Parameter Function	The FIDUCIAL_TYPE structure assigns a fiducial type name and defines the associated graphic file and size of an individual fiducial type. The structure may be used to define a single fiducial type, or a multiple of fiducial types.
Dependencies	GEOMETRIC_UNITS, GEOMETRIC_VIEW
Notes	A fiducial only exists as a graphic shape within a rectangle, the graphic data for the fiducial is held within an external graphic file. The co-

ordinate pairs for this rectangle are relative only to the fiducial shape type, and are used as references when placing the shape.

Reference Annexes E and I.

Single FIDUCIAL_TYPE definition syntax:

FIDUCIAL_TYPE *Fiducial_type_name = Fiducial_file_name, X-size, Y-size;*
FIDUCIAL_TYPE *Fiducial_type_name = Fiducial_file_name, X-size, Y-size;*

Multiple FIDUCIAL_TYPE definition syntax:

FIDUCIAL_TYPE {
Fiducial_type_name = Fiducial_file_name, X-size, Y-size;
Fiducial_type_name = Fiducial_file_name, X-size, Y-size;
Fiducial_type_name = Fiducial_file_name, X-size, Y-size;
}

where:

8.3.8.1 Fiducial_type_name

Textual reference name (as 7.1.3.2) for a fiducial type, which shall be unique within the DEVICE block.

8.3.8.2 Fiducial_file_name

This is the name of the file (as 7.1.3.2) that holds the fiducial as graphic data. The graphic data type shall be indicated by the file extension code, such as “BMP”, “GIF”, “DXF” etc. The data type is not specified with this standard, and it is up to the CAD/CAM software as to what can and cannot be displayed. The graphic shall be treated as being contained within a rectangle of X-size, Y-size dimensions.

8.3.8.3 X-size, Y-size

These real numeric parameters (as 7.1.3.3) comprise a co-ordinate pair, and determine the rectangle size of the fiducial graphic. The geometric centre of the fiducial graphic shall be determined as being (X-size/2, Y-size/2).

Example 1

```
FIDUCIAL_TYPE Fid1 = “tile.bmp”, 200, 250;
```

This example describes a fiducial, found as an external file “`tile.bmp`”, uniquely referred to as `Fid1`, with an X-size of 200 units, and a Y-size of 250 units. The units are defined by the GEOMETRIC_UNITS parameter.

Example 2

```
FIDUCIAL_TYPE  {
      fidu1 = “graphic1.bmp”, 200, 300;
      fidu2 = “graphic2.dxf”, 500, 500;
      fidu3 = “graphic3.gif”, 100, 100;
}
```

This multiple definition example describes three separate fiducial types and shapes:

- `fidu1` is contained in file “`graphic1.bmp`”, of size 200 by 300 units.
- `fidu2` is contained in file “`graphic2.dxf`”, of size 500 by 500 units.
- `fidu3` is contained in file “`graphic3.gif`”, of size 100 by 100 units.

8.3.9 FIDUCIAL Parameter

Parameter Name	**FIDUCIAL**
Parameter Type	Structure
Parameter Function	The FIDUCIAL structure defines the location and orientation of each individual graphic fiducial. As a structure, FIDUCIAL may be used to define a single fiducial, or a multiple of fiducials.
Dependencies	GEOMETRIC_VIEW, FIDUCIAL_TYPE, GEOMETRIC_UNITS, and GEOMETRIC_ORIGIN.
Notes	Each fiducial type name used shall have been previously declared in a FIDUCIAL_TYPE invocation, and co-ordinate pair information shall relate to the GEOMETRIC_ORIGIN of the device.
Reference	Annex E.

Single **FIDUCIAL** definition syntax:

FIDUCIAL F_*n* = *Fiducial_type_name*, *X-co.*, *Y-co.*, *orientation;*
FIDUCIAL F_*n* = *Fiducial_type_name*, *X-co.*, *Y-co.*, *orientation;*

Multiple FIDUCIAL definition syntax:

FIDUCIAL {
F_*n* = *Fiducial_type_name*, *X-co.*, *Y-co.*, *orientation;*
F_*n* = *Fiducial_type_name*, *X-co.*, *Y-co.*, *orientation;*
F_*n* = *Fiducial_type_name*, *X-co.*, *Y-co.*, *orientation;*
}

where:

8.3.9.1 F_*n*

Unique fiducial identifier, where "***n***" is the actual fiducial number (Integer, as in 7.1.3.4), numbering from top left anti-clockwise. Note that the underscore character is optional, it is used for clarity only.

8.3.9.2 Fiducial_type_name

Textual name (as in 7.1.3.2) of a referenced FIDUCIAL_TYPE shape (as 8.3.8.1), which shall have been previously declared, (refer to 7.1.4).

8.3.9.3 X-co

Numeric real X co-ordinate value (as 7.1.3.3) for location of the centre of the fiducial shape, in units defined by the GEOMETRIC_UNITS parameter. This value is relative to the X co-ordinate value of the GEOMETRIC_ORIGIN, and this parameter is identical in operation to that described in 8.4.5.4.

8.3.9.4 Y-co

Numeric real Y co-ordinate value (as 7.1.3.3) for location of the centre of the fiducial shape, in units defined by the GEOMETRIC_UNITS parameter. This value is relative to the Y co-ordinate value of the GEOMETRIC_ORIGIN, and this parameter is identical in operation to that described in 8.4.5.5.

8.3.9.5 Orientation

Orientation value, of integer values from 0 to 360 (see 7.1.3.4). This is the angle of clockwise rotation in degrees about the geometric reference centre of the fiducial shape. If the letters "**MX**" are included, then the orientation of the fiducial shape shall be mirrored in the X-axis, similarly if the letters "**MY**" are included, then the orientation of the fiducial shape shall be

mirrored in the Y-axis. Both "**MX**" and "**MY**" may be present simultaneously (refer to Annex D for a graphical representation), and all mirroring shall be done about the geometric reference centre of the fiducial shape. Note that the mirroring operation shall be carried out first, then the fiducial shall be rotated by the orientation angle. This parameter is identical in operation to that described in 8.4.5.6

Example 1

```
FIDUCIAL Fid007 = fidu1, 5000, 7000, MX0;
```

This example declares that fiducial **Fid007**.....

- is located at X = 5000 units, Y = 7000 units,
- is a FIDUCIAL_TYPE named "fidu1", mirrored on the X-axis and orientated at 0^{o}.

Example 2

```
FIDUCIAL F_18 = fiduX1, 1000, 2200, 90;
```

This example declares that fiducial **F_18** .....

- is located at X = 1000 units, Y = 2200 units
- is a FIDUCIAL_TYPE named "fiduX1", orientated (rotated) by 90^{o} clockwise.

Example 3

```
FIDUCIAL {
  Fid007-fidu1,  5000, 7000, MX0;
  F_18 = fiduX1, 1000, 2200, 90;
}
```

This multiple fiducial definition example declares identical fiducial data to that shown in Examples 1 and 2.

8.4 TERMINAL DATA

8.4.1 TERMINAL_COUNT Parameter

Parameter Name	**TERMINAL_COUNT**
Parameter Type	Variable
Parameter Function	Specifies the number of electrical terminal or points of connection.
Parameter Values	Integer, as in 7.1.3.4
Limitations	Shall be declared only once within a single DEVICE block.
Example	`TERMINAL_COUNT = 44;`
Notes	The TERMINAL_COUNT value shall be declared before any TERMINAL declaration or usage.

8.4.2 TERMINAL_TYPE_COUNT Parameter

Parameter Name	**TERMINAL_TYPE_COUNT**
Parameter Type	Variable
Parameter Function	Specifies the number of different connection or bond terminal types or shapes.
Parameter Values	Integer, as in 7.1.3.4
Limitations	Shall be declared only once within a single DEVICE block.
Example	`TERMINAL_TYPE_COUNT = 4;`
Notes	The TERMINAL_TYPE_COUNT value shall be declared before any TERMINAL_TYPE declaration or usage.

8.4.3 CONNECTION_COUNT Parameter

Parameter Name	**CONNECTION_COUNT**
Parameter Type	Variable, optional

Parameter Function	Specifies the maximum number of connections used by the TERMINAL parameter. This parameter actually specifies the highest value for the connection ***conn_N*** numbers used in the TERMINAL parameter declaration (refer to 8.4.5.2).
Parameter Values	Integer, as in 7.1.3.4
Limitations	Shall be declared only once within a single DEVICE block.
Example	`CONNECTION_COUNT = 4;`
Notes	The CONNECTION_COUNT value should be declared before any TERMINAL declaration or usage. It shall be used as a limit check for the connection ***conn_N*** numbers. If the CONNECTION_COUNT parameter is not declared, then no check will take place.

8.4.4 TERMINAL_TYPE Parameter

Parameter Name	**TERMINAL_TYPE**
Parameter Type	Structure
Parameter Function	The TERMINAL_TYPE structure assigns a type name and defines the shape and size of an individual terminal type. As a structure, TERMINAL_TYPE may be used to define a single terminal type, or a multiple of terminal types.
Dependencies	GEOMETRIC_UNITS, GEOMETRIC_VIEW, TERMINAL_TYPE_COUNT
Notes	The co-ordinate pairs are relative only to the terminal shape type, and are used as references when placing the shape.
Reference	Annex E.

Single TERMINAL_TYPE definition syntax:

TERMINAL_TYPE *Terminal_type_name* = *Terminal_shape_type*, *Co-ordinates*.,,,;
TERMINAL_TYPE *Terminal_type_name* = *Terminal_shape_type*, *Co-ordinates*.,,,;

Multiple TERMINAL_TYPE definition syntax:

TERMINAL_TYPE {
Terminal_type_name = *Terminal_shape_type*, *Co-ordinates* .,,,;
Terminal_type_name = *Terminal_shape_type*, *Co-ordinates* .,,,;
Terminal_type_name = *Terminal_shape_type*, *Co-ordinates* .,,,;
}

where:

8.4.4.1 Terminal_type_name

Textual reference name for a terminal type, as in 7.1.3.2, which shall be unique within the DEVICE block.

8.4.4.2 Terminal_shape_type

Determines the terminal shape type (only the first letter need be used):

Table 1 – Terminal shape types

Char	DET data	Shape
R	RECT	**R**ectangle,
C	CIRC	**C**ircle,
E	ELL	**E**llipse,
P	POLY	**P**olygon.

8.4.4.3 Co-ordinates

Real co-ordinate values (refer to Table 2), as in 7.1.3.3.

This specifies the relative co-ordinates for the terminal shape, and in doing so, determines the geometric reference centre for the shape. The geometric reference centre of the shape will be used by the TERMINAL co-ordinates (see 8.4.5.4 and 8.4.5.5) to place the shape, and all rotation and mirroring will be about this geometric reference centre. Only the polygon shape can have a geometric reference centre other than the natural "centre-of gravity".

Table 2 – Terminal shape co-ordinates

Shape	Co-ordinates	Geometric reference centre
Rectangle	X-size, Y-size	The geometric centre (0,0) will occur at X-size/2:Y-size/2
Circle	Diameter	The geometric centre (0,0) will occur at diameter/2:diameter/2
Ellipse	X-axis, Y-axis	The geometric centre (0,0) will occur at X-axis/2, Y-axis /2
Polygon	$X\text{-}co_1$, $Y\text{-}co_1$,..... $X\text{-}co_N$, $Y\text{-}co_N$	An "N" sided polygon will have N pairs of co-ordinates, geometrically centred upon (0,0), with the assumption that the polygon shape will be completed with the vector (Xco_N,Yco_N - Xco_1,Yco_1)

Example 1

```
TERMINAL_TYPE ShapeR1 = Rectangle,  200, 250;
```

This example describes a rectangle, uniquely referred to as ShapeR1, with an X-axis dimension of 200 units, and a Y-axis dimension of 250 units. The units are defined by the GEOMETRIC_UNITS parameter.

Example 2

```
TERMINAL_TYPE  {
      ShapeR1 = R, 200, 300;
      ShapeC2 = C, 250;
      ShapeE3 = E, 400, 200;
      ShapeP4 = P, (150, 200),(-150, 200),(-150,-150),(150,-150);
}
```

This example describes four separate terminal types and shapes:

- ShapeR1 is a rectangular terminal having an X-axis dimension of 200 units and a Y-axis dimension of 300 units,
- ShapeC2 is a circular terminal of 250 units in diameter,
- ShapeE3 is an elliptical terminal with the X-axis diameter of 400 units and Y-axis diameter of 200 units, and
- ShapeP4 is a 4-sided polygonal terminal shape, with a geometric reference centre of (0,0).

Note that only the first letter of the *Terminal_shape_type* descriptor is used, and that the *Terminal_type_names* are unique.

8.4.5 TERMINAL Parameter

Parameter Name **TERMINAL**
Parameter Type Structure

Parameter Function	The TERMINAL structure defines the name, location, orientation and electrical function of each individual connection terminal. As a structure, TERMINAL may be used to define a single terminal, or a multiple of terminals.
Dependencies	TERMINAL_COUNT, CONNECTION_COUNT, GEOMETRIC_VIEW, TERMINAL_TYPE, GEOMETRIC_UNITS, GEOMETRIC_ORIGIN.
Notes	Each terminal type name used shall have been previously declared in a TERMINAL_TYPE invocation, and co-ordinate pair information shall relate to the GEOMETRIC_ORIGIN of the device. Refer to Annex E.

Single TERMINAL definition syntax:

```
TERMINAL T_n = conn_N, Terminal_type_name, X-co., Y-co., orient.,
               Terminal_name, IO_type;
TERMINAL T_n = conn_N, Terminal_type_name, X-co., Y-co., orient.,
               Terminal_name, IO_type;
```

Multiple TERMINAL definition syntax:

```
TERMINAL {
          T_n = conn_N, Terminal_type_name, X-co., Y-co., orient.,
                Terminal_name, IO_type;
          T_n = conn_N, Terminal_type_name, X-co., Y-co., orient.,
                Terminal_name, IO_type;
          T_n = conn_N, Terminal_type_name, X-co., Y-co., orient.,
                Terminal_name, IO_type;
        }
```

where:

8.4.5.1 T_*n*

Unique terminal identifier, where "***n***" is the actual terminal number (Integer, as 7.1.3.4), numbering from top left anti-clockwise. Note that the underscore character is optional, it is used for clarity only.

8.4.5.2 conn_N

Connection number, non-unique integer, as 7.1.3.4. This connection number is merely a place reference, and may optionally refer to a package pin; need not be present, or may be zero, 0, or left blank if unknown. The singular advantage of this connection number is in specifying and identifying multiple terminals that need to be connected together. The value of this connection number should be checked to ensure that it does not exceed the value determined by CONNECTION_COUNT, when specified.

8.4.5.3 Terminal_type_name

Textual name, as 7.1.3.2, of a referenced TERMINAL_TYPE terminal shape (as 8.4.3.1), which shall have been previously declared, (refer to 7.1.4).

8.4.5.4 X-co

Numeric real X co-ordinate value, as 7.1.3.3, for location of centre of the terminal shape, in units defined by the GEOMETRIC_UNITS parameter. This value is relative to the X co-ordinate value of the GEOMETRIC_ORIGIN.

8.4.5.5 Y-co

Numeric real Y co-ordinate value, as 7.1.3.3, for location of centre of the terminal shape, in units defined by the GEOMETRIC_UNITS parameter. This value is relative to the Y co-ordinate value of the GEOMETRIC_ORIGIN.

8.4.5.6 Orient

Orientation value, of integer values from 0 to 360, as 7.1.3.4. This is the angle of clockwise rotation in degrees about the geometric reference centre of the terminal shape. If the letters "**MX**" are included, then the orientation of the terminal shape shall be mirrored in the X-axis, similarly if the letters "**MY**" are included, then the orientation of the terminal shape shall be mirrored in the Y-axis. Both "**MX**" and "**MY**" may be present simultaneously (refer to Annex D for a graphical representation), and all mirroring shall be done about the geometric reference centre of the terminal shape. Note that the mirroring operation shall be carried out first, then the terminal shall be rotated by the orientation angle.

8.4.5.7 Terminal_name

Textual name, non-unique, as 7.1.3.2. This terminal name is for user reference and graphic display only, and may be omitted.

8.4.5.8 IO_type

A single letter indicating the terminal pin function type as given in Table 3, not mandatory. Further characters may be added as required.

Table 3 – Terminal IO types

Letter	Terminal Function
I	**I**nput, digital
O	**O**utput, digital
B	**B**i-directional, digital
G	**G**round connection
V	Supply, may be any supply type
A	**A**nalog pin, input or output
N	**N**o-connect pin, leave disconnected
U	**U**ndetermined, user programmable I/O
T	**T**est pin, connect only under manufacturers advice
X	Internally connected, do not connect.
H	Digital functional input pin, to be held at a logic **H**igh
L	Digital functional input pin, to be held at a logic **L**ow

T_***n*** (see 8.4.5.1) shall always be unique, whereas the pin connection number (see 8.4.5.2), ***conn_N,*** and terminal name (see 8.4.5.7), ***Terminal_name,*** need not be unique.

Example 1

```
TERMINAL Pin007 = 9, ShapeR1, (5000, 7000), MX0, VCC1, V;
```

This example declares that terminal "**Pin007"**.....

- is connected to arbitrary connection # 9,
- is located at x = 5000 units, Y = 7000 units,

- is a TERMINAL_TYPE named “ShapeR1”, mirrored on the X-axis, orientated at 0°, and
- is called “VCC1”, and
- is a power supply pin.

Example 2

```
TERMINAL Conn08 = 17, ShapeP2, 5000, 7300, 90, qd2i, I;
TERMINAL Conn09 = 17, ShapeP2, 5000, 7800, 90, qd2o, O;
```

These example declare that terminals “**Conn08”** and “**Conn09”**.....

- are both connected to arbitrary connection # 17,
- are located at X = 5000 units, Y = 7300 units & X = 5000 units, Y = 7800 units respectively,
- are both a TERMINAL_TYPE named “ShapeP2”, orientated (rotated) by 90° clockwise,
- are named “qd2i” & “qd2o”
- and constitute a digital I/O bi-directional connection, with “qd2I” being an input (I) and “qd2o” being an output (O).

Example 3

```
TERMINAL Term10 = , ShapeP2, 5000, 7600, 0, , X;
```

This example declares that terminal “**Term10”**.....

- is unconnected (or with no specific connection reference),
- is located at X = 5000 units, Y = 7600 units,
- is a TERMINAL_TYPE named “ShapeP2”, orientated at 0°, and
- has no given name and
- is specified as not to be bonded to.

Example 4

```
TERMINAL T_44 = , ShapeR2, 25000, 97000, MXMY90, ,;
```

This example declares that terminal **T_44**.....

- is unconnected (or with no specific connection reference),
- is located at (25000:97000),
- is a TERMINAL_TYPE named “ShapeR2”, mirrored in both the X- and Y-axis, then orientated at (rotated clockwise by) 90°,
- has no given name, and
- has no specified electrical connection properties.

Example 5

```
TERMINAL {
      Pin007 = 9, ShapeR1, (5000, 7000), MX0, VCC1, V;
      Conn08 = 17, ShapeP2, (5000, 7300), 90, qd2i, I;
      Conn09 = 17, ShapeP2, (5000, 7800), 90, qd2o, O;
      Term10 = , ShapeP2, (5000, 7600), 0, , X;
      T_44 = , ShapeR2, (25000, 97000), MXMY90, ,;
}
```

This multiple terminal definition example declares identical terminal data to that shown in Examples 1 to 4.

8.4.6 TERMINAL_GROUP Parameter

Parameter Name	TERMINAL_GROUP
Parameter Type	Structure
Parameter Function	The TERMINAL_GROUP structure assigns a group of terminals to a single terminal-group identifier that may be used in place of a list of

	terminals. The structure may be used to define a single group or a multiple of groups. Refer to TERMINAL_GROUP Rules (voir 8.4.6.1) for detail.
Parameter Values	Two or more terminal or group identifiers, as defined in 8.4.5.1 and 8.4.6.3
Dependencies	TERMINAL (see 8.4.5)
Limitations	A TERMINAL_GROUP identifier must be unique, and may only referenced after it has been declared.
Reference	Annex B.

8.4.6.1 TERMINAL_GROUP Rules

8.4.6.1.1 Content

The elements forming a group may be terminal identifiers or group identifiers or any mixture of the two. A group must contain at least two elements.

8.4.6.1.2 Uniqueness

All elements within a group must be unique. A group may not contain groups that refer directly or indirectly to the same element.

8.4.6.1.3 Ordering

The order of elements within groups may be important. Where two or more groups are to be related in a permutation, then the elements within those groups must correspond across the groups.

8.4.6.1.4 Recursion

A group may not contain itself nor may it contain any group that refers directly or indirectly to itself

Single TERMINAL_GROUP definition syntax:

TERMINAL_GROUP **G_*n*** = _ *Terminal or Group identifier, ... Terminal or Group identifier;*
TERMINAL_GROUP **G_*n*** = _ *Terminal or Group identifier, ... Terminal or Group identifier;*

Multiple TERMINAL_GROUP definition syntax:

TERMINAL_GROUP {
G_*n* = _*Terminal or Group identifier, ... Terminal or Group identifier;*
G_*n* = _ *Terminal or Group identifier, ... Terminal or Group identifier;*
}

where:

8.4.6.2 G_*n*

Unique TERMINAL_GROUP identifier, where "***n***" is the actual TERMINAL_GROUP number (Integer, as 7.1.3.4). Note that the underscore character is optional, it is used for clarity only.

8.4.6.3 Terminal or Group identifier

Two or more terminal or terminal-group identifier names, as

- Any terminal identifier name must be pre-declared in a TERMINAL statement, refer to 8.4.4.1. A terminal identifier name may occur only once within a single group assignment.
- Any terminal-group identifier name must be pre-declared in a TERMINAL_GROUP statement, refer to 8.4.6.2. A terminal-group identifier name may occur only once within a single group assignment and assignment circularity or recursion is not permitted.

Example

```
TERMINAL_GROUP G_1 = T_10, T_11, T_12;
TERMINAL_GROUP G_2 = T_15, T_16, T_12;

TERMINAL_GROUP  {
     G_3 = T_1, T_2;
     G_4 = T_4, T_5;
     G_5 = G_1, T_3;
     G_6 = G_3, G_4, T_11;
}
```

Notes
In all cases, a single terminal identifier shall only occur with a group once. This is also relevant when considering the expansion of a terminal-group.
Refer to Annex B.

8.4.7 PERMUTABLE Parameter

Parameter Name	PERMUTABLE
Parameter Type	Structure
Parameter Function	The PERMUTABLE structure assigns a list of terminals or terminal groups that are permutable. The list can comprise terminal identifiers (as 8.4.5.1) or terminal-group identifiers (as 8.4.6.2) but not a mixture of the two types of identifier. Each PERMUABLE assignment declares that the terminals or terminal groups listed are exchangeable and/or swappable in terms of functionality. This is to facilitate alternative connection and routing by CAD software. Refer PERMUTABLE Rules (see 8.4.7.1) for detail. Refer to Annex B for further explanation.
Parameter Values	The PERMUTABLE parameter statement shall consist of either a list of two or more terminal identifiers (see 8.4.6.1) or two or more terminal-group identifiers (see 8.4.6.1), but shall not contain a mixture of terminal and terminal-group identifiers.
Dependencies	TERMINAL (see 8.4.5) TERMINAL_GROUP (see 8.4.6)
Notes	In all cases, a single terminal identifier shall only occur with a group once. This is relevant when considering the expansion of a terminal-group. For correct permutation of terminal-groups within a single PERMUTABLE assignment, it is mandatory that the sequence of terminals within each group corresponds directly to the sequence of terminals of the other terminal-groups within the assignment.
Reference	Annex B.

8.4.7.1 PERMUTABLE Rules

8.4.7.1.1 Content

The elements forming a permutation may be terminal identifiers or group identifiers but not a mixture of the two. A permutation must contain at least two elements.

8.4.7.1.2 Uniqueness

All elements within a permutation must be unique. A permutation may not contain groups that refer directly or indirectly to the same element.
Each element within a permutation must contain the same number of Terminal identifiers (whether within Terminal Groups or not) as each other.

8.4.7.1.3 Ordering

The order of elements within a permutation is not important.

Single PERMUTABLE definition syntax:

PERMUTABLE **P_*n*** = *Terminal identifier, ... Terminal identifier;*
PERMUTABLE **P_*n*** = *Terminal identifier, ... Terminal identifier;*
PERMUTABLE **P_*n*** = *Group identifier, ... Group identifier;*
PERMUTABLE **P_*n*** = *Group identifier, ... Group identifier;*

Multiple PERMUTABLE definition syntax:

PERMUTABLE {
P_*n* = *Terminal identifier, ... Terminal identifier;*
P_*n* = *Terminal identifier, ... Terminal identifier;*
P_*n* = *Group identifier, ... Group identifier;*
P_*n* = *Group identifier, ... Group identifier;*
}

Example

```
PERMUTABLE P_1 = T_1, T_2, T_3;
PERMUTABLE P_2 = GP1, GP2, GP3, GP5, GP4;

PERMUTABLE {
     P_1 = T_1, T_2;
     P_2 = T_4, T_5;
     P_3 = T_9, T_10;
     P_4 = T_12, T_13;
     P_5 = G_5, G_6, G_7, G_8;
}
```

8.5 MATERIAL DATA

8.5.1 TERMINAL_MATERIAL Parameter

Parameter Name	**TERMINAL_MATERIAL**
Parameter Type	Variable
Parameter Function	Specifies the material used for the final terminations on the device.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.

Example	`TERMINAL_MATERIAL = “Al”;`
	`TERMINAL_MATERIAL = “Copper”;`
	`TERMINAL_MATERIAL = “SAC405”;`

8.5.2 TERMINAL_MATERIAL_STRUCTURE Parameter

Parameter Name	TERMINAL_MATERIAL_STRUCTURE
Parameter Type	Variable
Parameter Function	Specifies the sequential structure of the materials used to build-up the terminations on the device.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.
Example	`TERMINAL_MATERIAL_STRUCTURE = “Al-Ti-Ni-Au”;`

8.5.3 DIE_SEMICONDUCTOR_MATERIAL Parameter

Parameter Name	**DIE_SEMICONDUCTOR_MATERIAL**
Parameter Type	Variable
Parameter Function	Specifies the semiconductor material, which is used for fabrication of the die.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.

Example	`DIE_SEMICONDUCTOR_MATERIAL = “Insulator”;`
	`DIE_SEMICONDUCTOR_MATERIAL = “Silicon”;`
	`DIE_SEMICONDUCTOR_MATERIAL = “Sapphire”;`

Notes	This parameter is only relevant if the die bulk material is different from the die substrate material.

8.5.4 DIE_SUBSTRATE_MATERIAL Parameter

Parameter Name	**DIE_SUBSTRATE_MATERIAL**
Parameter Type	Variable
Parameter Function	Specifies the bulk or substrate material on which the die is made where it differs from the DIE_SEMICONDUCTOR_MATERIAL.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.

Example	`DIE_SUBSTRATE_MATERIAL = “Silicon”;`
	`DIE_SUBSTRATE_MATERIAL = “Insulator”;`
	`DIE_SUBSTRATE_MATERIAL = “Sapphire”;`

8.5.5 DIE_SUBSTRATE_CONNECTION Parameter

Parameter Name	**DIE_SUBSTRATE_CONNECTION**
Parameter Type	Variable
Parameter Function	Specifies the electrical connection for the substrate, if any. By fabrication, the substrate may be already electrically connected, which should also be stated. Where a substrate connection should be made, the potential point for connection shall also be stated.
Parameter Values	One or more textual string data, as 7.1.3.1 The first parameter shall conform to values given in Table 4
Limitations	Shall be declared only once within a single DEVICE block.

Example	`DIE_SUBSTRATE_CONNECTION = “CONN”,”Ground”;`
	`DIE_SUBSTRATE_CONNECTION = “N/A”;`
	`DIE_SUBSTRATE_CONNECTION = “ISOL”;`
	`DIE_SUBSTRATE_CONNECTION = “OPT”,”Most Negative”;`
	`DIE_SUBSTRATE_CONNECTION = “OPT”,”Most Positive”;`
	`DIE_SUBSTRATE_CONNECTION = “CONN”,”Digital Vdd”;`
	`DIE_SUBSTRATE_CONNECTION = “OPT”,”Analog ground”;`

Table 4 – Substrate Connection Parameters

Parameter Value	Function
CONN	**Must** be electrically connected
ISOL	**Must** be electrically isolated
OPT	May be optionally connected
N/A	Not Applicable
N/K	Not Known (unknown)

Notes When either “CONN” or “OPT” is given as the first parameter value, the second parameter value shall be comprehensively stated, sufficiently adequate to be understood for electrical connectivity. Recommended parameter values are: “Most Positive”, “Most Negative”, “GND”, “AGND”, “DGND”, “VSS”, “VDD”, “VEE”, “VCC”, “AVSS”, “AVDD”, “AVEE”, “AVCC”, “DVSS”, “DVDD”, “DVEE”, “DVCC”,”VBIAS”, or “SPECIAL”.

Alternatively, a specific TERMINAL name “**T_n**”, as described in 8.4.5 may be used as the second parameter value for direct connection to the substrate.

8.5.6 DIE_PASSIVATION_MATERIAL Parameter

Parameter Name	**DIE_PASSIVATION_MATERIAL**
Parameter Type	Variable
Parameter Function	Specifies the die surface passivation material.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.

Example	`DIE_PASSIVATION_MATERIAL = “Silicon Nitride”;`
	`DIE_PASSIVATION_MATERIAL = “Oxy-Nitride”;`
	`DIE_PASSIVATION_MATERIAL = “Polyimide”;`

8.5.7 DIE_BACK_DETAIL Parameter

Parameter Name	**DIE_BACK_DETAIL**
Parameter Type	Variable
Parameter Function	Specifies the detail finish to the die backside, relevant for different attachment methods / package styles, to include both finish and material.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.

Example	`DIE_BACK_DETAIL = “Sawn”;`
	`DIE_BACK_DETAIL = “Au-Metallized”;`
	`DIE_BACK_DETAIL = “Backlapped”;`
	`DIE_BACK_DETAIL = “Polished”;`
	`DIE_BACK_DETAIL = “Gold Metal”;`
	`DIE_BACK_DETAIL = “None”;`

8.6 ELECTRICAL AND THERMAL RATING DATA

8.6.1 MAX_TEMP Parameter

Parameter Name	**MAX_TEMP**
Parameter Type	Variable
Parameter Function	Specifies the maximum temperature to which the device may be exposed during any part of die attach, device soldering or other manufacturing process.
Parameter Values	Real, as 7.1.3.3
Parameter Units	^{0}C, Celsius
Limitations	Shall be declared only once within a single DEVICE block.
Example	`MAX_TEMP = 280;`

8.6.2 MAX_TEMP_TIME Parameter

Parameter Name	**MAX_TEMP_TIME**
Parameter Type	Variable
Parameter Function	Specifies the maximum time for which the device may be exposed to the maximum temperature during any part of die attach, device soldering or other manufacturing process.
Parameter Values	Real, as 7.1.3.3
Parameter Units	Seconds
Dependencies	MAX_TEMP
Limitations	Shall be declared only once within a single DEVICE block.
Example	`MAX_TEMP_TIME = 10;`

8.6.3 POWER_RANGE Parameter

Parameter Name	**POWER_RANGE**
Parameter Type	Variable
Parameter Function	Specifies the power likely to be dissipated by the device.
Parameter Values	Real, as 7.1.3.3
Parameter Units	Watts
Limitations	Shall be declared only once within a single DEVICE block.
Example	`POWER_RANGE = 0.92;`
Notes	This parameter should be used with caution, as without fully specifying all measurement conditions, use of this value can be misunderstood. The value given should indicate the likely maximum power dissipated under “typical” worst-case conditions.

8.6.4 TEMPERATURE_RANGE Parameter

Parameter Name	**TEMPERATURE_RANGE**
Parameter Type	Variable
Parameter Function	Specifies the operation and specification range of the die.
Parameter Values	Minimum (Real, in ^{0}C), Maximum (Real, in ^{0}C), as clause 7.1.3.3

Parameter Units	^{0}C, Celsius
Limitations	Shall be declared only once within a single DEVICE block.
Example	`TEMPERATURE_RANGE = -40, 90;`
Notes	For use with bare die, this parameter should be used with caution, and is only intended to indicate the operational or specified temperature "grade" of the equivalent packaged part.

8.7 SIMULATION DATA

8.7.1 Simulator MODEL FILE Parameter

Parameter Name	**SIMULATOR_*simulator*_MODEL_FILE**
Parameter Type	Variable
Parameter Function	Specifies the name of the model file for use with *simulator*.
Parameter Values	Textual name data, as 7.1.3.2
Limitations	Shall be declared only once within a DEVICE block per *simulator* type.
Example	`SIMULATOR_SPICE_MODEL_FILE = "BC109.MOD";` `SIMULATOR_SPECTRE_MODEL_FILE = "RTBAA1.S";`
Notes	Only file names, without relative or absolute path names, shall be used.
Reference	Annex D.

8.7.2 Simulator MODEL FILE DATE Parameter

Parameter Name	**SIMULATOR_*simulator*_MODEL_FILE_DATE**
Parameter Type	Variable
Parameter Function	Specifies the creation or validation date of the model file.
Parameter Values	Date, as 7.1.3.5
Limitations	Shall be declared only once within a DEVICE block per *simulator* type.
Example	`SIMULATOR_SPICE_MODEL_FILE_DATE="19951021";` `SIMULATOR_VHDL_MODEL_FILE_DATE="1993-05-17";`
Notes	This date should match the date of the actual model file, to indicate that the correct model file is being used, and for use within a library make / build function.
Reference	Annex D.

8.7.3 Simulator NAME Parameter

Parameter Name	**SIMULATOR_*simulator*_NAME**
Parameter Type	Variable
Parameter Function	Specifies the name of either the generic or specific *simulator* to which the model file is appropriate.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a DEVICE block per *simulator* type.
Example	`SIMULATOR_SPICE_NAME = "pSpice";` `SIMULATOR_VERILOG_NAME = "Verilog-XL";`
Reference	Annex D.

8.7.4 Simulator VERSION Parameter

Parameter Name	**SIMULATOR_*simulator*_VERSION**
Parameter Type	Variable
Parameter Function	Specifies the version number of the *simulator* as specified by the SIMULATOR_*simulator*_NAME parameter, which was used to verify the model data.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a DEVICE block per *simulator* type.
Example	`SIMULATOR_SPICE_VERSION = "4.0.1";` `SIMULATOR_VERILOG_VERSION = "1.7B";`
Reference	Annex D.

8.7.5 Simulator COMPLIANCE Parameter

Parameter Name	**SIMULATOR_*simulator*_COMPLIANCE**
Parameter Type	Variable
Parameter Function	Specifies the minimum compliance level of the *simulator* required to both accurately reproduce and correlate with simulation results.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a DEVICE block per *simulator* type.
Example	`SIMULATOR_VHDL_COMPLIANCE = “VHDL ’93”;` `SIMULATOR_SPICE_COMPLIANCE = “2G6”;`
Reference	Annex D.

8.7.6 Simulator TERM_GROUP Parameter

Parameter Name	**SIMULATOR_*simulator*_TERM_GROUP**
Parameter Type	Variable
Parameter Function	Specifies the group or terminals to which the simulator model applies. If missing, the assumption is that the simulator model applies to the whole device.
Parameter Values	A list of terminals (as 8.4.5.1) and/or terminal groups (as 8.4.6.2)
Limitations	The terminals or terminal groups referenced shall have already been declared in either a TERMINAL or TERMINAL_GROUP parameter. Shall be declared only once within a DEVICE block per *simulator* type.
Example	`SIMULATOR_SPICE_TERM_GROUP = T_1, T_2, G_1;` `SIMULATOR_IBIS_TERM_GROUP = G_5, G_8, T_11, T_33;`
Reference	Annexes B and D.

8.8 HANDLING, PACKING, STORAGE and ASSEMBLY DATA

8.8.1 DELIVERY_FORM Parameter

Parameter Name	**DELIVERY_FORM**
Parameter Type	Variable
Parameter Function	Specifies the form in which the die is delivered.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.

Example	`DELIVERY_FORM = “Wafer”;`
	`DELIVERY_FORM = “Die”;`
	`DELIVERY_FORM = “Bare Die in Waffle Tray”;`
	`DELIVERY_FORM = “Sawn Wafer”;`
	`DELIVERY_FORM = “Tray”;`
	`DELIVERY_FORM = “Bandoleer”, “Waffle”;`

8.8.2 PACKING_CODE Parameter

Parameter Name	**PACKING_CODE**
Parameter Type	Variable
Parameter Function	Specifies the form of primary packing used for the supply of devices.
Parameter Values	Textual string data, as 7.1.3.1

Example	`PACKING_CODE = “WAFFLE”;`
	`PACKING_CODE = “GEL_PACK”;`
	`PACKING_CODE = “Bandoleer”;`

8.8.3 ASSEMBLY Parameters

Parameter Name	ASSY_ *text-identifier*
Parameter Type	Variable
Parameter Function	General textual parameter, to enable the user to include relevant and identified assembly techniques, parameters and operations.
Parameter Values	Textual string data, as 7.1.3.1
Examples	`ASSY_PROCESS_LIMITATIONS = “NONE”;`

Notes Recognised *ASSY_identifiers* are:

```
8.8.3.1    ASSY_PROCESS_LIMITATIONS
8.8.3.2    ASSY_STORAGE_LIMITATIONS
8.8.3.3    ASSY_ASSEMBLY_LIMITATIONS
8.8.3.4    ASSY_TEMPERATURE_LIMITATIONS
8.8.3.5    ASSY_BONDING_METHODS
8.8.3.6    ASSY_BONDING_MATERIALS
8.8.3.7    ASSY_ATTACH_METHODS
8.8.3.8    ASSY_ATTACH_MATERIALS
8.8.3.9    ASSY_GENERAL_REQUIREMENTS
8.8.3.10   ASSY_HANDLING_REQUIREMENTS
8.8.3.11   ASSY_PACKING_REQUIREMENTS
8.8.3.12   ASSY_STORAGE_REQUIREMENTS
8.8.3.13   ASSY_SHIPPING_REQUIREMENTS
```

8.9 WAFER SPECIFIC DATA

8.9.1 WAFER_SIZE Parameter

Parameter Name	**WAFER_SIZE**
Parameter Type	Variable
Parameter Function	Specifies the wafer diameter.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.
Example	`WAFER_SIZE = “6 inch”;` `WAFER_SIZE = “150mm”;`
Notes	As this is a dimensional value, but related to the fabrication and not the die size, the WAFER_SIZE parameter may be in units different to those defined by the GEOMETRIC_UNITS parameter, hence the data is presented as free-form text.

8.9.2 WAFER_THICKNESS Parameter

Parameter Name	**THICKNESS**
Parameter Type	Variable
Parameter Function	Determines the thickness (Z-dimension) of the wafer.
Parameter Values	Real Z-dimension value, as in 7.1.3.3
Dependencies	GEOMETRIC_UNITS
Limitations	Shall be declared only once within a single DEVICE block.
Example	`WAFER_THICKNESS = 10.5;` `WAFER_THICKNESS = 80;`
Notes	Dimension values are in GEOMETRIC_UNITS.

8.9.3 WAFER_THICKNESS_TOLERANCE Parameter

Parameter Name	**WAFER_THICKNESS_TOLERANCE**
Parameter Type	Variable
Parameter Function	Specifies the tolerance(s) of the WAFER_THICKNESS parameter that may be expected due to normal process variations.
Parameter Values	Real in GEOMETRIC_UNITS, as 7.1.3.3
Dependencies	GEOMETRIC_UNITS, WAFER_THICKNESS
Limitations	Shall be declared only once within a single DEVICE block.
Example	`WAFER_THICKNESS_TOLERANCE = 2.5;` `WAFER_THICKNESS_TOLERANCE = -1.2,1.5;`
Notes	One or two values may be given:- Where a single tolerance value is given, the tolerance shall be taken as an unsigned ± value. Where two tolerance values are given:- the first value shall be taken as the minimum, and the second value shall be taken as the maximum.

8.9.4 WAFER_DIE_STEP_SIZE Parameter

Parameter Name	**WAFER_DIE_STEP_SIZE**
Parameter Type	Variable
Parameter Function	Determines the X- and Y- step size dimensions, in GEOMETRIC_UNITS, for the die on wafer, as required by mechanical handling equipment.
Parameter Values	Real X-dimension, Real Y-dimension, (as 7.1.3.3)
Dependencies	GEOMETRIC_UNITS, GEOMETRIC_VIEW
Limitations	Shall be declared only once within a single DEVICE block.
Example	`WAFER_DIE_STEP_SIZE = 280, 530;` `WAFER_DIE_STEP_SIZE = 1500, 1500;`
Reference	Annex H.

8.9.5 WAFER_GROSS_DIE_COUNT Parameter

Parameter Name	**WAFER_GROSS_DIE_COUNT**
Parameter Type	Variable
Parameter Function	Specifies the number of whole and viable gross die (of the die type in question) available on the wafer.
Parameter Values	Integer, as 7.1.3.4
Limitations	Shall be declared only once within a single DEVICE block.
Example	`WAFER_GROSS_DIE_COUNT = 2027;`
Notes	This parameter value is only intended to give an indication of the number of relevant and viable die per wafer, and shall have no other implication.
Reference	Annex H.

8.9.6 WAFER_INDEX Parameter

Parameter Name	**WAFER_INDEX**
Parameter Type	Variable
Parameter Function	Specifies the type of index feature on a wafer which acts as a reference feature and its orientation with respect to the X-axis for the die. The first parameter determines the index feature type, and the second parameter specifies the approximate angular relationship between the assumed X-axis for the die and the index feature on the wafer. The angle shall be given, in degrees counted clockwise, of the index feature, taking the X-axis of the die as the reference.
Parameter Values	Textual string data, as 7.1.3.1, with value “Flat” or “Notch”, followed by a single integer value, in units of degrees, from 0 to 359, as 7.1.3.4.
Limitations	Shall be declared only once within a single DEVICE block.
Example	`WAFER_INDEX = “Flat”,90;` `WAFER_INDEX = “Notch”,0;`
Notes	This parameter is only intended as a guide, and so integer approximations shall be acceptable. Negative angle values are not permitted. Where more than one flat exists on the wafer, the orientation is with respect to the major, or prime, flat. This may also indicate the crystal [110] direction.
Reference	Annex H.

8.9.7 WAFER_RETICULE_STEP_SIZE Parameter

Parameter Name	**WAFER_RETICULE_STEP_SIZE**
Parameter Type	Variable
Parameter Function	Specifies X- and Y- step and repeat dimensions for a single reticule.
Parameter Values	Real X-dimension, Real Y-dimension, (as 7.1.3.3)
Dependencies	GEOMETRIC_UNITS, GEOMETRIC_VIEW
Limitations	Shall be declared only once within a single DEVICE block.
Example	`WAFER_RETICULE_STEP_SIZE = 2500, 8000;` `WAFER_RETICULE_STEP_SIZE = 15000, 27500;`

Notes	This parameter is relevant mainly for MPW (Multi Project Wafers), or instances where the WAFER_DIE_STEP_SIZE becomes invalid due to a non-integer relationship between reticule size and die size.
Reference	Annex H.

8.9.8 WAFER_RETICULE_GROSS_DIE_COUNT Parameter

Parameter Name	**WAFER_RETICULE_GROSS_DIE_COUNT**
Parameter Type	Variable
Parameter Function	Specifies the number of whole and viable gross die (of the die type in question) emplaced within a single reticule.
Parameter Values	Integer, as 7.1.3.4
Limitations	Shall be declared only once within a single DEVICE block.
Example	`WAFER_RETICULE_GROSS_DIE_COUNT = 40;`
Notes	This parameter is relevant mainly for MPW (Multi Project Wafers), or instances where there is more than one die type in the reticule. This parameter value is only intended to give an indication of the number of relevant and viable die per reticule, and shall have no other implication.
Reference	Annex H.

8.9.9 WAFER_INK Parameters

Parameter Name	WAFER_INK_*text-identifier*
Parameter Type	Variable
Parameter Function	General textual parameter, to enable the user to include relevant parameters relating to the wafer dot/ink size and colour. This is of particular relevance for visual systems for die selection, sorting and inspection.
Parameter Values	Textual string data, as 7.1.3.1

Examples	`WAFER_INK_COLOUR = “BLACK”;`
	`WAFER_INK_LOCATION = “CENTRAL”;`
	`WAFER_INK_SIZE_MAX = “0.8mm”;`
	`WAFER_INK_SORT_COLOUR = “BIN1, RED, UPPER RIGHT”;`
	`WAFER_INK_SORT_COLOUR = “BIN2, GREEN, LOWER LEFT”;`

Notes	Recognised *WAFER_INK_identifiers* are: 8.9.9.1 WAFER_INK_COLOUR 8.9.9.2 WAFER_INK_SIZE 8.9.9.3 WAFER_INK_SIZE_TOL 8.9.9.4 WAFER_INK_SIZE_MAX 8.9.9.5 WAFER_INK_LOCATION 8.9.9.6 WAFER_INK_LOCATION_TOL 8.9.9.7 WAFER_INK_HEIGHT_MAX 8.9.9.8 WAFER_INK_SORT_COLOUR

8.10 BUMP TERMINATION SPECIFIC DATA

Any data given that is pertinent to a “bumped” device (a device using added bump connection structures) can only reflect the bump parameters at the time of bump formation, and not necessarily the final parameters. Due to the nature of final bump connection, the bump parameters will be altered and deformed during the attachment process. Therefore all geometric and metallurgical parameters given should be considered for “virgin” bumps.

8.10.1 BUMP_MATERIAL Parameter

Parameter Name	**BUMP_MATERIAL**
Parameter Type	Variable
Parameter Function	Specifies the metallurgical material of the bump contact.
Parameter Values	Textual string data, as clause 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.

Example	`BUMP_MATERIAL = “Copper”;`
	`BUMP_MATERIAL = “Molybdenum Telluride”;`
	`BUMP_MATERIAL = “Au”;`
	`BUMP_MATERIAL = “SAC415”;`
	`BUMP_MATERIAL = “Pb-Sn”;`

Notes	Required only for device types with bump connection structures. Where a device has both bumped and non-bumped terminals, TERMINAL_MATERIAL should be used to describe the non-bumped terminals, and BUMP_MATERIAL should be used to describe the bumped terminal.

8.10.2 BUMP_HEIGHT Parameter

Parameter Name	**BUMP_HEIGHT**
Parameter Type	Variable
Parameter Function	Specifies the bump contact height above the passivation surface.
Parameter Values	Numeric real, in GEOMETRIC_UNITS, as 7.1.3.3
Dependencies	GEOMETRIC_UNITS
Limitations	Shall be declared only once within a single DEVICE block.
Example	`BUMP_HEIGHT = 150.0;`
Notes	Required only for device types with bump connection structures. The height of the bump can only be stated before the bump is modified by attachment and/or permanent connection.

8.10.3 BUMP_HEIGHT_TOLERANCE Parameter

Parameter Name	**BUMP_HEIGHT_TOLERANCE**
Parameter Type	Variable
Parameter Function	Specifies the tolerance of bump contact height above the passivation surface.
Parameter Values	Numeric real, in GEOMETRIC_UNITS, as 7.1.3.3
Dependencies	GEOMETRIC_UNITS, BUMP_HEIGHT
Limitations	Shall be declared only once within a single DEVICE block.
Example	`BUMP_HEIGHT_TOLERANCE = 35.0;` `BUMP_HEIGHT_TOLERANCE = -10.0,25.0;`
Notes	One or two values may be given:- Where a single tolerance value is given, the tolerance shall be taken as an unsigned ± value. Where two tolerance values are given:- the first value shall be taken as the minimum, and the second value shall be taken as the maximum. Required only for device types with bump connection structures.

8.10.4 BUMP_SHAPE Parameter

Parameter Name	**BUMP_SHAPE**
Parameter Type	Variable
Parameter Function	Specifies the shape of bump the contact height.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.
Example	`BUMP_SHAPE=”Pyramidical”;` `BUMP_SHAPE=”Ball”;`
Notes	Required only for device types with bump connection structures. The shape of the bump can only be stated before the bump is modified by attachment and/or permanent connection.

8.10.5 BUMP_SIZE Parameter

Parameter Name	**BUMP_SIZE**
Parameter Type	Variable
Parameter Function	Specifies the size of bump X-Y co-ordinates.
Parameter Values	Numeric real, in GEOMETRIC_UNITS, as 7.1.3.3

Dependencies	GEOMETRIC_UNITS
Limitations	Shall be declared only once within a single DEVICE block.
Example	`BUMP_SIZE= “150,150”;`
Notes	Required only for device types with bump connection structures. The size of the bump can only be stated before the bump is modified by attachment and/or permanent connection.

8.10.6 BUMP_SPECIFICATION_DRAWING Parameter

Parameter Name	**BUMP_SPECIFICATION_DRAWING**
Parameter Type	Variable
Parameter Function	Specifies the name(s) of the bump specification drawing file(s).
Parameter Values	Textual name data, as 7.1.3.2
Example	`BUMP_SPECIFICATION_DRAWING = “BumpShape1.JPG”;` `BUMP_SPECIFICATION_DRAWING = “BS1.JPG”, “BS2.TIF”;`
Notes	Only file names, without relative or absolute path names, shall be used. Required only for device types with bump connection structures. Any drawing of the bump can only reflect the state of the bump before being modified by attachment and/or permanent connection. Multiple parameters (specification drawing files) may be introduced within a single declaration, or multiple declarations maybe be employed to the same effect. There is no scaling or positional data requirement.
Reference	Annex I, Note 3 and 4.

8.10.7 BUMP_ATTACHMENT_METHOD Parameter

Parameter Name	**BUMP_ATTACHMENT_METHOD**
Parameter Type	Variable
Parameter Function	Specifies the type of attachment method required or suggested for this type of bump.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.
Example	`BUMP_ATTACHMENT_METHOD = “Thermocompression”;` `BUMP_ATTACHMENT_METHOD = “Solder Reflow”;`
Notes	Required only for device types with bump connection structures.

8.11 MINIMALLY PACKAGED DEVICE (MPD) SPECIFIC DATA

8.11.1 MPD_PACKAGE_MATERIAL Parameter

Parameter Name	**MPD_PACKAGE_MATERIAL**
Parameter Type	Variable
Parameter Function	Specifies the package body material used as final encapsulation of the MPD.
Parameter Values	Textual string data as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.

Example	`MPD_PACKAGE_MATERIAL = “Plastic”;`
	`MPD_PACKAGE_MATERIAL = “Epoxy”;`
	`MPD_PACKAGE_MATERIAL = “Ceramic”;`
	`MPD_PACKAGE_MATERIAL = “Metal Can”;`

Notes	Generally required for assembly purposes.

8.11.2 MPD_PACKAGE_STYLE Parameter

Parameter Name	**MPD_PACKAGE_STYLE**
Parameter Type	Variable
Parameter Function	Specifies the code for the package style as defined in a standard or as given by the manufacturer.
Parameter Values	Textual string data as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.

Example
```
MPD_PACKAGE_STYLE = “SOT-23”;
MPD_PACKAGE_STYLE = “MO”;
MPD_PACKAGE_STYLE = “TSOP-48”;
```

8.11.3 MPD_CONNECTION_TYPE Parameter

Parameter Name	**MPD_CONNECTION_TYPE**
Parameter Type	Variable
Parameter Function	Specifies the connection method to the MPD, such as lead, bump etc.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.
Example	`MPD_CONNECTION_TYPE = “Bump”;` `MPD_CONNECTION_TYPE = “TAB Lead”;`

8.11.4 MPD_MSL_LEVEL Parameter

Parameter Name	**MPD_MSL_LEVEL**
Parameter Type	Variable
Parameter Function	Specifies the Moisture Sensitivity Level of the MPD package, in accordance with IPC/JEDEC J-STD-033B:2007 – Handling, Packing, Shipping and Use of Moisture/Reflow Sensitive Surface Mount Devices, 2005.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Shall be declared only once within a single DEVICE block.
Example	`MPD_MSL_LEVEL = “3”;`

8.11.5 MPD_PACKAGE_DRAWING Parameter

Parameter Name	**MPD_PACKAGE_DRAWING**
Parameter Type	Variable
Parameter Function	Specifies the name(s) of the MPD package specification drawing file(s).
Parameter Values	Textual name data, as 7.1.3.2
Example	`MPD_PACKAGE_DRAWING = “PackageOutline.PDF”;` `MPD_PACKAGE_DRAWING = “PO1.JPG”,”PO1A.GIF”;`
Notes	Only file names, without relative or absolute path names, shall be used. Multiple parameters (package drawing files) may be introduced within a single declaration, or multiple declarations maybe be employed to the same effect. There is no scaling or positional data requirement.
Reference	Annex I, Notes 3 and 4.

8.12 QUALITY, RELIABILITY and TEST DATA

8.12.1 QUALITY Parameters

Parameter Name	**QUAL_** ***identifier***
Parameter Type	Variable
Parameter Function	General text parameter, to enable the user to include relevant and identified quality and reliability parameters.
Parameter Values	Textual string data, as 7.1.3.1
Limitations	Each identifier shall be declared only once within a single DEVICE block.
Examples	`QUAL_OUTGOING_QUALITY_LEVEL = “….”;`
Notes	Recognised *QUAL_identifiers* are: 8.12.1.1 QUAL_OUTGOING_QUALITY_LEVEL 8.12.1.2 QUAL_OUTGOING_QUALITY_UNITS 8.12.1.3 QUAL_OUTGOING_QUALITY_DESCRIPTION 8.12.1.4 QUAL_RELIABILITY_VALUE 8.12.1.5 QUAL_RELIABILITY_UNITS 8.12.1.6 QUAL_RELIABILITY_REFERENCE

8.12.1.7 QUAL_RELIABILITY_CONDITIONS
8.12.1.8 QUAL_RELIABILITY_CALC_METHOD
8.12.1.9 QUAL_STANDARDS_COMPLIANCE

8.12.2 TEST Parameters

Parameter Name **TEST_ *identifier***
Parameter Type Variable
Parameter Function General text parameter, to enable the user to include relevant and identified test related parameters.
Parameter Values Textual string data, as 7.1.3.1
Limitations Each identifier shall be declared only once within a single DEVICE block.
Examples
```
TEST_ADDITIONAL_REGUIREMENTS = “NONE”;
TEST_SCREEN_COMPLIANCE = ”MIL-883B”;
```

Notes Recognised *TEST_identifiers* are:

8.12.2.1 TEST_ELECTRICAL_CONDITIONS
8.12.2.2 TEST_ADDITIONAL_SCREENING
8.12.2.3 TEST_TESTABILITY_FEATURES
8.12.2.4 TEST_ADDITIONAL_REQUIREMENTS
8.12.2.5 TEST_YIELD_CODE
8.12.2.6 TEST_FLOW
8.12.2.7 TEST_TEMP
8.12.2.8 TEST_SCREEN
8.12.2.9 TEST_SCREEN_COMPLIANCE

8.13 OTHER DATA

8.13.1 TEXT Parameters

Parameter Name **TEXT_*-identifier***
Parameter Type Variable
Parameter Function General text parameter, to enable the user to include relevant and identified text not otherwise covered.
Parameter Values Textual string data, as 7.1.3.1
Limitations Each identifier shall be declared only once within a single DEVICE block.
Examples
```
TEXT_SPECIAL_REQUIREMENTS = “N/A”;
```

Notes Recognised *TEXT_identifiers* are:

8.13.1.1 TEXT_PRODUCT_STATUS
8.13.1.2 TEXT_FORM_OF_SUPPLY
8.13.1.3 TEXT_SPECIAL_REQUIREMENTS
8.13.1.4 TEXT_SPECIFIC_REQUIREMENTS
8.13.1.5 TEXT_STORAGE_CONDITIONS
8.13.1.6 TEXT_STORAGE_DURATION
8.13.1.7 TEXT_LONGTERM_STORAGE
8.13.1.8 TEXT_ORIGINAL_MANUFACTURER
8.13.1.9 TEXT_ORIGINAL_DESIGN_DATE

8.14 CONTROL DATA

8.14.1 PARSE Parameters

Parameter Name **PARSE_*-identifier***
Parameter Type Variable
Parameter Function Specific parameters for use as control flags or parameters by the parsing software only, these parameters have no DEVICE related meaning nor function.

Parameter Values: Textual string data, as 7.1.3.1

Notes: The PARSE_xxx parameters are only applicable to data occurring subsequent to their invocation; this allows the PARSE_xxx parameters to be changed during parsing throughout a single DEVICE block. It is therefore strongly advised that if the PARSE_xxx parameters are to be used, the initial invocation occurs before any other data within the device BLOCK. Multi-pass parsers must accommodate this feature.
These parse parameters may be modified or overridden by other options within the parsing software, such as command-line options.
Refer to Annex K for further details.

Examples:

```
PARSE_MODE = STRICT;
PARSE_ERROR_REPORT = TERSE;
PARSE_ERROR_TRAP = ALL;
PARSE_IGNORE = NONE;
PARSE_DEFINE_PARAMETER = “A_New_Parameter”;
PARSE_DEFINE_STRUCTURE = “A_New_Structure”;
```

8.14.1.1 PARSE_MODE

The PARSE_MODE parameter selects the severity of rule and syntax checking within a device BLOCK. Refer to Annex K for more detail.

The recognised PARSE_MODE parameters are:

```
STRICT
RELAXED
ENHANCED
USER
```

8.14.1.2 PARSE_ERROR_REPORT

The PARSE_ERROR_REPORT parameter determines how the warnings and errors uncovered during parsing of the device BLOCK are reported. A cumulative report of warnings and errors is expected once all parsing has been completed, this feature is not affected by this parameter. Refer to Annex K for further details.

The recognised PARSE_ ERROR_REPORT parameters are:

OFF	No errors are reported during parsing
TERSE	Only errors are reported during parsing
VERBOSE	All warnings and errors are reported during parsing

8.14.1.3 PARSE_ERROR_TRAP

The PARSE_ERROR_TRAP parameter determines how the parsing software should proceed once an error has been uncovered. Refer to Annex K for further details.

The recognised PARSE_ ERROR_TRAP parameters are:

ALL	Parser should not halt on error.
FIRST	Parser should halt on first error.

8.14.1.4 PARSE_IGNORE

The PARSE_IGNORE parameter determines whether the data is checked or not. Extreme caution must be exercised when using this parameter. Refer to Annex K for further details.

The recognised PARSE_ IGNORE parameters are:

NONE	All parsing checks are performed.
OFF	All parsing checks are performed (as NONE)
ALL	All parsing checks are ignored.
SYNTAX_ONLY	Only syntactical errors and line-termination checks.

8.14.1.5 PARSE_DEFINE_PARAMETER

The PARSE_DEFINE_PARAMETER parameter allows for the introduction and definition of new data parameters by name prior to them being incorporated within a new release of the DDX syntax. Refer to Annex K for further details.

Example:

```
PARSE_DEFINE_PARAMETER = “My_New_Parameter”;
PARSE_DEFINE_PARAMETER = “My_Second_Parameter”;

My_New_Parameter = “some data”;
My_Second_Parameter = “some other data”;
```

If the “new” parameter name conflicts with parameter names or reserved words already in existence, a warning should be flagged up, and the “new” parameter treated as defined in the current DDX standard, i.e. the PARSE_DEFINE_PARAMETER should be flagged as a warning, then ignored. This will allow the parsing software to accept these “new” parameters once they have been ratified without re-writing the DDX file.

The “new” name shall follow the general guidelines as set out in Clauses 5 to 7, and will only be valid within that device BLOCK. The “new” parameter will then be checked for syntax and parsed as if it were a standard parameter.

The PARSE_DEFINE_PARAMETER may occur as many times as needed, each statement may only introduce a single parameter, and must precede the use of that parameter.

Until ratified and incorporated with a new release of the DDX standard, all values pertaining to the “new” parameter will be treated as textual string data, as 7.1.3.1.

8.14.1.6 PARSE_DEFINE_STRUCTURE

The PARSE_DEFINE_STRUCTURE parameter allows for the introduction of new data structures by name prior to them being incorporated within a new release of the DDX syntax. Refer to Annex K for further details.

Example:

```
PARSE_DEFINE_STRUCTURE = “A_New_Structure”;
PARSE_DEFINE_STRUCTURE = “Another_Structure”;

A_New_Structure {
  NEW_STRUCTURE = “some data”, “some more data”;
  NEW_STRUCTURE = “some data”, “some more data”;
}
```

```
Another_Structure {
  ANOTHER_NEW_STRUCT = “some data”, “some more data”;
  ANOTHER_NEW_STRUCT = “some data”, “some more data”;
}
```

If the “new” structure name conflicts with structure names or reserved words already in existence, a warning should be flagged up, and the “new” structure treated as defined in the current DDX standard, i.e. the PARSE_DEFINE_STRUCTURE should be flagged as a warning, then ignored. This will allow the parsing software to accept these “new” structures once they have been ratified without re-writing the DDX file.

The “new” name shall follow the general guidelines as set out in Clauses 5 to 7, and will only be valid within that device BLOCK. The “new” structure will then be checked for syntax and parsed as if it were a structure.

The PARSE_DEFINE_STRUCTURE may occur as many times as needed, each statement may only introduce a single structure, and must precede the use of that structure.

The “*Structure_indent_name*” may be referenced as any other structure name, but again not within existing defined structure parameters until ratified.

Until ratified and incorporated with a new release of the DDX standard, all individual values contained within the “new” structure will be treated as textual string data, as 7.1.3.1.

Annex A
(informative)

An example of a DDX DEVICE block

```
DEVICE 7995 bare_die {
#
# Initial header data, with block and device history.
#
BLOCK_CREATION_DATE = “2000-12-25”;
BLOCK_VERSION = 1.0;
MANUFACTURER = “Fuzziwuzz Logic Ltd.”;
FUNCTION = “Special gate”;
DATA_SOURCE = “GOOD-DIE database”;
DATA_VERSION = “Initial Issue A”;
VERSION = “1.2.2”;

#
# Declaration of geometric view, units, size etc.,
#
GEOMETRIC_UNITS = millimetre;
GEOMETRIC_VIEW = “top”;
SIZE = 1.312, 1.050;
SIZE_TOLERANCE = 0.00, 0.0005, 0.00 0.0005;
THICKNESS = 0.360;
THICKNESS_TOLERANCE = 0.00, 0.0007;
GEOMETRIC_ORIGIN = 0,0;

#
# Additional details of Die type and usage.
#
DI*E_NAME = “XXZ322”;
DIE_MASK_REVISION = “Mask 1.0”;
MAX_TEMP = 280;
POWER_RANGE = 0.500;
DIE_SUBSTRATE_MATERIAL = “Silicon”;
DIE_TERMINAL_MATERIAL = “Al”;
IC_TECHNOLOGY = “bipolar”;
DIE_SUBSTRATE_CONNECTION = “Ground”;

#
# Delivery details.
#
DIE_BACK_DETAIL  = “Back-Lapped”;
DIE_DELIVERY_FORM = “Die, Wafer”;
WAFER_SIZE = “4 inch”;

#
# Definition of the number of bond pad types, bond pads
# and connections.
#
TERMINAL_TYPE_COUNT    = 5;
TERMINAL_COUNT         = 8;
CONNECTION_COUNT       = 14;
```

```
#
# Definition of the bond pad shapes and dimensions.
#
TERMINAL_TYPE  {
      PADR1 = Rectangle, 0.144, 0.104;
      PADR2 = Rectangle, 0.264, 0.104;
      PADR3 = Rectangle, 0.084, 0.084;
      PADC1 = Circle, 0.100;
      PADP1 = Polygon, (-0.0175,-0.042),(-0.042,-0.0175),
      (-0.042, 0.0175),(-0.0175, 0.042),
      ( 0.0175, 0.042),( 0.042, 0.0175),
      ( 0.042,-0.0175),( 0.0175,-0.042);
      }

#
# Bond pad placement, naming, orientation and connectivity
# details.
#
TERMINAL {
      T1 = 1 ,PADC1,-0.550, 0.416,  0,VCCA   ,P;
      T2 = 3 ,PADP1,-0.502, 0.190,  0,INPUTA ,I;
      T3 = 4 ,PADP1,-0.502,-0.192,  0,INPUTB ,I;
      T4 = 7 ,PADC1,-0.399,-0.442,  0,GNDA   ,G;
      T5 = 8 ,PADR2, 0.498,-0.442,  0,GNDB   ,G;
      T6 = 11,PADR3, 0.511,-0.171,  0,OUTPUTA,O;
      T7 = 12,PADR3, 0.511, 0.171,  0,OUTPUTB,O;
      T8 = 14,PADR1, 0.558, 0.416,  0,VCCB   ,P;
       }

#
# Details of a supplied pSpice simulation model.
#
SIMULATOR_SPICE_MODEL_FILE = “SP7995.MOD”;
SIMULATOR_SPICE_MODEL_FILE_DATE = “1997-09-17”;
SIMULATOR_SPICE_NAME = “pSpice”;
SIMULATOR_SPICE_VERSION = “4.0.1”;
SIMULATOR_SPICE_COMPLIANCE = “2G6”;

#
# Details of a supplied Spectre simulation model.
#
SIMULATOR_SPECTRE_MODEL_FILE = “SP7995.S”;
SIMULATOR_SPECTRE_MODEL_FILE_DATE = “1998-11-05”;
SIMULATOR_SPECTRE_NAME = “Spectre”;
SIMULATOR_SPECTRE_VERSION = “4.2.1, 1992”;
SIMULATOR_SPECTRE_COMPLIANCE = “2G6, Level-3”;

#
# Details of reference fiducial, supplied as a JIF graphic file
#
FIDUCIAL_TYPE fiduc1 = “7995FID1.JIF”, 0.072, 0.055;
FIDUCIAL F1 = fiduc1, -0.612, 0.470, 0;

# End of block
}
```

Annex B
(informative)

Groups and Permutation

A. Specific Rules for the TERMINAL_GROUP parameter

Rule A1. ***Content*** (refer to 8.4.6.1.1) – The elements forming a group may be terminal identifiers or group identifiers or any mixture of the two. A group must contain at least two elements.

Rule A2. ***Uniqueness*** (refer to 8.4.6.1.2) – All elements within a group must be unique. A group may not contain groups that refer directly or indirectly to the same element.

Rule A3. ***Ordering*** (refer to 8.4.6.1.3) – the order of elements within groups may be important. Where two or more groups are to be related in a permutation, then the elements within those groups must correspond across the groups.

Rule A4. ***Recursion*** (refer to 8.4.6.1.4) – A group may not contain itself nor may it contain any group that refers directly or indirectly to itself.

B. Specific Rules for the PERMUTABLE parameter

Rule B1. ***Content*** (refer to 8.4.7.1.1) – The elements forming a permutation may be terminal identifiers or group identifiers but not a mixture of the two. A permutation must contain at least two elements.

Rule B2. ***Uniqueness*** (refer to 8.4.7.1.2) – All elements within a permutation must be unique. A permutation may not contain groups that refer directly or indirectly to the same element. Each element within a permutation must contain the same number of Terminal identifiers (whether within Terminal Groups or not) as each other.

Rule B3. ***Ordering*** (refer to 8.4.7.1.3) – the order of elements within a permutation is not important.

The simplest method of describing the use and function of the TERMINAL_GROUP and PERMUTABLE parameters is by analysing a simple example. Here, a fragment of a DDX file for a 7400 device is given. The 7400 device is a quad 2-input NAND gate, and so for functionality, all four NAND gates are exchangeable. Throughout the analysis we refer to terminals, but for a packaged part the user may consider the term “pins” to be more appropriate.

```
DEVICE 74ACT00 bare_die {

  BLOCK_CREATION_DATE = "13/02/2006";
  BLOCK_VERSION = 1.0;
  DEVICE_NAME = "74ACT00";
  MANUFACTURER = "Fuzziwuzz Logic Ltd";
  FUNCTION = "Quad two-input advanced CMOS NAND gate";
  DEVICE_FORM = "bare_die";
  DATA_SOURCE = "GOOD-DIE Project database";
  VERSION = "1.3.0";
  GEOMETRIC_UNITS = micrometre;
  GEOMETRIC_VIEW = "top";
  SIZE = 1067, 1143;
  THICKNESS = 356;
  GEOMETRIC_ORIGIN = 0,0;
  DIE_NAME = "74ACT00";
  DIE_MASK_REVISION = "Mask T";
  MAX_TEMP = 150;
```

```
  POWER_RANGE = 0.2;
  IC_TECHNOLOGY = "CMOS";
  DIE_SEMICONDUCTOR_MATERIAL = "silicon";
  DIE_SUBSTRATE_CONNECTION = "CONN, Vcc";
  DIE_DELIVERY_FORM = "Die, wafer";
  TERMINAL_TYPE_COUNT = 1;

  TERMINAL_TYPE  {
      PADR1 = Rectangle, 97, 97;
      }

  TERMINAL_COUNT = 14;

  TERMINAL  {
      T_1 = 1, PADR1, -385, 422, 0, A1, I;
      T_2 = 2, PADR1, -385, 176, 0, B1, I;
      T_3 = 3, PADR1, -385, 11, 0, Y1, O;
      T_4 = 4, PADR1, -385, -236, 0, A2, I;
      T_5 = 5, PADR1, -208, -423, 0, B2, I;
      T_6 = 6, PADR1, -43, -423, 0, Y2, O;
      T_7 = 7, PADR1, 123, -423, 0, GND, G;
      T_8 = 8, PADR1, 385, -423, 0, Y3, O;
      T_9 = 9, PADR1, 385, -166, 0, B3, I;
      T_10 = 10, PADR1, 385, -1, 0, A3, I;
      T_11 = 11, PADR1, 385, 164, 0, Y4, O;
      T_12 = 12, PADR1, 385, 423, 0, B4, I;
      T_13 = 13, PADR1, 38, 423, 0, A4, I;
      T_14 = 14, PADR1, -129, 423, 0, VCC, P;
      }

TERMINAL_GROUP  {
      NAND_INA = T_1, T_2;
      NAND_INB = T_4, T_5;
      NAND_INC = T_9, T_10;
      NAND_IND = T_12, T_13;
      NAND_A = NAND_INA, T_3;
      NAND_B = NAND_INB, T_6;
      NAND_C = NAND_INC, T_8;
      NAND_D = NAND_IND, T_11;
      }

PERMUTABLE {
      P_1 = T_1, T_2;
      P_2 = T_4, T_5;
      P_3 = T_9, T_10;
      P_4 = T_12, T_13;
      P_5 = NAND_A, NAND_B, NAND_C, NAND_D;
      }
}
```

The 7400 device has 4 identical gates, Gates A to D, with the following characteristics:

7400 Device	INPUT1	INPUT2	OUTPUT
Gate A	T_1	T_2	T_3
Gate B	T_4	T_5	T_6
Gate C	T_9	T_10	T_8
Gate D	T_12	T_13	T_11

Regarding the DDX fragment above, we see that `TERMINAL_GROUP NAND_INA` groups the inputs for Gate A together, `NAND_INB` for Gate B and so on …

`TERMINAL_GROUP NAND_A` similarly groups the inputs for Gate A along with the relevant output terminal, and again `NAND_B` to `NAND_D` follow the Gates B to D in similar fashion. Note

that the mixing of terminal-groups and terminals within the same statement (in accordance with **Rule A1**), and that the terminal ordering (**Rule A3**) of input terminals and then output terminal, is adhered to.

Finally, the PERMUTABLE statement can be analysed thus:

- `P_1` states that the terminals `T_1` and `T_2` (Gate A) are interchangeable.
- `P_2` states that the terminals `T_4` and `T_5` (Gate B) are interchangeable.
- `P_3` states that the terminals `T_9` and `T_10` (Gate C) are interchangeable.
- `P_4` states that the terminals `T_12` and `T_13` (Gate D) are interchangeable.
- `P_5` states that groups `NAND_A` to `NAND_D` (Gates A to D) are interchangeable, so long as the sequence order of the terminals is adhered to.

Note that in the above example, we use "`P_1 = T_1, T_2;`" and not "`P_1 = NAND_INA;`", as the latter statement would violate **Rule B1** in not having at least two elements.

As **Rule B3** states, the ordering of the elements is not important, such that the statement

```
P_5 = NAND_A, NAND_B, NAND_C, NAND_D;
```

shall have the same interpretation as

```
P_5 = NAND_D, NAND_C, NAND_B, NAND_A;
```

As **Rule B2** states, the following would **NOT** be acceptable

```
P_7 = NAND_D, T_1, T_2;        (unequal elements, also breaks Rule B1)
P_8 = NAND_INA, NAND_B;        (unequal elements)
P_9 = NAND_INA, NAND_A;        (unequal elements and possible recursion)
P_10 = NAND_A, NAND_A; (possible recursion)
```

Annex C
(informative)

A Typical CAD view from the DDX file block example given in Annex A

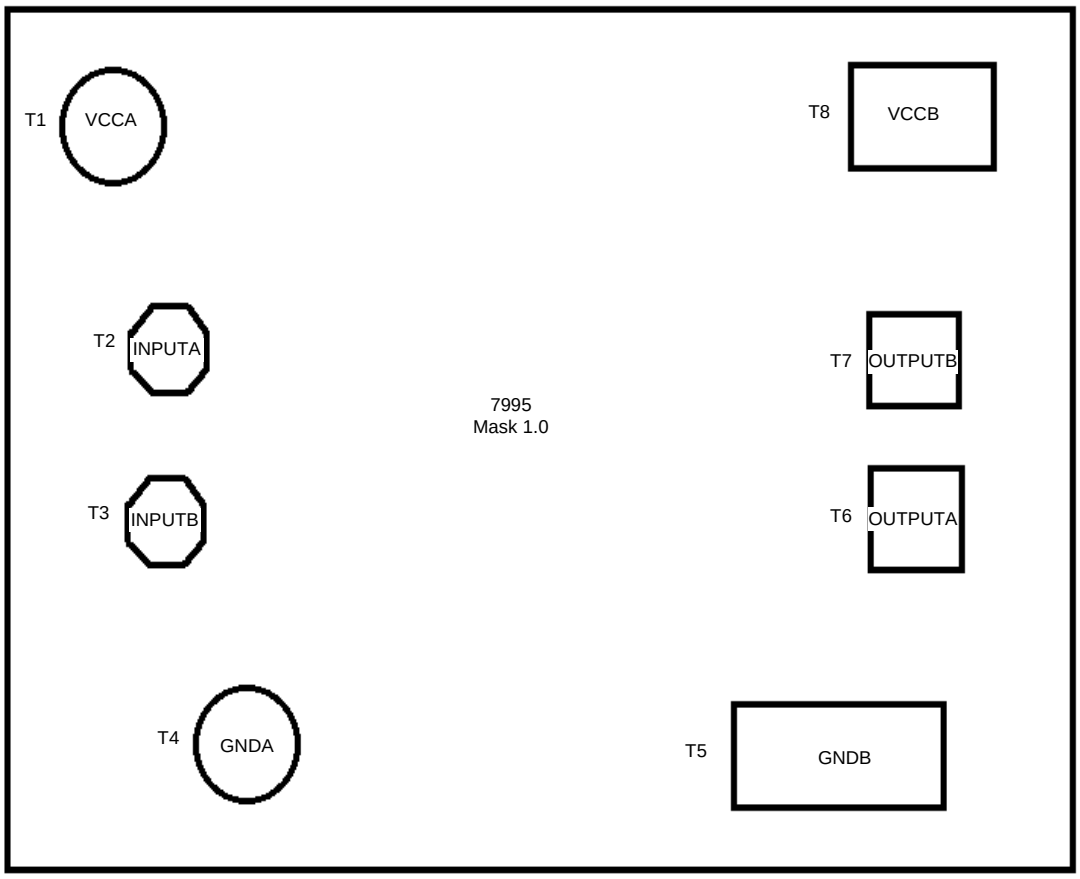

Figure C.1 – CAD representation of DDX example from Annex A

Note that Figure C.1. is not intended to be a geometrically accurate drawing, the purpose is solely to assist visualise the output from the DDX example given in Annex A. (The Die Fiducial graphic is not shown). The CAD system has chosen to display the **DEVICE_NAME** (refer to 8.1.1) and the **DIE_MASK_REVISION** (refer clause 8.2.3) parameters for the die. The individual terminal identifiers and terminal names have similarly been selected for display (refer to 8.4.5 for details).

Annex D
(informative)

Properties for Simulation

A DDX file can reference other external files, such as electrical simulation model files, mechanical model files and thermal modelling files.

To do this adequately, the following data shall be present:

- the file name and its creation date,
- the exact name of the simulator,
- its version
- the relevant compliance level to which this model applies,
- The terminals to which the model applies (electrical).

Within the model file there should be a textual note relating to the model's capability, applicability and accuracy, specifically noting any shortcomings of the model and its usage. The text "***simulator***" shall be replaced with either the actual or generic name of the simulator used and/or model data. Typical "***simulator"*** names might be Verilog, VHDL, SPICE, ELDO, BSDL, IBIS etc.

8.7.1 **SIMULATOR_*simulator*_MODEL_FILE**
Data presented as a file name. All file names given shall not include any absolute computer drive, computer path reference, or any details that cause the information to become computer or operating system specific or dependent.

8.7.2 **SIMULATOR_*simulator*_MODEL_FILE_DATE**
Data presented as a date, as per ISO 8601.

8.7.3 **SIMULATOR_*simulator*_NAME**
Data presented as text of the actual simulator used to prove the model.

8.7.4 **SIMULATOR_*simulator*_VERSION**
Data presented as text determining the actual version or revision of the simulator against which the model file was proven.

8.7.5 **SIMULATOR_*simulator*_COMPLIANCE**
Data presented as text which reflects the overall compliance of the simulator model, where applicable. "2G6", "VHDL '93", "IEEE 1364-2001" or "IEEE 1076.3-97" are examples of industry recognised compliance levels.

8.7.6 **SIMULATOR_*simulator*_TERM_GROUP**
Determines the terminals or terminal groups to which the simulator applies when the model file only covers a limited range of terminals, otherwise it is assumed that the model file covers the entire device. This parameter therefore permits the restriction of the simulation model to specific terminals or groups of terminals, and by it's absence, implies that the simulation model is applicable to all device terminals.

Example:

```
SIMULATOR_SPICE_MODEL_FILE = "RTBAE2.MOD";
SIMULATOR_SPICE_MODEL_FILE_DATE = "1997-09-17";
SIMULATOR_SPICE_NAME = "pSpice";
```

```
SIMULATOR_SPICE_VERSION = “4.0.1”;
SIMULATOR_SPICE_COMPLIANCE = “2G6”;
TERMINAL_GROUP G_1 = P8, P9, P10;
TERMINAL_GROUP G_2 = P28, P29, P30;
SIMULATOR_SPICE_TERM_GROUP = T_1, T_2, G_1, G_2;
```

Annex E
(informative)

TERMINAL and TERMINAL_TYPE graphical usage for CAD/CAM systems

The creation of a CAD/CAE "view" of the device from the data within the **DEVICE** block may be considered as having four separate steps. There is no geometric layer information contained within the DDX format, and it is assumed that all geometric data shall be displayed on a single layer. Textual information may be provided on a separate layer.

The **GEOMETRIC_UNITS** parameter (see 8.3.1) defines the appropriate scaling factors to be employed, and sets the scene for all further numerical input from that **DEVICE** block. The **GEOMETRIC_VIEW** parameter (see 8.3.2) determines whether the device is being viewed from the top or bottom.

The device outline is created from the **SIZE** parameter (see 8.3.4), and any geometric offset for further geometric placement is calculated from the **GEOMETRIC_ORIGIN** parameter (see 8.3.3). Data such as device name and type is collected and prepared for display purposes. Note that the **GEOMETRIC_ORIGIN** parameter is referenced to the XSIZE and YSIZE parameters, before tolerancing, as XSIZE/2, YSIZE/2. The **GEOMETRIC_ORIGIN** parameter values are added to this reference for all relative geometric parameters. The **SIZE_TOLERANCE** parameter (see 8.3.5) values may also be used to indicate the minimum and maximum device outline.

The **TERMINAL_COUNT**, **TERMINAL_TYPE_COUNT** and **CONNECTION_COUNT** parameters (see 8.4.1, 8.4.2 and 8.4.3) are essentially duplicated, as their values may be calculated from the **TERMINAL** (see 8.4.5) and **TERMINAL_TYPE** (see 8.4.4) structures. They are intended, however, to be used as a "sanity" check, to ensure correct parsing, and highlight any file or BLOCK corruption. The **TERMINAL_TYPE_COUNT** value can only be less than or equal to the **TERMINAL_TYPE** value. The **CONNECTION_COUNT** value has no such restriction.

The shapes of the terminals or connecting areas are separately defined, each shape having its own geometric reference centre. These shapes are defined with the **TERMINAL_TYPE** structure. Each of these terminal shapes can be placed anywhere within the device outline and, as there is no placement limitation, may be placed outside the device outline should the user so desire (e.g. pad placement for PCB-type connections). The **TERMINAL_TYPE** structure introduces **Terminal_Type_Names** (see 8.4.4.1 and 8.4.5.3) that shall be unique within the **DEVICE** block; these names are placeholders for subsequent usage by the **TERMINAL** structure, and so should be introduced prior to the **TERMINAL** structure statement using the placeholder name to permit single pass parsing of the **DEVICE** block. To prevent any possible ambiguity, it is strongly advised that the **Terminal_Type_Names** be limited to alphanumeric characters.

These terminals types (shapes), having been defined, may now be referenced by their **Terminal_Type_Name**, and geometrically placed by the **TERMINAL** structure in sequence (***T_n***), at given locations (the X & Y co-ordinates), orientated by mirroring (Figure E.1) and clockwise rotation (Figure E.2), and finally assigned a terminal name and I/O function for further use by the CAD system. The **Terminal_name** (see 8.4.5.7) would usually be visible as text, for visual convenience and need not be unique. The **Terminal_name** can also indicate terminals that will usually be electrically connected together, such as *"VDD", "VSS", "Ground"* etc.. A connection number (***conn_N***) may also be assigned and whilst this number bears no relation to the actual terminal number, it can be used for identifying common groups of pins, (such as pins that must be connected together), or actually referring to the original device's packaged pin-out (which again can be different to the die pad numbering). The functional I/O details (***IO_type***) (see 8.4.5.8) of each terminal are available for further connectivity checks by the CAD system. The unique **TERMINAL** identifier, (***T_n)*** (see 8.4.5.1)**,** can have any mix of

alphanumeric characters, and need not be in, nor follow, any particular sequence. The only requisite is that, as an identifier, it is unique within that **DEVICE** block, and to prevent any possible ambiguity, it is strongly advised that the unique identifier, (***T_n)*** be limited to alphanumeric characters.

Where applicable, the **DIE_SUBSTRATE_CONNECTION** (see 8.5.5) details should be displayed, alerting the user to the need to facilitate appropriate connection.

The creation and placement of fiducials is identical to that for terminals by using the **FIDUCIAL_TYPE** (see 8.3.8) and **FIDUCIAL** (see 8.3.9) parameter structures (q.v.). The main differences being that

- graphical fiducial data are only held within a separate graphic file, and
- that fiducials do not have any connectivity or electrical properties.

It is up to the CAD/CAM system to determine as to how to display the graphic content of the fiducial graphic file, as there are currently a plethora of graphical file standards. The fiducial graphic shall be scaled to fit within the size dimensions given by the **FIDUCIAL_TYPE** parameter structure, whose reference and placement centre will be taken as the midpoint of the X- and Y-size dimensions. The actual content and type of graphical data within this file is commonly determined by the file extent. i.e. TIFF, GIF, JPEG, BMP.

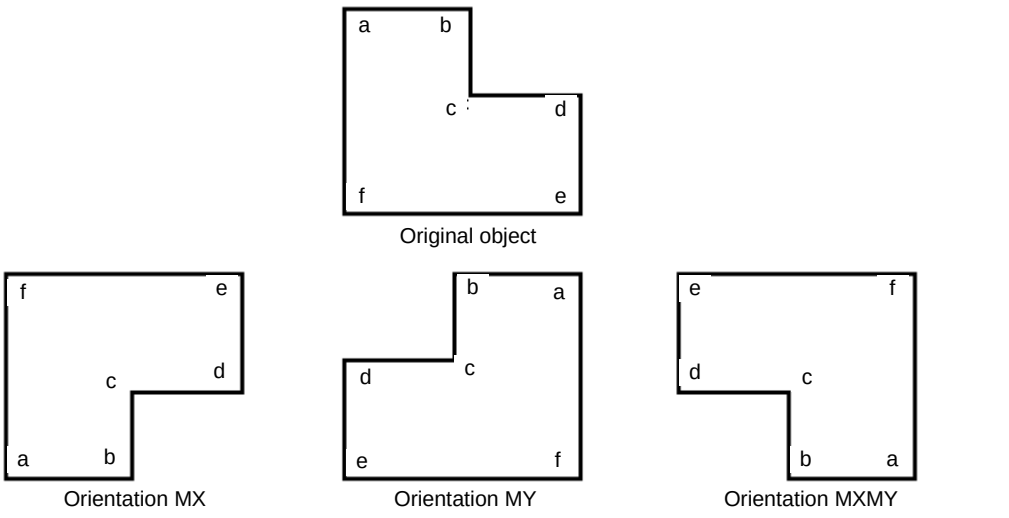

Figure E.1 – Highlighting the MX and MY orientation properties

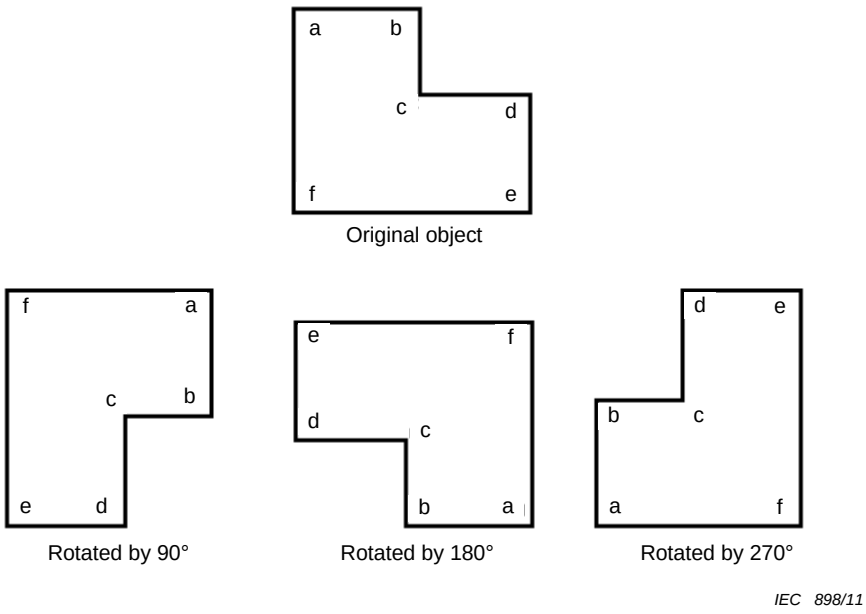

Figure E.2 – Highlighting the angular rotational orientation properties

Annex F
(informative)

Cross-reference with IEC 61360-4

The table below provides a read-across between the parameters in this standard and the DET definitions given in IEC 61360-4:2005.

Table F.1 – Parameter List

Clause/ Sub-clause	Parameter Name	Parameter Type	Data Type	DET Code
8.1	***BLOCK DATA***			
8.1.1	DEVICE_NAME	Header	Text	AAH547-001
8.1.2	DEVICE_FORM	Header	Text	AAD004-001
8.1.3	BLOCK_VERSION	Variable	Text	
8.1.4	BLOCK_CREATION_DATE	Variable	Date	
8.1.5	VERSION	Variable	Text	
8.2	***DEVICE_DATA***			
8.2.1	DIE_NAME	Variable	Text	AAD002-001
8.2.2	DIE_PACKAGED_PART_NAME	Variable	Text	AAD143-001
8.2.3	DIE_MASK_REVISION	Variable	Text	AAD003-001
8.2.4	MANUFACTURER	Variable	Text	AAD140-001
8.2.5	DATA_SOURCE	Variable	Text	AAD142-001
8.2.6	DATA_VERSION	Variable	Text	
8.2.7	FUNCTION	Variable	Text	
8.2.8	IC_TECHNOLOGY	Variable	Text	AAE686-005
8.2.8	DEVICE_PICTURE_FILE	Variable	File Name	
8.2.9	DEVICE_DATA_FILE	Variable	File Name	
8.3	***GEOMETRIC DATA***			
8.3.1	GEOMETRIC_UNITS	Variable	Real	AAD115-001
8.3.2	GEOMETRIC_VIEW	Variable	Text	AAD144-001
8.3.3	GEOMETRIC_ORIGIN	Variable	Real	AAD129 & 130-001
8.3.4	SIZE	Variable	Real	AAD070 & 071-001
8.3.5	SIZE_TOLERANCE	Variable	Real	AAD117-001
8.3.6	THICKNESS	Variable	Real	AAD072-001
8.3.7	THICKNESS_TOLERANCE	Variable	Real	AAD118-001
8.3.8	FIDUCIAL_TYPE	Structure		AAD156 to 159-001
8.3.9	FIDUCIAL	Structure		AAD160 to 162-001
8.4	***TERMINAL DATA***			
8.4.1	TERMINAL_COUNT	Variable	Integer	AAD145-001
8.4.2	TERMINAL_TYPE_COUNT	Variable	Integer	AAD116-001
8.4.3	CONNECTION_COUNT	Variable	Integer	
8.4.4	TERMINAL_TYPE	Structure		AAD024 to 030, AAD121-001

Clause/ Sub-clause	Parameter Name	Parameter Type	Data Type	DET Code
8.4.5	TERMINAL	Structure		AAD014 to 016, AAD019, AAD020-001
8.4.6	TERMINAL_GROUP	Structure		
8.4.7	PERMUTABLE	Structure		
8.5	***MATERIAL DATA***			
8.5.1	TERMINAL_MATERIAL (was DIE_TERMINAL_MATERIAL)	Variable	Text	AAD120-001, AAE634-005
8.5.2	TERMINAL_MATERIAL_STRUCTURE	Variable	Text	
8.5.3	DIE_SEMICONDUCTOR_MATERIAL	Variable	Text	AAD148-001
8.5.4	DIE_SUBSTRATE_MATERIAL	Variable	Text	AAD005-001
8.5.5	DIE_SUBSTRATE_CONNECTION	Variable	Text	AAD006 & 007-001
8.5.6	DIE_PASSIVATION_MATERIAL	Variable	Text	AAD078-001
8.5.7	DIE_BACK_DETAIL	Variable	Text	AAD119-001
8.6	***ELECTRICAL and THERMAL DATA***	Variable		
8.6.1	MAX_TEMP	Variable	Real	AAD149-001
8.6.2	MAX_TEMP_TIME	Variable	Real	
8.6.3	POWER_RANGE	Variable	Real	AAD151-001
8.6.4	TEMPERATURE_RANGE	Variable	Real	AAE891-005
8.7	***SIMULATOR DATA***			
8.7.1	SIMULATOR_*simulator*_MODEL_FILE	Variable	File Name	
8.7.2	SIMULATOR_*simulator*_MODEL_FILE_DATE	Variable	Text	
8.7.3	SIMULATOR_*simulator*_NAME	Variable	Text	
8.7.4	SIMULATOR_*simulator*_VERSION	Variable	Text	
8.7.5	SIMULATOR_*simulator*_COMPLIANCE	Variable	Text	
8.7.6	SIMULATOR_*simulator*_TERM_GROUP	Variable		
8.8	***HANDLING, PACKING, STORAGE and ASSEMBLY***			
8.8.1	DELIVERY_FORM (was DIE_DELIVERY_FORM)	Variable	Text	AAD155-001
8.8.2	PACKING_CODE	Variable	Text	AAD055-001
8.8.8	ASSEMBLY parameters	Variable	Text	
8.9	***WAFER SPECIFIC DATA***			
8.9.1	WAFER_SIZE	Variable	Text	AAD011-001
8.9.2	WAFER_THICKNESS	Variable	Real	
8.9.3	WAFER_THICKNESS_TOLERANCE	Variable	Real	
8.9.4	WAFER_DIE_STEP_SIZE	Variable	Real	AAD163-001, AAD164-001
8.9.5	WAFER_GROSS_DIE_COUNT	Variable	Integer	AAD165-001
8.9.6	WAFER_INDEX	Variable	Real	AAD166-001, AAD167-001
8.9.7	WAFER_RETICULE_STEP_SIZE	Variable	Real	AAD168-001, AAD169-001
8.9.8	WAFER_RETICULE_GROSS_DIE_COUNT	Variable	Integer	AAD170-001

Clause/ Sub-clause	Parameter Name	Parameter Type	Data Type	DET Code
8.9.9	WAFER_INK PARAMETERS	Variable	Text	
8.10	***BUMP TERMINATION SPECIFIC DATA***			
8.10.1	BUMP_MATERIAL	Variable	Text	AAD124-001
8.10.2	BUMP_HEIGHT	Variable	Real	AAD146-001
8.10.3	BUMP_HEIGHT_TOLERANCE	Variable	Real	AAD147-001
8.10.4	BUMP_SHAPE	Variable	Text	
8.10.5	BUMP_SIZE	Variable	Real	
8.10.6	BUMP_SPECIFICATION_DRAWING	Variable	File Name	
8.10.7	BUMP_ATTACHMENT_METHOD	Variable	Text	
8.11	***MINIMALLY PACKAGE DEVICE SPECIFIC DATA***			
8.11.1	MPD_PACKAGE_MATERIAL	Variable	Text	AAD150-001
8.11.2	MPD_PACKAGE_STYLE	Variable	Text	
8.11.3	MPD_CONNECTION_TYPE	Variable	Text	
8.11.4	MPD_MSL_LEVEL	Variable	Text	
8.11.5	MPD_PACKAGE_DRAWING	Variable	File Name	
Deleted	MPD_DELIVERY_FORM (refer to 8.8.1)	Variable		
Deleted	MPD_CONNECTION_MATERIAL (refer to 8.5.1)	Variable		
8.12	***QUALITY, RELIABILITY and TEST DATA***			
8.12.1	QUALITY Parameters	Variable	Text	
8.12.1	TEST Parameters	Variable	Text	
8.13	***OTHER DATA***			
8.13.1	TEXT	Variable	Text	

Annex G
(informative)

Notes on VERSION and NAME parameters

There are three “VERSION” related parameters, each of which contains specific revision data concerned with the different aspects of the data presented within a DDX BLOCK.

VERSION Subclause 8.1.5
This relates to the IEC 62258, Part 2, and its issue. Being a data transfer standard for use with CAD/CAE systems, it is anticipated that there will be updates and revisions, particularly the inclusion of additional data, on a regular basis. Refer to Clause 1 to determine the DDX version covered in this standard.

BLOCK_VERSION Subclause 8.1.3
This parameter relates solely to the revision status of the data with the BLOCK, and would expect to be “up-revved” as data within the BLOCK is modified. The **BLOCK_VERSION** parameter is specifically NOT concerned with the source of any data changes.

DATA_VERSION Subclause 8.2.6
This parameter covers the source of the data used; to indicate the technical “state-of-the-art” for the data contents. This parameter is therefore closely coupled with the **DATA_SOURCE** parameter.

BLOCK_CREATION_DATE Subclause 8.1.4
This parameter relates solely to the last date that data within the block was revised, and always accompanies, and should be linked to, the **BLOCK_VERSION** parameter.
Likewise, there are four “NAME” related parameters, each of which represents a different aspect of the name by which the device is known.

DEVICE_NAME Subclause 8.1.1
This is often referred to as the type number and is the identity normally used in data sheets and in parts lists.

DIE_NAME Subclause 8.2.1
This is the name given to the die itself and the set of masks used in its fabrication. This name is often part of the fiducials on the die surface.

DIE_MASK_REVISION Subclause 8.2.3
The revision or version code of the mask set used in fabricating the die.

DIE_PACKAGED_PART_NAME Subclause 8.2.2
This is generally similar to the DEVICE_NAME and is quoted as the name of a packaged part which usually contains the same die and has similar electrical characteristics.

Annex H
(informative)

Notes on WAFER parameters

When die are delivered as unsawn wafers, additional parameters can assist in the practicality of mechanical handling and quantity purchasing. Such parameters include the die step size, wafer size, wafer flat angle and gross die per wafer. Where the die step size is irregular, often caused by the reticule step, further parameters relating to the reticule size can become important.

Parameters relating to wafer delivery include:

WAFER_SIZE Subclause 8.9.1
This parameter gives the approximate wafer diameter. This is commonly quoted in millimetres or inches, and so need not be related to the **GEOMETRIC_UNITS** parameter.

WAFER_DIE_STEP_SIZE Subclause 8.9.4
The X and Y co-ordinates specified by this parameter are commonly used for die sawing and wafer probing, etc. The units again are determined by the **GEOMETRIC_UNITS** parameter.

WAFER_GROSS_DIE_COUNT Subclause 8.9.5
The gross die count can be an important parameter in the calculation of nett die per wafer, but does not reflect the good die per wafer, only the total of viable relevant die (whole die). Whilst theoretically there is a direct mathematical relationship between wafer and die size, test “drop-ins” etc., will affect the final count. The user should also be aware of cost-saving techniques which involve the reduction in the number of masks required that often result in fewer viable die.

WAFER_INDEX Subclause 8.9.6
This parameter pair defines the both type of physical feature present on a wafer and its orientation, which may then be used in an auto-location system in order to determine the placement and orientation of the wafer.
The first parameter specifies the type of feature, either a “flat” or “notch”, whilst the second parameter determines the approximate angle, to the nearest degree, of this index feature on the wafer with respect to the X-axis on the surface of the die. This can assist in wafer location recognition system for initial or crude orientation prior to final alignment by optical means. Where more than one flat exists on the wafer, the orientation is with respect to the major, or prime, flat, which may also be used to indicate the wafer lattice crystal [110] direction.

WAFER_RETICULE_STEP_SIZE Subclause 8.9.7
Where the reticule dimensions interfere with the regular stepping of the die size, the reticule step size should be given, in the same dimensions as the die step size (**GEOMETRIC_UNITS**).

WAFER_RETICULE_GROSS_DIE_COUNT Subclause 8.9.8
Where the **WAFER_RETICULE_STEP_SIZE** parameter is relevant, it may also be relevant to state the number of die, of the specified die type, within the reticule window. In the case of MPW runs, this may only be one die.

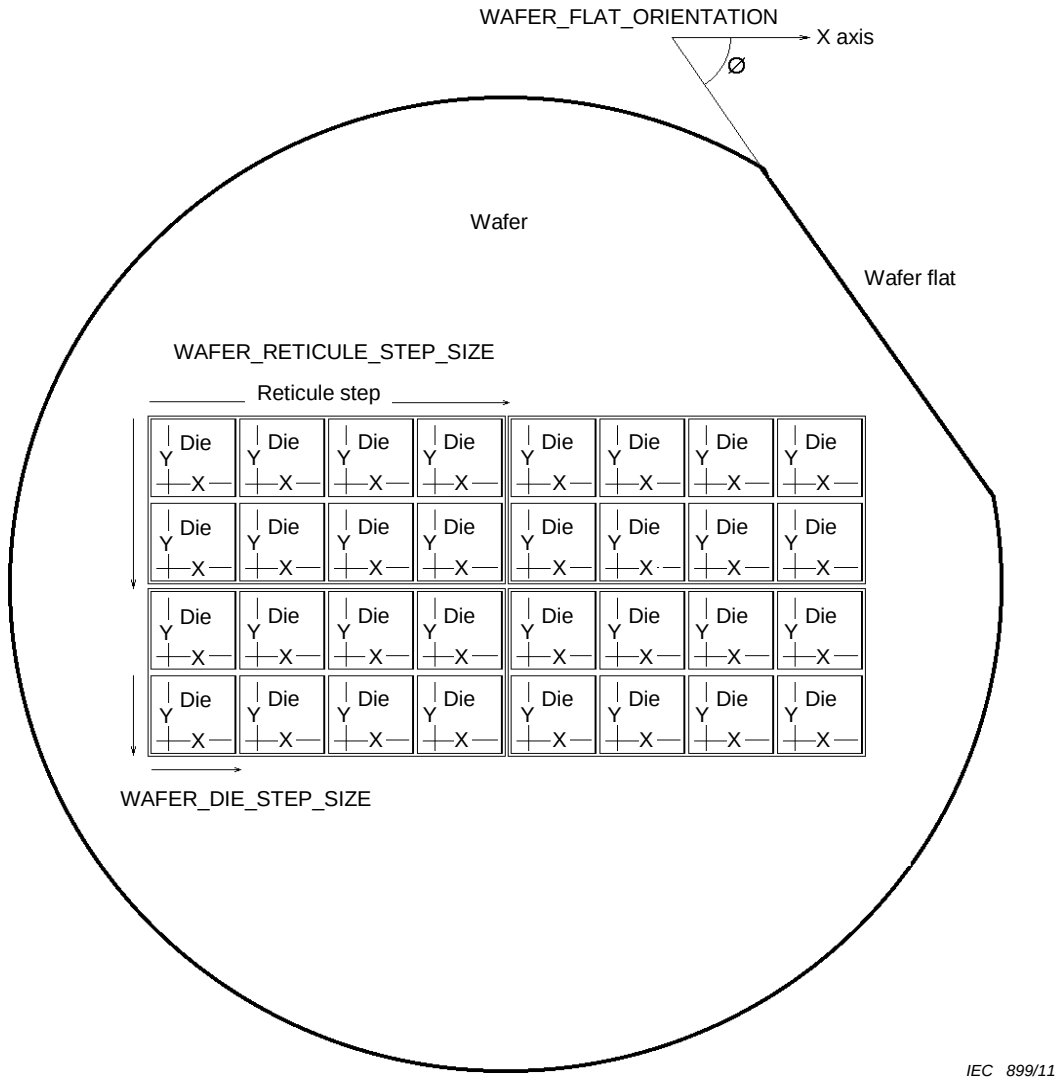

Figure H.1 – Illustrating the WAFER parameters

Figure H.1 shows a representation of where the reticule step size interferes with the die step dimensions, (the reticule here containing a total of 8 die in a 4 by 2 matrix). The gap between reticule and reticule results in an additional distance between adjacent die at the boundaries of the reticule. Figure H.1 also shows then wafer flat angle with respect to the assumed X-dimension of the die. Notional parameter values for Figure H.1 might be:

```
WAFER_SIZE = “150mm”;
WAFER_DIE_STEP_SIZE = 1000,1000;
WAFER_GROSS_DIE_COUNT = 2077;
WAFER_INDEX = “Flat”,45;
WAFER_RETICULE_STEP_SIZE = 4050, 2100;
WAFER_RETICULE_GROSS_DIE_COUNT = 8;
```

The reticule parameters given in this example here demonstrates a 50 unit discrepancy (**GEOMETRIC_UNITS)** in the X-dimension, and a 100 unit discrepancy (**GEOMETRIC_UNITS)** in the Y-dimension.

Annex I
(informative)

Additional notes

1) As it stands, the DDX format requires data to reside within a block structure, strictly delimited by braces. The only data required outside of these braces are the **DEVICE** keyword and the associated parameters of **DEVICE_NAME** and **DEVICE_FORM**. This then leaves scope for future expansion, such as the inclusion of non-DDX format data outside of the DDX block structure, provided that the DDX structure and format rules are conformed to. Such additional data may include file and block checksums to validate the data file contents, although full ASCII character checksum calculations need to take into account the differing line delimiters used by different software operating systems. This additional data should be catered for, and ignored, by any parsing routine specifically for reading DDX formatted data.

2) As there are many different character conventions for displaying or annotating a negated signal, such as “\”, “/”, “N”, “B” or “not/bar”, no particular method has been adopted in the DDX format for the display of such signal names. It is therefore reliant upon the information provider to adopt a single convention, and the CAE vendor to accommodate this convention in the graphical display, where required.

3) Referencing an external file. File name references are required by:

8.3.8.2	**FIDUCIAL_TYPE**
8.2.9	**DEVICE_PICTURE_FILE**
8.2.10	**DEVICE_DATA_FILE**
8.11.5	**MPD_PACKAGE_DRAWING**
8.10.6	**BUMP_SPECIFICATION_DRAWING**

When referencing an external file, such as a graphic, textual or document file, it is preferable that the external file type conforms to a standard format in common use. Convention holds that the document format and/or type is indicated by the named file extent.

4) Multiple file names. Multiple file names are permitted in:

8.2.9	**DEVICE_PICTURE_FILE**
8.2.10	**DEVICE_DATA_FILE**
8.11.5	**MPD_PACKAGE_DRAWING**
8.10.6	**BUMP_SPECIFICATION_DRAWING**

Where multiple file names are permitted, the file names can be introduced in a single function invocation, as:

```
DEVICE_DATA_FILE = “file1.txt”, “file2.txt”, … , “fileX.txt”;
```

or as multiple invocations of that function, as:

```
DEVICE_DATA_FILE = “file1.txt”;
DEVICE_DATA_FILE = “file2.txt”;
.
DEVICE_DATA_FILE = “fileN.txt”;
```

When using a single function invocation to introduce more than one file name, each file name should be surrounded by double quotes (as 6.3.7) and each file name shall be separated by a comma (as 6.3.2). Each line shall be terminated by a semicolon as 6.3.1.

Annex J
(informative)

DDX Version history

The following table, Table J.1, indicates the parameter version history. The issue column reflects the current DDX revision and version number at which that parameter was either introduced or last altered.

The current version for this DDX format is 1.3.0, as stated in Clause 1 of this standard.

The previous version of this DDX format was 1.0, as defined in ES59008, Part 6-1.

Table J.1 – Parameter Change History List

Clause/ Subclau-se	Parameter Name	ES59008-6-1 Reference	Previous Clause	Current Issue
8.1	***BLOCK DATA***			
8.1.1	DEVICE_NAME	8.2	8.5	1.0
8.1.2	DEVICE_FORM	8.1	8.4	1.0
8.1.3	BLOCK_VERSION	8.17	8.2	1.0
8.1.4	BLOCK_CREATION_DATE	8.16	8.1	1.0
8.1.5	VERSION	8.3	8.3	1.0
8.2	***DEVICE_DATA***			
8.2.1	DIE_NAME		8.8	1.2.1
8.2.2	DIE_PACKAGED_PART_NAME	8.36	8.9	1.0
8.2.3	DIE_MASK_REVISION	8.30	8.6	1.0
8.2.4	MANUFACTURER	8.4	8.7	1.0
8.2.5	DATA_SOURCE	8.19	8.11	1.0
8.2.6	DATA_VERSION	8.55	8.12	1.0
8.2.7	FUNCTION	8.5	8.10	1.0
8.2.8	IC_TECHNOLOGY	8.20	8.25	1.0
8.2.8	DEVICE_PICTURE_FILE			1.3.0
8.2.9	DEVICE_DATA_FILE			1.3.0
8.3	***GEOMETRIC DATA***			
8.3.1	GEOMETRIC_UNITS	8.6	8.13	1.0
8.3.2	GEOMETRIC_VIEW	8.7	8.14	1.0
8.3.3	GEOMETRIC_ORIGIN	8.10	8.17	1.0
8.3.4	SIZE	8.8	8.15	1.0
8.3.5	SIZE_TOLERANCE	8.21	8.18	1.0
8.3.6	THICKNESS	8.9	8.16	1.0
8.3.7	THICKNESS_TOLERANCE	8.22	8.19	1.0
8.3.8	FIDUCIAL_TYPE	8.47	8.51	1.0
8.3.9	FIDUCIAL	8.48	8.52	1.0
8.4	***TERMINAL DATA***			

Clause/ Subclause	Parameter Name	ES59008-6-1 Reference	Previous Clause	Current Issue
8.4.1	TERMINAL_COUNT	8.11	8.20	1.0
8.4.2	TERMINAL_TYPE_COUNT	8.12	8.21	1.0
8.4.3	CONNECTION_COUNT	8.13	8.22	1.0
8.4.4	TERMINAL_TYPE	8.14	8.23	1.0
8.4.5	TERMINAL	8.15	8.24	1.0
8.4.6	TERMINAL_GROUP		8.58	1.3.0
8.4.7	PERMUTABLE		8.61	1.3.0
8.5	***MATERIAL DATA***			
8.5.1	TERMINAL_MATERIAL (was DIE_TERMINAL_MATERIAL)		8.30	1.3.0
8.5.2	TERMINAL_MATERIAL_STRUCTURE			1.3.0
8.5.3	DIE_SEMICONDUCTOR_MATERIAL	8.34	8.26	1.0
8.5.4	DIE_SUBSTRATE_MATERIAL	8.32	8.27	1.0
8.5.5	DIE_SUBSTRATE_CONNECTION	8.31	8.28	1.0
8.5.6	DIE_PASSIVATION_MATERIAL	8.37	8.29	1.0
8.5.7	DIE_BACK_DETAIL	8.33	8.31	1.0
8.6	***ELECTRICAL and THERMAL DATA***			
8.6.1	MAX_TEMP	8.18	8.33	1.0
8.6.2	MAX_TEMP_TIME			1.3.0
8.6.3	POWER_RANGE	8.23	8.34	1.0
8.6.4	TEMPERATURE_RANGE	8.24	8.35	1.0
8.7	***SIMULATOR DATA***			
8.7.1	SIMULATOR_*simulator*_MODEL_FILE	8.25	8.36	1.0
8.7.2	SIMULATOR_*simulator*_MODEL_FILE_DATE	8.26	8.37	1.0
8.7.3	SIMULATOR_*simulator*_NAME	8.27	8.38	1.0
8.7.4	SIMULATOR_*simulator*_VERSION	8.28	8.39	1.0
8.7.5	SIMULATOR_*simulator*_COMPLIANCE	8.29	8.40	1.0
8.7.6	SIMULATOR_*simulator*_TERM_GROUP		8.59	1.3.0
8.8	***HANDLING, PACKING, STORAGE and ASSEMBLY***			
8.8.1	DELIVERY_FORM (was DIE_DELIVERY_FORM)	8.35	8.41	1.0
8.8.2	PACKING_CODE	8.46	8.42	1.0
8.8.8	ASSEMBLY parameters			1.3.0
8.9	***WAFER SPECIFIC DATA***			
8.9.1	WAFER_SIZE	8.45	8.32	1.0
8.9.2	WAFER_THICKNESS			1.3.0
8.9.3	WAFER_THICKNESS_TOLERANCE			1.3.0
8.9.4	WAFER_DIE_STEP_SIZE		8.53	1.2.1
8.9.5	WAFER_GROSS_DIE_COUNT		8.54	1.2.1
8.9.6	WAFER_INDEX		8.55	1.2.1
8.9.7	WAFER_RETICULE_STEP_SIZE		8.56	1.2.1
8.9.8	WAFER_RETICULE_GROSS_DIE_COUNT		8.57	1.2.1

Clause/ Subclause	Parameter Name	ES59008-6-1 Reference	Previous Clause	Current Issue
8.9.9	WAFER_INK			1.3.0
8.10	***BUMP TERMINATION SPECIFIC DATA***			
8.10.1	BUMP_MATERIAL	8.38	8.43	1.0
8.10.2	BUMP_HEIGHT	8.39	8.44	1.0
8.10.3	BUMP_HEIGHT_TOLERANCE	8.40	8.45	1.0
8.10.4	BUMP_SHAPE			1.3.0
8.10.5	BUMP_SIZE			1.3.0
8.10.6	BUMP_SPECIFICATION_DRAWING			1.3.0
8.10.7	BUMP_ATTACHMENT_METHOD			1.3.0
8.11	***MINIMALLY PACKAGE DEVICE SPECIFIC DATA***			
8.11.1	MPD_PACKAGE_MATERIAL	8.41	8.46	1.0
8.11.2	MPD_PACKAGE_STYLE		8.47	1.2.1
8.11.3	MPD_CONNECTION_TYPE	8.43	8.49	1.0
8.11.4	MPD_MSL_LEVEL			1.30
8.11.5	MPD_PACKAGE_DRAWING			1.30
Deleted	MPD_DELIVERY_FORM (refer to 8.8.1)	8.42	8.48	1.0
Deleted	MPD_CONNECTION_MATERIAL (refer to 8.5.1)	8.44	8.50	1.0
8.12	***QUALITY, RELIABILITY and TEST DATA***			
8.12.1	QUALITY Parameters			1.3.0
8.12.1	TEST Parameters	.		1.3.0
8.13	***OTHER DATA***			
8.13.1	TEXT		8.60	1.3.0
8.14	***CONTROL DATA***			
8.14.1	PARSE			1.3.0

Annex K
(informative)

Parse Control

The PARSE_ series of parameters are intended to add in-line control of the parsing software, primarily to allow for additions and extensions to the DDX software that have not, to date, been ratified and included in the current standard. Note that none of the PARSE_ parameters have any relevance or meaning to the device data.

To this end a number of PARSE_ parameters have been added:

8.14.1 **PARSE_MODE**
8.14.2 **PARSE_ERROR_REPORT**
8.14.3 **PARSE_ERROR_TRAP**
8.14.4 **PARSE_IGNORE**
8.14.5 **PARSE_DEFINE_PARAMETER**
8.14.6 **PARSE_DEFINE_STRUCTURE**

The **PARSE_MODE**, **PARSE_ERROR_REPORT**, **PARSE_ERROR_TRAP** and **PARSE_IGNORE** parameters may have multiple occurrences, and are intended to "switch" the desired parse mode as the DDX file or DDX block is linearly read.

The **PARSE_DEFINE_PARAMETER** and **PARSE_DEFINE_STRUCTURE** parameters serve to introduce new parameters and/or structures to the parsing software for inclusion in syntax checks, data validation and value assignment, solely for that specific DDX block. Once defined, it is expected that the new parameters and structures will fully conform to the syntax etc., as defined in Clauses 5 to 7. Until the new parameters and structures have been ratified and included in an update to this DDX standard, all data shall be treated as textual and will have no associated S.I unit.

The **PARSE_MODE** parameter is intended to control the way in which the syntax and data rules are interpreted and applied. In all cases, data that complies with this version of the DDX format (refer Clause 1) shall be free from errors or warnings. New data parameters or structures that have been introduced using the **PARSE_DEFINE**_xxx parameters may require some rule relaxation before an error-free parse result is achieved.

`PARSE_MODE = STRICT;`
Under "normal" circumstances, the _MODE parameter will default to "STRICT", and all currently compliant data must be error free, and errors will be issued if the PARSE_DEFINE options are used. These errors are to or structures than those given in the current standard,

`PARSE_MODE = RELAXED;`
The "RELAXED" value is given to permit inclusion of additional defined data that may not strictly comply with the rules as written. Instead of errors, warnings are to be issued when new parameters are included to serve notice to the user that the DDX block under scrutiny contains parameters not currently in the standard.

`PARSE_MODE = USER;`
`PARSE_MODE = ENHANCED;`
The "ENHANCED" and "USER" values are alternative options available to the discretion of the parse software provider and user.

The **PARSE_ERROR_REPORT** parameter determines the level of warnings and errors reported DURING parsing, it should have no effect on the end report. This is primarily to prevent warnings regarding parameters being used prior to their assignment, caused by the introduction of new parameters. These parameter options are self-explanatory.

```
PARSE_ERROR_REPORT = OFF;
PARSE_ERROR_REPORT = TERSE;
PARSE_ERROR_REPORT = VERBOSE;
```

The **PARSE_ERROR_TRAP** parameter determines how the parsing software should react once an error (as opposed to a warning) is discovered.

```
PARSE_ERROR_TRAP = ALL;
```
This is the expected “default” mode. The DDX block is parsed and all errors are reported.

```
PARSE_ERROR_TRAP = FIRST;
```
The “FIRST” value should cause the parsing software to halt upon the detection of the first error.

The **PARSE_IGNORE** parameter is meant to control whether checking is actually performed of part of the DDX block or not, and should accordingly be used with care.

```
PARSE_IGNORE = NONE;
PARSE_IGNORE = OFF;
```
All parsing checks are performed. This should be the default state for the parsing software.

```
PARSE_IGNORE = ALL;
```
All parsing checks are ignored, this is useful only to bypass rubbish data

```
PARSE_IGNORE = SYNTAX_ONLY;
```
Only syntactical errors and line-termination checks are performed. There are no checks on forward/backward referencing of variables, nor of parameter counts etc. This option should only be used when new parameters or structures are introduced that include variable referencing or non-textual data.

WARNINGS and ERRORS

It is advised that the following items be classed as **WARNINGS**, and all other errors classified as **ERRORS**:

WARNING 1. The occurrence of ASCII characters in the range 0x80 to 0xFF

WARNING 2. Where a data line is assumed to be terminated due to its length exceeding the parser’s line input buffer. Refer to 6.3.9.

WARNING 3. Where textual data, not enclosed within double quotes, includes line break characters, refer to 6.3.8.

WARNING 4. Where the textual file name does not conform to the character set as defined in 7.1.3.2.

WARNING 5. Where the **PARSE_DEFINE**_xxx introduces a parameter or structure that has been ratified and defined within the later versions of the DDX standard.

_____

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

_____

PRODUITS DE PUCES DE SEMICONDUCTEURS –

Partie 2: Formats d'échange de données

AVANT-PROPOS

1) La Commission Electrotechnique Internationale (CEI) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de la CEI). La CEI a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, la CEI – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de la CEI"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec la CEI, participent également aux travaux. La CEI collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
2) Les décisions ou accords officiels de la CEI concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de la CEI intéressés sont représentés dans chaque comité d'études.
3) Les Publications de la CEI se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de la CEI. Tous les efforts raisonnables sont entrepris afin que la CEI s'assure de l'exactitude du contenu technique de ses publications; la CEI ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de la CEI s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de la CEI dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de la CEI et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) La CEI elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de la CEI. La CEI n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
7) Aucune responsabilité ne doit être imputée à la CEI, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de la CEI, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de la CEI ou de toute autre Publication de la CEI, ou au crédit qui lui est accordé.
8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de la CEI peuvent faire l'objet de droits de brevet. La CEI ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de brevets et de ne pas avoir signalé leur existence.

La Norme internationale CEI 62258-2 a été établie par le comité d'études 47 de la CEI: Dispositifs à semiconducteurs.

La présente norme doit être lue en conjonction avec la CEI 62258-1.

Cette deuxième édition annule et remplace la première édition parue en 2005, dont elle constitue une révision technique.

Par rapport à la première édition, les paramètres suivants ont été mis à jour pour la présente édition:

Paragraphe	Nom du paramètre
8.2.9	DEVICE_PICTURE_FILE
8.2.10	DEVICE_DATA_FILE
8.4.6	TERMINAL_GROUP
8.4.7	PERMUTABLE
8.5.1	TERMINAL_MATERIAL (anciennement DIE_TERMINAL_MATERIAL)
8.5.2	TERMINAL_MATERIAL_STRUCTURE
8.6.2	MAX_TEMP_TIME
8.7.6	SIMULATOR_*simulator*_TERM_GROUP
8.8.3	ASSEMBLY
8.9.2	WAFER_THICKNESS
8.9.3	WAFER_THICKNESS_TOLERANCE
8.9.9	WAFER_INK
8.10.4	BUMP_SHAPE
8.10.5	BUMP_SIZE
8.10.6	BUMP_SPECIFICATION_DRAWING
8.10.7	BUMP_ATTACHMENT_METHOD
8.11.4	MPD_MSL_LEVEL
8.11.5	MPD_PACKAGE_DRAWING
8.12.1	QUALITY
8.12.2	TEST
8.13.1	TEXT
8.14.1	PARSE

Le texte de cette norme est issu des documents suivants:

FDIS	Rapport de vote
47/2085/FDIS	47/2095/RVD

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de cette norme.

Cette publication a été rédigée selon les Directives ISO/CEI, Partie 2.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de stabilité indiquée sur le site web de la CEI sous "http://webstore.iec.ch" dans les données relatives à la publication recherchée. A cette date, la publication sera

- reconduite,
- supprimée,
- remplacée par une édition révisée, ou
- amendée.

INTRODUCTION

La présente Norme internationale est fondée sur les travaux réalisés dans le cadre du projet ESPRIT 4, GOODDIE, qui a donné lieu à la publication de la série de Spécifications européennes ES 59008. Les organisations qui ont aidé à la préparation du présent document incluent les membres des projets ESPRIT ENCAST et ENCASIT, le Die Products Consortium, JEITA, JEDEC et ZVEI.

La structure de la présente Norme internationale telle qu'actuellement conçue est la suivante :

Sous le titre général : CEI 62258: Produits de puces de semiconducteurs

- Partie 1: Approvisionnement et utilisation
- Partie 2: Formats d'échange de données
- Partie 3: Bonnes pratiques recommandées pour la manipulation, le conditionnement et le stockage (Rapport technique)
- Partie 4: Questionnaire destiné aux utilisateurs et fournisseurs de puces (Rapport technique)
- Partie 5: Exigences pour l'information concernant la simulation électrique
- Partie 6: Exigences pour l'information concernant la simulation thermique
- Partie 7: Schéma du langage XML pour l'échange de données (Rapport technique)
- Partie 8: Schéma du modèle EXPRESS pour l'échange de données (Rapport technique)

D'autres parties peuvent être ajoutées si nécessaire.

PRODUITS DE PUCES DE SEMICONDUCTEURS –

Partie 2: Formats d'échange de données

1 Domaine d'application et objet

La présente partie de la CEI 62258 spécifie les formats de données qui peuvent être utilisés pour l'échange de données qui est couvert par d'autres parties de la présente série CEI 62258 ainsi que les définitions de tous les paramètres utilisés selon les principes et méthodes de la CEI 61360. Elle présente un format d'échange de données de dispositif, DDX (Device Data Exchange), dans le but premier de faciliter le transfert des données géométriques adéquates entre le fabricant de la puce et l'utilisateur de CAO/IAO et des modèles d'informations formels qui permettent l'échange de données dans d'autres formats tels que le format de fichier physique STEP, conformément à ISO 10303-21 et XML. Le format de données a été tenu intentionnellement flexible pour permettre un usage au-delà de ce domaine initial d'application.

Elle a été développée pour faciliter la production, la fourniture et l'utilisation des produits de puces de semiconducteurs, y compris, sans que cela soit limitatif:

- les tranches,
- les puces nues isolées,
- les puces et tranches avec structures de connexion fixées,
- les puces et tranches à encapsulation minimale ou partielle.

La présente norme reflète le format de données DDX en sa version **1.3.0**

2 Références normatives

Les documents de référence suivants sont indispensables pour l'application du présent document. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

CEI 62258-1, *Produits de puces de semiconducteurs – Partie 1: Approvisionnement et utilisation*

CEI 61360-4:2005, *Standard data element types with associated classification scheme for electric components – Part 4: IEC reference collection of standard data element types, component classes* (disponible en anglais seulement)

ISO 8601:2004, *Éléments de données et formats d'échange – Échange d'information – Représentation de la date et de l'heure*

ISO 6093:1985, *Traitement de l'information – Représentation des valeurs numériques dans les chaînes de caractères pour l'échange d'information*

IPC/JEDEC J-STD-033B:2007, *Handling, Packing, Shipping and Use of Moisture/Reflow Sensitive Surface Mount Devices*

ISO 10303-21:2002, *Systèmes d'automatisation industrielle et intégration – Représentation et échange de données de produits – Partie 21: Méthodes de mise en application: Encodage en texte clair des fichiers d'échange*

3 Termes et définitions

Pour les besoins du présent document, les termes et définitions donnés dans la CEI 62258-1 s'appliquent.

4 Exigences

Une référence spécifique pour les variables paramétriques est faite aux codes de types d'éléments de données (DET) de la CEI 61360, lesquels codes sont définis dans la Partie 4 de la CEI 61360.

5 Buts et usage du fichier de format d'échange de données de dispositif (DDX)

5.1 Pour faciliter le transfert de données par des supports électroniques du fournisseur de dispositif jusqu'à l'utilisateur final en vue de leur utilisation dans un système de CAO ou d'IAO, un format de fichier de données, à savoir le **D**evice **D**ata e**X**change, (**DDX**, format d'échange de données), doit être utilisé. Ce format de fichier de données a été intentionnellement maintenu flexible, afin de permettre des améliorations et additions ultérieures pour une utilisation future

5.2 Il est fortement recommandé que les fichiers d'échange de données de dispositif (**D**evice **D**ata e**X**change) aient les trois lettres **DDX** comme extension de fichier. De plus, un fichier d'échange de données de dispositif doit être appelé ici fichier **DDX**.

5.3 Les données devant être transférées d'un fournisseur de dispositif à un utilisateur doivent être contenues dans un seul fichier DDX lisible par un ordinateur. En outre, le contenu minimal de ce fichier doit suffire à un système de conception de logiciel CAO/IAO géométrique. Le fichier doit être lisible textuellement, afin de permettre une simple vérification manuelle.

5.4 Le fichier DDX et le contenu de données doivent être indépendants tant de la machine informatique que du système d'exploitation.

5.5 Le contenu du ficher DDX doit comprendre des informations mécaniques et d'interconnectivité, mais il peut en plus comprendre des données électriques et fonctionnelles.

5.6 Le fichier DDX peut contenir des données pour un ou plusieurs dispositifs et doit pouvoir être utilisé comme fichier bibliothèque par un système de conception de logiciel CAO/IAO. Le fichier peut contenir un ou plusieurs jeux de données pour le même type de dispositif, chacun de ces jeux ayant des formes de livraison différentes, telles que puces à bosses, puces nues, et emballage grandeur puce

5.7 Le fichier DDX doit pouvoir être créé de manière simple ou automatique, comme par exemple par un éditeur de texte ASCII ou une feuille de calcul.

5.8 Le fichier DDX doit pouvoir référencer des fichiers externes complémentaires, tels que fichiers de simulation et modèles thermiques.

5.9 Toutes les données doivent être définies de façon à permettre la conversion en d'autres formats d'échange et vice versa, tels que GDSII et CIF pour les données géométriques relatives aux puces. Une compatibilité aussi étroite que possible avec les données DIE

(échange d'informations de puces) existantes est souhaitable afin de faciliter une traduction simple de fichiers partiels de données DIE.

5.10 Les définitions des paramètres doivent être conformes à la CEI 61360 (voir Article 5 de la CEI 62258-1).

6 Format de fichier DDX et règles relatives au format de fichier

NOTE 1 La version 1.2.1 du DDX annule et remplace la version 1.0.0 contenue dans l'ES 59008-6-1.

NOTE 2 La version 1.3.0 du DDX annule et remplace la version 1.2.1 contenue dans la CEI 62258-2:2005.

Voir l'Article 1 pour la version DDX de la présente norme.

6.1 Validité des données

6.1.1 Toutes les données qui ne sont pas conformes à la syntaxe des données (voir 7.3) doivent être traitées comme une remarque et, donc, ignorées.

6.1.2 Toutes les données obligatoires doivent être présentes. Les données manquantes doivent être étiquetées comme étant une erreur, rendant ainsi ces données inutilisables.

6.1.3 Les opérations, calculs ou formules mathématiques ne doivent pas être permis au sein des données numériques.

6.2 Jeu de caractères

6.2.1 Le fichier DDX doit être un fichier texte compatible avec l'ASCII ayant la fin de ligne appropriée. La fin de ligne dépendra du système d'exploitation. DOS/Windows® utilise en général un caractère d'arrêt (ASCII 0Dh/0Ah) de retour chariot/changement de ligne <CR/LF> alors qu'UNIX® se repose invariablement et uniquement sur le caractère d'arrêt (ASCII 0x0A) de changement de ligne <LF>, le retour chariot <CR> (ASCII 0x0D) étant présent par implication.

6.2.2 Les caractères ASCII 0x00 à 0x7F sont permis; les caractères 0x80 à 0xFF de l'ASCII doivent être ignorés.

6.2.3 Toutes les données textuelles doivent être indépendantes de la casse.

6.2.4 Les caractères espace (ASCII 20h) et tabulation (ASCII 09h) doivent tous deux être traités comme des séparateurs espace; les multiples caractères espace et tabulation seront traités par la syntaxe comme un séparateur espace unique.

6.3 RÈGLES SYNTAXIQUES

6.3.1 Toutes les lignes de données doivent se terminer par un point-virgule: «;».

6.3.2 La virgule «,» doit être utilisée comme séparateur de données.

6.3.3 Les lignes commençant par un dièse «#» doivent être traitées comme un commentaire intentionnel. Toutes les données présentes sur ces lignes doivent être ignorées.

6.3.4 Les traits de soulignement «_» doivent être ignorés dans le nom d'une variable ou d'une propriété et ils peuvent servir de séparateurs intermédiaires de noms. Les traits de soulignement sont valides dans une chaîne textuelle et dans les données de nom.

6.3.5 Des accolades sont utilisées pour ouvrir ou fermer les structures ou les BLOCS. Une accolade d'ouverture «{» doit être utilisée pour débuter une structure ou un bloc et une accolade de fermeture «}» doit être utilisée pour terminer une structure ou un bloc.

6.3.6 Les parenthèses «()» doivent être permises, mais ignorées, dans les données numériques par souci de clarté (par exemple dans les paires de coordonnées).

6.3.7 Pour prendre en charge les sorties au format CSV (variable séparée par une virgule) caractéristique des feuilles de calcul, les données textuelles peuvent être mises entre des guillemets, «», et les paires conjuguées de guillemets doivent être ignorées.

6.3.8 Il n'y a pas de caractère spécifique pour la continuation de ligne. Une chaîne textuelle ouverte par des guillemets '«' doit être fermée par les guillemets conjugués '»', quel que soit le nombre de retours à la ligne dans le texte en question. Comme toutes les commandes DDX se terminent par un point-virgule, les données non textuelles sont réputées s'être terminées à ce point-virgule. Les données textuelles sont réputées se terminer au point-virgule suivant les guillemets de fermeture. Les données textuelles qui ne sont pas entourées de guillemets ne peuvent pas contenir de retour à la ligne ou de caractères de commande et elles doivent se terminer à la première occurrence d'un point-virgule. Les données textuelles placées après ce point-virgule seront traitées comme des données erronées et rejetées.

6.3.9 Pour la commodité pratique, la lisibilité et la facilité de l'analyse des lignes, il est recommandé que la longueur d'une ligne (entre caractères de fin de ligne) ne dépasse pas 255 caractères. En outre, il est fortement recommandé d'imposer une limite maximale de 1 023 caractères par ligne afin d'éviter que d'autres logiciels d'analyse ne rencontrent une erreur de dépassement de tampon d'entrée.

7 Contenu d'un fichier DDX

7.1 Règles relatives au contenu d'un fichier DDX

7.1.1 Structure en blocs

Les données ne doivent exister qu'au sein d'une structure de bloc, appelée «bloc DEVICE» et un ou plusieurs blocs DEVICE, contenant chacun des données, peuvent exister dans un même fichier. Chaque bloc DEVICE est unique et ne doit contenir que les données pertinentes pour un seul dispositif, ayant une forme de dispositif spécifique. Toutes les données dans chaque bloc DEVICE doivent être traitées comme étant locales et uniques au bloc en question seulement. (Voir 6.3.4)

7.1.2 Types des paramètres

Il y a deux types d'utilisation de paramètres pour les données, les structures et les variables, et ces paramètres doivent exister uniquement avec un bloc DEVICE:

- une structure détermine un ou plusieurs jeux de données ayant différents types de donnée.
- une variable équivaut à une ou plusieurs données d'un seul type de donnée

7.1.3 Types de donnée

Les types de donnée sont les suivants.

7.1.3.1 Données chaînes textuelles

Tous les caractères de l'ASCII entre ASCII 20h et ASCII 7Fh sont permis dans les données textuelles, les caractères de l'ASCII 80h inclus et au-dessus doivent être ignorés. Des caractères spéciaux de commande d'impression et d'affichage peuvent être pris en compte

pour permettre l'impression des caractères de soulignement ou de surlignement. Il est conseillé de placer les données chaînes textuelles entre des paires de guillemets (voir 6.3.7).

7.1.3.2 Données de nom textuel

Tous les noms doivent être uniques et doivent être constitués uniquement des caractères suivants issus du jeu de caractères ASCII:-

A à Z a à z 0 à 9 $ - % & ! @ _ .

Lorsque les données de nom textuel sont utilisées pour former le nom d'un fichier, il est conseillé de limiter le nom à huit caractères pour le nom de fichier et à trois caractères pour l'extension de fichier, avec le point «**.**» comme délimiteur nom/extension, au diapason d'un grand nombre de systèmes d'exploitation courants. Il est conseillé de placer les données de nom textuel entre des paires de guillemets (voir 6.3.7).

Remarquer que toutes les données de nom textuel sont indépendantes de la casse et que les espaces sont interdits dans un nom textuel.

7.1.3.3 Données numériques Real (réelles)

Les données numériques réelles doivent être conformes à l'ISO 6093:1985 et doivent être constituées des caractères:

0 à 9 + - . E e

Les valeurs des données peuvent être signées et utiliser la notation technique ou scientifique, mais elles ne doivent pas comporter d'unité dimensionnelle, par exemple:

90008, 9000.80, 9.0008E5, -5207, -5.207E3, 0.102, 102E-3

Remarquer que la virgule «**,**» est utilisée comme séparateur de données et, donc, elle ne doit pas être utilisée comme remplacement du point décimal «**.**».

7.1.3.4 Données numériques Integer (entières)

Les données numériques entières doivent être conformes à l'ISO 6093:1985 et seuls les caractères **0** à **9** sont permis. Les nombres entiers doivent être non signés et ne doivent pas non plus comporter d'unités dimensionnelles.

Pour des besoins pratiques, un nombre entier doit être limité à une résolution de 16 bits, c'est-à-dire que seules les valeurs entières comprises entre 0 inclus et 65536 inclus sont acceptables.

7.1.3.5 Données de date

Les données de date doivent être conformes au format de l'ISO 8601:2004, et peuvent comporter des informations d'heure également, par exemple:

“AAAA-MM-JJ”, “AAAAMMJJ”, “AAAA-MM-JJTHH:MM:SS”.

7.1.4 Déclarations aval

Afin de permettre une analyse en une seule passe, aucun identificateur de variable ou nom de variable ne doit être référencé avant d'avoir été défini.

7.1.5 Unités

Toutes les unités doivent appartenir au système SI, à part l'unité géométrique du micron (10^{-6} m), le pouce et le mil (10^{-3} pouce). Une seule unité de dimension doit être permise au sein d'un même bloc **DEVICE**. Remarquer que le pouce et le millième de pouce sont des unités non préférentielles et ne sont présents qu'en raison de la poursuite de leur usage courant.

7.1.6 Données de coordonnées

Dans toutes les données de coordonnées, la coordonnée **X** doit précéder la coordonnée **Y** et la coordonnée **Y** doit précéder la coordonnée **Z** (à savoir **X,Y** ou **X,Y,Z**).

La coordonnée **X** doit être l'axe horizontal (numériquement de gauche à droite), la coordonnée **Y** doit être l'axe vertical (numériquement de bas en haut) et la coordonnée **Z** doit être l'axe de profondeur (numériquement de proche en éloigné).

7.1.7 Mots réservés

Tous les noms de paramètres doivent être considérés comme réservés et aucun identificateur de variable ou nom de variable ne doit être autorisé à avoir le même nom. L'interdiction ne s'applique pas aux données textuelles de forme libre entre apostrophes ou entre guillemets (voir 7.1.3.1 et 7.1.3.2).

7.2 Syntaxe du bloc DEVICE DDX

DEVICE device_name device_form {
données pertinentes de la puce ……
}

Le fichier **DDX** peut contenir un ou plusieurs blocs **DEVICE**, toutes les données appartenant à un dispositif particulier doivent être intégrées dans le bloc correspondant. (Voir 6.1.1 et 7.1.1).

Un bloc **DEVICE** s'ouvre avec le mot-clé **DEVICE** et l'accolade d'ouverture «**{**», (comme monté), et le bloc **DEVICE** se ferme par l'accolade de fermeture «**}**».

Les données qui ne sont pas à l'intérieur de la structure de bloc **DEVICE** doivent être traitées comme une remarque, permettant l'addition ultérieure d'informations de somme de contrôle, la date de création de fichier et des données historiques etc., dans le fichier **DDX**, sans altérer les données du dispositif réel.

device_name est le nom donné par lequel le dispositif doit être appelé tandis que le **device_form** est la forme mécanique du dispositif auquel les données du bloc se rapportent.

Les données valides pour la variable **device_form** sont:

- **bare_die,**
- **bumped_die,**
- **lead_frame_die**
- **minimally_packaged_device** (ou **MPD**).

D'autres types de **device_form** peuvent être ajoutés à un stade ultérieur (voir la CEI 61360-4:2005, AAD004-001, «code de type de puce», pour de plus amples détails).

Un seul bloc **DEVICE** ayant le nom **device_name** du type **device_form** doit être présent dans un fichier **DDX** mais il est permis de dupliquer **device_name** ou **device_form**.

Exemple d'un agencement type de fichier **DDX** de blocs **DEVICE**:-

```
DEVICE name1 bare_die {
données pertinentes pour le dispositif «name1» comme puce nue..
}
DEVICE name1 bumped_die {
données pertinentes pour le dispositif «name1» comme puce à
bosses..
}
DEVICE name2 mpd {
données pertinentes pour le dispositif «name2» comme dispositif à
encapsulation minimale..
}
DEVICE name2 bare_die {
données pertinentes pour le dispositif «name2» comme puce nue..
}
DEVICE name1 mpd {
données pertinentes pour le dispositif «name1» comme dispositif à
encapsulation minimale..
}
DEVICE name3 bare_die {
données pertinentes pour le dispositif «name3» comme puce nue..
}
```

L'exemple ci-dessus comporte trois occurrences d'un bloc **DEVICE** pour le dispositif «name1», et deux occurrences d'un bloc **DEVICE** pour le dispositif “name2” mais chacun de ces blocs **DEVICE** spécifie un **device_form** différent. L'ordre ou la séquence des blocs **DEVICE** n'a pas d'importance.

7.3 Syntaxe des données DDX

Propriété = valeur [, valeur];

<**propriété**>[*signe égal comme séparateur*]<valeur/variable {*séparateur* <valeur/variable> }>[*fin de données*][*fin de ligne*]

<**propriété**>	::=	Nom du paramètre
[*espace*]	::=	{caractère espace (20h) ou caractère tabulation (09h)}0+
[*séparateur égal*]	::=	[espace]{égal **=**}[espace]
[*séparateur*]	::=	[espace]{virgule **,**}[espace]
[*fin de données*]	::=	[espace]{point-virgule;}
[*fin de ligne*]	::=	{CR ou CR/LF}

Par exemple:

```
Thickness          =      100.0;
Thickness          =      470;
geometric_units    =      micron;
geometricunits     =      micron;
GeometricUnits     =      “millimetres”;
terminal_type      =      T1, Circle, 220;
Terminal_Type      =      T2, Rectangle,  200 ,  250;
TerminalType       =      T2, O, (200, 250);
TERMINALTYPE       =      T2, O,  200  ,  250;
```

Ainsi, **terminal_type**, **Terminal_Type**, **TerminalType** et **TERMINALTYPE** font tous référence au même nom de paramètre.

8 Définitions des paramètres d'un bloc DEVICE

8.0 Notes d'usage général

8.0.1 Notes relatives à la forme du dispositif

Lorsqu'un paramètre est unique au **device_form**, tel que défini dans le bloc **DEVICE**, le paramètre sera précédé par…

8.0.1.1 **DIE_** le paramètre de données est unique à la forme de puces nues ou de puces à bosses

8.0.1.2 **BUMP_** le paramètre de données est unique aux puces à bosses seulement

8.0.1.3 **MPD_** le paramètre de données est unique aux dispositifs à encapsulation minimale, tels qu'un CSP

8.0.1.4 **WAFER_** le paramètre de données est unique à une puce nue livrée au niveau tranche

8.0.1.5 **LEAD_** le paramètre de données est unique à un dispositif de puce avec une grille de connexion fixée.

8.0.2 Eléments de paramètres de données

Avec la liste suivante de paramètres de données (voir 8.1 et suivants), il apparaît les éléments suivants:

8.0.2.1 le nom du paramètre, tel qu'utilisé par la syntaxe au sein d'un fichier **DDX**,

8.0.2.2 le type du paramètre, indiquant soit un type variable, soit un type de données de structure,

8.0.2.3 la fonction du paramètre, déterminant son utilisation et sa signification,

8.0.2.4 la valeur du paramètre indiquant le type de données attendu,

8.0.2.5 toute limitation de paramètre, indiquant toute limitation au sein du bloc **DEVICE**,

8.0.2.6 les dépendances de paramètre, soulignant les paramètres qu'il est nécessaire de déclarer avant invocation,

8.0.2.7 un ou plusieurs exemples pratiques, et

8.0.2.8 toute remarque pertinente.

Un bref tableau de paramètres est donné dans l'Annexe F, et un exemple pratique d'un bloc complet **DDX DEVICE** est donné dans l'Annexe A, la sortie graphique attendue étant fournie dans l'Annexe C.

Tous les paramètres doivent être conformes aux codes correspondants de la CEI 61360 du type d'élément de données (DET), tels que définis dans la CEI 61360-4:2005. Se référer à l'Annexe F pour un tableau de références croisées.

8.0.3 Termes et conventions

Un point de connexion électrique s'appelle une borne. Il peut s'agir d'un plot de connexion pour une puce nue et cela peut également se référer à la zone de plaques de contact ou d'empreintes de plots requise par un support d'interconnexion. Par convention, la borne placée au coin supérieur gauche d'une puce a le numéro 1 et la numérotation des bornes ou des broches progresse en séquence dans le sens contraire des aiguilles d'une montre.

Les dimensions en coordonnée X se rapportent à la longueur dans le plan horizontal, avec des valeurs positives augmentant vers la droite. Les dimensions en coordonnée Y se rapportent à la largeur dans le plan horizontal avec des valeurs positives qui augmentent en s'éloignant de

la vue de l'utilisateur. Les dimensions en coordonnée Z se rapportent à la hauteur dans la direction verticale avec des valeurs positives qui augmentent vers le haut (vers l'observateur).

Comme point de référence, tous les composants, y compris les puces à bosses, sont généralement vus de dessus, avec le côté actif vers le haut.

8.0.4 Résumé des règles générales

8.0.4.1 Toutes les données valides doivent être contenues dans un bloc **DEVICE** (voir 7.1.1).

8.0.4.2 Tout paramètre local ou unique, tel qu'un nom, doit être défini avant son utilisation.

8.0.4.3 Tous les paramètres doivent se conformer aux codes DET correspondants de la CEI 61360, tels que définis dans la partie 4 de la CEI 61360 (voir 5.8 et Annexe F).

8.0.4.4 Les unités de mesure, **GEOMETRIC_UNITS**, doivent être définies avant qu'une quelconque variable géométrique ne soit définie.

8.0.4.5 L'origine géométrique, **GEOMETRIC_ORIGIN**, et la vue géométrique, **GEOMETRIC_VIEW**, doivent être définies avant que de quelconques coordonnées géométriques ne soient définies.

8.0.4.6 Le paramètre **TERMINAL_COUNT** doit être défini avant qu'il ne soit fait référence à de quelconques paramètres **TERMINAL** et le nombre de paramètres **TERMINAL** ne doit pas dépasser la valeur de **TERMINAL_COUNT**.

8.0.4.7 Le paramètre **TERMINAL_TYPE_COUNT** doit être défini avant qu'il ne soit fait référence à de quelconques paramètres **TERMINAL_TYPE** et le nombre de paramètres **TERMINAL_TYPE** ne doit pas dépasser la valeur de **TERMINAL_TYPE_COUNT**.

8.1 BLOCK DATA (DONNÉES DE BLOC)

8.1.1 Paramètre DEVICE_NAME

Il est défini dans l'en-tête de bloc **DEVICE** comme étant le paramètre **device_name**.

Nom de paramètre	**device_name**
Type du paramètre	Variable, se référer à 7.2 sur les blocs **DEVICE**
Fonction du paramètre	Définit le numéro de type ou le nom de référence selon le fabricant de dispositif
Valeurs du paramètre	Données de nom textuel, comme en 7.1.3.2
Exemple	`SN74LS04`
Référence	Voir 7.1.7, 7.2 et Annexe G.

8.1.2 Paramètre DEVICE_FORM

Il est défini dans l'en-tête de bloc **DEVICE** par le paramètre **device_form**.

Nom de paramètre	**device_form**
Type du paramètre	Variable, se référer à 7.2 sur les blocs **DEVICE**
Fonction du paramètre	Définit la forme physique du dispositif.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Exemple	`bare_die, bumped_die, MPD`
Référence	Paragraphe 7.2 et Annexe G.

8.1.3 Paramètre BLOCK_VERSION

Nom de paramètre	**BLOCK_VERSION**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le numéro de version et/ou le numéro d'émission du bloc DEVICE.

Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`BLOCK_VERSION = “1.0A”;`
Référence	Annexe G.

8.1.4 Paramètre BLOCK_CREATION_DATE

Nom de paramètre	**BLOCK_CREATION_DATE**
Type du paramètre	Variable
Fonction du paramètre	Spécifie la date à laquelle le bloc DEVICE a été créé et modifié pour la dernière fois.
Valeurs du paramètre	Date, conformément à 7.1.3.5
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemples	`BLOCK_CREATION_DATE = “1997-12-25”;`
Référence	Annexe G.

8.1.5 Paramètre VERSION

Nom de paramètre	**VERSION**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le numéro de version de la norme DDX (actuellement à la version 1.3.0), à laquelle ce bloc DEVICE est conforme.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`VERSION = “1.3.0”;`
Notes	Voir Article 1 pour la version de document de la présente Norme. Remarquer que le présent document peut ne pas être la version la plus récente et donc se référer aux Autorités de normalisation en cas de doute
Référence	Annexe G.

8.2 DONNÉES DE DISPOSITIF

8.2.1 Paramètre DIE_NAME

Nom de paramètre	**DIE_NAME**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le nom du jeu de puce ou de masque ayant servi à produire la puce. Il peut être différent du paramètre DEVICE_NAME.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`DIE_NAME = “XXC345”;`
Référence	Annexe G.

8.2.2 Paramètre DIE_PACKAGED_PART_NAME

Nom de paramètre	**DIE_PACKAGED_PART_NAME**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le nom de pièce des fabricants pour une pièce emballée équivalente, le cas échéant. Il peut être différent du paramètre DEVICE_NAME.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Exemple	`DIE_PACKAGED_PART_NAME = “SN5405JN”;`
Notes	Utilisé pour se référer à une pièce de puce encapsulée identique, pas une fonction simplement similaire, lorsqu'elle est livrée par le même fabricant sous forme encapsulée.

8.2.3 Paramètre DIE_MASK_REVISION

Nom de paramètre	**DIE_MASK_REVISION**
Type du paramètre	Variable

Fonction du paramètre	Spécifie les détails relatifs à la révision du masque associés à la version particulière en question sur la puce.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Doit être déclaré une seule fois dans un seul bloc **DEVICE**.
Exemple	`DIE_MASK_REVISION = “RTDAC1”;` `DIE_MASK_REVISION = “9033-2-101-M5/2”;`
Notes	Destiné à s'assurer que les données de puce correspondent à la version géométrique de la puce réelle.
Référence	Annexe G.

8.2.4 Paramètre MANUFACTURER

Nom de paramètre	**MANUFACTURER**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le fabricant ou l'usine de fabrication du dispositif.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`MANUFACTURER = “Fuzziwuz Logic Inc.”;`

8.2.5 Paramètre DATA_SOURCE

Nom de paramètre	**DATA_SOURCE**
Type du paramètre	Variable
Fonction du paramètre	Spécifie la source des données du dispositif, notamment lorsqu'elle diffère du fabricant ou de l'usine de fabrication.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`DATA_SOURCE = “AnyChip Technology Ltd.”;` `DATA_SOURCE = “GOOD-DIE database”;`

8.2.6 Paramètre DATA_VERSION

Nom de paramètre	**DATA_VERSION**
Type du paramètre	Variable
Fonction du paramètre	Spécifie la révision de la source de données utilisée pour déterminer les paramètres au sein du bloc DEVICE. Ce paramètre est lié à 8.2.5, Paramètre DATA_SOURCE
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`DATA_VERSION = “Initial Issue 1.0”;`
Référence	Se référer à l'Annexe G.

8.2.7 Paramètre FUNCTION

Nom de paramètre	**FUNCTION**
Type du paramètre	Variable
Fonction du paramètre	Brève description de la fonction des dispositifs
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`FUNCTION = “16 Bit Microprocessor”;`
Notes	La CEI 61360-4:2005 spécifie certaines classes fonctionnelles et d'application qui peuvent être utilisées.

8.2.8 Paramètre IC_TECHNOLOGY

Nom de paramètre	**IC_TECHNOLOGY**
Type du paramètre	Variable
Fonction du paramètre	Spécifie la technologie de fabrication du dispositif
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`IC_TECHNOLOGY = “CMOS”;` `IC_TECHNOLOGY = “bipolar”;`

```
IC_TECHNOLOGY = “bicmos”;
IC_TECHNOLOGY = “GaAs”;
```

8.2.9 Paramètre DEVICE_PICTURE_FILE

Nom de paramètre	**DEVICE_PICTURE_FILE**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le(s) nom(s) du (des) fichiers de graphiques ou d'images représentant le dispositif.
Valeurs du paramètre	Données de nom textuel, comme en 7.1.3.2
Exemple	`DEVICE_PICTURE_FILE = “DIE001.JPG”;` `DEVICE_PICTURE_FILE = “AA0B0C.SF”, “FRED.GIF”;`
Notes	Le fichier d'images peut être une photographie réelle ou une représentation graphique du dispositif. Il convient que l'extension du fichier indique le format utilisé. Seuls doivent être utilisés les noms de fichier, sans nom de chemin relatif ou absolu. Plusieurs paramètres (noms de fichier d'images) peuvent être introduits dans une déclaration unique ou bien plusieurs déclarations peuvent être utilisées pour le même effet. Il n'existe aucune exigence de mise à l'échelle ou positionnelle.
Référence	Annexe I, Notes 3 et 4.

8.2.10 Paramètre DEVICE_DATA_FILE

Nom de paramètre	**DEVICE_DATA_FILE**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le(s) nom(s) d'un ou plusieurs fichiers contenant des données pertinentes, telles que les spécifications techniques, les documents d'approvisionnement ou des fiches signalétiques générales.
Valeurs du paramètre	Données de nom textuel, comme en 7.1.3.2
Exemple	`DEVICE_DATA_FILE = “SpecSheet.PDF”;` `DEVICE_DATA_FILE = “AA0B0C.DOC”, “AA0B0C.TXT”;`
Notes	Le fichier de données peut avoir n'importe quel format reconnu. Il convient que l'extension du fichier indique le format utilisé. Seuls doivent être utilisés les noms de fichier, sans nom de chemin relatif ou absolu. Plusieurs paramètres (noms de fichier de données) peuvent être introduits dans une déclaration unique ou bien plusieurs déclarations peuvent être utilisées pour le même effet.
Référence	Annexe I, Notes 3 et 4.

8.3 DONNÉES GÉOMÉTRIQUES

8.3.1 Paramètres GEOMETRIC_UNITS

Nom de paramètre	**GEOMETRIC_UNITS**
Type du paramètre	Variable
Fonction du paramètre	Spécifie les unités géométriques qui doivent s'appliquer à toutes les valeurs géométriques au sein du bloc DEVICE.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1. L'une des valeurs suivantes: ▪ micromètre ou micron, ▪ mètre, ▪ millimètre, ▪ pouce ▪ mil (1.0E-3 pouce)
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`GEOMETRIC_UNITS = microns;` `GEOMETRIC_UNITS = mil;`
Notes	Ce paramètre GEOMETRIC_UNITS doit être déclaré avant d'utiliser une quelconque unité géométrique. Le micron est en général l'unité

	dimensionnelle par défaut pour les dimensions des puces tandis que le pouce et le mil sont des unités non préférentielles.
Référence	Paragraphe 7.1.5 et Annexe E.

8.3.2 Paramètre GEOMETRIC_VIEW

Nom de paramètre	**GEOMETRIC_VIEW**
Type du paramètre	Variable
Fonction du paramètre	Spécifie la vue géométrique qui doit s'appliquer à toutes les formes géométriques au sein du bloc DEVICE.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1. L'une des valeurs suivantes: • **TOP** c'est-à-dire côté actif vers le haut, et • **BOTTOM** c'est-à-dire côté actif vers le bas.
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`GEOMETRIC_VIEW = top;` `GEOMETRIC_VIEW = “bottom”;`
Notes	Le paramètre GEOMETRIC_VIEW doit être déclaré avant de créer une quelconque forme géométrique. Il est courant de voir une puce nue et une pièce emballée en vue de dessus “TOP” alors que cela sera en vue de dessous “BOTTOM” (c'est-à-dire à travers le substrat) pour une puce à bosses.
Référence	Annexe E.

8.3.3 Paramètre GEOMETRIC_ORIGIN

Nom de paramètre	**GEOMETRIC_ORIGIN**
Type du paramètre	Variable
Fonction du paramètre	Détermine l'origine géométrique en X et en Y par rapport à laquelle toutes les autres coordonnées sont référencées. L'origine est donnée par rapport au centre géométrique de la puce dans les unités spécifiées par le paramètre GEOMETRIC_UNITS.
Valeurs du paramètre	Origine de coordonnée en nombre réel X, origine de coordonnée en nombre réel Y, conformément à 7.1.3.3.
Dépendances	GEOMETRIC_UNITS, SIZE
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`GEOMETRIC_ORIGIN = -6000,-7500;`
Notes	Les valeurs de dimensions sont exprimées en GEOMETRIC_UNITS. Les valeurs des paires de coordonnées d'origine se rapportent au centre géométrique (voir Figure 1 pour une explication plus complète).
Référence	Annexe E.

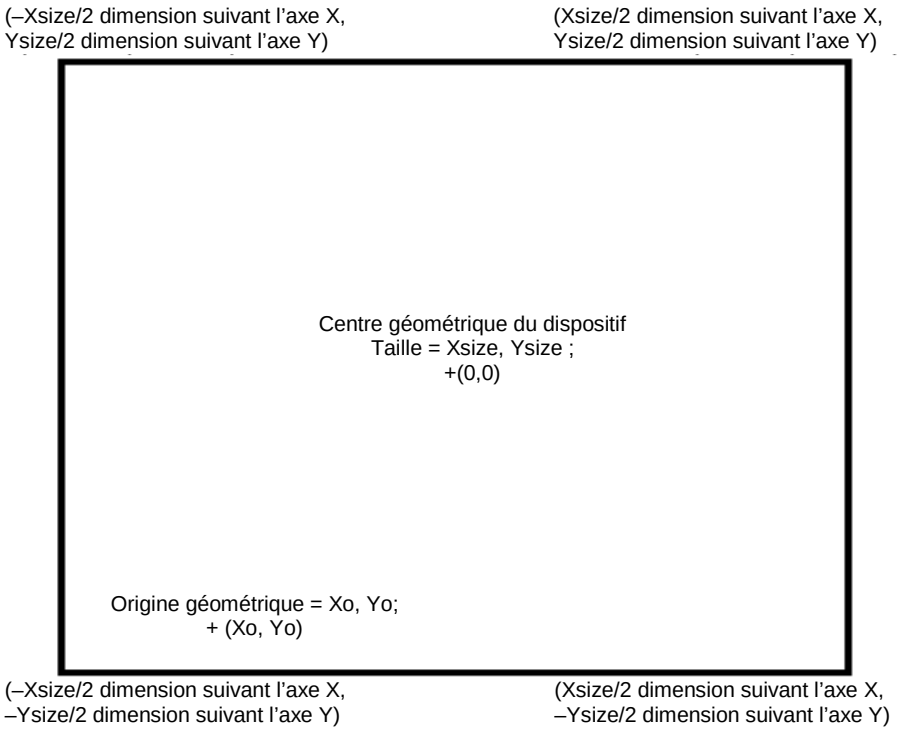

Figure 1 – Relation entre le centre géométrique et l'origine géométrique

Le paramètre GEOMETRIC_ORIGIN est traité comme un décalage pour toutes les données de coordonnées et, de ce fait, les valeurs de GEOMETRIC_ORIGIN sont ***ajoutées*** à ***toutes*** les paires individuelles de coordonnées pour donner la position en X et en Y par rapport au centre géométrique du dispositif.

La principale utilisation du paramètre GEOMETRIC_ORIGIN est de permettre le centrage de l'origine sur une borne ou autre caractéristique géométrique, plutôt qu'en une position géométrique arbitraire car, dans la pratique, SIZE peut être l'objet d'une tolérance asymétrique et, par voie de conséquence, toutes les références liées à ce centre géométrique pourraient elles-aussi être soumises à une erreur de tolérance.

8.3.4 Paramètre SIZE

Nom de paramètre	**SIZE**
Type du paramètre	Variable
Fonction du paramètre	Détermine les dimensions X et Y du dispositif, et spécifie facultativement sa forme comme étant une ellipse.
Valeurs du paramètre	Dimension X en nombre réel, dimension Y en nombre réel (conformément à 7.1.3.3), {Ellipse}
Dépendances	GEOMETRIC_UNITS, GEOMETRIC_VIEW
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`SIZE = 250, 500.5;` `SIZE = 350, 350, E;`
Notes	Les valeurs de dimensions sont exprimées en GEOMETRIC_UNITS. Dans le dernier exemple, le caractère «E» comme 3e paramètre définit une ellipse, dans cette instance une puce circulaire de diamètre 350 GEOMETRIC_UNITS.
Référence	Annexe E.

8.3.5 Paramètre SIZE_TOLERANCE

Nom de paramètre	**SIZE_TOLERANCE**

Type du paramètre	Variable
Fonction du paramètre	Spécifier la/les tolérance(s) géométrique(s) des valeurs du paramètre SIZE
Valeurs du paramètre	Réelles en GEOMETRIC_UNITS, conformément à 7.1.3.3.
Dépendances	GEOMETRIC_UNITS, SIZE, GEOMETRIC_VIEW
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.

Exemple	`SIZE_TOLERANCE = 0.5;`
	`SIZE_TOLERANCE = -0.2,0.5;`
	`SIZE_TOLERANCE = -0.2,0.5,-0.1,0.4;`

Notes	Une, deux ou quatre valeurs peuvent être données: Lorsqu'une valeur de tolérance unique est donnée, la tolérance doit être prise comme étant une valeur non signée ± pour l'axe X et aussi pour l'axe Y. Lorsque deux valeurs de tolérance sont données: la première valeur doit être prise comme étant le minimum pour l'axe X et aussi pour l'axe Y, la seconde valeur doit être prise comme étant le maximum pour l'axe X et aussi pour l'axe Y. Lorsque quatre valeurs de tolérance sont données: la première valeur doit être prise comme étant le minimum pour l'axe X, la deuxième valeur doit être prise comme étant le maximum pour l'axe X, la troisième valeur doit être prise comme étant le minimum pour l'axe Y; la quatrième valeur doit être prise comme étant le maximum pour l'axe Y.

8.3.6 Paramètre THICKNESS

Nom de paramètre	**THICKNESS**
Type du paramètre	Variable
Fonction du paramètre	Détermine l'épaisseur (dimension Z) du dispositif.
Valeurs du paramètre	Valeur de la dimension Z en nombre réel, conformément à 7.1.3.3
Dépendances	GEOMETRIC_UNITS
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`THICKNESS = 10.5;`
Notes	Les valeurs de dimensions sont exprimées en GEOMETRIC_UNITS.

8.3.7 Paramètre THICKNESS_TOLERANCE

Nom de paramètre	**THICKNESS_TOLERANCE**
Type du paramètre	Variable
Fonction du paramètre	Spécifie l'une ou les plusieurs tolérances du paramètre THICKNESS qui peuvent être attendues en raison des variations normales de processus.
Valeurs du paramètre	Réelles en GEOMETRIC_UNITS, conformément à 7.1.3.3.
Dépendances	GEOMETRIC_UNITS, THICKNESS
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`THICKNESS_TOLERANCE = 2.5;` `THICKNESS_TOLERANCE = -1.2,1.5;`
Notes	Une ou deux valeurs peuvent être données: Lorsqu'une tolérance unique est donnée, la tolérance doit être prise comme étant une valeur non signée ±. Lorsque deux valeurs de tolérance sont données: la première valeur doit être prise comme le minimum et la seconde valeur doit être prise comme le maximum.

8.3.8 Paramètre FIDUCIAL_TYPE

Nom de paramètre	**FIDUCIAL_TYPE**

Type du paramètre	Structure
Fonction du paramètre	La structure FIDUCIAL_TYPE attribue un nom de type de repère conventionnel et définit le fichier graphique associé et la taille d'un type individuel de repère conventionnel. La structure peut être utilisée pour définir un seul type de repère conventionnel ou une multitude de types de repère conventionnel.
Dépendances	GEOMETRIC_UNITS, GEOMETRIC_VIEW
Notes	Un repère conventionnel existe uniquement comme forme graphique dans un rectangle, les données graphiques pour le repère conventionnel sont contenues dans un fichier graphique externe. Les paires de coordonnées pour ce rectangle sont uniquement rapportées au type de forme de repère conventionnel et sont utilisées comme des références lors du placement de la forme.
Référence	Annexes E et I.

Syntaxe de la définition d'un FIDUCIAL_TYPE unique:

FIDUCIAL_TYPE *Fiducial_type_name = Fiducial_file_name, X-size, Y-size;*
FIDUCIAL_TYPE *Fiducial_type_name = Fiducial_file_name, X-size, Y-size;*

Syntaxe de la définition d'un FIDUCIAL_TYPE multiple:

FIDUCIAL_TYPE {
Fiducial_type_name = Fiducial_file_name, X-size, Y-size;
Fiducial_type_name = Fiducial_file_name, X-size, Y-size;
Fiducial_type_name = Fiducial_file_name, X-size, Y-size;
}

où:

8.3.8.1 Fiducial_type_name

Nom de référence textuel (conformément à 7.1.3.2) pour un type de repère conventionnel, lequel nom doit être unique dans le bloc DEVICE.

8.3.8.2 Fiducial_file_name

Il s'agit du nom du fichier (conformément à 7.1.3.2) qui contient le repère conventionnel sous forme de données graphiques. Le type des données graphiques doit être indiqué par le code d'extension de fichier, tel que “BMP”, “GIF”, “DXF” etc. Le type de données n'est pas spécifié dans la présente Norme et il incombe au logiciel de CAO/IAO de décider de ce qui peut ou ne peut pas être affiché. Le graphique doit être traité comme étant contenu dans un rectangle de dimensions X-size et Y-size.

8.3.8.3 X-size, Y-size

Ces paramètres numériques réels (conformément à 7.1.3.3) comprennent une paire de coordonnées et déterminent la taille du rectangle du graphique de repère conventionnel. Le centre géométrique du graphique de repère conventionnel doit être déterminé comme étant en (X-size/2, Y-size/2).

Exemple 1

```
FIDUCIAL_TYPE Fid1 = “tile.bmp”, 200, 250;
```

Cet exemple décrit un repère conventionnel, trouvé comme un fichier externe «`tile.bmp`», appelé de manière unique `Fid1`, avec un X-size de 200 unités, et un Y-size de 250 unités. Les unités sont définies par le paramètre GEOMETRIC_UNITS.

Exemple 2

```
FIDUCIAL_TYPE  {
      fidu1 = “graphic1.bmp”, 200, 300;
      fidu2 = “graphic2.dxf”, 500, 500;
      fidu3 = “graphic3.gif”, 100, 100;
}
```

Cet exemple de plusieurs définitions décrit trois types et tailles séparés de repère conventionnel.

- `fidu1` est contenu dans le fichier «`graphic1.bmp`», d'une taille de 200 unités par 300 unités.
- `fidu2` est contenu dans le fichier «`graphic2.dxf`», d'une taille de 500 unités par 500 unités.
- `fidu3` est contenu dans le fichier «`graphic3.gif`», d'une taille de 100 unités par 100 unités.

8.3.9 Paramètre FIDUCIAL

Nom de paramètre	**FIDUCIAL**
Type du paramètre	Structure
Fonction du paramètre	La structure FIDUCIAL définit l'emplacement et l'orientation de chaque repère conventionnel graphique. En tant que structure, FIDUCIAL peut être utilisé pour définir un repère conventionnel unique ou une multitude de repères conventionnels.
Dépendances	GEOMETRIC_VIEW, FIDUCIAL_TYPE, GEOMETRIC_UNITS, et GEOMETRIC_ORIGIN.
Notes	Chaque nom de type de repère conventionnel utilisé doit avoir été préalablement déclaré dans une invocation de FIDUCIAL_TYPE et les informations relatives aux paires de coordonnées doivent se rapporter au GEOMETRIC_ORIGIN du dispositif.
Référence	Annexe E.

Syntaxe de la définition d'un **FIDUCIAL** unique:

FIDUCIAL F_*n* = *Fiducial_type_name*, *X-co.*, *Y-co.*, *orientation;*
FIDUCIAL F_*n* = *Fiducial_type_name*, *X-co.*, *Y-co.*, *orientation;*

Syntaxe de la définition d'un FIDUCIAL multiple:

FIDUCIAL {
F_*n* = *Fiducial_type_name*, *X-co.*, *Y-co.*, *orientation;*
F_*n* = *Fiducial_type_name*, *X-co.*, *Y-co.*, *orientation;*
F_*n* = *Fiducial_type_name*, *X-co.*, *Y-co.*, *orientation;*
}

où:

8.3.9.1 F_*n*

Identificateur de repère conventionnel unique, où “***n***” est le numéro effectif du repère conventionnel (nombre entier, conformément à 7.1.3.4), numérotation à partir du coin supérieur gauche dans le sens contraire des aiguilles d'une montre. Noter que le trait de soulignement est facultatif, il est utilisé pour la clarté seulement.

8.3.9.2 Fiducial_type_name

Nom textuel (conformément à 7.1.3.2) d'une forme référencée de FIDUCIAL_TYPE (conformément à 8.3.8.1), lequel nom doit avoir été préalablement déclaré (voir 7.1.4).

8.3.9.3 X-co

Valeur numérique réelle de la coordonnée X (comme en 7.1.3.3) pour l'emplacement du centre de la forme de repère conventionnel, en unités définies par le paramètre GEOMETRIC_UNITS. Cette valeur est rapportée à la valeur de la coordonnée X du GEOMETRIC_ORIGIN, et ce paramètre a un fonctionnement identique à celui décrit en 8.4.5.4.

8.3.9.4 Y-co

Valeur numérique réelle de la coordonnée Y (conformément à 7.1.3.3) pour l'emplacement du centre de la forme de repère conventionnel, en unités définies par le paramètre GEOMETRIC_UNITS. Cette valeur est rapportée à la valeur de la coordonnée Y du GEOMETRIC_ORIGIN, et ce paramètre a un fonctionnement identique à celui décrit en 8.4.5.5.

8.3.9.5 Orientation

Valeur d'orientation, en valeurs entières de 0 à 360 (voir 7.1.3.4). Il s'agit de l'angle de rotation dans le sens horaire, exprimé en degrés, autour du centre de référence géométrique de la forme de repère conventionnel. Si les lettres "**MX**" sont incluses, l'orientation de la forme de repère conventionnel doit être miroitée dans l'axe X; de même, si les lettres "**MY**" sont incluses, l'orientation de la forme de repère conventionnel doit être miroitée dans l'axe Y. "**MX**" et "**MY**" peuvent être présents simultanément (voir Annexe D pour une représentation graphique) et l'ensemble du miroitage doit être effectué autour du centre de référence géométrique de la forme de repère conventionnel. Noter que l'opération de miroitage doit être réalisée en premier lieu puis le repère conventionnel doit être tourné de l'angle d'orientation. Ce paramètre a un fonctionnement identique à celui décrit en 8.4.5.6

Exemple 1

```
FIDUCIAL Fid007 = fidu1, 5000, 7000, MX0;
```

Cet exemple déclare que le repère conventionnel **Fid007**.....

- est situé à X = 5 000 unités, Y = 7 000 unités,
- est un FIDUCIAL_TYPE dénommé "fidu1", miroité sur l'axe X et orienté à 0°.

Exemple 2

```
FIDUCIAL F_18 = fiduX1, 1000, 2200, 90;
```

Cet exemple déclare que le repère conventionnel **F_18** .....

- est situé à X = 1 000 unités, Y = 2 200 unités,
- est un FIDUCIAL_TYPE dénommé "fiduX1", orienté (tourné) de 90° dans le sens horaire.

Exemple 3

```
FIDUCIAL {
  Fid007-fidu1,  5000, 7000, MX0;
  F_18 = fiduX1, 1000, 2200, 90;
}
```

Cet exemple de définition de repère conventionnel multiple déclare des données conventionnelles identiques à celles montrées dans les Exemples 1 et 2.

8.4 DONNÉES RELATIVES AUX BORNES

8.4.1 Paramètre TERMINAL_COUNT

Nom de paramètre	**TERMINAL_COUNT**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le nombre de bornes électriques ou de points de connexion.

Valeurs du paramètre	Integer, conformément à 7.1.3.4.
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`TERMINAL_COUNT = 44;`
Notes	La valeur de TERMINAL_COUNT doit être déclarée avant la déclaration ou l'utilisation de tout TERMINAL.

8.4.2 Paramètre TERMINAL_TYPE_COUNT

Nom de paramètre	**TERMINAL_TYPE_COUNT**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le nombre de formes ou types différents de borne de connexion ou de liaison.
Valeurs du paramètre	Integer, conformément à 7.1.3.4.
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`TERMINAL_TYPE_COUNT = 4;`
Notes	La valeur de TERMINAL_TYPE_COUNT doit être déclarée avant la déclaration ou l'utilisation d'un quelconque TERMINAL_TYPE.

8.4.3 Paramètre CONNECTION_COUNT

Nom de paramètre	**CONNECTION_COUNT**
Type du paramètre	Variable, facultatif
Fonction du paramètre	Spécifie le nombre maximal de connecteurs utilisés par le paramètre TERMINAL. Ce paramètre spécifie en fait la valeur la plus élevée pour le nombre ***conn_N*** de connexions utilisées dans la déclaration du paramètre TERMINAL (voir 8.4.5.2).
Valeurs du paramètre	Integer, conformément à 7.1.3.4.
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`CONNECTION_COUNT = 4;`
Notes	Il convient de déclarer la valeur de CONNECTION_COUNT avant la déclaration ou l'utilisation de tout TERMINAL. Il doit être utilisé comme le contrôle limite pour les nombres ***conn_N*** de connexions. Si le paramètre CONNECTION_COUNT n'est pas déclaré, aucun contrôle n'aura lieu.

8.4.4 Paramètre TERMINAL_TYPE

Nom de paramètre	**TERMINAL_TYPE**
Type du paramètre	Structure
Fonction du paramètre	La structure TERMINAL_TYPE attribue un nom de type et définit la forme et la taille d'un type individuel de borne. En tant que structure, TERMINAL_TYPE peut être utilisé pour définir un type de borne unique ou une multitude de types de borne.
Dépendances	GEOMETRIC_UNITS, GEOMETRIC_VIEW, TERMINAL_TYPE_COUNT
Notes	Les paires de coordonnées sont uniquement rapportées au type de forme de borne et sont utilisées comme des références lors du placement de la forme.
Référence	Annexe E.

Syntaxe de la définition d'un TERMINAL_TYPE unique:

TERMINAL_TYPE *Terminal_type_name* = *Terminal_shape_type*, *Co-ordinates*.,,,;
TERMINAL_TYPE *Terminal_type_name* = *Terminal_shape_type*, *Co-ordinates*.,,,;

Syntaxe de la définition d'un TERMINAL_TYPE multiple:

TERMINAL_TYPE {
Terminal_type_name = *Terminal_shape_type*, *Co-ordinates* .,,,;
Terminal_type_name = *Terminal_shape_type*, *Co-ordinates* .,,,;

Terminal_type_name = *Terminal_shape_type*, *Co-ordinates* .,,,;
}

où:

8.4.4.1 Terminal_type_name

Nom de référence textuel pour un type de borne (conformément à 7.1.3.2), lequel nom doit être unique dans le bloc DEVICE.

8.4.4.2 Terminal_shape_type

Détermine le type de forme de borne (il n'est nécessaire d'utiliser que la première lettre):

Tableau 1 – Types de forme de borne

Carac-tère	Donnée DET	Forme
R	RECT	**R**ectangle,
C	CIRC	**C**ercle,
E	ELL	**E**llipse,
P	POLY	**P**olygone.

8.4.4.3 Coordonnées *(Co-ordinates)*

Valeurs de coordonnées en nombre réel (voir Tableau 2), conformément à 7.1.3.3.

Elles spécifient les coordonnées relatives pour la forme de borne et, ce faisant, déterminent le centre de référence géométrique pour la forme. Le centre de référence géométrique pour la forme sera utilisé par les coordonnées TERMINAL (voir 8.4.5.4 et 8.4.5.5) pour placer la forme, et toutes les opérations de rotation et de miroitage seront réalisées autour du centre de référence géométrique. Seule la forme polygonale peut avoir un centre de référence géométrique autre que le «centre de gravité» naturel.

Tableau 2 – Coordonnées de la forme de borne

Forme	Coordonnées	Centre de référence géométrique
Rectangle	X-size, Y-size	Le centre géométrique (0,0) se placera à X-size/2:Y-size/2
Cercle	Diamètre	Le centre géométrique (0,0) se placera à diamètre/2:diamètre/2
Ellipse	X-axis, Y-axis	Le centre géométrique (0,0) se placera à X-axis/2:Y-axis/2
Polygone	$X\text{-}co_1$, $Y\text{-}co_1$,..... $X\text{-}co_N$, $Y\text{-}co_N$	Un polygone de “N” cotés aura *N* paires de coordonnées géométriquement centrées sur (0,0), avec l'hypothèse que la forme polygonale sera complétée par le vecteur (Xco_N,Yco_N - Xco_1,Yco_1)

Exemple 1

```
TERMINAL_TYPE ShapeR1 = Rectangle,  200, 250;
```

Cet exemple décrit un rectangle, appelé de manière univoque «ShapeR1», avec une dimension sur l'axe X de 200 unités et une dimension sur l'axe Y -de 250 unités. Les unités sont définies par le paramètre GEOMETRIC_UNITS.

Exemple 2

```
TERMINAL_TYPE  {
      ShapeR1 = R, 200, 300;
      ShapeC2 = C, 250;
      ShapeE3 = E, 400, 200;
     ShapeP4 = P, (150, 200),(-150, 200),(-150,-150),( 150,-150);
}
```

Cet exemple décrit quatre types et formes distincts de borne:

- ShapeR1 est une borne rectangulaire ayant une dimension sur l'axe X de 200 unités et une dimension sur l'axe Y de 300 unités,
- ShapeC2 est une borne circulaire de 250 unités de diamètre,
- ShapeE3 est une borne elliptique avec un diamètre sur l'axe X de 400 unités et un diamètre sur l'axe Y de 200 unités, et
- ShapeP4 est une forme de borne polygonale à quatre côtés, avec un centre de référence géométrique à (0,0).

Noter que seule la première lettre du descripteur *Terminal_shape_type* est utilisée et que les noms *Terminal_type_name* sont uniques.

8.4.5 Paramètre TERMINAL

Nom de paramètre	**TERMINAL**
Type du paramètre	Structure
Fonction du paramètre	La structure TERMINAL définit le nom, l'emplacement, l'orientation et la fonction électrique de chaque borne de connexion. En tant que structure, TERMINAL peut être utilisé pour définir une seule borne ou une multitude de bornes.
Dépendances	TERMINAL_COUNT, CONNECTION_COUNT, GEOMETRIC_VIEW, TERMINAL_TYPE, GEOMETRIC_UNITS, GEOMETRIC_ORIGIN.
Notes	Chaque nom de type de repère conventionnel utilisé doit avoir été préalablement déclaré dans une invocation de FIDUCIAL_TYPE et les informations relatives aux paires de coordonnées doivent se rapporter au GEOMETRIC_ORIGIN du dispositif. Se référer à l'Annexe E.

Syntaxe de la définition d'un TERMINAL unique:

```
TERMINAL T_n = conn_N, Terminal_type_name, X-co., Y-co., orient.,
               Terminal_name, IO_type;
TERMINAL T_n = conn_N, Terminal_type_name, X-co., Y-co., orient.,
               Terminal_name, IO_type;
```

Syntaxe de la définition d'un TERMINAL multiple:

```
TERMINAL {
          T_n = conn_N, Terminal_type_name, X-co., Y-co., orient.,
                Terminal_name, IO_type;
          T_n = conn_N, Terminal_type_name, X-co., Y-co., orient.,
                Terminal_name, IO_type;
          T_n = conn_N, Terminal_type_name, X-co., Y-co., orient.,
                Terminal_name, IO_type;
        }
```

où:

8.4.5.1 T_*n*

Identificateur de borne unique, où "***n***" est le numéro effectif de la borne (nombre entier, conformément à 7.1.3.4), numérotation à partir du coin supérieur gauche dans le sens contraire des aiguilles d'une montre. Noter que le trait de soulignement est facultatif, il est utilisé pour la clarté seulement.

8.4.5.2 conn_N

Numéro de connexion, nombre entier non unique, conformément à 7.1.3.4. Ce numéro de connexion est simplement une référence d'emplacement et peut facultativement se rapporter à une broche de boîtier; peut ne pas être présent ou bien peut être zéro, 0, ou laisser vide si inconnu. L'avantage singulier de ce numéro de connexion est de spécifier et identifier plusieurs bornes qu'il est nécessaire de raccorder ensemble. Il convient de vérifier la valeur de ce numéro de connexion afin de s'assurer qu'il ne dépasse pas la valeur déterminée par CONNECTION_COUNT, lorsqu'elle est spécifiée.

8.4.5.3 Terminal_type_name

Nom textuel (conformément à 7.1.3.2) d'une forme référencée de TERMINAL_TYPE (conformément à 8.4.3.1), lequel nom doit avoir été préalablement déclaré (voir 7.1.4).

8.4.5.4 X-co

Valeur numérique réelle de la coordonnée X (conformément à 7.1.3.3) pour l'emplacement du centre de la forme de borne, en unités définies par le paramètre GEOMETRIC_UNITS. Cette valeur se rapporte à la valeur de la coordonnée X du GEOMETRIC_ORIGIN.

8.4.5.5 Y-co

Valeur numérique réelle de la coordonnée Y (conformément à 7.1.3.3) pour l'emplacement du centre de la forme de borne, en unités définies par le paramètre GEOMETRIC_UNITS. Cette valeur se rapporte à la valeur de la coordonnée Y du GEOMETRIC_ORIGIN.

8.4.5.6 Orient

Valeur d'orientation, en valeurs entières de 0 à 360, conformément à 7.1.3.4. Il s'agit de l'angle de rotation dans le sens horaire, exprimé en degrés, autour du centre de référence géométrique de la forme de borne. Si les lettres “**MX**” sont incluses, l'orientation de la forme de borne doit être miroitée dans l'axe X; de même, si les lettres “**MY**” sont incluses, l'orientation de la forme de borne doit être miroitée dans l'axe Y. “**MX**” et “**MY**” peuvent être présents simultanément (voir Annexe D pour une représentation graphique) et l'ensemble du miroitage doit être effectué autour du centre de référence géométrique de la forme de borne. Noter que l'opération de miroitage doit être réalisée en premier lieu puis la borne doit être tournée de l'angle d'orientation.

8.4.5.7 Terminal_name

Nom textuel, non unique, conformément à 7.1.3.2. Ce nom de borne sert à la référence utilisateur et à l'affichage graphique seulement et peut être omis.

8.4.5.8 IO_type

Lettre unique indiquant le type de fonction des broches de borne tel que donné au Tableau 3, non obligatoire. D'autres caractères peuvent être ajoutés selon les besoins.

Tableau 3 – Types d'I/O (c'est-à-dire E/S) de borne

Lettre	Fonction des bornes
I	**I**nput (c'est-à-dire Entrée), numérique
O	**O**utput (c'est-à-dire Sortie), numérique
B	**B**i-directional (c'est-à-dire Bidirectionnelle), numérique
G	**G**round connection (c'est-à-dire Raccordement à la terre)
V	Alimentation, il peut s'agir de n'importe quel type d'alimentation.
A	Broche **A**nalogique, entrée ou sortie
N	**N**o-connect pin (c'est-à-dire broche sans connexion), laisser déconnectée
U	**U**ndetermined (c'est-à-dire Indéterminée), I/O (c'est-à-dire E/S programmable par l'utilisateur
T	**T**est pin (c'est-à-dire broche d'essai), connecter seulement sur avis des fabricants
X	Raccordée intérieurement, ne pas raccorder
H	Broche d'entrée fonctionnelle numérique, à maintenir au niveau logique **H**aut
L	Broche d'entrée fonctionnelle numérique, à maintenir au niveau logique **L**ow (c'est-à-dire Bas)

T_n (voir 8.4.5.1) doit toujours être unique alors que le numéro de connexion de broche (Paragraphe 8.4.5.2), ***conn_N***, et le nom de borne (voir 8.4.5.7),***Terminal_name***, peuvent ne pas être uniques.

Exemple 1

```
TERMINAL Pin007 = 9, ShapeR1, (5000, 7000), MX0, VCC1, V;
```

Cet exemple déclare que la borne "**Pin007**".....

- est raccordée à la connexion arbitraire N° 9,
- est situé à X = 5 000 unités, Y = 7 000 unités,
- est un TERMINAL_TYPE dénommé "ShapeR1", miroité sur l'axe X et orienté à 0°, et
- est appelée "VCC1", et
- est une broche d'alimentation électrique.

Exemple 2

```
TERMINAL Conn08 = 17, ShapeP2, 5000, 7300, 90, qd2i, I;
TERMINAL Conn09 = 17, ShapeP2, 5000, 7800, 90, qd2o, O;
```

Ces exemples déclarent que les bornes "**Conn08**" et "**Conn09**".....

- sont toutes deux raccordées à la connexion arbitraire N° 17,
- sont situées respectivement à X = 5000 unités, Y = 7300 unités et à X = 5000 unités, Y = 7800 unités,
- sont toutes deux d'un TERMINAL_TYPE dénommé "ShapeP2", orienté (tourné) de 90° dans le sens horaire,
- sont dénommées "qd2i" et "qd2o"
- et constituent une connexion bidirectionnelle I/O numérique "qd2I" étant une entrée (I) et "qd2o" une sortie (O).

Exemple 3

```
TERMINAL Term10 = , ShapeP2, 5000, 7600, 0, , X;
```

Cet exemple déclare que la borne "**Term10**".....

- n'est pas raccordée (ou n'a pas de référence de connexion spécifique),
- est située à X = 5000 unités, Y = 7600 unités,
- est d'un TERMINAL_TYPE dénommé "ShapeP2", orienté à 0°, et
- n'a pas de nom attribué et
- est spécifiée comme une borne à laquelle rien ne doit être relié.

Exemple 4

```
TERMINAL T_44 = , ShapeR2, 25000, 97000, MXMY90, ,;
```

Cet exemple déclare que la borne **T_44**.....

- n'est pas raccordée (ou n'a pas de référence de connexion spécifique),
- est située à (25000:97000),
- est d'un TERMINAL_TYPE dénommé "ShapeR2", miroité à la fois dans l'axe X et dans l'axe Y, puis orienté à (tournée dans le sens horaire de) 90°,
- n'a pas de nom attribué, et
- n'a pas de propriétés de connexion électrique spécifiées.

Exemple 5

```
TERMINAL {
      Pin007 = 9, ShapeR1, (5000, 7000), MX0, VCC1, V;
      Conn08 = 17, ShapeP2, (5000, 7300), 90, qd2i, I;
      Conn09 = 17, ShapeP2, (5000, 7800), 90, qd2o, O;
      Term10 = , ShapeP2, (5000, 7600), 0, , X;
      T_44 = , ShapeR2, (25000, 97000), MXMY90, ,;
}
```

Cet exemple de définition de borne multiple déclare des données de borne identiques à celles montrées dans les Exemples 1 à 4.

8.4.6 Paramètre TERMINAL_GROUP

Nom de paramètre	TERMINAL_GROUP
Type du paramètre	Structure
Fonction du paramètre	La structure TERMINAL_GROUP affecte à un groupe de bornes un seul identificateur de groupe de bornes qui peut être utilisé à la place d'une liste de bornes. La structure peut être utilisée pour définir un seul groupe ou une multitude de groupes. Se référer aux règles de TERMINAL_GROUP (voir 8.4.6.1) pour les détails.
Valeurs du paramètre	Deux ou plusieurs identificateurs de borne ou de groupe, tels que définis en 8.4.5.1 et 8.4.6.3
Dépendances	TERMINAL (voir 8.4.5)
Limitations	Un identificateur de TERMINAL_GROUP doit être unique et peut seulement être référencé après qu'il a été déclaré.
Référence	Annexe B.

8.4.6.1 Règles relatives à TERMINAL_GROUP

8.4.6.1.1 Contenu

Les éléments formant un groupe peuvent être des identificateurs de bornes, des identificateurs de groupes ou tout mélange des deux. Un groupe doit contenir au moins deux éléments.

8.4.6.1.2 Unicité

Tous les éléments d'un groupe doivent être uniques. Un groupe ne peut contenir de groupes qui se réfèrent directement ou indirectement au même élément.

8.4.6.1.3 Ordre

L'ordre des éléments dans des groupes peut être important. Lorsque deux ou plusieurs groupes doivent être reliés dans une permutation, les éléments au sein de ces groupes doivent correspondre à travers les groupes.

8.4.6.1.4 Récursivité

Un groupe ne peut se contenir lui-même ni ne peut-il contenir de groupe qui se réfère directement ou indirectement à lui-même.

Syntaxe de la définition d'un TERMINAL_GROUP unique:

TERMINAL_GROUP **G_*n*** = _ *Terminal ou Group identifier, ... Terminal or Group identifier;*
TERMINAL_GROUP **G_*n*** = _ *Terminal ou Group identifier, ... Terminal or Group identifier;*

Syntaxe de la définition d'un TERMINAL_GROUP multiple:

TERMINAL_GROUP {
G_*n* = _*Terminal ou Group identifier, ... Terminal or Group identifier;*
G_*n* = _ *Terminal ou Group identifier, ... Terminal ou Group identifier;*
}

où:

8.4.6.2 G_*n*

L'identificateur de TERMINAL_GROUP unique "***n***" est le numéro réel du TERMINAL_GROUP (Nombre entier, conformément à 7.1.3.4). Noter que le trait de soulignement est facultatif, il est utilisé pour la clarté seulement.

8.4.6.3 Identificateur de borne ou de groupe *(Terminal ou Group identifier)*

Deux ou plusieurs noms d'identificateur de borne ou de groupe de bornes

- Tout nom d'identificateur de borne doit être déclaré au préalable dans une déclaration de TERMINAL (voir 8.4.4.1). Un nom d'identificateur de borne ne peut apparaître qu'une seule fois dans une affectation de groupe unique.
- Tout nom d'identificateur de groupe de bornes doit être déclaré au préalable dans une déclaration de TERMINAL_GROUP (voir 8.4.6.2). Un nom d'identificateur de groupe de bornes peut apparaître une seule fois dans une seule affectation de groupe. et la circularité ou récursivité d'affectation est interdite.

Exemple

```
TERMINAL_GROUP G_1 = T_10, T_11, T_12;
TERMINAL_GROUP G_2 = T_15, T_16, T_12;

TERMINAL_GROUP  {
     G_3 = T_1, T_2;
     G_4 = T_4, T_5;
     G_5 = G_1, T_3;
     G_6 = G_3, G_4, T_11;
}
```

Notes

Dans tous les cas, un identificateur de borne unique doit seulement apparaître une seule fois dans un groupe. Cela est également pertinent lorsqu'il s'agit de considérer l'expansion d'un groupe de bornes.
Se référer à l'Annexe B.

8.4.7 Paramètre PERMUTABLE

Nom de paramètre	PERMUTABLE
Type du paramètre	Structure
Fonction du paramètre	La structure PERMUTABLE affecte une liste de bornes ou de groupes de bornes qui sont permutables. La liste peut comprendre des identificateurs de bornes (voir 8.4.5.1) ou des identificateurs de groupes de bornes (conformément à 8.4.6.2) mais pas un mélange des deux types d'identificateur. Chaque affectation PERMUTABLE déclare que les bornes ou groupes de bornes énuméré(e)s sont échangeables et/ou permutables en termes de fonctionnalité. Cela sert à faciliter la connexion et le routage alternatifs par un logiciel de CAO. Se référer à Règles de PERMUTABLE (voir 8.4.7.1) pour les détails. Se référer à l'Annexe B pour d'autres explications.
Valeurs du paramètre	La déclaration du paramètre PERMUTABLE doit consister soit en une liste de deux ou plusieurs identificateurs de bornes (voir 8.4.6.1), soit en deux ou plusieurs identificateurs de groupes de bornes (voir 8.4.6.1), mais ne doit pas contenir de mélange d'identificateurs de bornes et d'identificateurs de groupes de bornes.
Dépendances	TERMINAL (voir 8.4.5) TERMINAL_GROUP (voir 8.4.6)
Notes	Dans tous les cas, un identificateur de borne unique doit seulement apparaître une seule fois dans un groupe. Cela est pertinent lorsqu'il s'agit de considérer l'expansion d'un groupe de bornes. Pour la permutation correcte de groupes de bornes dans une affectation de PERMUTABLE unique, il est obligatoire que la séquence de bornes dans chaque groupe corresponde directement à la séquence de bornes des autres groupes de bornes au sein de l'affectation
Référence	Annexe B.

8.4.7.1 Règles relatives à PERMUTABLE

8.4.7.1.1 Contenu

Les éléments formant une permutation peuvent être des identificateurs de bornes ou des identificateurs de groupes mais pas un mélange des deux. Une permutation doit contenir au moins deux éléments.

8.4.7.1.2 Unicité

Tous les éléments d'une permutation doivent être uniques. Une permutation ne peut contenir de groupes qui se réfèrent directement ou indirectement au même élément.
Chaque élément d'une permutation doit contenir le même nombre d'identificateurs de bornes (au sein de groupes de bornes ou non) que les autres.

8.4.7.1.3 Ordre

L'ordre des éléments dans une permutation n'a pas d'importance.

Syntaxe de la définition d'un PERMUTABLE unique:

PERMUTABLE **P_*n*** = *Terminal identifier, ... Terminal identifier;*
PERMUTABLE **P_*n*** = *Terminal identifier, ... Terminal identifier;*
PERMUTABLE **P_*n*** = *Group identifier, ... Group identifier;*
PERMUTABLE **P_*n*** = *Group identifier, ... Group identifier;*

Syntaxe de la définition d'un PERMUTABLE multiple:

PERMUTABLE {
P_*n* = *Terminal identifier, ... Terminal identifier;*
P_*n* = *Terminal identifier, ... Terminal identifier;*
P_*n* = *Group identifier, ... Group identifier;*
P_*n* = *Group identifier, ... Group identifier;*
}

Exemple
```
PERMUTABLE P_1 = T_1, T_2, T_3;
PERMUTABLE P_2 = GP1, GP2, GP3, GP5, GP4;

PERMUTABLE {
     P_1 = T_1, T_2;
     P_2 = T_4, T_5;
     P_3 = T_9, T_10;
     P_4 = T_12, T_13;
     P_5 = G_5, G_6, G_7, G_8;
}
```

8.5 DONNÉES RELATIVES AUX MATÉRIAUX

8.5.1 Paramètre TERMINAL_MATERIAL

Nom de paramètre	**TERMINAL_MATERIAL**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le matériau utilisé pour les terminaisons finales sur le dispositif
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.

Exemple	`TERMINAL_MATERIAL = “Al”;`
	`TERMINAL_MATERIAL = “Cuivre”;`
	`TERMINAL_MATERIAL = “SAC405”;`

8.5.2 Paramètre TERMINAL_MATERIAL_STRUCTURE

Nom de paramètre	TERMINAL_MATERIAL_STRUCTURE
Type du paramètre	Variable
Fonction du paramètre	Spécifie la structure séquentielle des matériaux utilisés pour établir les terminaisons sur le dispositif.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`TERMINAL_MATERIAL_STRUCTURE = “Al-Ti-Ni-Au”;`

8.5.3 Paramètre DIE_SEMICONDUCTOR_MATERIAL

Nom de paramètre	**DIE_SEMICONDUCTOR_MATERIAL**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le matériau semiconducteur qui est utilisé pour la fabrication de la puce.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.

Exemple	`DIE_SEMICONDUCTOR_MATERIAL = “Insulator”;`
	`DIE_SEMICONDUCTOR_MATERIAL = “Silicon”;`
	`DIE_SEMICONDUCTOR_MATERIAL = “Sapphire”;`

Notes	Ce paramètre est pertinent seulement si le matériau de base de la puce est différent du matériau du substrat de la puce.

8.5.4 Paramètre DIE_SUBSTRATE_MATERIAL

Nom de paramètre	**DIE_SUBSTRATE_MATERIAL**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le matériau de base ou de substrat sur lequel la puce est fabriquée lorsqu'il diffère du DIE_SEMICONDUCTOR_MATERIAL.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Doit être déclaré une seule fois dans un seul bloc **DEVICE**.

Exemple	`DIE_SUBSTRATE_MATERIAL = “Silicon”;`
	`DIE_SUBSTRATE_MATERIAL = “Insulator”;`
	`DIE_SUBSTRATE_MATERIAL = “Sapphire”;`

8.5.5 Paramètre DIE_SUBSTRATE_CONNECTION

Nom de paramètre **DIE_SUBSTRATE_CONNECTION**
Type du paramètre Variable
Fonction du paramètre Spécifie la connexion électrique pour le substrat, s'il en existe. Par fabrication, le substrat peut déjà être connecté électriquement, ce qu'il convient également de déclarer. Lorsqu'il convient d'établir une connexion électrique, le point de potentiel pour la connexion doit aussi être déclaré.
Valeurs du paramètre Une ou plusieurs données de chaîne textuelle, conformément à 7.1.3.1. Le premier paramètre doit être conforme aux valeurs données dans le Tableau 4.
Limitations Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple

```
DIE_SUBSTRATE_CONNECTION = “CONN”,”Ground”;
DIE_SUBSTRATE_CONNECTION = “N/A”;
DIE_SUBSTRATE_CONNECTION = “ISOL”;
DIE_SUBSTRATE_CONNECTION = “OPT”,”Most Negative”;
DIE_SUBSTRATE_CONNECTION = “OPT”,”Most Positive”;
DIE_SUBSTRATE_CONNECTION = “CONN”,”Digital Vdd”;
DIE_SUBSTRATE_CONNECTION = “OPT”,”Analog ground”;
```

Tableau 4 – Paramètres de connexion au substrat

Valeur du paramètre	Fonction
CONN	**Doit** être connecté électriquement.
ISOL	**Doit** être isolé électriquement
OPT	Peut être connecté facultativement
N/A	Non applicable
N/K	Non connu (inconnu)

Notes Lorsque “CONN” ou “OPT” est donné comme première valeur de paramètre, la deuxième valeur de paramètre doit être complètement déclarée, de manière suffisamment adéquate pour être comprise pour la connectivité électrique. Les valeurs de paramètre recommandées sont: “Most Positive”, “Most Negative”, “GND”, “AGND”, “DGND”, “VSS”, “VDD”, “VEE”, “VCC”, “AVSS”, “AVDD”, “AVEE”, “AVCC”, “DVSS”, “DVDD”, “DVEE”, “DVCC”,”VBIAS”, ou “SPECIAL”.
En variante, un nom de **TERMINAL** spécifique “**T_n**”, tel que décrit en 8.4.5, peut être utilisé comme seconde valeur de paramètre pour la connexion directe au substrat.

8.5.6 Paramètre DIE_PASSIVATION_MATERIAL

Nom de paramètre **DIE_PASSIVATION_MATERIAL**
Type du paramètre Variable
Fonction du paramètre Spécifie le matériau de passivation utilisé pour la surface de la puce.
Valeurs du paramètre Données de chaîne textuelle, conformément à 7.1.3.1
Limitations Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple

```
DIE_PASSIVATION_MATERIAL = “Silicon Nitride”;
DIE_PASSIVATION_MATERIAL = “Oxy-Nitride”;
DIE_PASSIVATION_MATERIAL = “Polyimide”;
```

8.5.7 Paramètre DIE_BACK_DETAIL

Nom de paramètre **DIE_BACK_DETAIL**
Type du paramètre Variable
Fonction du paramètre Spécifie la finition en détail au verso de la puce, pertinente pour différentes méthodes de fixation/différents types de boîtier, afin d'inclure à la fois la finition et le matériau.
Valeurs du paramètre Données de chaîne textuelle, conformément à 7.1.3.1
Limitations Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple

```
DIE_BACK_DETAIL = “Sawn”;
```

```
DIE_BACK_DETAIL = “Au-Metallized”;
DIE_BACK_DETAIL = “Backlapped”;
DIE_BACK_DETAIL = “Polished”;
DIE_BACK_DETAIL = “Gold Metal”;
DIE_BACK_DETAIL = “None”;
```

8.6 DONNÉES RELATIVES AUX CARACTÉRISTIQUES ASSIGNÉES ÉLECTRIQUES ET THERMIQUES

8.6.1 Paramètre MAX_TEMP

Nom de paramètre	**MAX_TEMP**
Type du paramètre	Variable
Fonction du paramètre	Spécifie la température maximale à laquelle le dispositif peut être exposé au cours d'une partie de la fixation de la puce, du brasage du dispositif ou autre processus de fabrication
Valeurs du paramètre	Real, conformément à 7.1.3.3.
Unités du paramètre	^{0}C, Celsius
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`MAX_TEMP = 280;`

8.6.2 Paramètre MAX_TEMP_TIME

Nom de paramètre	**MAX_TEMP_TIME**
Type du paramètre	Variable
Fonction du paramètre	Spécifie la durée maximale pendant laquelle le dispositif peut être exposé à la température maximale au cours d'une partie de la fixation de la puce, du brasage du dispositif ou autre processus de fabrication.
Valeurs du paramètre	Real, conformément à 7.1.3.3.
Unités du paramètre	Secondes
Dépendances	MAX_TEMP
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`MAX_TEMP_TIME = 10;`

8.6.3 Paramètre POWER_RANGE

Nom de paramètre	**POWER_RANGE**
Type du paramètre	Variable
Fonction du paramètre	Spécifie la puissance susceptible d'être dissipée par le dispositif.
Valeurs du paramètre	Real, conformément à 7.1.3.3.
Unités du paramètre	Watts
Limitations	Doit être déclaré une seule fois dans un seul bloc **DEVICE**.
Exemple	`POWER_RANGE = 0.92;`
Notes	Il convient d'utiliser ce paramètre avec prudence car, faute de spécifier complètement toutes les conditions de mesure, l'utilisation de cette valeur peut être mal comprise. Il convient que la valeur indique la puissance maximale vraisemblablement dissipée dans les conditions types de cas le plus défavorable.

8.6.4 Paramètre TEMPERATURE_RANGE

Nom de paramètre	**TEMPERATURE_RANGE**
Type du paramètre	Variable
Fonction du paramètre	Spécifie la plage de fonctionnement et de spécification de la puce.
Valeurs du paramètre	Minimum (Réel, en ^{0}C), Maximum (Réel, en ^{0}C), conformément à 7.1.3.3.
Unités du paramètre	^{0}C, Celsius
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`TEMPERATURE_RANGE = -40, 90;`
Notes	Dans le cas des puces nues, il convient d'utiliser ce paramètre avec prudence et il est seulement destiné à indiquer le niveau de

température opérationnelle ou spécifiée de la pièce emballée équivalente.

8.7 DONNÉES DE SIMULATION

8.7.1 Paramètre MODEL FILE de simulateur

Nom de paramètre	**SIMULATOR_*simulator*_MODEL_FILE**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le nom du fichier modèle à utiliser avec *simulator*.
Valeurs du paramètre	Données de nom textuel, comme en 7.1.3.2
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE par type de *simulator*.
Exemple	`SIMULATOR_SPICE_MODEL_FILE = “BC109.MOD”;` `SIMULATOR_SPECTRE_MODEL_FILE = “RTBAA1.S”;`
Notes	Seuls doivent être utilisés les noms de fichier, sans nom de chemin relatif ou absolu.
Référence	Annexe D.

8.7.2 Paramètre MODEL FILE DATE de simulateur

Nom de paramètre	**SIMULATOR_*simulator*_MODEL_FILE_DATE**
Type du paramètre	Variable
Fonction du paramètre	Spécifie la date de création ou de validation du fichier modèle.
Valeurs du paramètre	Date, conformément à 7.1.3.5
Limitations	Doit être déclaré une seule fois dans un seul bloc **DEVICE** par type de *simulator*.
Exemple	`SIMULATOR_SPICE_MODEL_FILE_DATE=”19951021”;` `SIMULATOR_VHDL_MODEL_FILE_DATE=”1993-05-17”;`
Notes	Il convient que cette date coïncide avec la date du fichier modèle réel, pour indiquer que le fichier modèle correct est en cours d'utilisation, et pour une utilisation dans une fonction de création/établissement de bibliothèque.
Référence	Annexe D.

8.7.3 Paramètre NAME de simulateur

Nom de paramètre	**SIMULATOR_*simulator*_NAME**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le nom du *simulator* générique ou spécifique pour lequel le ficher modèle est approprié.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE par type de *simulator*.
Exemple	`SIMULATOR_SPICE_NAME = “pSpice”;` `SIMULATOR_VERILOG_NAME = “Verilog-XL”;`
Référence	Annexe D.

8.7.4 Paramètre VERSION de simulateur

Nom de paramètre	**SIMULATOR_*simulator*_VERSION**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le numéro de version du *simulator* tel que spécifié par le paramètre SIMULATOR_*simulator*_NAME qui avait été utilisé pour vérifier les données de modèle.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE par type de *simulator*.
Exemple	`SIMULATOR_SPICE_VERSION = “4.0.1”;` `SIMULATOR_VERILOG_VERSION = “1.7B”;`
Référence	Annexe D.

8.7.5 Paramètre COMPLIANCE de simulateur

Nom de paramètre	**SIMULATOR_*simulator*_COMPLIANCE**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le niveau minimal de conformité du *simulateur* requis pour à la fois reproduire avec précision les résultats de la simulation et se corréler à ceux-ci.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE par type de *simulator*.
Exemple	`SIMULATOR_VHDL_COMPLIANCE = “VHDL ’93”;` `SIMULATOR_SPICE_COMPLIANCE = “2G6”;`
Référence	Annexe D.

8.7.6 Paramètre TERM_GROUP de simulateur

Nom de paramètre	**SIMULATOR_*simulator*_TERM_GROUP**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le groupe ou les bornes au(x)quel(les) s'applique le modèle de simulateur. S'il est absent, on suppose que le modèle s'applique à l'ensemble du dispositif.
Valeurs du paramètre	Liste de bornes (conformément à 8.4.5.1) et/ou de groupes de bornes (conformément à 8.4.6.2)
Limitations	Les bornes ou groupes de bornes référencé(e)s doivent avoir déjà été déclaré(e)s soit dans un paramètre TERMINAL, soit dans un paramètre TERMINAL_GROUP. Doit être déclaré une seule fois dans un seul bloc DEVICE par type de *simulator*.
Exemple	`SIMULATOR_SPICE_TERM_GROUP = T_1, T_2, G_1;` `SIMULATOR_IBIS_TERM_GROUP = G_5, G_8, T_11, T_33;`
Référence	Annexes B et D.

8.8 DONNÉES RELATIVES À LA MANUTENTION, AU CONDITIONNEMENT, AU STOCKAGE ET À L'ASSEMBLAGE

8.8.1 Paramètre DELIVERY_FORM

Nom de paramètre	**DELIVERY_FORM**
Type du paramètre	Variable
Fonction du paramètre	Spécifie la forme dans laquelle la puce est livrée.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.

Exemple	`DELIVERY_FORM = “Wafer”;`
	`DELIVERY_FORM = “Die”;`
	`DELIVERY_FORM = “Bare Die in Waffle Tray”;`
	`DELIVERY_FORM = “Sawn Wafer”;`
	`DELIVERY_FORM = “Tray”;`
	`DELIVERY_FORM = “Bandoleer”, “Waffle”;`

8.8.2 Paramètre PACKING_CODE

Nom de paramètre	**PACKING_CODE**
Type du paramètre	Variable
Fonction du paramètre	Spécifie la forme du conditionnement primaire utilisé pour la livraison des dispositifs.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1

Exemple	`PACKING_CODE = “WAFFLE”;`
	`PACKING_CODE = “GEL_PACK”;`
	`PACKING_CODE = “Bandoleer”;`

8.8.3 Paramètres ASSEMBLY

Nom de paramètre	ASSY_ *text-identifier*
Type du paramètre	Variable
Fonction du paramètre	Paramètre textuel général, pour permettre à l'utilisateur d'inclure des techniques, paramètres et fonctionnements pertinents et identifiés.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Exemples	`ASSY_PROCESS_LIMITATIONS = “NONE”;`

Notes — Les identificateurs *ASSY_identifiers* reconnus sont:

```
8.8.3.1    ASSY_PROCESS_LIMITATIONS
8.8.3.2    ASSY_STORAGE_LIMITATIONS
8.8.3.3    ASSY_ASSEMBLY_LIMITATIONS
8.8.3.4    ASSY_TEMPERATURE_LIMITATIONS
8.8.3.5    ASSY_BONDING_METHODS
8.8.3.6    ASSY_BONDING_MATERIALS
8.8.3.7    ASSY_ATTACH_METHODS
8.8.3.8    ASSY_ATTACH_MATERIALS
8.8.3.9    ASSY_GENERAL_REQUIREMENTS
8.8.3.10   ASSY_HANDLING_REQUIREMENTS
8.8.3.11   ASSY_PACKING_REQUIREMENTS
8.8.3.12   ASSY_STORAGE_REQUIREMENTS
8.8.3.13   ASSY_SHIPPING_REQUIREMENTS
```

8.9 DONNÉES SPÉCIFIQUES AUX TRANCHES

8.9.1 Paramètre WAFER_SIZE

Nom de paramètre	**WAFER_SIZE**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le diamètre de la tranche.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`WAFER_SIZE = “6 inch”;` `WAFER_SIZE = “150mm”;`
Notes	Sachant qu'il s'agit d'une valeur dimensionnelle mais reliée à la fabrication et non à la taille de la puce, il est permis d'exprimer le paramètre WAFER_SIZE dans des unités différentes de celles définies par le paramètre GEOMETRIC_UNITS et, donc, les données sont présentées en texte de forme libre.

8.9.2 Paramètre WAFER_THICKNESS

Nom de paramètre	**THICKNESS**
Type du paramètre	Variable
Fonction du paramètre	Détermine l'épaisseur (dimension Z) de la tranche.
Valeurs du paramètre	Valeur de la dimension Z en nombre réel, conformément à 7.1.3.3.
Dépendances	GEOMETRIC_UNITS
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`WAFER_THICKNESS = 10.5;` `WAFER_THICKNESS = 80;`
Notes	Les valeurs de dimensions sont exprimées en GEOMETRIC_UNITS.

8.9.3 Paramètre WAFER_THICKNESS_TOLERANCE

Nom de paramètre	**WAFER_THICKNESS_TOLERANCE**
Type du paramètre	Variable
Fonction du paramètre	Spécifie l'une ou les plusieurs tolérances du paramètre WAFER_THICKNESS qui peuvent être attendues en raison des variations normales de processus.
Valeurs du paramètre	Réelles en GEOMETRIC_UNITS, conformément à 7.1.3.3.
Dépendances	GEOMETRIC_UNITS, WAFER_THICKNESS
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.

Exemple	`WAFER_THICKNESS_TOLERANCE = 2.5;` `WAFER_THICKNESS_TOLERANCE = -1.2,1.5;`
Notes	Une ou deux valeurs peuvent être données: Lorsqu'une tolérance unique est donnée, la tolérance doit être prise comme étant une valeur non signée ±. Lorsque deux valeurs de tolérance sont données: la première valeur doit être prise comme le minimum et la seconde valeur doit être prise comme le maximum.

8.9.4 Paramètre WAFER_DIE_STEP_SIZE

Nom de paramètre	**WAFER_DIE_STEP_SIZE**
Type du paramètre	Variable
Fonction du paramètre	Détermine les dimensions de taille de pas en X et en Y, en GEOMETRIC_UNITS, pour la puce sur la tranche, telles que requises par le matériel de manutention mécanique.
Valeurs du paramètre	Dimension X en nombre réel, dimension Y en nombre réel (conformément à 7.1.3.3)
Dépendances	GEOMETRIC_UNITS, GEOMETRIC_VIEW
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`WAFER_DIE_STEP_SIZE = 280, 530;` `WAFER_DIE_STEP_SIZE = 1500, 1500;`
Référence	Annexe H.

8.9.5 Paramètre WAFER_GROSS_DIE_COUNT

Nom de paramètre	**WAFER_GROSS_DIE_COUNT**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le nombre de puces brutes entières et viables (ayant le type de puce en question) disponibles sur la tranche.
Valeurs du paramètre	Integer, conformément à 7.1.3.4.
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`WAFER_GROSS_DIE_COUNT = 2027;`
Notes	Cette valeur de paramètre est seulement destinée à donner une indication du nombre de puces pertinentes et viables par tranche et ne doit pas avoir d'autre implication.
Référence	Annexe H.

8.9.6 Paramètre WAFER_INDEX

Nom de paramètre	**WAFER_INDEX**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le type de la caractéristique d'index sur une tranche qui agit comme une caractéristique de référence et son orientation par rapport à l'axe X pour la puce. Le premier paramètre détermine le type de la caractéristique d'index tandis que le second paramètre spécifie la relation angulaire approchée entre l'axe X supposé pour la puce et la caractéristique d'index sur la tranche. L'angle doit être donné, en degrés comptés dans le sens horaire, de la caractéristique d'index, en prenant l'axe X de la puce comme référence.
Valeurs du paramètre	Données en chaîne textuelle, conformément à 7.1.3.1, avec la valeur “Flat” ou “Notch”, suivie d'une seule valeur entière, en unités de degrés, de 0 à 359, conformément à 7.1.3.4.
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`WAFER_INDEX = “Flat”,90;` `WAFER_INDEX = “Notch”,0;`
Notes	Ce paramètre est seulement censé être un guide et, donc, des approximations entières doivent être acceptables. Les angles négatifs sont interdits. S'il existe plus d'un méplat sur la tranche, l'orientation

est par rapport au méplat majeur ou principal. Cela peut aussi indiquer la direction cristallographique [110].

Référence Annexe H.

8.9.7 Paramètre WAFER_RETICULE_STEP_SIZE

Nom de paramètre **WAFER_RETICULE_STEP_SIZE**
Type du paramètre Variable
Fonction du paramètre Spécifie les dimensions répétées de pas en X et en Y et les dimensions répétées pour un réticule unique.
Valeurs du paramètre Dimension X en nombre réel, dimension Y en nombre réel (conformément à 7.1.3.3)
Dépendances GEOMETRIC_UNITS, GEOMETRIC_VIEW
Limitations Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple
```
WAFER_RETICULE_STEP_SIZE = 2500, 8000;
WAFER_RETICULE_STEP_SIZE = 15000, 27500;
```
Notes Ce paramètre est surtout pertinent pour les tranches MPW (Multi Project Wafers, c'est-à-dire tranches multiprojet), ou des instances où WAFER_DIE_STEP_SIZE devient invalide en raison d'une relation non entière entre la taille du réticule et la taille de la puce.
Référence Annexe H.

8.9.8 Paramètre WAFER_RETICULE_GROSS_DIE_COUNT

Nom de paramètre **WAFER_RETICULE_GROSS_DIE_COUNT**
Type du paramètre Variable
Fonction du paramètre Spécifie le nombre de puces brutes entières et viables (ayant le type de puce en question) placées dans un réticule unique.
Valeurs du paramètre Integer, conformément à 7.1.3.4.
Limitations Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple
```
WAFER_RETICULE_GROSS_DIE_COUNT = 40;
```
Notes Ce paramètre est surtout pertinent pour les tranches MPW (tranches multiprojet) ou les instances où le réticule contient plus d'un type de puce. Cette valeur de paramètre est seulement destinée à donner une indication du nombre de puces pertinentes et viables par réticule et ne doit pas avoir d'autre implication.
Référence Annexe H.

8.9.9 Paramètres WAFER_INK

Nom de paramètre WAFER_INK_*text-identifier*
Type du paramètre Variable
Fonction du paramètre Paramètre textuel général, pour permettre à l'utilisateur d'inclure des paramètres pertinents relatifs à la taille et à la couleur des points/encre de la tranche. Cela revêt une importance particulière pour les systèmes visuels de sélection, tri et examen des puces.
Valeurs du paramètre Données de chaîne textuelle, conformément à 7.1.3.1
Exemples
```
WAFER_INK_COLOUR = “BLACK”;
WAFER_INK_LOCATION = “CENTRAL”;
WAFER_INK_SIZE_MAX = “0.8mm”;
WAFER_INK_SORT_COLOUR = “BIN1, RED, UPPER RIGHT”;
WAFER_INK_SORT_COLOUR = “BIN2, GREEN, LOWER LEFT”;
```

Notes Les identificateurs *WAFER_INK_identifiers* reconnus sont:

```
8.9.9.1    WAFER_INK_COLOUR
8.9.9.2    WAFER_INK_SIZE
8.9.9.3    WAFER_INK_SIZE_TOL
8.9.9.4    WAFER_INK_SIZE_MAX
8.9.9.5    WAFER_INK_LOCATION
8.9.9.6    WAFER_INK_LOCATION_TOL
8.9.9.7    WAFER_INK_HEIGHT_MAX
8.9.9.8    WAFER_INK_SORT_COLOUR
```

8.10 DONNÉES SPÉCIFIQUES AUX TERMINAISONS À BOSSES

Toute donnée fournie qui est pertinente pour un dispositif «à bosses» (un dispositif utilisant des structures de connexion à bosses ajoutées) peut seulement refléter les paramètres de bosses au moment de la formation des bosses et pas nécessairement les paramètres finals. En raison de la nature de la connexion à bosses finale, les paramètres de bosse seront altérés et déformés au cours du processus de fixation. Il convient donc de considérer tous les paramètres géométriques et métallurgiques donnés comme relatifs aux bosses «vierges»

8.10.1 Paramètre BUMP_MATERIAL

Nom de paramètre	**BUMP_MATERIAL**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le matériau métallurgique du plot de contact.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.

Exemple	`BUMP_MATERIAL = “Copper”;`
	`BUMP_MATERIAL = “Molybdenum Telluride”;`
	`BUMP_MATERIAL = “Au”;`
	`BUMP_MATERIAL = “SAC415”;`
	`BUMP_MATERIAL = “Pb-Sn”;`

Notes	Requis seulement pour les types de dispositifs avec des structures de connexion à bosses. Lorsqu'un dispositif comporte à la fois des bornes à bosses et sans bosses, il convient d'utiliser TERMINAL_MATERIAL pour décrire les bornes sans bosses et d'utiliser BUMP_MATERIAL pour décrire la borne à bosse.

8.10.2 Paramètre BUMP_HEIGHT

Nom de paramètre	**BUMP_HEIGHT**
Type du paramètre	Variable
Fonction du paramètre	Spécifie la tolérance sur la hauteur du plot de contact au-dessus de la surface de passivation.
Valeurs du paramètre	Réelles numériques, en GEOMETRIC_UNITS, conformément à 7.1.3.3.
Dépendances	GEOMETRIC_UNITS
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`BUMP_HEIGHT = 150.0;`
Notes	Requis seulement pour les types de dispositifs avec des structures de connexion à bosses. La hauteur de la bosse peut seulement être déclarée avant que la bosse ne soit modifiée par une fixation et/ou une connexion permanente.

8.10.3 Paramètre BUMP_HEIGHT_TOLERANCE

Nom de paramètre	**BUMP_HEIGHT_TOLERANCE**
Type du paramètre	Variable
Fonction du paramètre	Spécifie la tolérance sur la hauteur du plot de contact au-dessus de la surface de passivation.
Valeurs du paramètre	Réelles numériques, en GEOMETRIC_UNITS, conformément à 7.1.3.3.
Dépendances	GEOMETRIC_UNITS, BUMP_HEIGHT
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`BUMP_HEIGHT_TOLERANCE = 35.0;` `BUMP_HEIGHT_TOLERANCE = -10.0,25.0;`
Notes	Une ou deux valeurs peuvent être données: Lorsqu'une tolérance unique est donnée, la tolérance doit être prise comme étant une valeur non signée ±. Lorsque deux valeurs de tolérance sont données: la première valeur doit être prise comme le minimum et la seconde valeur doit être prise comme le maximum.

Requis seulement pour les types de dispositifs avec des structures de connexion à bosses.

8.10.4 Paramètre BUMP_SHAPE

Nom de paramètre	**BUMP_SHAPE**
Type du paramètre	Variable
Fonction du paramètre	Spécifie la forme de bosse du contact en hauteur.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`BUMP_SHAPE=”Pyramidical”;` `BUMP_SHAPE=”Ball”;`
Notes	Requis seulement pour les types de dispositifs avec des structures de connexion à bosses. La forme de la bosse peut seulement être déclarée avant que la bosse ne soit modifiée par une fixation et/ou une connexion permanente.

8.10.5 Paramètre BUMP_SIZE

Nom de paramètre	**BUMP_SIZE**
Type du paramètre	Variable
Fonction du paramètre	Spécifie la taille de bosse en coordonnées X et -Y.
Valeurs du paramètre	Réelles numériques, en GEOMETRIC_UNITS, conformément à 7.1.3.3.
Dépendances	GEOMETRIC_UNITS
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`BUMP_SIZE= “150,150”;`
Notes	Requis seulement pour les types de dispositifs avec des structures de connexion à bosses. La taille de la bosse peut seulement être déclarée avant que la bosse ne soit modifiée par une fixation et/ou une connexion permanente.

8.10.6 Paramètre BUMP_SPECIFICATION_DRAWING

Nom de paramètre	**BUMP_SPECIFICATION_DRAWING**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le(s) nom(s) du/des fichier(s) de dessin de spécification des bosses.
Valeurs du paramètre	Données de nom textuel, comme en 7.1.3.2
Exemple	`BUMP_SPECIFICATION_DRAWING = “BumpShape1.JPG”;` `BUMP_SPECIFICATION_DRAWING = “BS1.JPG”, “BS2.TIF”;`
Notes	Seuls doivent être utilisés les noms de fichier, sans nom de chemin relatif ou absolu. Requis seulement pour les types de dispositifs avec des structures de connexion à bosses. Tout dessin de la bosse peut seulement refléter l'état de la bosse avant sa modification par une fixation et/ou une connexion permanente. Plusieurs paramètres (fichiers de dessins de spécification) peuvent être introduits dans une déclaration unique ou bien plusieurs déclarations peuvent être utilisées pour le même effet. Il n'existe aucune exigence de mise à l'échelle ou positionnelle
Référence	Annexe I, Note 3 et 4.

8.10.7 Paramètre BUMP_ATTACHMENT_METHOD

Nom de paramètre	**BUMP_ATTACHMENT_METHOD**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le type de méthode de fixation requis ou conseillé pour ce type de bosse.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`BUMP_ATTACHMENT_METHOD = “Thermocompression”;` `BUMP_ATTACHMENT_METHOD = “Solder Reflow”;`

Notes	Requis seulement pour les types de dispositifs avec des structures de connexion à bosses.

8.11 DONNÉES SPÉCIFIQUES AUX DISPOSITIFS À ENCAPSULATION RÉDUITE (MPD, MINIMALLY PACKAGED DEVICE)

8.11.1 Paramètre MPD_PACKAGE_MATERIAL

Nom de paramètre	**MPD_PACKAGE_MATERIAL**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le matériau du corps d'emballage utilisé comme encapsulation finale du MPD.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1.
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.

Exemple	`MPD_PACKAGE_MATERIAL = “Plastic”;`
	`MPD_PACKAGE_MATERIAL = “Epoxy”;`
	`MPD_PACKAGE_MATERIAL = “Ceramic”;`
	`MPD_PACKAGE_MATERIAL = “Metal Can”;`

Notes	En général, requis pour des besoins d'assemblage

8.11.2 Paramètre MPD_PACKAGE_STYLE

Nom de paramètre	**MPD_PACKAGE_STYLE**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le code pour le type de boîtier tel que défini dans une norme ou tel que donné par le fabricant.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1.
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.

Exemple	`MPD_PACKAGE_STYLE = “SOT-23”;`
	`MPD_PACKAGE_STYLE = “MO”;`
	`MPD_PACKAGE_STYLE = “TSOP-48”;`

8.11.3 Paramètre MPD_CONNECTION_TYPE

Nom de paramètre	**MPD_CONNECTION_TYPE**
Type du paramètre	Variable
Fonction du paramètre	Spécifie une méthode de connexion au MPD, telle que fil conducteur, bosse, etc.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`MPD_CONNECTION_TYPE = “Bump”;` `MPD_CONNECTION_TYPE = “TAB Lead”;`

8.11.4 Paramètre MPD_MSL_LEVEL

Nom de paramètre	**MPD_MSL_LEVEL**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le niveau de sensibilité à l'humidité de l'emballage MPD, conformément à l'IPC/JEDEC J-STD-033B:2007 – Handling, Packing, Shipping and Use of Moisture/Reflow Sensitive Surface Mount Devices, 2005.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemple	`MPD_MSL_LEVEL = “3”;`

8.11.5 Paramètre MPD_PACKAGE_DRAWING

Nom de paramètre	**MPD_PACKAGE_DRAWING**
Type du paramètre	Variable
Fonction du paramètre	Spécifie le(s) nom(s) du(des) fichier(s) de dessin de spécification des emballages de MPD.

Valeurs du paramètre	Données de nom textuel, conformément à 7.1.3.2
Exemple	`MPD_PACKAGE_DRAWING = “PackageOutline.PDF”;` `MPD_PACKAGE_DRAWING = “PO1.JPG”,”PO1A.GIF”;`
Notes	Seuls doivent être utilisés les noms de fichier, sans nom de chemin relatif ou absolu. Plusieurs paramètres (fichiers de dessins de boîtier) peuvent être introduits dans une déclaration unique ou bien plusieurs déclarations peuvent être utilisées pour le même effet. Il n'existe aucune exigence de mise à l'échelle ou positionnelle
Référence	Annexe I, Notes 3 et 4.

8.12 DONNÉES DE QUALITÉ, FIABILITÉ ET ESSAIS

8.12.1 Paramètres QUALITY

Nom de paramètre	**QUAL_*identificateur***
Type du paramètre	Variable
Fonction du paramètre	Paramètre textuel général, pour permettre à l'utilisateur d'inclure des paramètres pertinents et identifiés de qualité et de fiabilité.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Chaque identificateur doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemples	`QUAL_OUTGOING_QUALITY_LEVEL = “….”;`
Notes	Les identificateurs *QUAL_identifiers* reconnus sont:

```
8.12.1.1   QUAL_OUTGOING_QUALITY_LEVEL
8.12.1.2   QUAL_OUTGOING_QUALITY_UNITS
8.12.1.3   QUAL_OUTGOING_QUALITY_DESCRIPTION
8.12.1.4   QUAL_RELIABILITY_VALUE
8.12.1.5   QUAL_RELIABILITY_UNITS
8.12.1.6   QUAL_RELIABILITY_REFERENCE
8.12.1.7   QUAL_RELIABILITY_CONDITIONS
8.12.1.8   QUAL_RELIABILITY_CALC_METHOD
8.12.1.9   QUAL_STANDARDS_COMPLIANCE
```

8.12.2 Paramètres TEST

Nom de paramètre	**TEST_*identificateur***
Type du paramètre	Variable
Fonction du paramètre	Paramètre textuel général, pour permettre à l'utilisateur d'inclure des paramètres pertinents et identifiés relatifs aux essais.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Chaque identificateur doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemples	`TEST_ADDITIONAL_REGUIREMENTS = “NONE”;` `TEST_SCREEN_COMPLIANCE = ”MIL-883B”;`
Notes	Les identificateurs *TEST_identifiers* reconnus sont:

```
8.12.2.1   TEST_ELECTRICAL_CONDITIONS
8.12.2.2   TEST_ADDITIONAL_SCREENING
8.12.2.3   TEST_TESTABILITY_FEATURES
8.12.2.4   TEST_ADDITIONAL_REQUIREMENTS
8.12.2.5   TEST_YIELD_CODE
8.12.2.6   TEST_FLOW
8.12.2.7   TEST_TEMP
8.12.2.8   TEST_SCREEN
8.12.2.9   TEST_SCREEN_COMPLIANCE
```

8.13 AUTRES DONNÉES

8.13.1 Paramètres TEXT

Nom de paramètre	**TEXT_*identificateur***
Type du paramètre	Variable
Fonction du paramètre	Paramètre textuel général, pour permettre à l'utilisateur d'inclure du texte pertinent et identifié non couvert autrement.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Limitations	Chaque identificateur doit être déclaré une seule fois dans un seul bloc DEVICE.
Exemples	`TEXT_SPECIAL_REQUIREMENTS = “N/A”;`

Notes Les identificateurs *TEXT_identifiers* reconnus sont:

```
8.13.1.1   TEXT_PRODUCT_STATUS
8.13.1.2   TEXT_FORM_OF_SUPPLY
8.13.1.3   TEXT_SPECIAL_REQUIREMENTS
8.13.1.4   TEXT_SPECIFIC_REQUIREMENTS
8.13.1.5   TEXT_STORAGE_CONDITIONS
8.13.1.6   TEXT_STORAGE_DURATION
8.13.1.7   TEXT_LONGTERM_STORAGE
8.13.1.8   TEXT_ORIGINAL_MANUFACTURER
8.13.1.9   TEXT_ORIGINAL_DESIGN_DATE
```

8.14 DONNÉES DE CONTRÔLE

8.14.1 Paramètres PARSE

Nom de paramètre	**PARSE_*identificateur***
Type du paramètre	Variable
Fonction du paramètre	Paramètres spécifiques à utiliser comme fanions ou paramètres de contrôle par le logiciel d'analyse uniquement, ces paramètres n'ont pas de signification ou de fonction liée au dispositif.
Valeurs du paramètre	Données de chaîne textuelle, conformément à 7.1.3.1
Notes	Les paramètres PARSE_xxx sont applicables uniquement aux données apparaissant après leur invocation; cela permet de modifier les paramètres PARSE_xxx au cours de l'analyse de bout en bout d'un bloc DEVICE unique. Il est donc fortement conseillé que si les paramètres PARSE_xxx sont utilisés, l'invocation initiale se produise avant toute autre donnée dans le bloc DEVICE. Les analyseurs à plusieurs passes doivent prendre en charge cette caractéristique. Ces paramètres d'analyse peuvent être modifiés ou supplantés par d'autres options dans le logiciel d'analyse, telles que les options de ligne de commande. Se référer à l'Annexe K pour d'autres détails.

Exemples

```
PARSE_MODE = STRICT;
PARSE_ERROR_REPORT = TERSE;
PARSE_ERROR_TRAP = ALL;
PARSE_IGNORE = NONE;
PARSE_DEFINE_PARAMETER = “A_New_Parameter”;
PARSE_DEFINE_STRUCTURE = “A_New_Structure”;
```

8.14.1.1 PARSE_MODE

Le paramètre PARSE_MODE sélectionne la sévérité de la règle et de la vérification de syntaxe au sein d'un bloc DEVICE. Se référer à l'Annexe K pour plus de détails.

Les paramètres PARSE_MODE reconnus sont:

```
STRICT
RELAXED
ENHANCED
USER
```

8.14.1.2 PARSE_ERROR_REPORT

Le paramètre PARSE_ERROR_REPORT détermine comment les avertissements et erreurs découverts au cours de l'analyse du bloc DEVICE sont rapportés. Un compte-rendu cumulatif des avertissements et erreurs est attendu lorsque toute l'analyse a été menée à bien, cette caractéristique n'est pas altérée par ce paramètre. Se référer à l'Annexe K pour d'autres détails.

Les paramètres PARSE_ ERROR_REPORT reconnus sont:

`OFF`	Aucune erreur n'est rapportée au cours de l'analyse.
`TERSE`	Seules les erreurs sont rapportées au cours de l'analyse.
`VERBOSE`	Tous les avertissements et erreurs sont rapportés au cours de l'analyse.

8.14.1.3 PARSE_ERROR_TRAP

Le paramètre PARSE_ERROR_TRAP détermine comment il convient que le logiciel d'analyse procède lorsqu'une erreur a été découverte. Se référer à l'Annexe K pour d'autres détails.

Les paramètres PARSE_ ERROR_TRAP reconnus sont

`ALL`	Il convient qu'un analyseur ne s'arrête pas sur erreur.
`FIRST`	Il convient que l'analyseur s'arrête dès la première erreur.

8.14.1.4 PARSE_IGNORE

Le paramètre PARSE_IGNORE détermine si les données sont vérifiées ou non. Il faut faire preuve d'une extrême prudence en utilisant ce paramètre. Se référer à l'Annexe K pour d'autres détails.

Les paramètres PARSE_IGNORE reconnus sont:

`NONE`	Tous les contrôles d'analyse sont exécutés.
`OFF`	Tous les contrôles d'analyse sont exécutés (comme `NONE`)
`ALL`	Tous les contrôles d'analyse sont ignorés.
`SYNTAX_ONLY`	Seuls les contrôles des erreurs de syntaxe et des fins de ligne sont effectués.

8.14.1.5 PARSE_DEFINE_PARAMETER

Le paramètre PARSE_DEFINE_PARAMETER permet l'introduction et la définition de nouveaux paramètres de données par leur nom avant qu'ils ne soient incorporés dans une nouvelle version de la syntaxe DDX. Se référer à l'Annexe K pour d'autres détails.

Exemple:

```
PARSE_DEFINE_PARAMETER = “My_New_Parameter”;
```

```
PARSE_DEFINE_PARAMETER = “My_Second_Parameter”;

My_New_Parameter = “some data”;
My_Second_Parameter = “some other data”;
```

Si le nom du «nouveau» paramètre est en conflit avec des noms de paramètres ou des mots réservés déjà existants, il convient qu'un avertissement soit signalé et que le «nouveau»paramètre traité tel que défini dans la norme DDX courante, c'est-à-dire qu'il convient que le PARSE_DEFINE_PARAMETER soit signalé comme un avertissement, puis ignoré. Cela permet au logiciel d'analyse d'accepter ces «nouveaux» paramètres une fois qu'ils ont été ratifiés sans réécrire le fichier DDX.

Le «nouveau» nom doit suivre les recommandations générales énoncées dans les Articles 5 à 7 et ne sera valide qu'au sein du bloc DEVICE en question. La syntaxe du «nouveau» paramètre sera alors contrôlée et le paramètre sera analysé comme s'il s'agissait d'un paramètre standard.

Le PARSE_DEFINE_PARAMETER peut apparaître autant de fois que nécessaire, chaque déclaration peut seulement introduire 'un seul paramètre, et doit précéder l'utilisation du paramètre en question.

En attendant d'être ratifiées et incorporées avec une nouvelle version de la norme DDX, toutes les valeurs relatives au «nouveau» paramètre seront traitées comme données de chaîne textuelle, comme en 7.1.3.1.

8.14.1.6 PARSE_DEFINE_STRUCTURE

Le paramètre PARSE_DEFINE_STRUCTURE permet l'introduction de nouvelles structures de données par leur nom avant qu'elles ne soient incorporées dans une nouvelle version de la syntaxe DDX. Se référer à l'Annexe K pour d'autres détails.

Exemple:

```
PARSE_DEFINE_STRUCTURE = “A_New_Structure”;
PARSE_DEFINE_STRUCTURE = “Another_Structure”;

A_New_Structure {
  NEW_STRUCTURE = “some data”, “some more data”;
  NEW_STRUCTURE = “some data”, “some more data”;
}
Another_Structure {
  ANOTHER_NEW_STRUCT = “some data”, “some more data”;
  ANOTHER_NEW_STRUCT = “some data”, “some more data”;
}
```

Si le nom de la «nouvelle» structure est en conflit avec des noms de structures ou des mots réservés déjà existants, il convient qu'un avertissement soit signalé et que la «nouvelle» structure traitée telle que définie dans la norme DDX courante, c'est-à-dire qu'il convient que le PARSE_DEFINE_STRUCTURE soit signalé comme un avertissement, puis ignoré. Cela permet au logiciel d'analyse d'accepter ces «nouvelles» structures une fois qu'elles ont été ratifiées sans réécrire le fichier DDX.

Le «nouveau» nom doit suivre les recommandations générales énoncées dans les Articles 5 à 7 et ne sera valide qu'au sein du bloc DEVICE en question. La syntaxe de la «nouvelle» structure sera alors

contrôlée et la structure sera analysée «comme s'il s'agissait d'une structure standard.

Le PARSE_DEFINE_STRUCTURE peut apparaître autant de fois que nécessaire, chaque déclaration peut seulement introduire une seule structure, et doit précéder l'utilisation de la structure en question.

Le "*Structure_indent_name*" peut être référencé comme n'importe quel autre nom de structure, mais pas dans des paramètres de structure définis existants avant ratification

En attendant d'être ratifiées et incorporées avec une nouvelle version de la norme DDX, toutes les valeurs individuelles contenues dans la «nouvelle» structure seront traitées comme données de chaîne textuelle, comme en 7.1.3.1.

Annexe A
(informative)

Exemple d'un bloc DEVICE DDX

```
DEVICE 7995 bare_die {
#
# Données d'en-tête initiales, avec l'historique du bloc et du
# dispositif.
#
BLOCK_CREATION_DATE = “2000-12-25”;
BLOCK_VERSION = 1.0;
MANUFACTURER = “Fuzziwuzz Logic Ltd.”;
FUNCTION = “Special gate”;
DATA_SOURCE = “GOOD-DIE database”;
DATA_VERSION = “Initial Issue A”;
VERSION = “1.2.2”;

#
# Déclaration de la vue géométrique, unités, taille, etc.,
#
GEOMETRIC_UNITS = millimetre;
GEOMETRIC_VIEW = “top”;
SIZE = 1.312, 1.050;
SIZE_TOLERANCE = 0.00, 0.0005, 0.00 0.0005;
THICKNESS = 0.360;
THICKNESS_TOLERANCE = 0.00, 0.0007;
GEOMETRIC_ORIGIN = 0,0;

#
# Détails complémentaires relatifs au type et à l'utilisation de
# la puce.
#
DI*E_NAME = “XXZ322”;
DIE_MASK_REVISION = “Mask 1.0”;
MAX_TEMP = 280;
POWER_RANGE = 0.500;
DIE_SUBSTRATE_MATERIAL = “Silicon”;
DIE_TERMINAL_MATERIAL = “Al”;
IC_TECHNOLOGY = “bipolar”;
DIE_SUBSTRATE_CONNECTION = “Ground”;

#
# Détails relatifs à la livraison.
#
DIE_BACK_DETAIL  = “Back-Lapped”;
DIE_DELIVERY_FORM = “Die, Wafer”;
WAFER_SIZE = “4 inch”;

#
# Définition du nombre de types de plots de connexion, de plots
# et de connexion et de connexions.
#
TERMINAL_TYPE_COUNT    = 5;
TERMINAL_COUNT         = 8;
CONNECTION_COUNT       = 14;
```

```
#
# Définitions de la forme et des dimensions des plots de
# connexion.
#
TERMINAL_TYPE  {
      PADR1 = Rectangle, 0.144, 0.104;
      PADR2 = Rectangle, 0.264, 0.104;
      PADR3 = Rectangle, 0.084, 0.084;
      PADC1 = Circle, 0.100;
      PADP1 = Polygon, (-0.0175,-0.042),(-0.042,-0.0175),
      (-0.042, 0.0175),(-0.0175, 0.042),
      ( 0.0175, 0.042),( 0.042, 0.0175),
      ( 0.042,-0.0175),( 0.0175,-0.042);
      }

#
# Détails relatifs au placement, dénomination, orientation et
# connectivité des plots de connexion.
#
TERMINAL {
      T1 = 1 ,PADC1,-0.550, 0.416,  0,VCCA   ,P;
      T2 = 3 ,PADP1,-0.502, 0.190,  0,INPUTA ,I;
      T3 = 4 ,PADP1,-0.502,-0.192,  0,INPUTB ,I;
      T4 = 7 ,PADC1,-0.399,-0.442,  0,GNDA   ,G;
      T5 = 8 ,PADR2, 0.498,-0.442,  0,GNDB   ,G;
      T6 = 11,PADR3, 0.511,-0.171,  0,OUTPUTA,O;
      T7 = 12,PADR3, 0.511, 0.171,  0,OUTPUTB,O;
      T8 = 14,PADR1, 0.558, 0.416,  0,VCCB   ,P;
      }

#
# Détails relatifs à un modèle de simulation pSpice fourni.
#
SIMULATOR_SPICE_MODEL_FILE = “SP7995.MOD”;
SIMULATOR_SPICE_MODEL_FILE_DATE = “1997-09-17”;
SIMULATOR_SPICE_NAME = “pSpice”;
SIMULATOR_SPICE_VERSION = “4.0.1”;
SIMULATOR_SPICE_COMPLIANCE = “2G6”;

#
# Détails relatifs à un modèle de simulation Spectre fourni.
#
SIMULATOR_SPECTRE_MODEL_FILE = “SP7995.S”;
SIMULATOR_SPECTRE_MODEL_FILE_DATE = “1998-11-05”;
SIMULATOR_SPECTRE_NAME = “Spectre”;
SIMULATOR_SPECTRE_VERSION = “4.2.1, 1992”;
SIMULATOR_SPECTRE_COMPLIANCE = “2G6, Level-3”;

#
# Détails relatifs au repère conventionnel de référence, fournis
# sous forme de fichier graphique JIF
#
FIDUCIAL_TYPE fiduc1 = “7995FID1.JIF”, 0.072, 0.055;
FIDUCIAL F1 = fiduc1, -0.612, 0.470, 0;

# Fin de bloc
}
```

Annexe B
(informative)

Groupes et permutation

A. Règles (Rule) spécifiques pour le paramètre TERMINAL_GROUP

Règle A1. ***Contenu*** (voir 8.4.6.1.1) – Les éléments formant un groupe peuvent être des identificateurs de borne ou des identificateurs de groupe ou n'importe quel mélange des deux. Un groupe doit contenir au moins deux éléments.

Règle A2. ***Unicité*** (voir 8.4.6.1.2) – Tous les éléments au sein d'un groupe doivent être uniques. Un groupe ne peut contenir de groupes qui se réfèrent directement ou indirectement au même élément.

Règle A3. ***Ordre*** (voir 8.4.6.1.3) – l'ordre des éléments au sein des groupes peut être important. Lorsque deux ou plusieurs groupes doivent être reliés dans une permutation, les éléments au sein de ces groupes doivent correspondre à travers les groupes.

Règle A4. ***Récursivité*** (voir 8.4.6.1.4) – Un groupe ne peut se contenir lui-même ni ne peut-il contenir de groupe qui se réfère directement ou indirectement à lui-même.

B. Règles spécifiques pour le paramètre PERMUTABLE

Règle B1. ***Contenu*** (voir 8.4.7.1.1) – Les éléments formant une permutation peuvent être des identificateurs de borne ou des identificateurs de groupe mais pas un mélange des deux. Une permutation doit contenir au moins deux éléments.

Règle B2. ***Unicité*** (voir 8.4.7.1.2) – Tous les éléments au sein d'une permutation doivent être uniques. Une permutation ne peut contenir de groupes qui se réfèrent directement ou indirectement au même élément. Chaque élément d'une permutation doit contenir le même nombre d'identificateurs de bornes (au sein de groupes de bornes ou non) que les autres.

Règle B3. ***Ordre*** (voir 8.4.7.1.3) – l'ordre des éléments au sein d'une permutation n'est pas important.

Le moyen le plus simple pour décrire l'utilisation et le fonctionnement des paramètres TERMINAL_GROUP et PERMUTABLE consiste à analyser un exemple simple. Ici, il est donné un fragment de fichier DDX pour un dispositif 7400. Le dispositif 7400 est une porte NAND à deux entrées à quatre gâchettes et ainsi pour la fonctionnalité toutes les quatre portes NAND sont interchangeables. Tout au long de l'analyse, référence est faite à des bornes mais pour une pièce encapsulée, l'utilisateur peut considérer le terme «broches» plus approprié.

```
DEVICE 74ACT00 bare_die {

  BLOCK_CREATION_DATE = "13/02/2006";
  BLOCK_VERSION = 1.0;
  DEVICE_NAME = "74ACT00";
  MANUFACTURER = "Fuzziwuzz Logic Ltd";
  FUNCTION = "Quad two-input advanced CMOS NAND gate";
  DEVICE_FORM = "bare_die";
  DATA_SOURCE = "GOOD-DIE Project database";
  VERSION = "1.3.0";
  GEOMETRIC_UNITS = micrometre;
  GEOMETRIC_VIEW = "top";
  SIZE = 1067, 1143;
  THICKNESS = 356;
  GEOMETRIC_ORIGIN = 0,0;
```

```
  DIE_NAME = "74ACT00";
  DIE_MASK_REVISION = "Mask T";
  MAX_TEMP = 150;
  POWER_RANGE = 0.2;
  IC_TECHNOLOGY = "CMOS";
  DIE_SEMICONDUCTOR_MATERIAL = "silicon";
  DIE_SUBSTRATE_CONNECTION = "CONN, Vcc";
  DIE_DELIVERY_FORM = "Die, wafer";
  TERMINAL_TYPE_COUNT = 1;

  TERMINAL_TYPE  {
      PADR1 = Rectangle, 97, 97;
      }

  TERMINAL_COUNT = 14;

  TERMINAL  {
      T_1 = 1, PADR1, -385, 422, 0, A1, I;
      T_2 = 2, PADR1, -385, 176, 0, B1, I;
      T_3 = 3, PADR1, -385, 11, 0, Y1, O;
      T_4 = 4, PADR1, -385, -236, 0, A2, I;
      T_5 = 5, PADR1, -208, -423, 0, B2, I;
      T_6 = 6, PADR1, -43, -423, 0, Y2, O;
      T_7 = 7, PADR1, 123, -423, 0, GND, G;
      T_8 = 8, PADR1, 385, -423, 0, Y3, O;
      T_9 = 9, PADR1, 385, -166, 0, B3, I;
      T_10 = 10, PADR1, 385, -1, 0, A3, I;
      T_11 = 11, PADR1, 385, 164, 0, Y4, O;
      T_12 = 12, PADR1, 385, 423, 0, B4, I;
      T_13 = 13, PADR1, 38, 423, 0, A4, I;
      T_14 = 14, PADR1, -129, 423, 0, VCC, P;
      }

TERMINAL_GROUP  {
      NAND_INA = T_1, T_2;
      NAND_INB = T_4, T_5;
      NAND_INC = T_9, T_10;
      NAND_IND = T_12, T_13;
      NAND_A = NAND_INA, T_3;
      NAND_B = NAND_INB, T_6;
      NAND_C = NAND_INC, T_8;
      NAND_D = NAND_IND, T_11;
      }

PERMUTABLE {
      P_1 = T_1, T_2;
      P_2 = T_4, T_5;
      P_3 = T_9, T_10;
      P_4 = T_12, T_13;
      P_5 = NAND_A, NAND_B, NAND_C, NAND_D;
      }
}
```

Le dispositif 7400 comporte quatre portes identiques, Portes A à D, avec les caractéristiques suivantes:

Dispositif 7400	INPUT1 (c'est-à-dire Entrée 1)	INPUT2 (c'est-à-dire Entrée 2)	OUTPUT (c'est-à-dire Sortie)
Porte A	T_1	T_2	T_3
Porte B	T_4	T_5	T_6
Porte C	T_9	T_10	T_8
Porte D	T_12	T_13	T_11

Avec le fragment DDX ci-dessus, il apparaît à l'évidence que `TERMINAL_GROUP NAND_INA` regroupe les entrées pour la Porte A ensemble, `NAND_INB` pour la Porte B et ainsi de suite.

`TERMINAL_GROUP NAND_A` regroupe de même les entrées pour la Porte A avec la borne de sortie correspondante, et de nouveau `NAND_B` à `NAND_D` suivent les Portes B à D d'une manière similaire. Noter que les règles relatives au mélange de groupes de bornes et de bornes dans la même déclaration (conformément à la **Règle A1**) et à l'ordre de borne (**Règle A3**) des bornes d'entrée et ensuite la borne de sortie sont respectées.

Enfin, la déclaration de `PERMUTABLE` peut être analysée ainsi:

- `P_1` énonce que les bornes `T_1` et `T_2` (Porte A) sont interchangeables.
- `P_2` énonce que les bornes `T_4` et `T_5` (Porte B) sont interchangeables.
- `P_3` énonce que les bornes `T_9` et `T_10` (Porte C) sont interchangeables.
- `P_4` énonce que les bornes `T_12` et `T_13` (Porte D) sont interchangeables.
- `P_5` énonce que les groupes `NAND_A` à `NAND_D` (Portes A à D) sont interchangeables, pour autant que l'ordre séquentiel des bornes soit respecté.

Noter que dans l'exemple ci-dessus, il est utilisé "`P_1 = T_1, T_2;`" et pas "`P_1 = NAND_INA;`", car la seconde déclaration violerait la **Règle B1** en ne comportant pas au moins deux éléments.

Comme l'énonce la **Règle B3**, l'ordre des éléments n'a pas d'importance et, de ce fait, la déclaration

```
P_5 = NAND_A, NAND_B, NAND_C, NAND_D;
```

doit avoir la même interprétation que

```
P_5 = NAND_D, NAND_C, NAND_B, NAND_A;
```

Comme l'énonce la **Règle B2**, ce qui suit **NE SERAIT PAS** acceptable

```
P_7 = NAND_D, T_1, T_2;        (éléments inégaux, viole aussi la Règle B1)
P_8 = NAND_INA, NAND_B;        (éléments inégaux)
P_9 = NAND_INA, NAND_A;        (éléments inégaux et possible récursivité)
P_10 = NAND_A, NAND_A;         (possible récursivité)
```

Annexe C
(informative)

Vue CAO type à partir de l'exemple de bloc de fichier DDX donné dans l'Annexe A

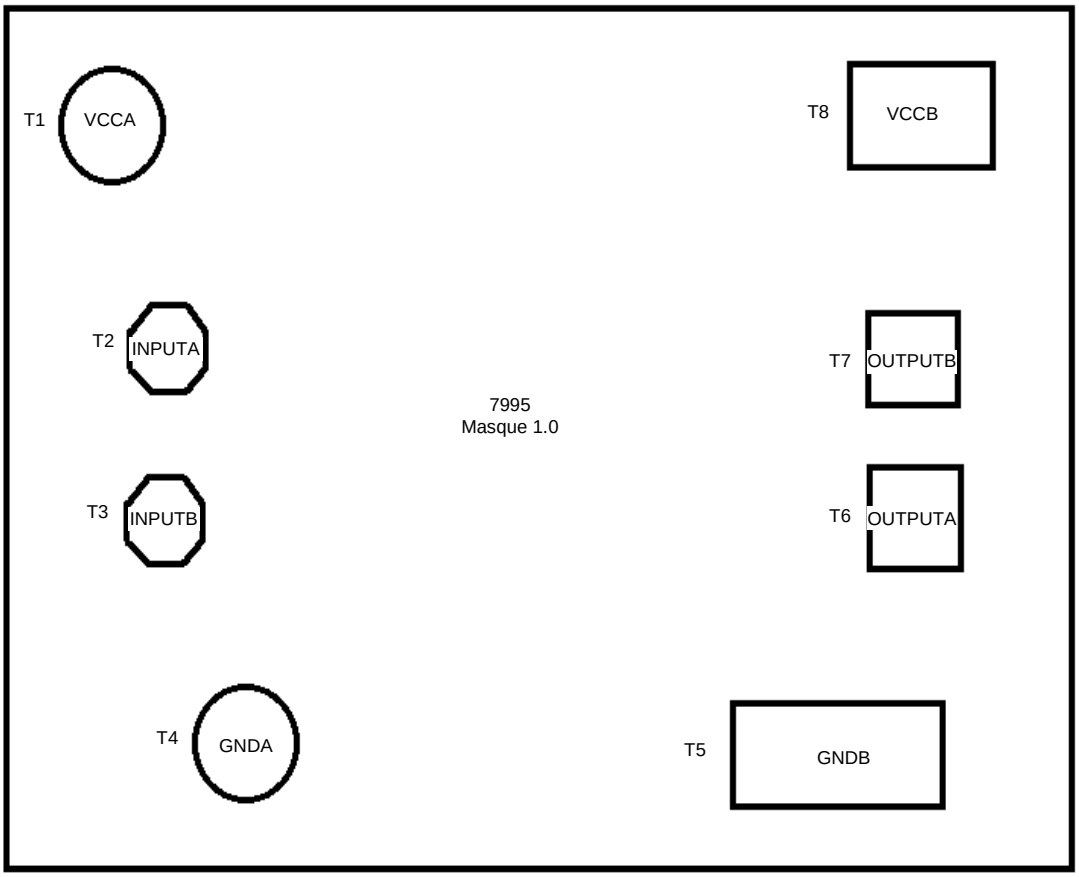

Figure C.1 – Représentation CAO de l'exemple de DDX issu de l'Annexe A

Noter que la Figure C.1. n'est pas censée être un dessin exact du point de vue géométrique, le but étant seulement d'aider à visualiser la sortie à partir de l'exemple de DDX donné dans l'Annexe A. (Le graphique Repère conventionnel de puce n'est pas montré.) Le système CAO a choisi d'afficher les paramètres **DEVICE_NAME** (voir 8.1.1) et **DIE_MASK_REVISION** (voir 8.2.3) pour la puce. Les identificateurs des bornes individuelles et les noms de borne ont été choisis d'une manière similaire pour l'affichage (voir 8.4.5 pour les détails).

Annexe D
(informative)

Propriétés pour la Simulation

Un fichier DDX peut référencer d'autres fichiers modèles de simulation électrique et des fichiers de modélisation thermique.

Pour ce faire d'une manière adéquate, les données suivantes doivent être présentes:

– le nom du fichier et sa date de création,
– le nom exact du simulateur,
– sa version,
– le niveau pertinent de conformité auquel ce modèle s'applique,
– la borne à laquelle le modèle s'applique (électrique).

Il convient qu'il existe dans le fichier modèle une note textuelle relative à la capacité du modèle, à son applicabilité et à son exactitude, indiquant en particulier tout inconvénient du modèle et de son utilisation. Le texte "***simulator***" doit être remplacé par le nom réel ou générique du simulateur utilisé et/ou des données de modèle. Les noms types de "***simulator***" pourraient être Verilog, VHDL, SPICE, ELDO, BSDL, IBIS, etc.

8.7.1 **SIMULATOR_*simulator*_MODEL_FILE**
Données présentées comme nom de fichier. Tous les noms de fichiers donnés ne doivent comporter ni lettre de lecteur d'ordinateur ni référence de chemin absolu ni détails quelconques pouvant rendre les informations spécifiques ou dépendantes d'un ordinateur ou d'un système d'exploitation.

8.7.2 **SIMULATOR_*simulator*_MODEL_FILE_DATE**
Données présentées sous forme de date, conformément à l'ISO 8601.

8.7.3 **SIMULATOR_*simulator*_NAME**
Données présentées sous forme de texte d'un simulateur réel utilisé pour éprouver le modèle.

8.7.4 **SIMULATOR_*simulator*_VERSION**
Données présentées sous forme de texte déterminant la version ou la révision du simulateur par rapport auquel le fichier de modèle a été éprouvé.

8.7.5 **SIMULATOR_*simulator*_COMPLIANCE**
Données présentées sous forme de texte qui reflète la conformité globale du modèle de simulateur, le cas échéant. "2G6", "VHDL 93", "IEEE 1364-2001" ou "IEEE 1076.3-97" sont des exemples de niveaux de conformité reconnus dans l'industrie.

8.7.6 **SIMULATOR_*simulator*_TERM_GROUP**
Détermine les bornes ou groupes de bornes au(x)quel(le)s le simulateur s'applique lorsque le fichier modèle couvre seulement une gamme limitée de bornes. Autrement, il est supposé que le fichier modèle couvre l'ensemble du dispositif. Ce paramètre permet donc la limitation du modèle de simulation à des bornes ou groupes de bornes spécifiques. Son absence implique que le modèle de simulation est applicable à toutes les bornes du dispositif.

Exemple:

```
SIMULATOR_SPICE_MODEL_FILE = “RTBAE2.MOD”;
SIMULATOR_SPICE_MODEL_FILE_DATE = “1997-09-17”;
SIMULATOR_SPICE_NAME = “pSpice”;
SIMULATOR_SPICE_VERSION = “4.0.1”;
SIMULATOR_SPICE_COMPLIANCE = “2G6”;
TERMINAL_GROUP G_1 = P8, P9, P10;
TERMINAL_GROUP G_2 = P28, P29, P30;
SIMULATOR_SPICE_TERM_GROUP = T_1, T_2, G_1, G_2;
```

Annexe E
(informative)

Utilisation graphique de TERMINAL et TERMINAL_TYPE pour les systèmes CAO/FAO

La création d'une «vue» CAO/IAO du dispositif à partir des données contenues dans le bloc **DEVICE** peut être considérée comporter quatre étapes distinctes. Il n'y a aucune information de couche géométrique dans le format DDX et il est supposé que toutes les données géométriques doivent être affichées sur une seule couche. Des informations textuelles peuvent être fournies sur une couche séparée.

Le paramètre **GEOMETRIC_UNITS** (voir 8.3.1) définit les facteurs d'échelle appropriés à utiliser et plante le décor pour toutes les entrées numériques ultérieures provenant de ce bloc **DEVICE**. Le paramètre **GEOMETRIC_VIEW** (voir 8.3.2) détermine si le dispositif est vu de dessus ou de dessous.

Le contour du dispositif est créé à partir du paramètre **SIZE** (voir 8.3.4), et tout décalage géométrique pour le placement géométrique ultérieur est calculé à partir du paramètre **GEOMETRIC_ORIGIN** (voir 8.3.3). Les données telles que le nom et le type du dispositif sont recueillies et préparées à des fins d'affichage. Noter que le paramètre **GEOMETRIC_ORIGIN** est référencé aux paramètres XSIZE et YSIZE, avant tolérancement, comme étant XSIZE/2, YSIZE/2. Les valeurs du paramètre **GEOMETRIC_ORIGIN** sont ajoutées à cette référence pour tous les paramètres géométriques relatifs. Les valeurs du paramètre **SIZE_TOLERANCE** (voir 8.3.5) peuvent aussi être utilisées pour indiquer le contour maximal et minimal du dispositif.

Les paramètres **TERMINAL_COUNT**, **TERMINAL_TYPE_COUNT** et **CONNECTION_COUNT** (voir 8.4.1, 8.4.2 et 8.4.3) sont essentiellement dupliqués car leurs valeurs peuvent être calculées à partir des structures **TERMINAL** (voir 8.4.5) et **TERMINAL_TYPE** (voir 8.4.4). Ils sont cependant destinés à être utilisés comme contrôle d'intégrité pour assurer une analyse correcte et souligner toute corruption de fichier ou de bloc. La valeur **TERMINAL_TYPE_COUNT** peut seulement être inférieure ou égale à la valeur **TERMINAL_TYPE**. La valeur **CONNECTION_COUNT** n'est soumise à aucune restriction de ce type.

Les formes des bornes ou des zones de connexion sont définies séparément, chaque forme ayant son propre centre de référence géométrique. Ces formes sont définies avec la structure **TERMINAL_TYPE**. Chacune de ces formes de borne peut être placée n'importe où dans le contour du dispositif et, comme il n'y a aucune limitation d'emplacement, peut être placée à l'extérieur du contour du dispositif si l'utilisateur le souhaite (par exemple placement de plages de contact pour les connexions du type carte de circuit imprimé PCB). La structure **TERMINAL_TYPE** introduit des **Terminal_Type_Names** (voir 8.4.4.1 et 8.4.5.3) qui doivent être uniques au sein du bloc **DEVICE;** ces noms sont des paramètres fictifs pour un usage ultérieur par la structure **TERMINAL** et donc il convient de les introduire avant la déclaration de la structure **TERMINAL** en utilisant le nom du paramètre fictif pour permettre une analyse en une seule passe du bloc **DEVICE**. Afin d'éviter toute ambiguïté éventuelle, il est fortement conseillé de limiter les **Terminal_Type_Names** à des caractères alphanumériques

Ces types (formes) de bornes, ayant été définis, peuvent alors être référencés par leur **Terminal_Type_Name**, et placés géométriquement par la structure **TERMINAL** en séquence (***T_n***), à des emplacements donnés (les coordonnées X et Y), orientés par miroitage (Figure E.1) et rotation dans le sens horaire (Figure E.2), et recevoir un nom de borne et une fonction E/S pour une utilisation ultérieure par le système de CAO. Le **Terminal_name** (voir 8.4.5.7) serait habituellement visible sous forme de texte pour la commodité visuelle et peut ne pas

être unique. Le **Terminal_name** peut aussi indiquer des bornes qui sont habituellement connectées électriquement ensemble, telles que "*VDD*", "*VSS*", "*Ground*", etc. Un numéro de connexion (***conn_N***) peut aussi être attribué et bien que ce numéro n'ait aucune relation avec le numéro de borne réel, il peut être utilisé pour identifier des groupes communs de broches, (telles que les broches qui doivent être connectées ensemble) ou pour se référer effectivement à un brochage encapsulé du dispositif d'origine (qui, une fois encore, peut être différent par rapport à la numérotation des pastilles de connexion,). Les détails fonctionnels d'E/S (***IO_type***) (voir 8.4.5.8) pour chaque borne sont disponibles pour d'autres contrôles de connectivité par le système de CAO. L'identificateur unique **TERMINAL**, (***T_n***) (voir 8.4.5.1)**,** peut comporter n'importe quel mélange de caractères alphanumériques et peut ne pas se présenter dans une séquence particulière quelconque ni en suivre une. La seule condition est qu'en tant qu'identificateur, il soit unique au sein du bloc **DEVICE** en question et, pour éviter toute ambiguïté éventuelle, il est fortement conseillé de limiter l'identificateur unique (***T_n***) à des caractères alphanumériques.

Si applicable, il convient que les détails de **DIE_SUBSTRATE_CONNECTION** (voir 8.5.5) s'affichent, alertant l'utilisateur sur la nécessité de faciliter la connexion appropriée.

La création et le placement des repères conventionnels sont identiques à ceux du cas des bornes en utilisant les structures (q.v.). de paramètres de **FIDUCIAL_TYPE** (voir 8.3.8) et **FIDUCIAL** (voir 8.3.9). Les différences principales sont que

- les données conventionnelles graphiques sont seulement contenues dans un fichier graphique séparé, et
- que les repères conventionnels n'ont pas de connectivité ou de propriétés électriques.

Il incombe au système de CAO/FAO de déterminer la façon d'afficher le contenu graphique du fichier graphique conventionnel, car il existe actuellement une pléthore de normes relatives aux fichiers graphiques. Le graphique conventionnel doit être mis à l'échelle pour s'ajuster dans les dimensions d taille données par la structure de paramètre **FIDUCIAL_TYPE** dont le centre de référence et de placement sera pris comme le point milieu des dimensions de taille X et Y. Le contenu et le type réels des données graphiques dans ce fichier sont habituellement déterminés par l'extension du fichier, à savoir TIFF, GIF, JPEG, BMP.

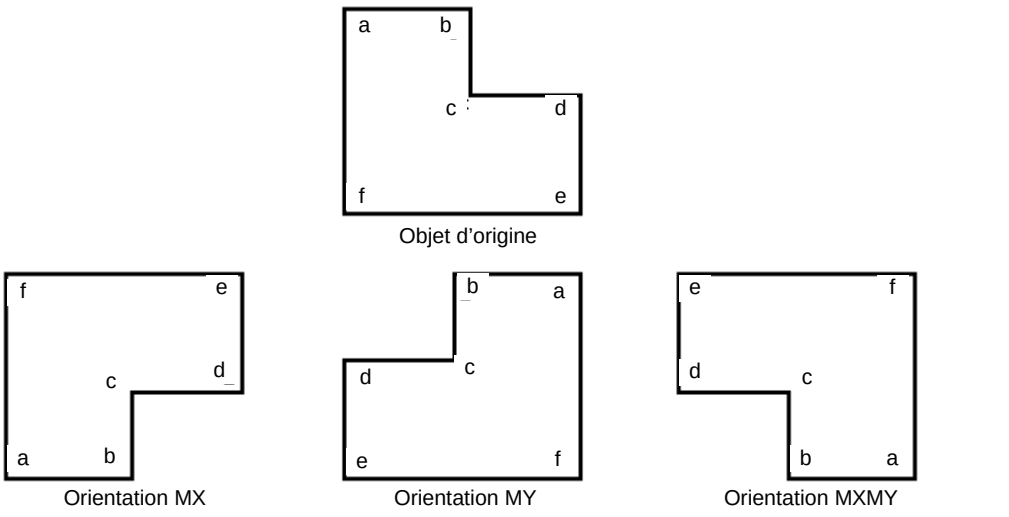

Figure E.1 – Mise en évidence des propriétés d'orientation MX et MY

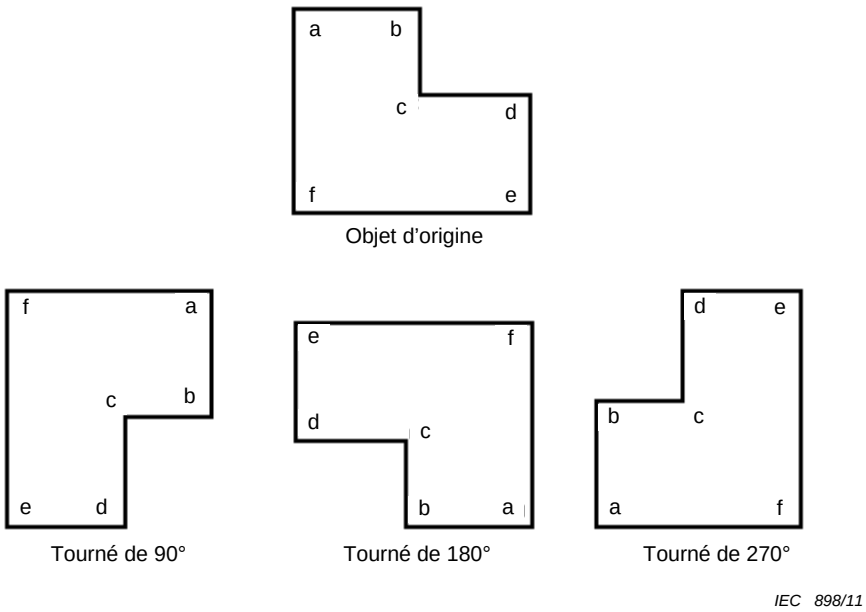

Figure E.2 – Mise en évidence des propriétés d'orientation de rotation angulaire

Annexe F
(informative)

Correspondance avec la CEI 61360-4

Le tableau ci-dessous fournit une analogie entre les paramètres contenus dans la présente norme et les définitions DET données dans la CEI 61360-4:2005.

Tableau F.1 – Liste de paramètres

Article/ Paragra-phe	Nom du paramètre	Type du paramètre	Type de données	Code DET
8.1	***DONNÉES DE BLOC***			
8.1.1	DEVICE_NAME	Header (c'est-à-dire En-tête)	Texte	AAH547-001
8.1.2	DEVICE_FORM	Header (c'est-à-dire En-tête)	Texte	AAD004-001
8.1.3	BLOCK_VERSION	Variable	Texte	
8.1.4	BLOCK_CREATION_DATE	Variable	Date	
8.1.5	VERSION	Variable	Texte	
8.2	***DEVICE_DATA***			
8.2.1	DIE_NAME	Variable	Texte	AAD002-001
8.2.2	DIE_PACKAGED_PART_NAME	Variable	Texte	AAD143-001
8.2.3	DIE_MASK_REVISION	Variable	Texte	AAD003-001
8.2.4	MANUFACTURER	Variable	Texte	AAD140-001
8.2.5	DATA_SOURCE	Variable	Texte	AAD142-001
8.2.6	DATA_VERSION	Variable	Texte	
8.2.7	FUNCTION	Variable	Texte	
8.2.8	IC_TECHNOLOGY	Variable	Texte	AAE686-005
8.2.8	DEVICE_PICTURE_FILE	Variable	File Name (nom de fichier)	
8.2.9	DEVICE_DATA_FILE	Variable	File Name (nom de fichier)	
8.3	***DONNÉES GÉOMÉTRIQUES***			
8.3.1	GEOMETRIC_UNITS	Variable	Real	AAD115-001
8.3.2	GEOMETRIC_VIEW	Variable	Texte	AAD144-001
8.3.3	GEOMETRIC_ORIGIN	Variable	Real	AAD129 & 130-001
8.3.4	SIZE	Variable	Real	AAD070 & 071-001
8.3.5	SIZE_TOLERANCE	Variable	Real	AAD117-001
8.3.6	THICKNESS	Variable	Real	AAD072-001
8.3.7	THICKNESS_TOLERANCE	Variable	Real	AAD118-001
8.3.8	FIDUCIAL_TYPE	Structure		AAD156 à 159-001
8.3.9	FIDUCIAL	Structure		AAD160 à 162-001
8.4	***DONNÉES RELATIVES AUX BORNES***			
8.4.1	TERMINAL_COUNT	Variable	Integer	AAD145-001

Article/ Paragra-phe	Nom du paramètre	Type du paramètre	Type de données	Code DET
8.4.2	TERMINAL_TYPE_COUNT	Variable	Integer	AAD116-001
8.4.3	CONNECTION_COUNT	Variable	Integer	
8.4.4	TERMINAL_TYPE	Structure		AAD024 à 030, AAD121-001
8.4.5	TERMINAL	Structure		AAD014 à 016, AAD019, AAD020-001
8.4.6	TERMINAL_GROUP	Structure		
8.4.7	PERMUTABLE	Structure		
8.5	***DONNÉES RELATIVES AUX MATÉRIAUX***			
8.5.1	TERMINAL_MATERIAL (anciennement DIE_TERMINAL_MATERIAL)	Variable	Texte	AAD120-001, AAE634-005
8.5.2	TERMINAL_MATERIAL_STRUCTURE	Variable	Texte	
8.5.3	DIE_SEMICONDUCTOR_MATERIAL	Variable	Texte	AAD148-001
8.5.4	DIE_SUBSTRATE_MATERIAL	Variable	Texte	AAD005-001
8.5.5	DIE_SUBSTRATE_CONNECTION	Variable	Texte	AAD006 & 007-001
8.5.6	DIE_PASSIVATION_MATERIAL	Variable	Texte	AAD078-001
8.5.7	DIE_BACK_DETAIL	Variable	Texte	AAD119-001
8.6	***DONNÉES ÉLECTRIQUES et THERMIQUES***	Variable		
8.6.1	MAX_TEMP	Variable	Real	AAD149-001
8.6.2	MAX_TEMP_TIME	Variable	Real	
8.6.3	POWER_RANGE	Variable	Real	AAD151-001
8.6.4	TEMPERATURE_RANGE	Variable	Real	AAE891-005
8.7	***SIMULATOR DATA***			
8.7.1	SIMULATOR_*simulator*_MODEL_FILE	Variable	File Name (nom de fichier)	
8.7.2	SIMULATOR_*simulator*_MODEL_FILE_DATE	Variable	Texte	
8.7.3	SIMULATOR_*simulator*_NAME	Variable	Texte	
8.7.4	SIMULATOR_*simulator*_VERSION	Variable	Texte	
8.7.5	SIMULATOR_*simulator*_COMPLIANCE	Variable	Texte	
8.7.6	SIMULATOR_*simulator*_TERM_GROUP	Variable		
8.8	***MANUTENTION,CONDITIONNEMENT, STOCKAGE et ASSEMBLAGE***			
8.8.1	DELIVERY_FORM (anciennement DIE_DELIVERY_FORM)	Variable	Texte	AAD155-001
8.8.2	PACKING_CODE	Variable	Texte	AAD055-001
8.8.8	Paramètres ASSEMBLY	Variable	Texte	
8.9	***DONNÉES SPÉCIFIQUES AUX TRANCHES***			
8.9.1	WAFER_SIZE	Variable	Texte	AAD011-001
8.9.2	WAFER_THICKNESS	Variable	Real	
8.9.3	WAFER_THICKNESS_TOLERANCE	Variable	Real	
8.9.4	WAFER_DIE_STEP_SIZE	Variable	Real	AAD163-001, AAD164-001
8.9.5	WAFER_GROSS_DIE_COUNT	Variable	Integer	AAD165-001

Article/ Paragra-phe	Nom du paramètre	Type du paramètre	Type de données	Code DET
8.9.6	WAFER_INDEX	Variable	Real	AAD166-001, AAD167-001
8.9.7	WAFER_RETICULE_STEP_SIZE	Variable	Real	AAD168-001, AAD169-001
8.9.8	WAFER_RETICULE_GROSS_DIE_COUNT	Variable	Integer	AAD170-001
8.9.9	Paramètres WAFER_INK	Variable	Texte	
8.10	***DONNÉES SPÉCIFIQUES AUX TERMINAISONS À BOSSES***			
8.10.1	BUMP_MATERIAL	Variable	Texte	AAD124-001
8.10.2	BUMP_HEIGHT	Variable	Real	AAD146-001
8.10.3	BUMP_HEIGHT_TOLERANCE	Variable	Real	AAD147-001
8.10.4	BUMP_SHAPE	Variable	Texte	
8.10.5	BUMP_SIZE	Variable	Real	
8.10.6	BUMP_SPECIFICATION_DRAWING	Variable	File Name (nom de fichier)	
8.10.7	BUMP_ATTACHMENT_METHOD	Variable	Texte	
8.11	***DONNÉES SPÉCIFIQUES AUX DISPOSITIFS À ENCAPSULATION MINIMALE***			
8.11.1	MPD_PACKAGE_MATERIAL	Variable	Texte	AAD150-001
8.11.2	MPD_PACKAGE_STYLE	Variable	Texte	
8.11.3	MPD_CONNECTION_TYPE	Variable	Texte	
8.11.4	MPD_MSL_LEVEL	Variable	Texte	
8.11.5	MPD_PACKAGE_DRAWING	Variable	File Name (nom de fichier)	
Supprimé	MPD_DELIVERY_FORM (se référer à 8.8.1)	Variable		
Supprimé	MPD_CONNECTION_MATERIAL (se référer à 8.5.1)	Variable		
8.12	***DONNÉES DE QUALITÉ, FIABILITÉ et ESSAIS***			
8.12.1	Paramètres QUALITY	Variable	Texte	
8.12.1	Paramètres TEST	Variable	Texte	
8.13	***AUTRES DONNÉES***			
8.13.1	TEXT	Variable	Texte	

Annexe G
(informative)

Notes sur les paramètres VERSION et NAME

Il y a trois paramètres relatifs à la “VERSION”, chacun contenant des données de révision spécifiques concernées par les différents aspects des données présentées au sein d'un BLOCK DDX.

VERSION Paragraphe 8.1.5
Il se rapporte à la CEI 62258, Partie 2, et à sa parution. S'agissant d'une norme de transfert de données à utiliser avec les systèmes de CAO/IAO, il est envisagé qu'il y aura des mises à jour et des révisions, en particulier l'inclusion de données complémentaires, à intervalles réguliers. Se référer à l’Article 1 pour déterminer la version DDX couverte dans la présente norme.

BLOCK_VERSION Paragraphe 8.1.3
Ce paramètre se rapporte uniquement à l'état de révision des données avec le BLOCK et serait censé être ” révisé en accéléré" à mesure que les données au sein de BLOCK sont modifiées. Le paramètre **BLOCK_VERSION** N'EST PAS concerné spécifiquement par la source des modifications éventuelles des données.

DATA_VERSION Paragraphe 8.2.6
Ce paramètre couvre la source des données utilisées; pour indiquer l'état de l'art technique pour le contenu des données. Ce paramètre est donc étroitement couplé au paramètre **DATA_SOURCE**.

BLOCK_CREATION_DATE Paragraphe 8.1.4
Ce paramètre se rapporte uniquement à la dernière date à laquelle les données dans ce bloc ont été révisées et accompagne toujours le paramètre **BLOCK_VERSION**, auquel il convient de le lier.
De même, il y a quatre paramètres relatifs à “NAME”, chacun de ceux-ci représentant un aspect différent du nom du dispositif.

DEVICE_NAME Paragraphe 8.1.1
Il est souvent appelé numéro de type et il est l'identité utilisée normalement dans les fiches techniques et dans les listes de pièces de rechange.

DIE_NAME Paragraphe 8.2.1
IL s'agit du nom donné à la puce elle-même et au jeu de masques utilisé dans sa fabrication. Ce nom fait souvent partie de repères conventionnels sur la surface de la puce.

DIE_MASK_REVISION Paragraphe 8.2.3
Code de révision ou de version du jeu de masques utilisé dans la fabrication de la puce.

DIE_PACKAGED_PART_NAME Paragraphe 8.2.2
Généralement similaire à DEVICE_NAME, il est cité comme étant le nom d'une pièce emballée qui contient habituellement la même puce et a des caractéristiques électriques similaires.

Annexe H
(informative)

Notes sur les paramètres WAFER

Lorsque les puces sont livrées sous forme de tranches non débitées, des paramètres supplémentaires peuvent aider à la modalité pratique de la manutention mécanique et à l'achat en grande quantité. Ces paramètres incluent la taille de pas de la puce, la taille des tranches, l'angle de méplat de la tranche et la quantité brute de puces par tranche. Lorsque la taille du pas de puce est irrégulière, souvent en raison du pas de réticule, des paramètres supplémentaires relatifs à la taille de réticule peuvent devenir importants.

Les paramètres relatifs à la livraison des tranches sont comprennent :

WAFER_SIZE Paragraphe 8.9.1
Ce paramètre donne le diamètre approché de la tranche. Il est habituellement exprimé en millimètres ou en pouces et, de ce fait, peut ne pas être relié au paramètre **GEOMETRIC_UNITS**.

WAFER_DIE_STEP_SIZE Paragraphe 8.9.4
Les coordonnées X et Y spécifiées par ce paramètre sont communément utilisées pour le sciage de puces et la vérification des tranches, etc. Les unités sont de nouveau déterminées par le paramètre **GEOMETRIC_UNITS**.

WAFER_GROSS_DIE_COUNT Paragraphe 8.9.5
Le nombre de puces brutes peut être un paramètre important dans le calcul du nombre net de puces par tranche mais ne doit pas refléter le nombre de bonnes puces par tranche, seulement le nombre total de puces viables pertinentes (puces entières). Alors qu'il existe théoriquement une relation mathématique entre taille de tranche et taille de puce, les parasites d'essai, etc., altéreront le décompte final. Il convient que l'utilisateur soit au fait des techniques de réduction de coûts qui impliquent la réduction du nombre de masques requis qui conduit souvent à un nombre plus faible de puces viables.

WAFER_INDEX Paragraphe 8.9.6
Cette paire de paramètres définit le type de caractéristique physique présent sur une tranche et son orientation, laquelle paire peut ensuite être utilisée dans un système de localisation automatique afin de déterminer le placement et l'orientation de la tranche.
Le premier paramètre spécifie le type de caractéristique, soit un “flat” (méplat), soit un “notch” (entaille), alors que le second paramètre détermine l'angle approché, au degré près, de cette caractéristique d'index sur la tranche par rapport à l'axe X sur la surface de la puce. Cela peut aider dans un système de reconnaissance de localisation de tranche pour une orientation initiale ou grossière avant un alignement final par des moyens optiques. Lorsque la tranche comporte plus d'un méplat, l'orientation est réalisée par rapport au méplat majeur, ou principal, qui peut aussi être utilisé pour indiquer la direction cristallographique [110] du réseau de la tranche.

WAFER_RETICULE_STEP_SIZE Paragraphe 8.9.7
Lorsque les dimensions de réticule interfèrent avec l'espacement régulier en pas des puces, il convient que la taille du pas de réticule soit exprimée dans les mêmes dimensions que la taille du pas de puce (**GEOMETRIC_UNITS**).

WAFER_RETICULE_GROSS_DIE_COUNT Paragraphe 8.9.8
Lorsque le paramètre **WAFER_RETICULE_STEP_SIZE** est pertinent, il peut être aussi pertinent d'énoncer le nombre de puces ayant le type de puce spécifié dans la fenêtre de réticule. Dans le cas des MPW, cela peut seulement être une puce.

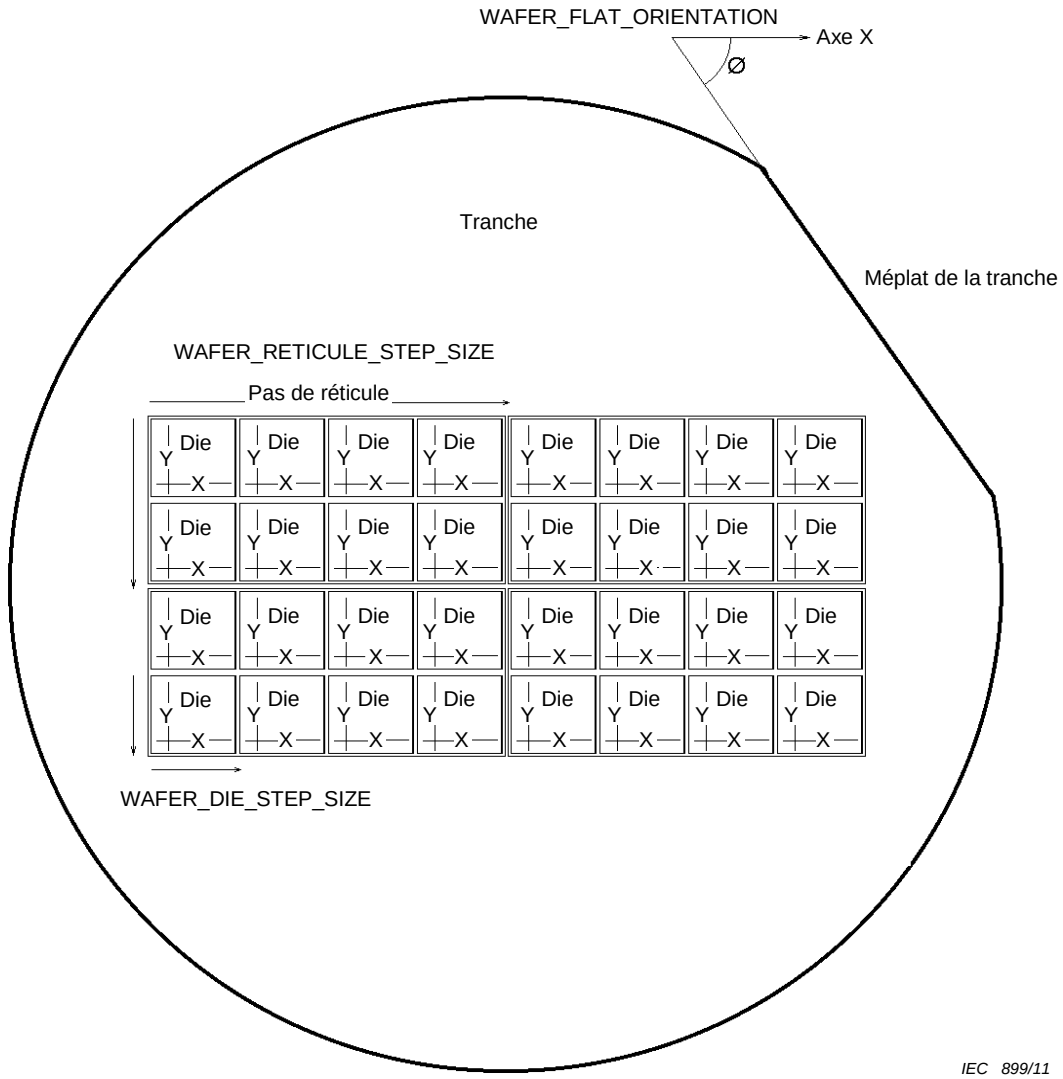

Figure H.1 – Illustration des paramètres WAFER

La Figure H.1 montre une représentation du cas où la taille du pas de réticule interfère avec les dimensions du pas de puce (le réticule contient ici un total de huit puces en une matrice 4 par 2). L'écart entre réticules conduit a une distance supplémentaire entre des puces adjacentes aux frontières du réticule. La Figure H.1 montre l'angle de méplat de la tranche par rapport à la dimension X supposée de la puce. Des valeurs de paramètres fictives pour la Figure H.1 pourraient être:

```
WAFER_SIZE = “150mm”;
WAFER_DIE_STEP_SIZE = 1000,1000;
WAFER_GROSS_DIE_COUNT = 2077;
WAFER_INDEX = “Flat”,45;
WAFER_RETICULE_STEP_SIZE = 4050, 2100;
WAFER_RETICULE_GROSS_DIE_COUNT = 8;
```

Les paramètres de réticule donnés dans cet exemple font apparaître ici une divergence de 50 unités (**GEOMETRIC_UNITS**) dans la dimension X et une divergence de 100 unités (**GEOMETRIC_UNITS**) dans la dimension Y.

Annexe I
(informative)

Notes complémentaires

1) Tel qu'il est, le format DDX exige que les données résident au sein d'une structure de bloc, strictement délimitée par des accolades. Les seules données exigées à l'extérieur de ces accolades sont le mot-clé **DEVICE** et les paramètres associés **DEVICE_NAME** et **DEVICE_FORM**. Cela laisse alors du champ pour une future expansion, telle que l'inclusion de données au format autre que DDX à l'extérieur de la structure de bloc DDX, à condition de réserve de respecter les règles relatives à la structure et au format DDX. De telles données complémentaires peuvent inclure des sommes de contrôle des fichiers et des blocs pour valider le contenu des fichiers de données, bien qu'il soit nécessaire pour les calculs complets des sommes de contrôle pour les caractères ASCII de prendre en compte les délimiteurs de ligne différents utilisés par différents systèmes d'exploitation logicielle. Il convient que ces données complémentaires soient recherchées et ignorées par toute routine d'analyse spécifiquement pour lire des données au format DDX.

2) Sachant qu'il existe un grand nombre de différentes conventions relatives aux caractères servant à l'affichage ou à l'annotation d'un signal invalidé, telles que “\\”, “/”, “N”, “B” ou “not/bar”, il n'a été adopté aucune méthode particulière dans le format DDX pour l'affichage de tels noms de signaux. Il appartient donc au fournisseur d'informations d'adopter une convention unique et au vendeur IAO de prendre en charge cette convention dans l'affichage graphique, le cas échéant.

3) Référence à un fichier externe. Les références à des noms de fichier sont exigées par:

8.3.8.2	**FIDUCIAL_TYPE**
8.2.9	**DEVICE_PICTURE_FILE**
8.2.10	**DEVICE_DATA_FILE**
8.11.5	**MPD_PACKAGE_DRAWING**
8.10.6	**BUMP_SPECIFICATION_DRAWING**

Pour référencer un fichier externe, tel qu'un fichier graphique, texte ou document, il est préférable que le type de fichier externe se conforme à un format normalisé d'usage commun. La convention veut que le format et/ou type de document soit indiqué par l'extension du fichier nommé

4) Noms multiples de fichiers. Les noms multiples de fichiers sont permis dans:

8.2.9	**DEVICE_PICTURE_FILE**
8.2.10	**DEVICE_DATA_FILE**
8.11.5	**MPD_PACKAGE_DRAWING**
8.10.6	**BUMP_SPECIFICATION_DRAWING**

Lorsque des noms multiples de fichiers sont permis, les noms de fichier peuvent être introduits dans une invocation unique de fonction, comme:

```
DEVICE_DATA_FILE = “file1.txt”, “file2.txt”, … , “fileX.txt”;
```

ou sous forme d'invocations multiples de la fonction en question, comme:

```
      DEVICE_DATA_FILE = “file1.txt”;
DEVICE_DATA_FILE = “file2.txt”;
.
DEVICE_DATA_FILE = “fileN.txt”;
```

Lorsqu'on utilise une invocation unique de fonction pour introduire plus d'un nom de fichier, il convient que chaque nom de fichier soit entouré de guillemets (conformément à 6.3.7) et chaque nom doit être séparé par une virgule (conformément à 6.3.2). Chaque ligne doit se terminer par un point-virgule conformément à 6.3.1.

Annexe J
(informative)

Historique des versions DDX

Le tableau suivant, Tableau J.1, indique l'historique des versions des paramètres. La colonne parution reflète la version courante du DDX et le numéro de version auquel le paramètre en question a été soit introduit, soit modifié la dernière fois.

La version courante pour ce format DDX est 1.3.0, comme indiqué à l'Article 1 de la présente norme.

La version précédente de ce format DDX était 1.0, comme défini dans l'ES59008, Partie 6-1.

Tableau J.1 – Liste de l'historique des modifications des paramètres

Article/ Paragra-phe	Nom du paramètre	Référence ES59008-6-1	Article/ Paragraphe précédent	Parution courante
8.1	***DONNÉES DE BLOC***			
8.1.1	DEVICE_NAME	8.2	8.5	1.0
8.1.2	DEVICE_FORM	8.1	8.4	1.0
8.1.3	BLOCK_VERSION	8.17	8.2	1.0
8.1.4	BLOCK_CREATION_DATE	8.16	8.1	1.0
8.1.5	VERSION	8.3	8.3	1.0
8.2	***DEVICE_DATA***			
8.2.1	DIE_NAME		8.8	1.2.1
8.2.2	DIE_PACKAGED_PART_NAME	8.36	8.9	1.0
8.2.3	DIE_MASK_REVISION	8.30	8.6	1.0
8.2.4	MANUFACTURER	8.4	8.7	1.0
8.2.5	DATA_SOURCE	8.19	8.11	1.0
8.2.6	DATA_VERSION	8.55	8.12	1.0
8.2.7	FUNCTION	8.5	8.10	1.0
8.2.8	IC_TECHNOLOGY	8.20	8.25	1.0
8.2.8	DEVICE_PICTURE_FILE			1.3.0
8.2.9	DEVICE_DATA_FILE			1.3.0
8.3	***DONNÉES GÉOMÉTRIQUES***			
8.3.1	GEOMETRIC_UNITS	8.6	8.13	1.0
8.3.2	GEOMETRIC_VIEW	8.7	8.14	1.0
8.3.3	GEOMETRIC_ORIGIN	8.10	8.17	1.0
8.3.4	SIZE	8.8	8.15	1.0
8.3.5	SIZE_TOLERANCE	8.21	8.18	1.0
8.3.6	THICKNESS	8.9	8.16	1.0
8.3.7	THICKNESS_TOLERANCE	8.22	8.19	1.0
8.3.8	FIDUCIAL_TYPE	8.47	8.51	1.0

Article/ Paragra-phe	Nom du paramètre	Référence ES59008-6-1	Article/ Paragraphe précédent	Parution courante
8.3.9	FIDUCIAL	8.48	8.52	1.0
8.4	***DONNÉES RELATIVES AUX BORNES***			
8.4.1	TERMINAL_COUNT	8.11	8.20	1.0
8.4.2	TERMINAL_TYPE_COUNT	8.12	8.21	1.0
8.4.3	CONNECTION_COUNT	8.13	8.22	1.0
8.4.4	TERMINAL_TYPE	8.14	8.23	1.0
8.4.5	TERMINAL	8.15	8.24	1.0
8.4.6	TERMINAL_GROUP		8.58	1.3.0
8.4.7	PERMUTABLE		8.61	1.3.0
8.5	***DONNÉES RELATIVES AUX MATÉRIAUX***			
8.5.1	TERMINAL_MATERIAL (anciennement DIE_TERMINAL_MATERIAL)		8.30	1.3.0
8.5.2	TERMINAL_MATERIAL_STRUCTURE			1.3.0
8.5.3	DIE_SEMICONDUCTOR_MATERIAL	8.34	8.26	1.0
8.5.4	DIE_SUBSTRATE_MATERIAL	8.32	8.27	1.0
8.5.5	DIE_SUBSTRATE_CONNECTION	8.31	8.28	1.0
8.5.6	DIE_PASSIVATION_MATERIAL	8.37	8.29	1.0
8.5.7	DIE_BACK_DETAIL	8.33	8.31	1.0
8.6	***DONNÉES ÉLECTRIQUES et THERMIQUES***			
8.6.1	MAX_TEMP	8.18	8.33	1.0
8.6.2	MAX_TEMP_TIME			1.3.0
8.6.3	POWER_RANGE	8.23	8.34	1.0
8.6.4	TEMPERATURE_RANGE	8.24	8.35	1.0
8.7	***SIMULATOR DATA***			
8.7.1	SIMULATOR_*simulator*_MODEL_FILE	8.25	8.36	1.0
8.7.2	SIMULATOR_*simulator*_MODEL_FILE_DATE	8.26	8.37	1.0
8.7.3	SIMULATOR_*simulator*_NAME	8.27	8.38	1.0
8.7.4	SIMULATOR_*simulator*_VERSION	8.28	8.39	1.0
8.7.5	SIMULATOR_*simulator*_COMPLIANCE	8.29	8.40	1.0
8.7.6	SIMULATOR_*simulator*_TERM_GROUP		8.59	1.3.0
8.8	***MANUTENTION, CONDITIONNEMENT, STOCKAGE et ASSEMBLAGE***			
8.8.1	DELIVERY_FORM (anciennement DIE_DELIVERY_FORM)	8.35	8.41	1.0
8.8.2	PACKING_CODE	8.46	8.42	1.0
8.8.8	Paramètres ASSEMBLY			1.3.0
8.9	***DONNÉES SPÉCIFIQUES AUX TRANCHES***			
8.9.1	WAFER_SIZE	8.45	8.32	1.0
8.9.2	WAFER_THICKNESS			1.3.0
8.9.3	WAFER_THICKNESS_TOLERANCE			1.3.0
8.9.4	WAFER_DIE_STEP_SIZE		8.53	1.2.1

Article/ Paragra-phe	Nom du paramètre	Référence ES59008-6-1	Article/ Paragraphe précédent	Parution courante
8.9.5	WAFER_GROSS_DIE_COUNT		8.54	1.2.1
8.9.6	WAFER_INDEX		8.55	1.2.1
8.9.7	WAFER_RETICULE_STEP_SIZE		8.56	1.2.1
8.9.8	WAFER_RETICULE_GROSS_DIE_COUNT		8.57	1.2.1
8.9.9	WAFER_INK			1.3.0
8.10	***DONNÉES SPÉCIFIQUES AUX TERMINAISONS À BOSSES***			
8.10.1	BUMP_MATERIAL	8.38	8.43	1.0
8.10.2	BUMP_HEIGHT	8.39	8.44	1.0
8.10.3	BUMP_HEIGHT_TOLERANCE	8.40	8.45	1.0
8.10.4	BUMP_SHAPE			1.3.0
8.10.5	BUMP_SIZE			1.3.0
8.10.6	BUMP_SPECIFICATION_DRAWING			1.3.0
8.10.7	BUMP_ATTACHMENT_METHOD			1.3.0
8.11	***DONNÉES SPÉCIFIQUES AUX DISPOSITIFS À ENCAPSULATION MINIMALE***			
8.11.1	MPD_PACKAGE_MATERIAL	8.41	8.46	1.0
8.11.2	MPD_PACKAGE_STYLE		8.47	1.2.1
8.11.3	MPD_CONNECTION_TYPE	8.43	8.49	1.0
8.11.4	MPD_MSL_LEVEL			1.30
8.11.5	MPD_PACKAGE_DRAWING			1.30
Supprimé	MPD_DELIVERY_FORM (se référer à 8.8.1)	8.42	8.48	1.0
Supprimé	MPD_CONNECTION_MATERIAL (se référer à 8.5.1)	8.44	8.50	1.0
8.12	***DONNÉES DE QUALITÉ, FIABILITÉ et ESSAIS***			
8.12.1	Paramètres QUALITY			1.3.0
8.12.1	Paramètres TEST	.		1.3.0
8.13	***AUTRES DONNÉES***			
8.13.1	TEXT		8.60	1.3.0
8.14	***DONNÉES DE CONTRÔLE***			
8.14.1	PARSE			1.3.0

Annexe K
(informative)

Contrôle d'analyse

Les séries de paramètres PARSE_parameters sont destinés à ajouter un contrôle en ligne du logiciel d'analyse, principalement pour permettre des ajouts et des extensions aux logiciels DDX qui n'ont pas, à ce jour, été ratifiés et inclus dans la norme courante. Noter qu'aucun des paramètres PARSE_ parameters n'a de pertinence ou de signification pour les données du dispositif.

À cet effet, il a été ajouté un certain nombre de PARSE_ parameters:

8.14.1 **PARSE_MODE**
8.14.2 **PARSE_ERROR_REPORT**
8.14.3 **PARSE_ERROR_TRAP**
8.14.4 **PARSE_IGNORE**
8.14.5 **PARSE_DEFINE_PARAMETER**
8.14.6 **PARSE_DEFINE_STRUCTURE**

Les paramètres **PARSE_MODE**, **PARSE_ERROR_REPORT**, **PARSE_ERROR_TRAP** et **PARSE_IGNORE** peuvent avoir des occurrences multiples et sont destinés à “commuter” le mode d'analyse souhaité à mesure que le fichier DDX ou le bloc DDX est lu de façon linéaire.

Les paramètres **PARSE_DEFINE_PARAMETER** et **PARSE_DEFINE_STRUCTURE** servent à présenter de nouveaux paramètres et/ou de nouvelles structures au logiciel d'analyse en vue de leur inclusion dans les vérifications de la syntaxe, la validation des données et l'affectation de valeurs, uniquement pour ce bloc DDX spécifique. Une fois qu'ils sont définis, les nouveaux paramètres et structures sont censés se conformer totalement à la syntaxe etc., tels que définis dans les Articles 5 à 7. En attendant que les nouveaux paramètres et structures aient été ratifiés et inclus dans une mise à jour de la présente norme DDX, toutes les données doivent être traitées comme étant textuelles et n'auront pas d'unité S.I. associée.

Le paramètre **PARSE_MODE** est destiné à contrôler la manière dont les règles relatives à la syntaxe et aux données sont interprétées et appliquées. Dans tous les cas, les données conformes à la présente version du format DDX (voir l’Article 1) doivent être exemptes d'erreurs ou d'avertissements. Les nouveaux paramètres ou structures de données qui ont été introduits à l'aide des paramètres **PARSE_DEFINE_**xxx peuvent nécessiter une certaine relaxation de règles avant qu'un résultat d'analyse exempt d'erreurs soit obtenu.

`PARSE_MODE = STRICT;`
Dans des circonstances «normales», le paramètre _MODE aura la valeur par défaut “STRICT”, et toutes les données actuellement conformes doivent être exemptes d'erreurs et des erreurs seront émises si les options de PARSE_DEFINE sont utilisées. Ces erreurs sont ?????structures autres que celles données dans la norme courante.

`PARSE_MODE = RELAXED;`
La valeur “RELAXED” est donnée pour permettre l'inclusion de données définies supplémentaires qui peuvent ne pas se conformer strictement aux règles telles qu'elles sont écrites. Au lieu d'erreurs, des avertissements doivent être émis lorsque de nouveaux paramètres sont inclus pour signaler à l'utilisateur que le bloc DDX examiné contient des paramètres qui ne figurent pas actuellement dans la norme.

`PARSE_MODE = USER;`
`PARSE_MODE = ENHANCED;`
Les valeurs “ENHANCED” et “USER” sont des options alternatives disponibles à la discrétion du fournisseur et de l'utilisateur de logiciel d'analyse.

Le paramètre **PARSE_ERROR_REPORT** détermine le niveau des avertissements et des erreurs rapportés PENDANT l'analyse; il convient qu'il n'ait pas d'effet sur le rapport final. Cela sert principalement à empêcher que les avertissements relatifs à des paramètres soient utilisés avant leur affectation, en raison de l'introduction de nouveaux paramètres. Ces options de paramètre s'expliquent d'elles-mêmes.

```
PARSE_ERROR_REPORT = OFF;
PARSE_ERROR_REPORT = TERSE;
PARSE_ERROR_REPORT = VERBOSE;
```

Le paramètre **PARSE_ERROR_TRAP** détermine comment il convient que le logiciel d'analyse réagisse une fois qu'une erreur (par opposition à avertissement) a été décelée.

```
PARSE_ERROR_TRAP = ALL;
```
Il s'agit du mode «par défaut» attendu. Le bloc DDX est analysé et toutes les erreurs sont rapportées.

```
PARSE_ERROR_TRAP = FIRST;
```
Il convient que la valeur “FIRST” fasse s'arrêter le logiciel d'analyse dès la détection de la première erreur.

Le paramètre **PARSE_IGNORE** est destiné à contrôler si la vérification d'une partie du bloc DDX est réellement effectuée ou non et il convient donc de l'utiliser avec prudence.

```
PARSE_IGNORE = NONE;
PARSE_IGNORE = OFF;
```
Tous les contrôles d'analyse sont exécutés. Il convient que cela soit l'état par défaut pour les logiciels d'analyse.

```
PARSE_IGNORE = ALL;
```
Tous les contrôles d'analyse sont ignorés, ce qui n'est utile que pour contourner les données inutiles

```
PARSE_IGNORE = SYNTAX_ONLY;
```
Seuls les contrôles des erreurs de syntaxe et des fins de ligne sont effectués. Il n'y a aucune vérification des déclarations aval/amont des variables ni des nombres de paramètres etc. Il convient d'utiliser cette option seulement en cas d'introduction de nouveaux paramètres ou de nouvelles structures qui incluent un référencement variable ou des données autres que textuelles.

AVERTISSEMENTS et ERREURS

Il est conseillé de classer les éléments suivants comme étant des **WARNINGS** (avertissements) et de classer toutes les autres erreurs comme étant des **ERRORS** (erreurs):

WARNING 1. Occurrence de caractères ASCII appartenant à la plage 0x80 à 0xFF

WARNING 2. Lorsqu'une ligne de données est censée être terminée en raison de fait que sa longueur dépasse le tampon d'entrés de lignes de l'analyseur. Se référer à 6.3.9.

WARNING 3. Lorsque des données textuelles, non entourées de guillemets, comportent des caractères de retour de ligne, voir 6.3.8.

WARNING 4. Lorsque le nom de fichier textuel n'est pas conforme au jeu de caractères défini en 7.1.3.2.

WARNING 5. Lorsque **PARSE_DEFINE**_xxx introduit une structure qui a été ratifiée et définie dans des versions plus récentes de la norme DDX.

_____

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

3, rue de Varembé
PO Box 131
CH-1211 Geneva 20
Switzerland

Tel: + 41 22 919 02 11
Fax: + 41 22 919 03 00
info@iec.ch
www.iec.ch