

BS IEC 62016:2003



BSI Standards Publication

Core model of the electronics domain

NO COPYING WITHOUT BSI PERMISSION EXCEPT AS PERMITTED BY COPYRIGHT LAW

raising standards worldwide™

Full document details: BS IEC 62016:2003



National foreword

This British Standard is the UK implementation of IEC 62016:2003.

The UK participation in its preparation was entrusted to Technical Committee GEL/93, Design automation.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

Compliance with a British Standard cannot confer immunity from legal obligations.

ISBN 978 0 580 73033 7

ICS 25.040.01

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 August 2011.

© BSI 2011

Amendments and Corrigenda issued since publication

Date	Text affected
------	---------------

INTERNATIONAL STANDARD

**IEC
62016**

First edition
2003-12

Core model of the electronics domain



Reference number
IEC 62016:2003(E)

Publication numbering

As from 1 January 1997 all IEC publications are issued with a designation in the 60000 series. For example, IEC 34-1 is now referred to as IEC 60034-1.

Consolidated editions

The IEC is now publishing consolidated versions of its publications. For example, edition numbers 1.0, 1.1 and 1.2 refer, respectively, to the base publication, the base publication incorporating amendment 1 and the base publication incorporating amendments 1 and 2.

Further information on IEC publications

The technical content of IEC publications is kept under constant review by the IEC, thus ensuring that the content reflects current technology. Information relating to this publication, including its validity, is available in the IEC Catalogue of publications (see below) in addition to new editions, amendments and corrigenda. Information on the subjects under consideration and work in progress undertaken by the technical committee which has prepared this publication, as well as the list of publications issued, is also available from the following:

- **IEC Web Site** (www.iec.ch)

- **Catalogue of IEC publications**

The on-line catalogue on the IEC web site (http://www.iec.ch/searchpub/cur_fut.htm) enables you to search by a variety of criteria including text searches, technical committees and date of publication. On-line information is also available on recently issued publications, withdrawn and replaced publications, as well as corrigenda.

- **IEC Just Published**

This summary of recently issued publications (http://www.iec.ch/online_news/justpub/jp_entry.htm) is also available by email. Please contact the Customer Service Centre (see below) for further information.

- **Customer Service Centre**

If you have any questions regarding this publication or need further assistance, please contact the Customer Service Centre:

Email: custserv@iec.ch
Tel: +41 22 919 02 11
Fax: +41 22 919 03 00

CONTENTS

FOREWORD.....	5
INTRODUCTION.....	7
1 Scope and object.....	8
2 Reference documents.....	9
3 General modelling issues	9
3.1 Ownership and reference	9
3.2 Uniqueness by value	12
3.3 Default values	12
3.4 Optional versus empty sets	12
3.5 Model topology.....	12
4 Concepts.....	13
4.1 The information base.....	13
4.2 Global ports and global port bundles	13
4.3 Libraries	13
4.4 Cells.....	14
4.5 Clusters and cell representation sets.....	14
4.6 Cell representations	14
4.7 Master ports and master port bundles	14
4.8 Instances	15
4.9 Instance ports and instance port bundles	15
5 Connectivity	16
5.1 Logical connectivity	16
5.2 Structural connectivity with wide instances	17
5.3 Structural connectivity of connectivity views	19
6 The design hierarchy mechanism	21
6.1 The design hierarchy.....	21
6.2 Annotations	23
7 Core Model for electronic design	24
7.1 Libraries	24
7.2 Interfacing to cells	25
7.3 Cell definition hierarchies	27
7.4 Instantiation	27
7.5 Logical connectivity	28
7.6 Structural connectivity	28
7.7 Global connectivity	32
7.8 Design and configuration.....	33
7.9 Annotation.....	34
8 Core Model EXPRESS-G.....	34
8.1 Partial EXPRESS-G of cell	35
8.2 Partial EXPRESS-G of cell_representation	36
8.3 Partial EXPRESS-G of cluster	37
8.4 Partial EXPRESS-G of cluster_configuration	38
8.5 Partial EXPRESS-G of cluster_interface	39
8.6 Partial EXPRESS-G of connectivity_generic_bus	40

8.7	Partial EXPRESS-G of connectivity_generic_net	41
8.8	Partial EXPRESS-G of design	42
8.9	Partial EXPRESS-G of global_port	43
8.10	Partial EXPRESS-G of information_base	44
8.11	Partial EXPRESS-G of instance.....	45
8.12	Partial EXPRESS-G of instance_configuration.....	46
8.13	Partial EXPRESS-G of library	47
8.14	Partial EXPRESS-G of master_port_annotate.....	48
8.15	Partial EXPRESS-G of name_information.....	49
8.16	Partial EXPRESS-G of occurrence_annotate	50
8.17	Partial EXPRESS-G of occurrence_annotate	51
8.18	Partial EXPRESS-G of occurrence_annotate	52
8.19	Partial EXPRESS-G of occurrence_hierarchy_annotate.....	53
8.20	Partial EXPRESS-G of port_structure	54
8.21	Partial EXPRESS-G of property.....	55
8.22	Partial EXPRESS-G of property_override	56
8.23	Partial EXPRESS-G of signal	57
9	Core Model schemas.....	58
9.1	connectivity_structure_model	58
9.2	connectivity_view_model	59
9.3	design_hierarchy_model.....	60
9.4	design_management_model	62
9.5	documentation_model	63
9.6	hierarchy_model.....	64
9.7	information_base_model	67
9.8	library_model	68
9.9	logical_connectivity_model	69
9.10	support_definition_model	70
10	Core Model information model	72
10.1	connectivity_structure_model	72
10.2	design_hierarchy_model.....	89
10.3	documentation_model	114
10.4	hierarchy_model.....	116
10.5	information_base_model	157
10.6	support_definition_model	166
11	Index	188
	Figure 1 – The owner relationship	10
	Figure 2 – Owner relationship with multiple potential owners	11
	Figure 3 – The reference mechanism	11
	Figure 4 – Uniqueness by value	12
	Figure 5 – An example of signal hierarchy	17
	Figure 6 – A net joins a port on an instance in the commoned style	18
	Figure 7 – A bus joins a port bundle on an instance in the commoned style	18
	Figure 8 – A bus joins a port bundle on an instance in the fanned-out style	19
	Figure 9 – Internal and external libraries.....	25
	Figure 10 – Port bundling – 1.....	26

Figure 11 – Port bundling – 2.....	26
Figure 12 – Instantiation	27
Figure 13 – Connectivity net	28
Figure 14 – Connectivity bus.....	29
Figure 15 – Connectivity bus – Commoning	29
Figure 16 – Connectivity bus – Logical equivalent of commoning	30
Figure 17 – Connectivity bus – Fanning-out	30
Figure 18 – Connectivity bus – Logical equivalent of fanning-out	31
Figure 19 – Connectivity bus-slice	31
Figure 20 – Connectivity ripper	32
Figure 21 – Global port scoping	33

INTERNATIONAL ELECTROTECHNICAL COMMISSION

CORE MODEL OF THE ELECTRONICS DOMAIN

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 provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with an IEC Publication.
- 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 62016 has been prepared by IEC technical committee 93: Design automation.

The document was released by the feeder organization, Government Electronics and Information Technology Association, a sector of the Electronic Industries Alliance (EIA), for committee draft, comment, and review by members of IEC TC93 ¹.

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

FDIS	Report on voting
93/172/FDIS	93/176/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.

¹ The EIA feeder organization retains the copyright and intellectual property of this work but provides permission for IEC to reproduce and exploit the information contained herein.

This standard does not follow the rules for the structure of International Standards given in the ISO/IEC Directives, Part 2.

The committee has decided that the contents of this publication will remain unchanged until 2012-07. At this date, the publication will be

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

INTRODUCTION

The Core Model of the electronics domain provides a common basis for design information handled by CAD systems within the electronic domain. It is the purpose of this model to provide a conceptual representation of the electronics domain, so that the compliant CAD systems handle a similar set of concepts, thus making inter-communication, sharing and exchange of design information a much easier task. It is not the purpose of this model to describe implementation details or to provide a data representation of electronics domain information.

The Core Model of the electronics domain, Edition 1.0, is referred to as the “Core Model” throughout this document. The Core Model, in part, has been created by enhancing the industry connectivity consensus model, EDIF CFI DR Alignment Model Version 1.0 (www.edif.org).

The chosen description language for this Core Model is EXPRESS, as defined by ISO 10303-11.

It is necessary to describe the Core Model as an information model in order to provide a formal definition of the design information that shall be recognized by the compliant CAD systems. The benefits of a formal description derive from its ability to provide an unambiguous representation of concepts, attributes and relationships, and the global rules and constraints that may be applied. By having such a description, it is possible to check the consistency and the correctness of the model as well as to provide a reliable starting-point for further development. It also facilitates the design of correct electronics CAD implementations based on this Core Model, as the actual implementation methods can be checked against the model.

This Core Model includes connectivity, hierarchy and design information for the electronics domain. Future parts of this Core Model standard may be extended to include other categories of information (for example, *cell_representation*, schematic representation, the PCB domain, symbols and display information).

In order to facilitate the creation of other parts of this Core Model standard, some objects have been used in this Core Model to facilitate support for other parts of the electronics domain. There are two types of such objects.

- Entities, such as *cell_representation*, are important concepts that provide support for defining other Core Model parts.
- Constraints: Some of constraints of this Core Model use conditions that are always true. They have been written in this way in order to ensure that they remain valid when the model is extended.

CORE MODEL OF THE ELECTRONICS DOMAIN

1 Scope and object

This International Standard provides the semantics definitions for the following categories of information related to electronic circuit designs. Each category of design information is modelled as an EXPRESS schema.

The Core Model consists of 10 schemas. Each of them is presented in this document as a separate chapter. At the beginning of each chapter, a description of the corresponding schema is provided.

- The *hierarchy_model* schema describes the hierarchical information of a cell, i.e. the way a cell may be divided into other cells.

A circuit may be divided into cells which, in turn, may be further subdivided into other cells, thus creating a hierarchy. The hierarchy information describes the cells, the possible cell representations and their instances.

- The *design_hierarchy_model* schema describes the annotation on an occurrence hierarchy.

The definition of a design requires that specific representations (views) of design objects in the hierarchy are selected. This unambiguously creates a configured design hierarchy. This concept is similar to the configuration of a design in VHDL and is related to view selection mechanisms of other electronics design domain information models in industry. The design hierarchy identifies top-level design cells and may provide annotated design-specific data into the elaborated hierarchy.

- The *connectivity_view_model* schema describes the connectivity information of a cell.

This describes the way in which the circuits are connected in order that information or energy may flow from one part of a design or product to another. The Core Model subdivides this information into

- the *logical_connectivity_model* schema which describes the connectivity for a given level of a hierarchy.

Logical connectivity information describes the bit level, abstract electrical connectivity for a given level of a hierarchy, in terms of signals and signal groups.

- the *connectivity_structure_model* schema which describes the structural connectivity of a connectivity view.

Structural connectivity information describes the connectivity, for a given level of a hierarchy, from the structural point of view. The structural connectivity is specified in terms of busses, nets and rippers. Such a structural representation is used to provide support for physical implementation and annotation.

- The *library_model* schema describes the technology information contained in a library, as well as the reusable objects and data.

A library provides a means of grouping cell definitions. A library may be used to group other classes of reusable objects and information as well. Information in a library may be related in terms of technology information.

- The *information_base_model* schema describes the information in an information base.

The *information_base* describes the kind of information that can be found directly in an information base.

In addition, the following information is also included in the model.

- The *design_management_model* schema provides the design management information. This information is needed to trace back to the source or the owner of the data.
- The *documentation_model* schema describes the documentation provided for an object.
- The *support_definition_model* schema contains the definition of some auxiliary entities, types and functions that are used by several schemas.

Names of objects used in this Core Model standard were chosen to be the same as the names of the similar objects and concepts in existing electronics domain information models wherever possible.

2 Reference documents

IEC 61690-1:2000, *Electronic design interchange format (EDIF) – Part 1: Version 300*

IEC 61690-2:2000, *Electronic design interchange format (EDIF) – Part 2: Version 400*

ISO 13030-11:1994, *Industrial automation systems and integration – Product representation and exchange – Part 11: Description methods: The EXPRESS-I language reference manual*

3 General modelling issues

This standard describes the general modelling techniques and conventions used by the Core Model. The most important concept discussed here is that of relationship between entities (ownership and reference). In addition, issues such as uniqueness in aggregates, default values, empty sets and model topology are presented.

3.1 Ownership and reference

An object is said to be the “owner” of another object, if the latter can only exist in the context of the former. From the point of view of the information model, this means that an instance of the “owned” object can exist if, and only if, there is an instance of the owner object. Any given instance of an object must have exactly one owner. Some examples of owner relationships are provided below.

By contrast, if an object references another object, the existence of the latter is not dependent on the existence of the former. In the information model this means that the existence of an instance of the referenced object is not related to the existence of any instance of the object that references it. A given instance may be referenced by any number of other instances unless stated otherwise by a constraint.

Generally speaking, the referencing mechanism provides a way of sharing data whereas the ownership dependence is used to create a scope for the objects and to control their existence.

The difference between the two types of relationship is important because of their effects in an actual implementation. If the implementation provides a static representation of data (such as an EDIF file), an owned object can be textually contained in the definition of the owner, whereas a referenced object is used by the referencing object but may be declared in another context. If the system allows dynamic representation of the data (like CFI DR PI), the difference between ownership and reference is reflected in the process of object creation and destruction. Indeed, whenever an object is destroyed, its owned objects are destroyed too, unlike the referenced objects which continue to exist.

The EXPRESS language does not provide a direct method of specifying whether an object is owned or referenced. However, the Alignment Model represents the two types of relationship in different ways by using the techniques and conventions described below.

3.1.1 Ownership

Let us consider the example presented in Figure 1 which shows the owner relationship between a *library* and its contained *cells*:

```
ENTITY library
  cells : OPTIONAL SET [1:?] OF cell;
  ...
END_ENTITY;

ENTITY cell
  ...
  INVERSE
    containing_library : library FOR cells;
END_ENTITY;
```

Figure 1 – The owner relationship

In this example, ownership is modelled straightforwardly by using the EXPRESS INVERSE clause. Its meaning is that, for every instance of a *cell*, there is exactly one instance of the *library* that contains that *cell* instance.

There are cases, however, when an object may have several potential owners. The example in Figure 2 shows that a *documentation* object may be created in the context of a *connectivity_generic_net* or a *connectivity_generic_bus*, etc.

The owner relationship is modelled by using the INVERSE clause. Its meaning is that there may exist, at most, one instance of the *connectivity_generic_net* and, at most, one instance of a *connectivity_generic_bus*, etc, that contain a given *documentation* instance. However, any instance of the *documentation* may exist as a member of the “document” SET in a *connectivity_generic_net* or as a member of the “document” SET in a *connectivity_generic_bus*, etc, but not as all of them. Therefore, a domain rule (the WHERE clause) is used in order to ensure that there is only one owner of the *documentation* instance in the database. Of course, there are other objects that reference that *documentation* instance.

```
ENTITY connectivity_generic_net
  ABSTRACT SUPERTYPE OF (ONEOF(connectivity_net,
                                connectivity_sub_net));
  ...
  document : OPTIONAL SET [1:?] OF documentation;
  ...
END_ENTITY;

ENTITY connectivity_generic_bus
  ABSTRACT SUPERTYPE OF (ONEOF(connectivity_bus,
                                connectivity_bus_slice,
                                connectivity_sub_bus));
  ...
  document : OPTIONAL SET [1:?] OF documentation;
  ...
END_ENTITY;

ENTITY documentation;
  ...
  INVERSE
    containing_generic_net:
      SET [0:1] OF connectivity_generic_net FOR document;
    containing_generic_bus:
      SET [0:1] OF connectivity_generic_bus FOR document;
  ...
  WHERE
    containment_constraint:
      (* Each "documentation" is defined in only one place. *)
      SIZEOF(containing_generic_net) +
      SIZEOF(containing_generic_bus) +
  ...
END_ENTITY;
```

Figure 2 – Owner relationship with multiple potential owners

3.1.2 Reference

The reference mechanism can be used to share common data between several objects. Let us consider a *master_logical_port* which has a default connection to a *global_port*. Since it is possible for several *master_logical_ports* to have the same default connection, the *global_port* is referenced by the *master_logical_port*. In this case, the *global_port* does not contain an INVERSE attribute for the *master_logical_port* because it is not owned by it. In order to increase the readability of the model, the references to other objects are tagged as such. Figure 3 gives an example of object referencing.

```
ENTITY master_logical_port;
  default_connection : OPTIONAL global_port; -- reference
  ...
END_ENTITY;
```

Figure 3 – The reference mechanism

3.2 Uniqueness by value

EXPRESS defines the uniqueness in aggregates to be “by reference”. This means that a set cannot contain the same object twice but it may contain two different objects with exactly the same value. Therefore, a special EXPRESS function is used whenever uniqueness by value is required, as shown in Figure 4.

```
ENTITY library
  document          : OPTIONAL SET [1:?] OF documentation;
  status_of_copyright : OPTIONAL SET [1:?] OF copyright;
  status_of_written   : OPTIONAL SET [1:?] OF written;
  ...
WHERE
  unique_status_of_copyright:
    value_unique(status_of_copyright);
  unique_status_of_written:
    value_unique(status_of_written);
  unique_document:
    value_unique(document);
END_ENTITY;
```

Figure 4 – Uniqueness by value

The example shows that a *library* should not contain two identical documentation, copyright information or author information, i.e. it cannot contain two instances of *documentation*, *copyright* or *written* with the same value.

3.3 Default values

The default values, which are used by the CAD systems, are relevant to the actual implementations rather than to the information models. Therefore, the Core Model does not contain a specification of these values.

3.4 Optional versus empty sets

In EXPRESS, a set which may have no elements can be described by either

- OPTIONAL SET [1:upper_limit] OF base_type, or by
- SET [0:upper_limit] OF base_type.

However, the concepts modelled by the two descriptions are quite different. An OPTIONAL SET shows the fact that the set may or may not exist whereas a SET[0 : upper_limit] shows that the set always exists but it may be empty. The Core Model uses the former method.

3.5 Model topology

The topology is the graph representation of the relationships between the concepts defined in the model. Two possible approaches have been considered.

- A tree representation in which there is a concept (the root of the tree) that is a generalization of all the other concepts defined in the model. An example of Information Model that uses this topology is the CFI DR IM.
- A forest representation in which there are several categories of concepts that do not have any common features. An example of Information Model that uses this topology is the EDIF IM.

The forest representation has been chosen for the Core Model. An advantage of this representation (as opposed to the tree representation) is that it reduces the maximum depth of the leaves in the model hierarchy.

4 Concepts

This chapter describes the fundamental concepts defined by the Core Model. These concepts may be thought of as creating a hierarchy which is abstract at its higher level and becomes progressively more detailed. At the highest level of the structure, an *information_base* may contain *designs* and *libraries*. Each *library* is a collection of *cells* which are grouped according to a set of common characteristics. A *cell* is the main design object which may be instantiated later in another *cell*, thus creating a design hierarchy. A *cell* may be connected to other *cells* by means of its interface. These concepts are described in the following sections.

4.1 The information base

An *information_base* describes the data that can be found in the database of a compliant CAD system. This information may include *libraries*, *designs*, *global_ports* and *global_port_bundles* and *unit* definitions. The Core Model is an information model that describes one information base. All the entities defined in the Core Model belong, directly or indirectly, to the *information_base* object. The model of the *information_base* entity can be found in the *information_base_model* schema. A partial EXPRESS-G diagram is also available.

4.2 Global ports and global port bundles

Certain ports such as GND, VCC or clocks are used in common by several modules of an electronic circuit. The Core Model describes this by using the concept of global port, which is modelled by the *global_port* entity. The *global_ports* are defined in the *information_base* and are, therefore, visible in all the *cells* and *designs* defined in this information base.

It is sometimes convenient to address several ports as if they were a single port. In order to do this the Core Model uses the concept of port bundle. A port bundle provides a means of creating a structured port by grouping other ports and/or port bundles defined in the same context. In the case of *global_ports*, the port bundle is modelled by the *global_port_bundle* entity.

Both the *global_port* and the *global_port_bundle* are described in the *information_base_model* schema. A partial EXPRESS-G diagram is also available.

4.3 Libraries

A *library* provides a means of grouping design units (*cell* definitions). In future versions of the Core Model a *library* will contain other classes of reusable objects, as well.

A *library* may be locally defined, in which case the definition of the contained *cells* is included in the information base. *Libraries* may also be externally defined, which implies that the details of the *cell* definitions are not available within the information base.

The *library_model* schema describes the model for the *library* entity. A partial EXPRESS-G diagram is also available. According to where the contained data can be found, the library entity is subtyped into *internal_library* and *external_library*. The *internal_library* corresponds to the case where the library is locally defined. Therefore, since all the data is present in the *information_base*, an *internal_library* may contain implementation information. An *external_library* corresponds to the case where the library data is not contained in the *information_base* and, therefore, it does not include implementation details.

4.4 Cells

A *cell* is the basic design unit described by the Core Model. In order to be able to represent complex circuits, a *cell* can be divided into other *cells*, thus creating a hierarchy. The model of the *cell* entity is given in the *hierarchy_model* schema. A partial EXPRESS-G *cell* entity is subtyped into *internal_cell* and *external_cell* according to whether it describes information defined locally or not. An *internal_cell* contains locally defined data and may, therefore, contain implementation information. An *external_cell* does not include implementation details. Since an *internal_cell* contains data defined in the current *information_base*, it can only belong to an *internal_library*. Similarly, an *external_cell* belongs to an *external_library*.

4.5 Clusters and cell representation sets

A *cell* is described by means of *cell_representations*. *Cell_representations* that share the same interface can be grouped into *clusters*. A *cell* can contain more than one *cluster*. A *cluster* may be instantiated in another cell view, thus creating a design hierarchy. A *cluster* is subtyped into *internal_cluster* and *external_cluster* which are used by *internal_cell* and *external_cell* respectively. An *internal_cluster* may only contain *internal_cell_representations*, and an *external_cluster* only *external_cell_representations*. The model of the *cluster* entity is given in the *hierarchy_model* schema. A partial EXPRESS-G diagram is also available.

Cell representation sets are used to group *cell_representations* of the same *cell* that have a close relationship based on reasons other than a common interface. Simple annotation is provided within the *cell representation set* in order to specify the reason for the grouping. The model for the *cell_representation_set* entity can be found in the *hierarchy_model* schema.

4.6 Cell representations

A *cell_representation* describes chosen aspects of a *cell*. At present, the Core Model supports only one type of *cell_representation*, the connectivity view. Future versions of the model will be extended, however, with other view types such as schematic and pcb layout, as well as with symbols. A *cell_representation* may be either an *internal_cell_representation* or an *external_cell_representation*. An *internal_cell_representation* belongs to an *internal_cluster* and describes an *internal_cell*. Similarly, an *external_cell_representation* belongs to an *external_cluster* and describes an *external_cell*. An *internal_cell_representation* can establish a direct relationship with other *cell_representations* in order to provide versioning and data management control information. Hence, an *internal_cell_representation* may be a new version of another *cell_representation* of the same type or may be derived from another *cell_representation*. If the *internal_cell_representation* is derived from another *cell_representation*, their type can be different. The interface of a *cell_representation* is obtained from its containing *cluster*.

The model of the *cell_representation* entity can be found in the *hierarchy_model* schema. A partial EXPRESS-G *internal_connectivity_view* and *external_connectivity_view* are given in the *connectivity_view_model* schema.

4.7 Master ports and master port bundles

The interface of a *cluster* consists of ports and port bundles. These objects represent the only places where a connection can be made to a *cell*. The interface of a *cluster* is described by the *cluster_interface* entity.

In the Core Model, each port in the cluster interface is described by a *master_logical_port* entity which describes its logical connectivity information. A *master_logical_port* can be associated with one or several *master_structure_ports* which describe the structural connectivity information of the port. It is possible that a *master_logical_port* is not associated with any *master_structure_port*.

The size of the *master_logical_port* and *master_structure_port* is always one. The *master_logical_port* entity is subtyped into *input_master_logical_port*, *output_master_logical_port*, *bidirectional_master_logical_port* and *unspecified_direction_master_logical_port*, which correspond to the possible directions of the port. In addition, a port contains information such as its designator, its properties and its external load capacitance. It is possible to specify a *global_port* as the default connection for a *master_logical_port*.

It is sometimes convenient to address several ports as if they were a single port. In order to be able to do this, a grouping method is provided. The Core Model uses the concept of port bundle in order to describe a structured port. A port bundle in the cluster interface is described by a *master_logical_port_bundle* entity, which groups *master_logical_ports* and/or other *master_logical_port_bundles* defined in the same interface. The same *master_logical_port* may be referenced by more than one *master_logical_port_bundle*. However, it cannot be referenced more than once by the same *master_logical_port_bundle*, either directly or indirectly.

The availability of a *master_logical_port_bundle* for structural connectivity is indicated by the *master_structure_port_bundle* entity. A *master_logical_port_bundle* can be associated with more than one *master_structure_port_bundle*.

A structured port can be defined in the interface of a *cluster* by using the *port_structure* entity. A *port_structure* describes a possible structuring of the structure ports and port bundles of a cluster interface. It can represent either an ordered or an unordered structured port.

The models for the *master_logical_port*, *master_structure_port*, *master_logical_port_bundle*, *master_structure_port_bundle* and *port_structure* entities are given in the *hierarchy_model* schema. Partial EXPRESS-G diagrams for *cluster_interface* and *port_structure* are also available.

4.8 Instances

A *cluster* may be instantiated in a cell view, thus enabling the creation of a design hierarchy. The *instance* entity models the instance of a *cluster*. The interface of the *instance* consists of *instance_structure_ports* and *instance_structure_port_bundles* which must reference *master_structure_ports* and *master_structure_port_bundles* defined in the interface of the instantiated *cluster*. If the width of the *instance* is greater than one, the *instance* represents an arrayed *instance*.

The advantage of the instantiation mechanism is that the choice of a particular *cell_representation* can be delayed until the complete *design* is configured. For example, a connectivity view may contain *instances* of *clusters* of other *cells*. When the connectivity view is configured, *instances* are configured by selecting suitable views.

An *internal_connectivity_view* can define, for each of its instances, a *connectivity_instance_implementation* which describes the structural information associated with an *instance*. Since the only type of *cell_representation* defined in the current model is the connectivity view, the only significant structural information is the port structuring.

The model for the *instance* entity can be found in the *hierarchy_model* schema. A partial EXPRESS-G diagram is also available.

4.9 Instance ports and instance port bundles

The interface of the *instance* consists of *instance_structure_ports* and *instance_structure_port_bundles*. The *instance_structure_port* is defined by either a *connectivity_generic_bus* or by a *connectivity_generic_net* and associates a *master_structure_port* with an *instance*. The information contained in the *master_logical_port* associated with the referenced *master_structure_port* may be overridden by the *instance* if it defines an *instance_port_attributes* entity associated with the *master_logical_port*.

The *instance_port_attributes* entity is subtyped into *input_instance_port_attributes*, *output_instance_port_attributes*, *bidirectional_instance_port_attributes* and *unspecified_direction_instance_port_attributes*, corresponding to the possible types of the associated *master_logical_port*.

An *instance_structure_port_bundle* entity references a *master_structure_port_bundle* which must be defined in the interface of the instantiated *cluster*. The *instance* may override some of the information contained in the *master_logical_port_bundle* associated with the referenced *master_structure_port_bundle* by defining an *instance_port_bundle_attributes* entity.

The Core Model defines the *instance* entity to be an indexed collection of *cluster* instances. The number of elements in the collection is given by the “width” attribute in the *instance* entity. A port in the interface of a member of the *instance* is modelled by the *instance_member_logical_port* and *instance_member_structure_port* entities.

The models for the *instance_structure_port*, *instance_member_logical_port*, *instance_member_structure_port*, *instance_structure_port_bundle*, *instance_port_attributes* and *instance_port_bundle_attributes* entities are given in the *hierarchy_model* schema.

5 Connectivity

The only view type supported by the model at present is the connectivity view. Future versions of the model may contain other views that support connectivity definitions such as schematic and pcblayout. The Core Model differentiates between two levels of connectivity information within the connectivity view.

- Logical connectivity information

The logical connectivity describes the bit level, abstract electrical connectivity for a given level of a hierarchy, in terms of signals and signal bundles. Every type of view that supports connectivity definitions contains similar connectivity information.

- Structural connectivity information

The structural connectivity describes the connectivity, for a given level of a hierarchy, from the structural point of view. The structural connectivity is specified in terms of buses, nets and rippers. Such a structural representation is used to provide support for physical implementation and annotation. The structural connectivity information is view-type specific. Indeed, the manner in which the connectivity view is structured is unlikely to be the same as that of a future schematic or pcb view.

5.1 Logical connectivity

The model for the logical connectivity can be found in the *logical_connectivity_model* schema. A partial EXPRESS-G diagram is also available. The logical connectivity of a view is described in terms of *signals* and *signal_bundles*.

5.1.1 Signals

A *signal* is defined as an object that provides a means by which all the *global_ports*, *instance_member_logical_ports* and *master_logical_ports* may be logically electrically common at a given level of hierarchy, and to enable the logical flowing, sharing and exchanging of information and energy between the applicable blocks.

A *signal* is always one bit wide and is defined within an *internal_connectivity_view*. All *global_ports* joined by a *signal* are defined in the containing *information_base*. All *instance_member_logical_ports* joined by a *signal* must reference *instances* in the containing view. All *master_logical_ports* joined by a *signal* are defined in the interface of the containing *cluster*.

5.1.2 Signal_bundle

A *signal_bundle* provides a grouping mechanism for *signals*. A *signal_bundle* is defined within a view and is used to support structural connectivity. However, it does not provide additional connectivity information. A *signal_bundle* is specified as an ordered list of one or more *signals* or *signal_bundles* in the same containing view. The structure of a *signal_bundle* is independent of port ordering or port structure. No two *signal_bundles* may have the same structure of members. The same *signal* may be referenced by more than one *signal_bundle* and may be referenced more than once in the same *signal_bundle*. Since a *signal_bundle* may contain *signals* and *signal_bundles*, it creates a signal hierarchy as shown in the example in Figure 5.

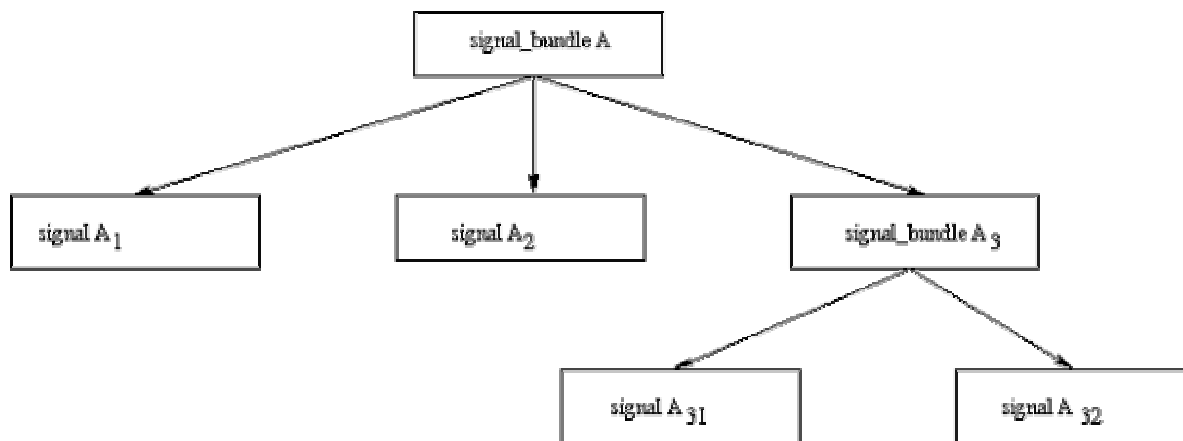


Figure 5 – An example of signal hierarchy

Such a hierarchical structure is used to support the structural connectivity expressed by buses.

5.2 Structural connectivity with wide instances

When an interconnect (net or bus) joins a port or a port bundle on a wide *instance*, the pattern of connectivity will be either in the commoned style or in the fanned-out style. For the commoned style, the joined *instance_structure_ports* or *instance_structure_port_bundles* reference *master_structure_ports* or *master_structure_port_bundles* whose size is the same as that of the interconnect. For the fanned-out style, the joined *instance_structure_ports* are flattened to lists of *instance_member_structure_ports*. The length of these lists is equal to the size of the interconnect. In the case of *instance_structure_port_bundles*, their size is equal to the size of the *master_logical_port_bundle* associated with the *master_structure_port_bundle* referenced by the *instance_structure_port_bundle*. The size of the *instance_structure_port_bundle* must be equal to the size of the joined interconnect (whose size is equal to the size of its associated *signal* or *signal_bundle*). A net is associated with a *signal* and, therefore, its size is always one. A bus is associated with a *signal_bundle* and, therefore, its size is equal to the length of the list created by flattening the *signal_bundle*. Several examples are given below.

5.2.1 Joining ports on instances in the commoned style

Figure 6 presents an example of a net joining a port on a wide *instance*, using the commoned style. The wide *instance* “I” has a width of three. The instantiated *cluster* defines a *master_structure_port* ‘P’ in the *cluster_interface*.

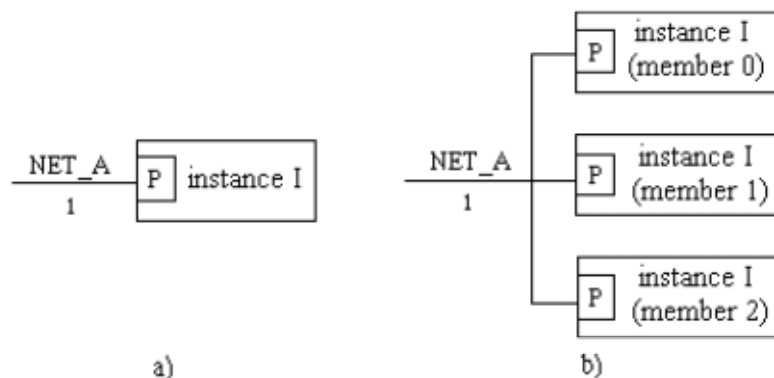


Figure 6 – A net joins a port on an instance in the commoned style

If the net “NET_A” joins port “P” in *instance* “I” as shown in Figure 6a, the underlying connectivity of “NET_A” will be equivalent to joining port “P” on the 0th member of *instance* “I”, port “P” on the 1st member of *instance* “I” and port “P” on the 2nd member of *instance* “I”, as shown in Figure 6b.

5.2.2 Joining port bundles on instances in the commoned style

Figure 7 shows an example of a bus joining a port bundle on a *instance* using the commoned style. The wide *instance* “I” has a width of three. The interface of the instantiated *cluster* contains two *master_structure_ports* “P1”, “P2” and a *master_structure_port_bundle* “PB” which groups “P1” and “P2”.

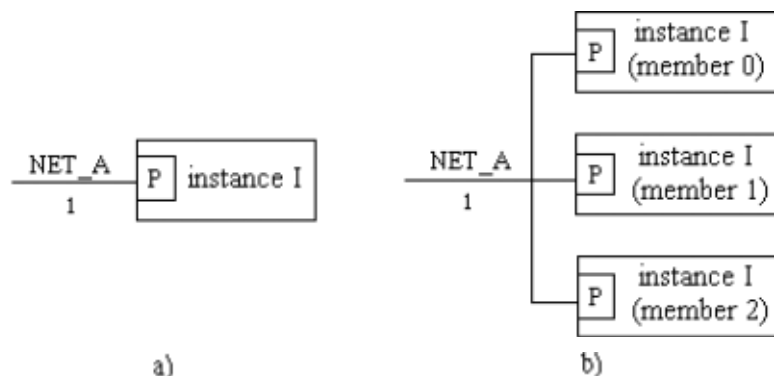


Figure 7 – A bus joins a port bundle on an instance in the commoned style

When a bus joins a port bundle on an *instance*, using the commoned style, the joined *instance_structure_port_bundle* must reference a *master_structure_port_bundle* whose size is the same as that of the bus. The size of the bus is determined by flattening its associated *signal_bundle*. In the example in Figure 7a, a bus “BUS_A” joins a port bundle “PB” on an *instance* “I”. The size of “PB” is the same as the size of “BUS_A”. This is equivalent to the situation presented in Figure 7b where the 0th bit of “BUS_A” joins port “P1” on the 0th member of *instance* “I”, port “P1” on the 1st member of *instance* “I” and port “P1” on the 2nd member of *instance* “I”, and the 1st bit of “BUS_A” joins port “P2” on the 0th member of *instance* “I”, port “P2” on the 1st member of *instance* “I” and port “P2” on the 2nd member of *instance* “I”.

5.2.3 Joining port bundles on instances in the fanned-out style

Figure 8 shows an example of a bus joining a port bundle on an *instance*, using the fanned-out style. The wide *instance* “I” has a width of three. The interface of the instantiated *cluster* defines two *master_structure_ports* “P1”, “P2” and a *master_structure_port_bundle* “PB” which groups “P1” and “P2”.

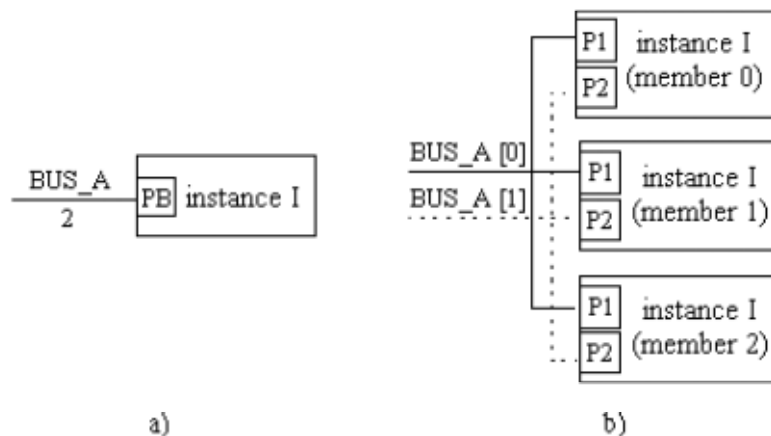


Figure 8 – A bus joins a port bundle on an instance in the fanned-out style

When a bus joins a port bundle on an *instance*, using the fanned-out style, the size of port list resulting from expanding the *master_logical_port_bundle* associated with the *master_structure_port_bundle* referenced by the joined *instance_structure_port_bundle* must be the same as that of the bus. The size of the port list is the product of the *instance* width and the referenced *master_structure_port_bundle* size. In the example in Figure 8a, a bus “BUS_C” joins the port bundle “PB” on the *instance* “I”. “PB” implicitly represents a list of six ports (i.e. *instance* width × *master_structure_port_bundle* size = 3 × 2). The elements of the port bundle “PB” correspond to consecutive members of the list of ports. The size of bus “BUS_C” is also 6. The example is equivalent to the situation presented in Figure 8b where the 0th bit of the bus joins port “P1” on the 0th member of *instance* “I”, the 1st bit joins port “P2” on the 0th member of *instance* “I”, the 2nd bit joins port “P” on the 1st member of *instance* “I”, the 3rd bit joins port “P2” on the 1st member of *instance* “I”, the 4th bit joins port “P1” on the 2nd member of *instance* “I” and the 5th bit joins port “P2” on the 2nd member of *instance* “I”.

5.3 Structural connectivity of connectivity views

The model of the structural connectivity is found in the *connectivity_structure_model* schema. The structural connectivity information is described by connectivity nets, connectivity buses and connectivity rippers. The structural representation is used to provide support for physical implementation and annotation.

5.3.1 Connectivity net and connectivity sub-net

A net is a structured representation of a *signal*. Each net is associated with a *signal* and may join single ports.

Sometimes it is necessary to divide a net into various parts so that properties may be added to one part but not to any of the others. Hence, a net may be divided into sub-nets which may be further subdivided into sub-nets. However, sub-nets do not give new connectivity information. The ports associated with a sub-net are referenced in the immediately containing net or sub-net. No two sub-nets at a given level refer to the same set of ports.

In the information model, the concept of net and sub-net are generalized into the concept of generic-net. A *connectivity_generic_net* is either a *connectivity_net* or a *connectivity_sub_net*. Each *connectivity_generic_net* is associated with a *signal*. It may join *global_ports*, *instance_member_structure_ports*, *instance_structure_ports*, *master_structure_ports* and *connectivity_rippers*. A *connectivity_generic_net* may contain *connectivity_sub_nets* which may be further divided into more *connectivity_sub_nets*.

A *connectivity_net* is defined within an *internal_connectivity_view*. Its associated *signal* and its joined *connectivity_rippers* are defined in the containing view. Furthermore, the joined *global_ports*, *instance_member_structure_ports* and *master_structure_ports* are mentioned in its associated *signal*. If a *connectivity_net* joins an *instance_structure_port*, then the list of *instance_member_structure_ports* joined implicitly is a subset of the *instance_member_structure_ports* joined by its associated *signal*.

A *connectivity_sub_net* is defined within another *connectivity_generic_net*. The associated *signal* of a *connectivity_sub_net* is the same as the one in its enclosing structure. Its joined ports or rippers are a subset of the ports or rippers referenced by its containing net.

A partial EXPRESS-G diagram of *connectivity_generic_net* and its subtypes is available.

5.3.2 Connectivity bus, connectivity bus slice and connectivity sub-bus

A bus is a structured representation of a collection of *signals*. Each bus is associated with a *signal_bundle* and may join wide ports.

A bus may be subdivided in two ways. It may be divided into bus-slices or into sub-buses. A bus-slice is used to define a part of a bus. Its associated *signal_bundle* is a member of the *signal_bundle* associated with its immediately containing bus. No two bus-slices at a given level refer to the same *signal_bundle*. A sub-bus is similar in nature to a sub-net in that it allows additional structural information to be given about a part of a bus. A sub-bus is associated with the same *signal_bundle* as that in the immediately containing bus. No two sub-buses at a given level reference the same set of ports or rippers.

In the information model, the concepts of bus, bus-slice and sub-bus are generalized into the concept of generic-bus. A *connectivity_generic_bus* is either a *connectivity_bus*, a *connectivity_bus_slice* or a *connectivity_sub_bus*. Each *connectivity_generic_bus* is associated with a *signal_bundle*. It may join *connectivity_rippers* and wide ports such as *global_port_bundles*, *local_master_port_bundles*, *instance_structure_port_bundles* and *master_structure_port_bundles*. It may be divided into *connectivity_bus_slices* or *connectivity_sub_buses* which may themselves be further subdivided.

All *global_port_bundles*, *local_master_port_bundles*, and *master_structure_port_bundles* joined by a *connectivity_generic_bus* have the same port size. This is the same as the size of the associated *signal_bundle*. Each of the logical ports associated with the ports of the joined port structures is referenced or defined in the corresponding signal in the flattened *signal_bundle*. If a generic-bus joins an *instance_structure_port_bundle*, the pattern of connectivity will be either in the commoned style or in the fanned-out style. For the commoned style, the joined *instance_structure_port_bundles* reference *master_structure_port_bundles* whose port size is the same as the size of the flattened *signal_bundle*. The *instance_structure_port_bundles* size is equal to the size of the *master_logical_port_bundle* associated with the *master_structure_port_bundle* referenced by the *instance_structure_port_bundle*. The size of the *instance_structure_port_bundle* must be equal to the size of the joined flattened *signal_bundle*.

A *connectivity_bus* is defined within an *internal_connectivity_view*. Its associated *signal_bundle* and its joined rippers are defined in the containing view.

A *connectivity_bus_slice* is defined within another *connectivity_generic_bus*, and its associated *signal_bundle* is a member of the *signal_bundle* associated with its enclosing structure.

A *connectivity_sub_bus* is defined within another *connectivity_generic_bus* and its associated *signal_bundle* is the same as that of its enclosing bus. Its joined rippers or ports are a subset of the rippers or ports referenced by the enclosing bus.

A partial EXPRESS-G diagram of *connectivity_generic_bus* and its subtypes is available.

5.3.3 Connectivity ripper

A ripper associates a bus/bus-slice/sub-bus structure with other bus/bus-slice/sub-bus structures or with net/sub-net structures. Rippers can be explicitly joined by any number of net and bus structures, defined in the same view.

In the information model, the *connectivity_ripper* entity has two INVERSE attributes which identify the associated net and bus structures. All the related net or bus structures belong to the same view. Each structure is related to other structures by sharing at least one common *signal*.

6 The design hierarchy mechanism

The Core Model contains a design hierarchy mechanism which is similar to the EDIF design hierarchy and includes some of the capabilities given by the CFI DR view selection.

The need for a design hierarchy mechanism derives from the fact that a *cluster* contains one interface but may have multiple views. Therefore, in order to specify an unambiguous tree structure for a given design, it is necessary to be able to specify for each *instance* which view is to be used. Furthermore, it is necessary to be able to specify alternative configurations for *instances* so that different design trees may be represented.

In addition to choosing specific configurations of specific views it may also be necessary to specify that, in a particular configuration, a given *instance* may be a leaf and use none of the views of its cluster. It is also necessary to specify whether the configured *instance* belongs to an *internal_cell* or to an *external_cell*. Finally, not all configuration information may be available at a given moment. Therefore a mechanism for explicitly having unconfigured *instances* is needed.

6.1 The design hierarchy

A *design* is a set of hierarchies of *cell_representations*. Each hierarchy unambiguously describes an electrical circuit. The hierarchy is modelled by the *design_hierarchy* entity. A design hierarchy contains the description (annotation) of the top-level node of the hierarchy and an associated cluster configuration. A *design* may contain many design hierarchies. For each *design_hierarchy*, the chosen *cluster_configuration* is defined in the *cluster_interface* of the top *cell* of the *design*. According to the information contained in the top level, a *design_hierarchy* is subtyped into

- *external_design_hierarchy*. This corresponds to the case where the top level selects an *external_cluster_configuration*, which means that the actual *cell* information is external to the information base. According to the information that is available, the *external_design_hierarchy* is further sub-typed into
 - *expandable_external_design_hierarchy*. This corresponds to the case where an *expandable_external_cluster_configuration* is selected. The expandable external cluster configuration selects an *external_cell_representation* as the top view of the hierarchy. Although the information is external and, therefore, does not allow any further configuration, the top level may, potentially, be further expanded.
 - *leaf_external_design_hierarchy*. This corresponds to the case where the design hierarchy selects a *leaf_external_cluster_configuration*. The leaf external cluster configuration specifies that there is no further expansion of the hierarchy. Therefore, this design hierarchy corresponds to the case where the design consists of only one cell which is described externally.
- *internal_design_hierarchy*. This corresponds to the case where the top level selects an *internal_cluster_configuration*. According to the information contained in the cluster configuration, the design hierarchy may be further subdivided into

- *expandable_internal_design_hierarchy*. This corresponds to the case where it selects an *expandable_internal_cluster_configuration*. The expandable internal cluster configuration selects an *internal_cell_representation* as the top view of the hierarchy. Some of the instances contained in the top view may have associated instance configurations that are contained in the *expandable_internal_cluster_configuration*. Each of these *instance_configurations* selects another *cluster_configuration*, thus elaborating the hierarchy.
- *expandable_internal_connectivity_design_hierarchy*. This corresponds to the case where the design hierarchy selects an *expandable_internal_connectivity_view_configuration* which provides information for the expansion of a hierarchy that starts with an *internal_connectivity_view*.
- *leaf_internal_design_hierarchy*. This corresponds to the case where the design hierarchy selects a *leaf_internal_cluster_configuration*. The leaf internal cluster configuration specifies that there is no further expansion of the hierarchy. Therefore, this design hierarchy corresponds to the case where the *design* consists of only one *cell* which is described in the information base.

A partial EXPRESS-G diagram of *design* is available.

The *design_hierarchy* contains the chosen *cluster_configuration* for the top *cell*. The role of the *cluster_configuration* entity is to configure a *cluster* by providing additional information and to specify how the hierarchy is to be continued. The *cluster_configuration* is subtyped into:

- *external_cluster_configuration*. This is further subtyped into
 - *expandable_external_cluster_configuration*
 - *leaf_external_cluster_configuration*
- *internal_cluster_configuration*. This is further subtyped into
 - *expandable_internal_cluster_configuration*
 - *expandable_internal_connectivity_view_configuration*
 - *leaf_internal_cluster_configuration*

A partial EXPRESS-G diagram of *cluster_configuration* and its subtypes is available.

An expandable cluster configuration identifies a view for the further expansion of the hierarchy. The chosen view belongs to the *cluster* that contains the expandable cluster configuration.

In the case of *expandable_internal_cluster_configuration*, each *instance* defined within the chosen view is either configured or explicitly unconfigured. In order to configure an *instance*, it is necessary to specify an *instance_configuration*. An *instance_configuration* selects a valid *cluster_configuration* for the configured *instance*. The chosen *cluster_configuration* is defined in the instantiated *cluster* of the configured *instance*. To unconfigure an *instance*, it is necessary to state explicitly that the *instance* is required to be unconfigured. Thus, an *expandable_internal_cluster_configuration* may contain several *unconfigured_instances* and several *instance_configurations*. If an instance has an *instance_configuration* which references a leaf configuration, there is no further structure in the design tree below that instance. An *expandable_internal_connectivity_view_configuration* gives information for the expansion of a hierarchy that has a connectivity view *cell_representation* at the top level.

A partial EXPRESS-G diagram of *instance_configuration* and its subtypes is available.

In a given expansion, an *instance* is configured at most once. An expandable cluster configuration may also create a scope for global ports, at the corresponding level in the hierarchy.

6.2 Annotations

The top level occurrence of a *design_hierarchy* is annotated by an *occurrence_hierarchy_annotate* entity. Master ports, interconnects and signals defined within the top view may also be annotated. A *master_port_annotate* attaches or modifies attributes and properties which are associated with a *master_logical_port* occurrence in the top-level occurrence. An *interconnect_annotate* associates physical characteristics with an interconnect structure within an occurrence hierarchy. A *signal_annotate* attaches or modifies properties which are associated with a *signal* occurrence within an occurrence hierarchy.

The *occurrence_hierarchy_annotate* entity is subtyped into

- *external_occurrence_hierarchy_annotate*. This is further subtyped into
 - *expandable_external_occurrence_hierarchy_annotate*.
This corresponds to the case where the annotate belongs to an *expandable_external_design_hierarchy*.
 - *leaf_external_occurrence_hierarchy_annotate*.
This corresponds to the case where the annotate belongs to a *leaf_external_design_hierarchy*.
- *internal_occurrence_hierarchy_annotate*. This is further subtyped into
 - *expandable_internal_occurrence_hierarchy_annotate*.
This corresponds to the case where the annotate belongs to an *expandable_internal_design_hierarchy*.
 - *expandable_internal_connectivity_occurrence_hierarchy_annotate*.
This corresponds to the case where the annotate belongs to an *expandable_internal_connectivity_design_hierarchy*.
 - *leaf_internal_occurrence_hierarchy_annotate*.
This corresponds to the case where the annotate belongs to a *leaf_internal_design_hierarchy*.

A partial EXPRESS-G diagram of *occurrence_hierarchy_annotates* and its subtypes is available. An *expandable_internal_design_hierarchy* is the only truly expandable hierarchy since it contains information that is locally defined. However, only the *expandable_internal_connectivity_design_hierarchy* can be annotated since the type of the annotated view is known. Therefore, its associated *expandable_internal_connectivity_occurrence_hierarchy_annotate* may contain *occurrence_annotates*. They are used to provide annotation information for all the nodes in a design hierarchy which are not top level nodes.

The *occurrence_annotate* entity is subtyped into

- *external_occurrence_annotate*. This is further subtyped into
 - *expandable_external_occurrence_annotate*
 - *leaf_external_occurrence_annotate*.
- *internal_occurrence_annotate*. This is further subtyped into
 - *expandable_internal_occurrence_annotate*
 - *expandable_internal_connectivity_occurrence_annotate*
 - *leaf_internal_occurrence_annotate*.

Partial EXPRESS-G diagrams of *occurrence_annotate* and its subtypes are available.

The *occurrence_annotate* selects the annotated occurrence by referencing an *instance_configuration* which specifies the annotated instance and the chosen *cluster_configuration* for that annotated instance. Instance ports on the annotated instance may be annotated by *instance_port_annotates*. An *instance_port_annotate* is used to attach or modify properties and attributes which are associated with an *instance_structure_port* occurrence within an occurrence hierarchy. In an *expandable_internal_occurrence_annotate* interconnects and signals defined within the chosen view may be annotated by *interconnect_annotates* and *signal_annotates* respectively.

Any leaf level occurrence of a *design_hierarchy* is annotated by a leaf occurrence annotate. A leaf level occurrence is selected by referencing an *instance_configuration* which points to a leaf cluster configuration.

7 Core Model for electronic design

The Core Model for electronic design describes the concepts which are necessary for the definition of bit-level electronic designs. These concepts include

- libraries
- re-usable design units (cells)
- cell definition hierarchies
- connectivity
- designs

7.1 Libraries

Within the Core Model, the units of designs, called cells, are grouped into libraries. All of the cells in a library are considered by the originating system to be related in some way. There are no pre-defined semantics to such groupings within the Core Model. An information base may contain one or more libraries.

Where a library, and the cells within it, are fully specified within the information base, the library and its contained objects are said to be “internal”.

Alternatively, if the contents of the library are known to all systems accessing the information base, then only a minimal amount of information needs to be specified in the information base. Such a library and its contained objects are termed “external”. The external library specifications within the information base can be considered as stubs or handles for the actual library which exists outside the information base.

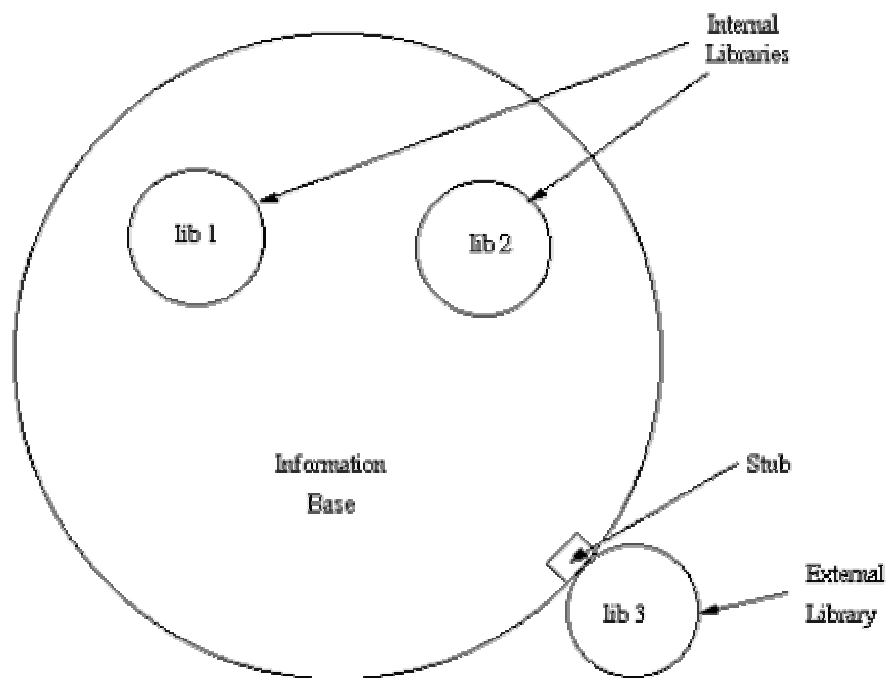


Figure 9 – Internal and external libraries

In the case of an external library, the only information which is specified about the library and its contents is that necessary to form relationships to the objects within it. For example, it is necessary to know the ports of objects within the external library so that connections can be made to those objects. However, details about the connectivity within the library's objects are omitted since they are assumed to be specified in the outside library.

7.2 Interfacing to cells

Within design systems, modules or gates can be connected together by specifying connections between their ports or pins. In addition, it is possible for different descriptions of the same gate to have different sets of pins. For example, a description of a two-input NAND gate could have three ports (two input and one output), or five (two input, one output, power and ground).

The concept of a port is considered to be split into master logical and master structure ports. A master structure port makes a master logical port available for use within structural connectivity.

Within the Core Model, the manner in which a cell can interface to other cells is specified by a set of master logical ports. Each master logical port can carry a single-bit of data. These master logical ports are grouped into a cluster interface. Different descriptions, termed *cell_representations*, of the cell which share the same set of master logical ports may be grouped into clusters.

The single-bit master logical ports can be grouped into master logical port bundles. A master logical port bundle references the list of master logical ports and/or other master logical port bundles which are within it. These master logical port bundles may be made available for wide-connections via a referencing master structure port bundle.

For example, the inputs to an adder could be considered as a single master structure port bundle as shown below. This master structure port bundle makes available the master logical port bundle which consists of a single carry in bit and two eight-bit wide master logical port bundles representing the numbers to be added. Each of the eight-bit master logical port bundles consists of eight single-bit master logical ports.

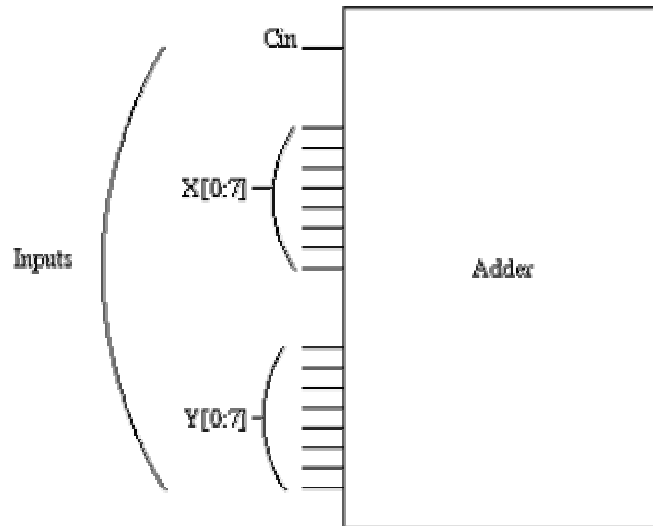


Figure 10 – Port bundling – 1

As may be expected from the above example, a master logical port may only be a member once of a given bundle.

For a given design unit, there may be several ways in which the same master logical ports may be grouped for different purposes. For example, it may be reasonable to group the two bit[0]s of the input numbers, the two bit[1]s and so on.

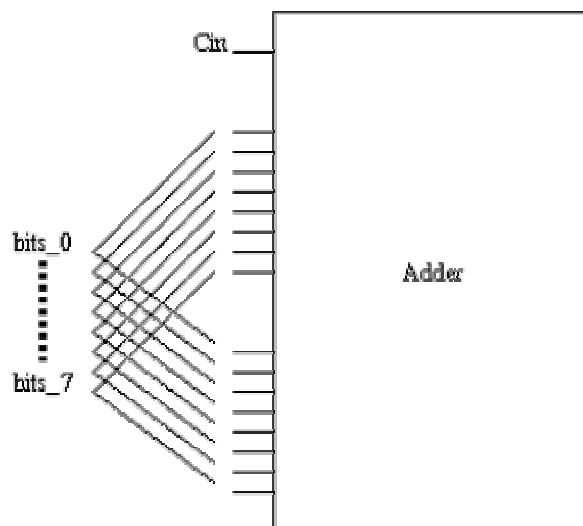


Figure 11 – Port bundling – 2

In order to support this alternative, there would be eight master structure port bundles each corresponding to two master logical ports.

The two groupings of ports described above are mutually exclusive i.e. they cannot be used for the same cell at the same time. For this reason, each set of master structure ports and master structure port bundle is specified within a different port structure. So the two example groupings of ports would be specified in two separate port structures. A port structure may additionally specify the order in which the originating system considers the ports to be structured.

7.3 Cell definition hierarchies

A cell can specify different possible realizations or implementations of its logic. In the current version of the Core Model, the only possible realization of an internal cell is as a connectivity view.

7.4 Instantiation

An internal connectivity view can be made up of sub-units. Within the Core Model, these sub-units are specified as instances of other cells. An instance may be viewed as an indication that one or more occurrences of the instantiated cell will be included in the overall design for each instance within an occurrence of the connectivity view. The number of occurrences of the instantiated cell is specified by the instance width.

For example, in the figure below, the definition of cell C includes a three-wide instance of cell A. Since cell C is instantiated by a five-wide instance (i3) within cell D, the top-level design, within the design there are 21 occurrences of cell A. Similarly, there are seven occurrences of cell B.

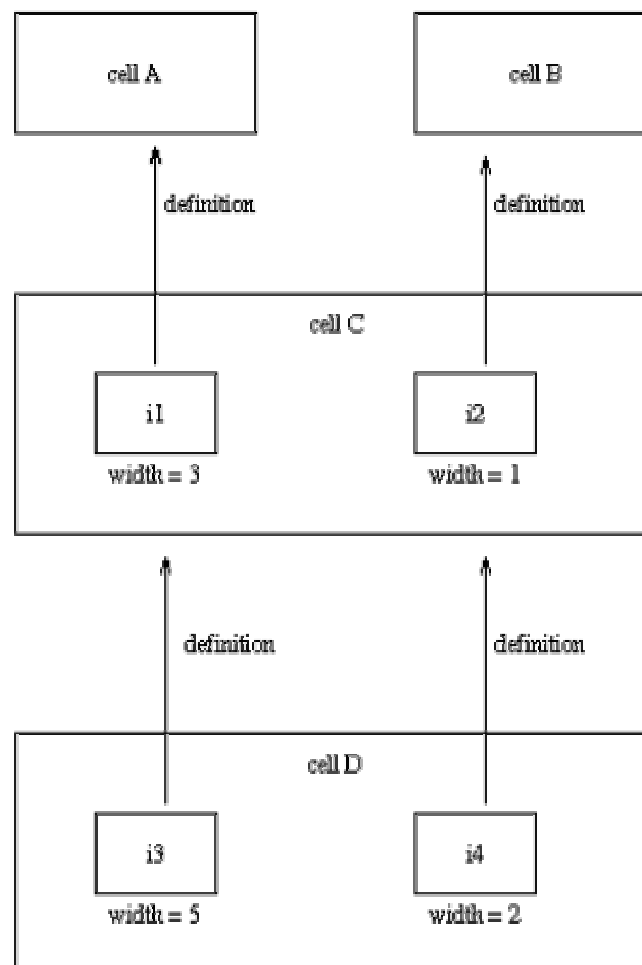


Figure 12 – Instantiation

In order for an instance to be sensibly used within a connectivity view, it is necessary to choose a particular logical interface of the instantiated cell, i.e. the instance identifies a particular cluster of the instantiated cell.

Attributes of the instantiated cluster and its logical ports may be overridden or added to within the instance. These changes only apply to the instance and corresponding occurrences i.e. they alter the copies of the instantiated cell not the instantiated cell itself.

7.5 Logical connectivity

Within a connectivity view, the bit-wide logical interconnection of the logical ports of the view and of any instances within it, is specified by signals. A signal within the Core Model just indicates a bit-wide connection. In particular, the scope of this Core Model definition does not

- indicate any value changes on the connection;
- allow the specification of type information about the data carried on the signal.

A signal may connect logical ports of the connectivity view. These are termed master logical ports. In strict terms, a cluster interface specifies master logical port(bundle)s and master structure port(bundle)s. A signal may also connect up to logical ports of instances within the connectivity view. Since an instance may represent more than one occurrence, it is necessary for the signal to specify which basic member of the instance the port interfaces to. This complex reference to a logical port of a member of the instance is termed an instance member logical port.

Signals may be grouped into signal bundles, in a similar manner to that described above for logical ports and logical port bundles. The instances, signals and signal bundles together specify the logical connectivity of the connectivity view.

7.6 Structural connectivity

The logical connectivity is sufficient to describe the connections between the master logical ports of the connectivity view and the instance member logical ports of the instances. However, sometimes systems describe details about the manner in which the logical signal should be realized. Such implementation information includes the specification that a given signal is realized by several sub-connections with different properties as shown below. The structured realization of a signal is termed a net.

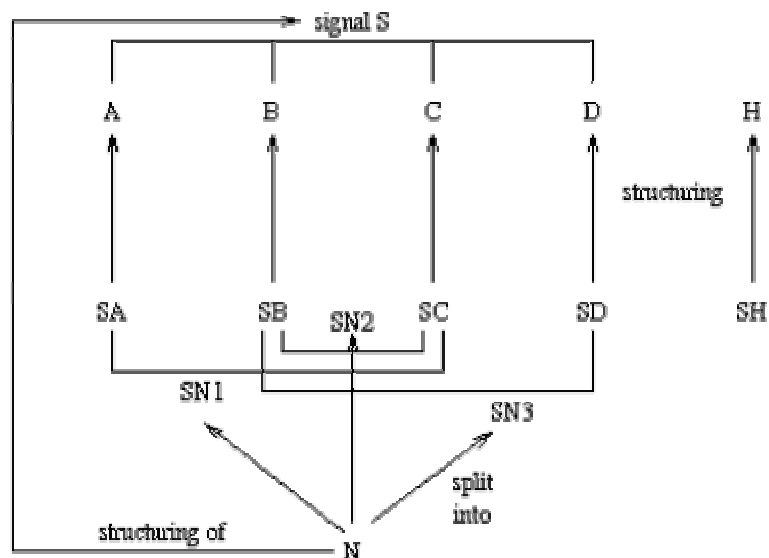


Figure 13 – Connectivity net

In the above figure, the logical signal S connecting together the logical port set [A, B, C, D] is realized by the net N which is split into three sub-nets, SN1, SN2 and SN3 connecting the structure ports {SA, SC}, {SC, SB} and {SB, SD} respectively.

As may be expected, there are constraints to ensure that the structure ports connected by N structure a subset of the logical ports joined by the signal S, i.e. N could not join to SH.

Groups of signals may also be realized by a single object termed a connectivity bus. Since the connectivity bus is wide, it connects together wide structured objects, i.e. master structure port bundles and instance structure port bundles. In addition, there is the capability within the Core Model for a connectivity view to specify *ad hoc* groupings of master ports, termed local master port bundles, which can also be connected by buses.

The simplest case of bus connectivity is where the joined objects are of the same size as the bus as shown below.

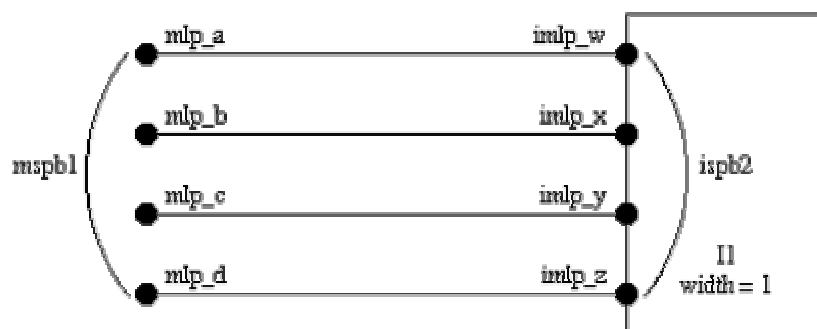


Figure 14 – Connectivity bus

The master structure port bundle, mspb1 is joined by the bus to the instance structure port bundle of instance I1. In order for this to be a valid bus, master logical port mlp_a must be connected by a signal to instance member logical port imlp_w, mlp_b to imlp_x etc. It should be emphasized that structural connectivity never establishes “connectedness”; it structures the logical connectivity specified by signals.

When a bus connects to an instance structure port bundle, there are two ways in which the signals of the signal bundle which the bus realizes may connect up to the individual ports of the instance structure port bundle. The size of the port bundle may equal that of the bus, in which case the first element of the port bundles are connected together as shown below. This type of connection is termed *commoning* because the corresponding members of the port bundles of the instance members are connected.

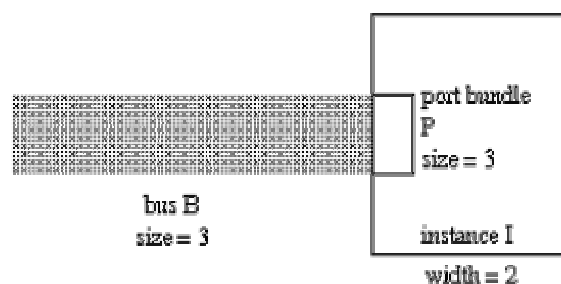


Figure 15 – Connectivity bus – Commoning

In the above example, the first signal associated with bus B is connected to the first element of port bundle P of each member of instance I, i.e. since the width of instance I is 2, the signal is connected to 2 instance member logical ports as shown below.

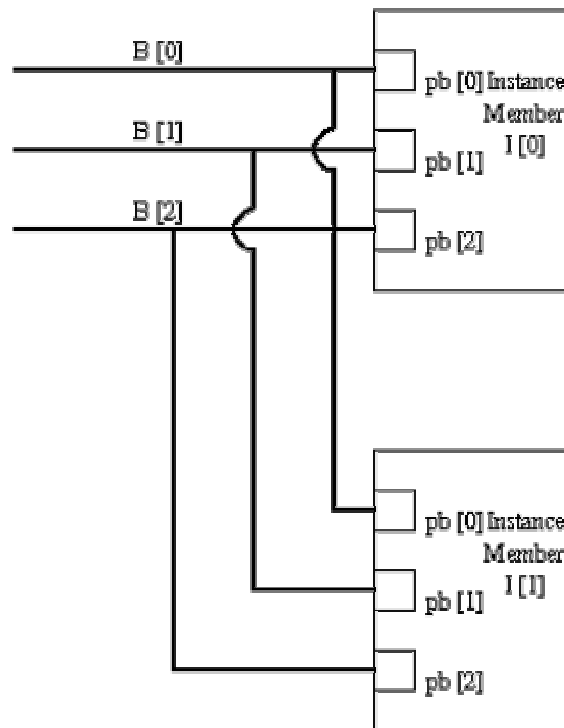


Figure 16 – Connectivity bus – Logical equivalent of commoning

Alternatively, the port bundle may be smaller than the bus and all of the members of the port bundle of all of the members of the instance may be needed to connect up to the bus as shown below. This type of connection is termed fanning out as the bus may be considered to fan out to the members of the instances.

A special case of fanning-out may occur where the bus is connected up to a single port of a wide instance. Each port of the members of the instance is then connected up to a different signal of the bus as shown below.

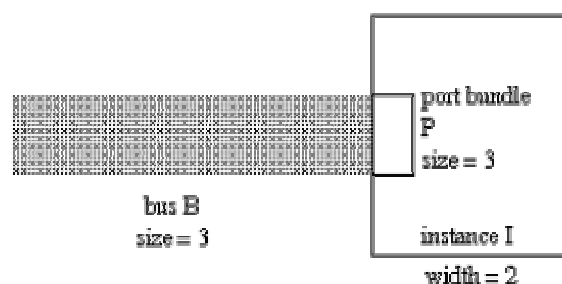


Figure 17 – Connectivity bus – Fanning-out

In the above example, the first signal associated with bus B is connected to an instance member logical port corresponding to port P of the first member of instance I, i.e. since the width of the instance is 3, there are three signals associated with bus B as shown below

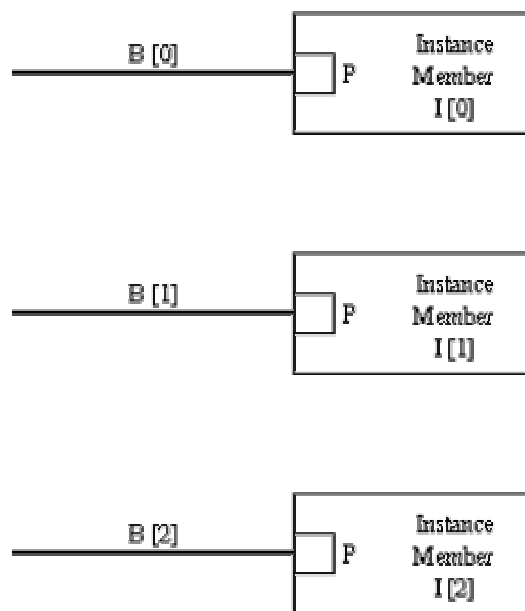


Figure18 – Connectivity bus – Logical equivalent of fanning-out

There is a degenerate case of a connectivity bus which structures a signal bundle which contains only one signal i.e. the connectivity bus is one-bit wide. In this case, it can join instance member structure ports and master structure ports.

Since a connectivity bus exists to specify a structured realization of a signal bundle, its sub-structure can be specified by connectivity sub-buses and/or connectivity bus-slices.

A connectivity sub-bus is similar in concept to a connectivity sub-net and may be used to specify details such as criticality, or merely to indicate that the originating system considers the connectivity bus to be broken down in this manner.

In the case of a connectivity sub-bus, the connectivity which is realized by the connectivity sub-bus does not state all of the structure ports joined. Conversely a connectivity bus-slice considers a sub-set of the sets of structure ports joined by the overall connectivity bus. An example of this is shown below.

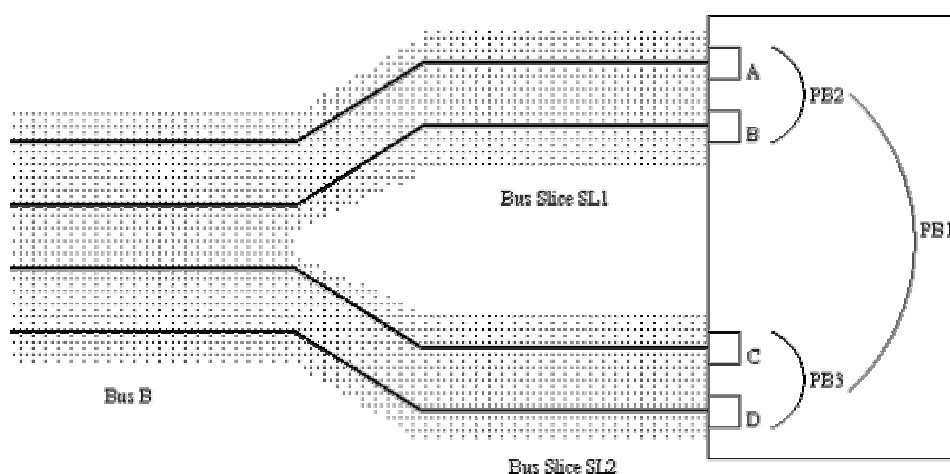


Figure 19 – Connectivity bus-slice

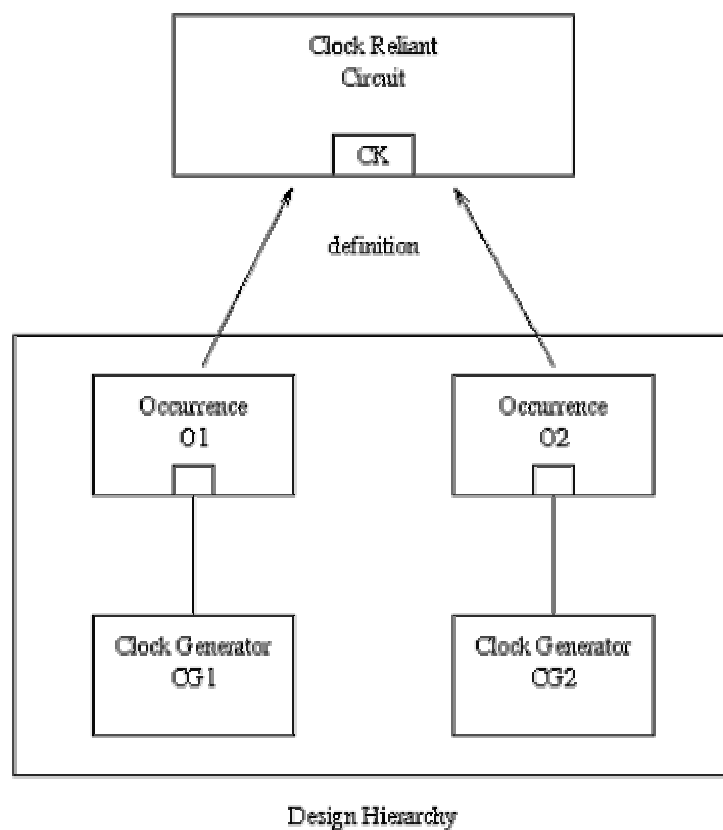


Figure 21 – Global port scoping

O1 and O2 are occurrences of the same cluster which specifies circuitry which relies upon the presence of a clock signal. For example, the port CK may be explicitly connected to the global port clock. Global port scoping allows O1 and O2 to have their global port clock scoped given that the requirement for CK to be connected to a clock is satisfied by the connections to the two separate clock generators.

A global port will commonly be scoped because a master logical port of the current level of the design hierarchy will realize the necessary connection, for example, by feeding in a clock signal as shown above. This is not necessarily the case, however; for example, a set of modules in the subset of the hierarchy may read and write data onto a common bus without that data being required outside the subset of the hierarchy.

7.8 Design and configuration

The libraries specify sets of cells. In many cases, however, it is necessary to identify the top level cell of a design. Since a cell may interact with the outside world in different ways, as specified by the available interfaces, there may be several different manners in which the design may be considered; these are termed design hierarchies.

Different design hierarchies may also be specified according to the way in which the instances within the cell are configured.

A cluster may have several cell representations which interact in the same manner (as specified by the cluster interface). As a result, the instantiation of a cluster does not determine the exact cell representation which will occur within the design hierarchy. As a result, an instance may be specified as being

- Unconfigured
The system has not determined whether or not the instance will be associated with a particular cell representation.
- A leaf
No corresponding cell representation should be used to expand the hierarchy.
- Expandable
A particular cell representation is to further expand the hierarchy.

There are two different ways in which instance configuration could have been specified.

1. By defining the configuration for each occurrence of an instance within the design hierarchy.
2. By the definition within a cluster of the configurations which the system considers to be reasonable. A design hierarchy then identifies the chosen configuration of a cluster of the top-level design cell.

Within the Core Model, the second mechanism has been chosen.

7.9 Annotation

In the context of a design hierarchy, new information or overriding values for previously defined values may be specified for parts of the design hierarchy.

Within any occurrence, properties and attributes of the cell, cluster and its ports may be added or overridden.

Within an occurrence which has been configured to a specific cell representation properties of the cell representation may be overridden.

Within an occurrence which has been configured to an internal connectivity view, properties and attributes of the signals, nets and buses, and sub-occurrences, i.e. instance members, may be overridden.

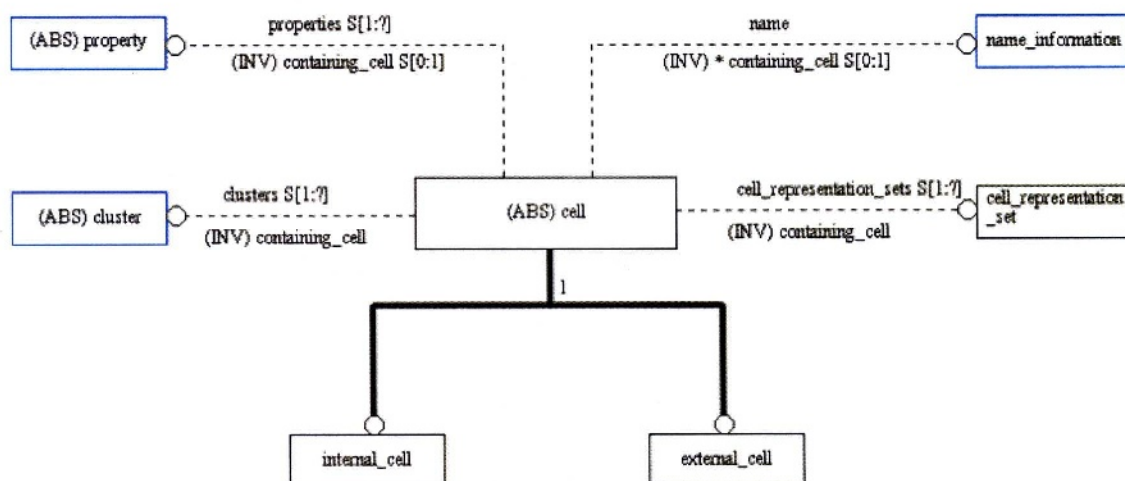
8 Core Model EXPRESS-G

This section contains partial EXPRESS-G of some of the key objects of this Core Model of the electronics domain.

8.1 Partial EXPRESS-G of cell

The following EXPRESS-G diagram is incomplete and does not necessarily include all the relationships of cell. It is included for informative purposes only. In case of any discrepancy with the EXPRESS description, the EXPRESS shall take precedence.

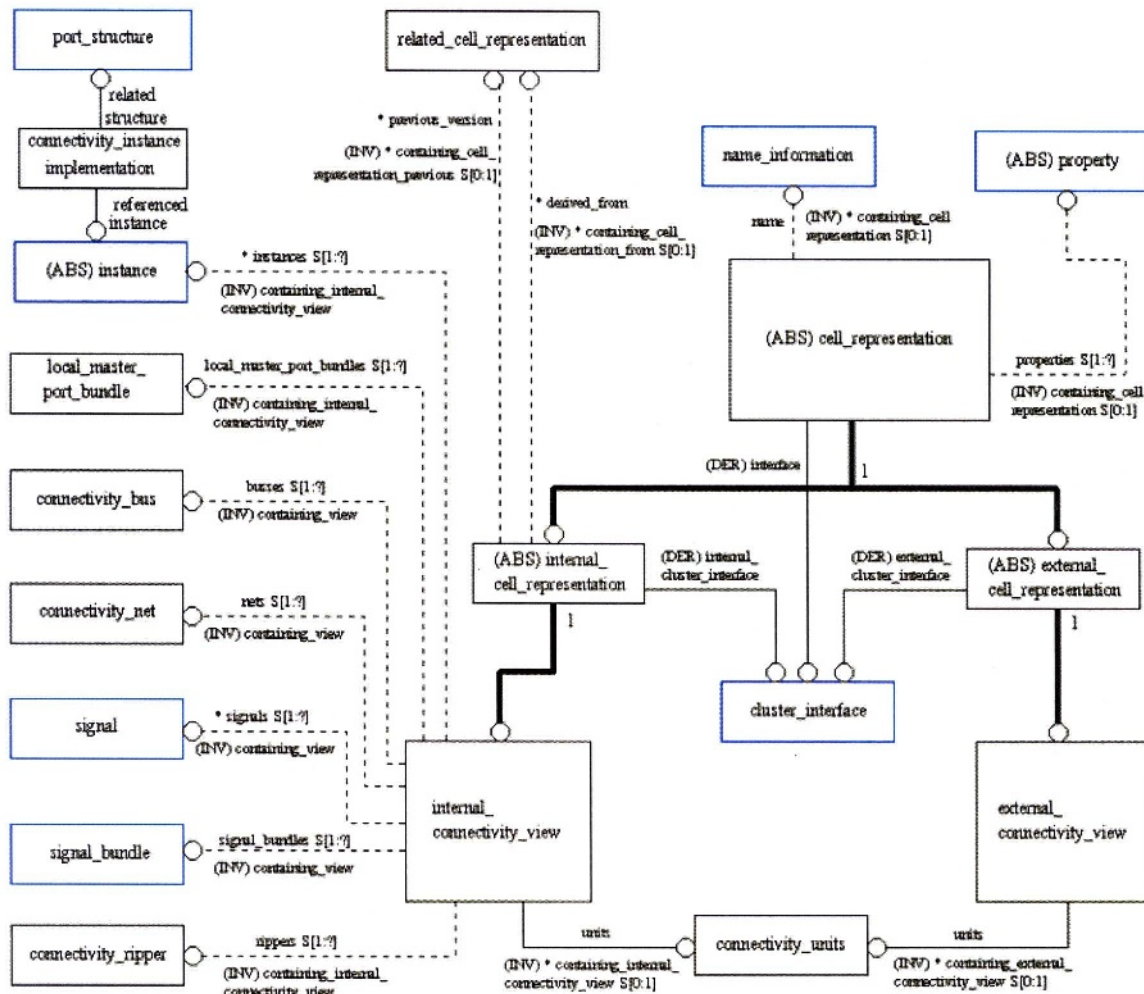
The blue boxes are linked to further EXPRESS-G diagrams. The black boxes are, apart from simple EXPRESS types, linked to the EXPRESS description of the object.



8.2 Partial EXPRESS-G of cell_representation

The following EXPRESS-G diagram is incomplete and does not necessarily include all the relationships of cell_representation. It is included for informative purposes only. In case of any discrepancy with the EXPRESS description, the EXPRESS shall take precedence.

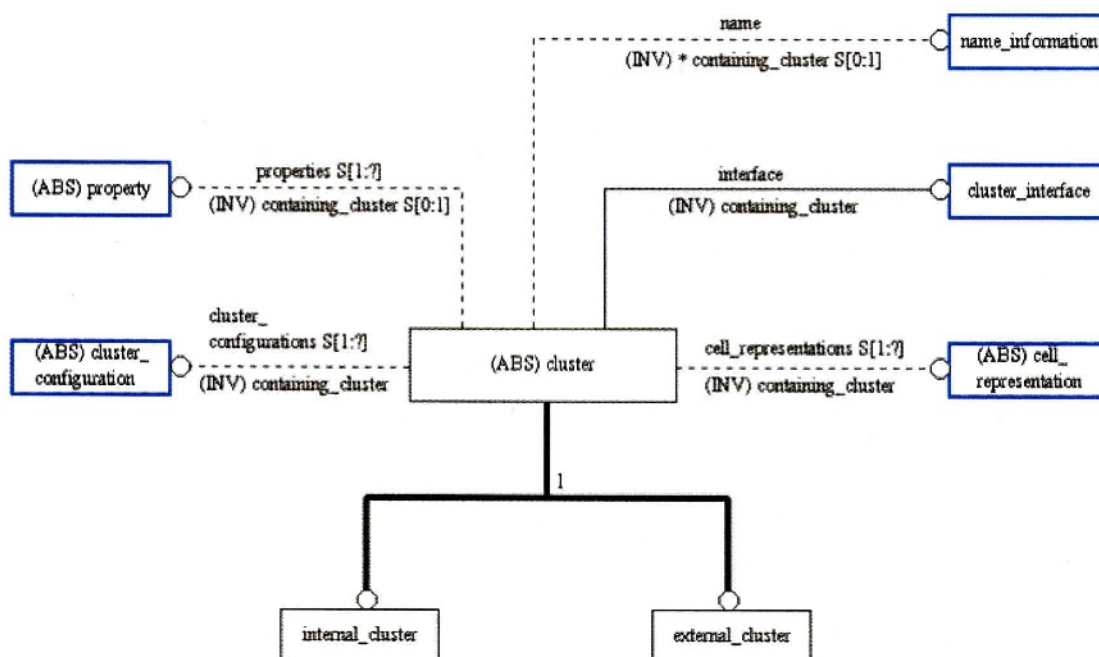
The blue boxes are linked to further EXPRESS-G diagrams. The black boxes are, apart from simple EXPRESS types, linked to the EXPRESS description of the object.



8.3 Partial EXPRESS-G of cluster

The following EXPRESS-G diagram is incomplete and does not, necessarily, include all the relationships of cluster. It is included for informative purposes only. In case of any discrepancy with the EXPRESS description, the EXPRESS shall take precedence.

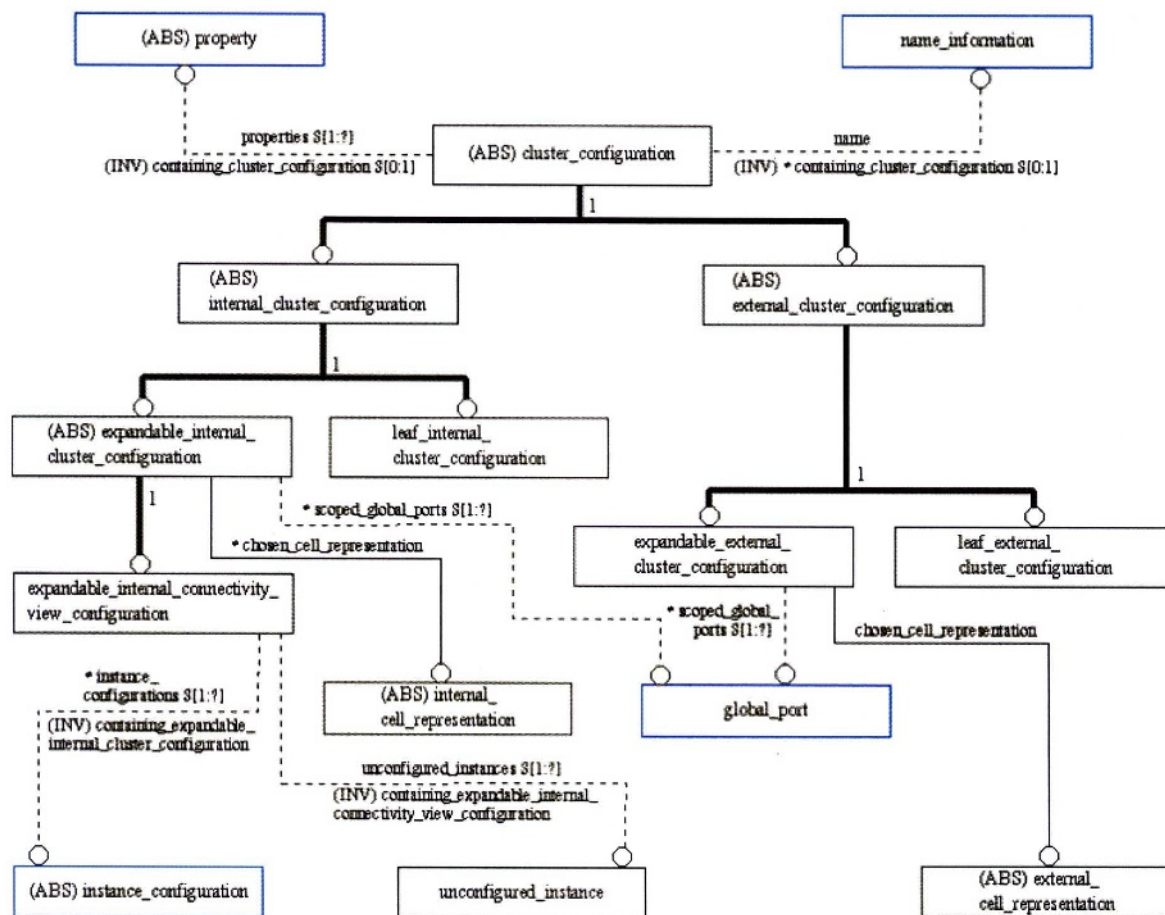
The blue boxes are linked to further EXPRESS-G diagrams. The black boxes are, apart from simple EXPRESS types, linked to the EXPRESS description of the object.



8.4 Partial EXPRESS-G of cluster_configuration

The following EXPRESS-G diagram is incomplete and does not necessarily include all the relationships of cluster_configuration. It is included for informative purposes only. In case of any discrepancy with the EXPRESS description, the EXPRESS shall take precedence.

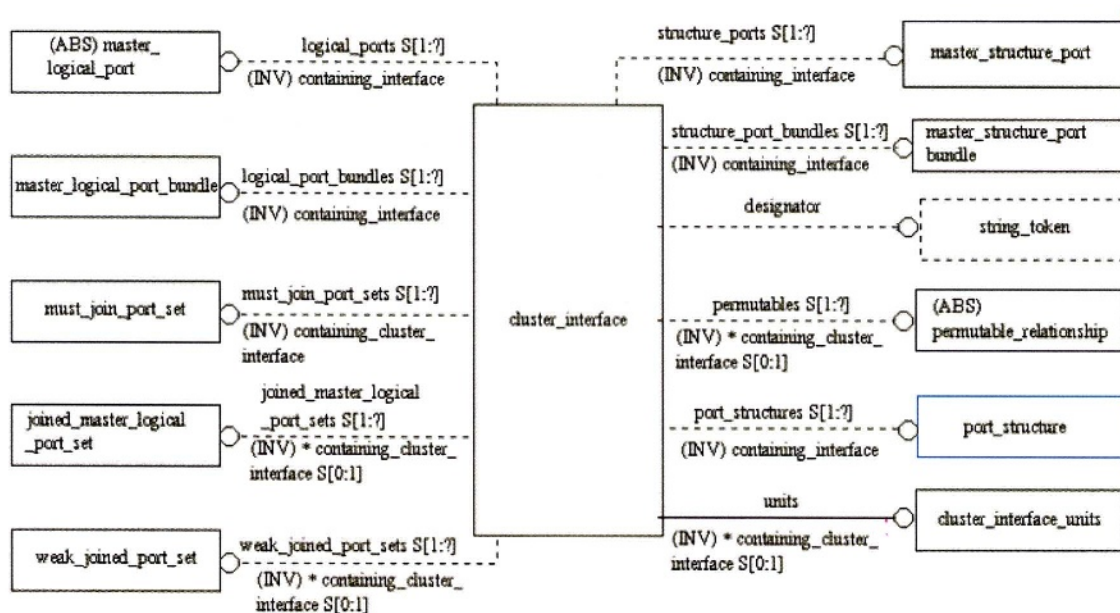
The blue boxes are linked to further EXPRESS-G diagrams. The black boxes are, apart from simple EXPRESS types, linked to the EXPRESS description of the object.



8.5 Partial EXPRESS-G of cluster_interface

The following EXPRESS-G diagram is incomplete and does not necessarily include all the relationships of cluster_interface. It is included for informative purposes only. In case of any discrepancy with the EXPRESS description, the EXPRESS shall take precedence.

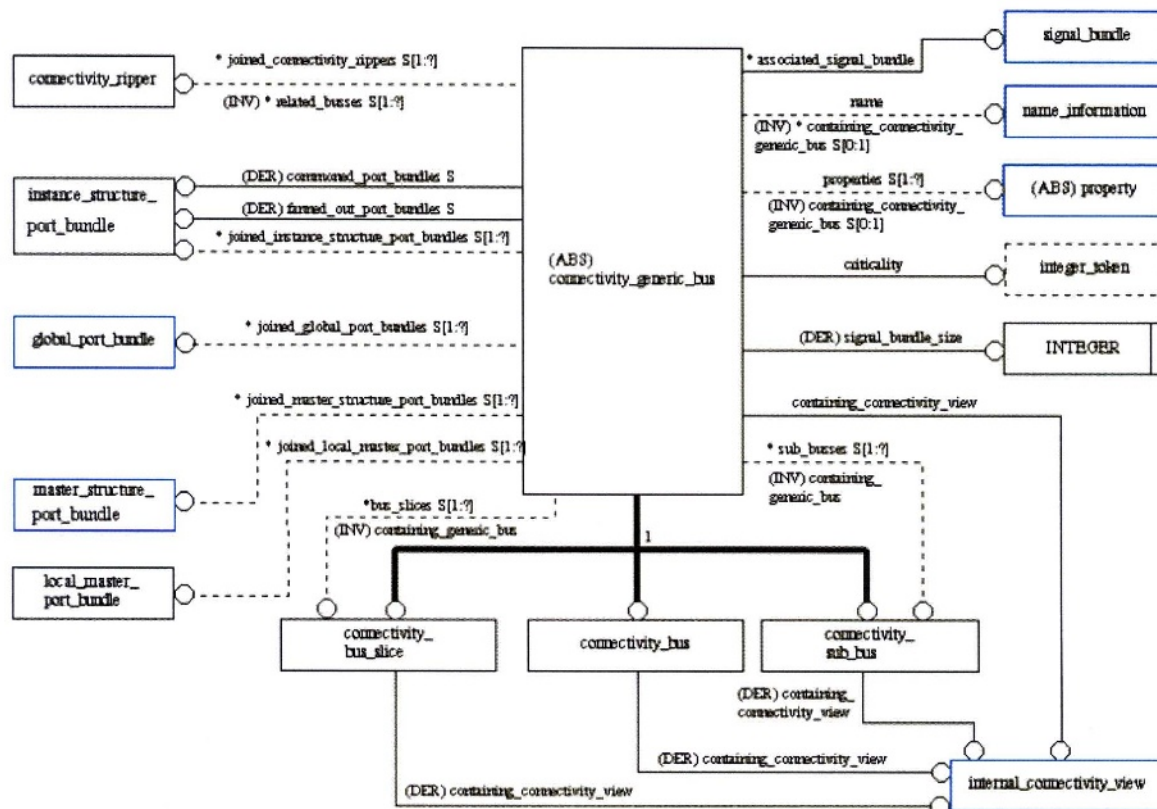
The blue boxes are linked to further EXPRESS-G diagrams. The black boxes are, apart from simple EXPRESS types, linked to the EXPRESS description of the object.



8.6 Partial EXPRESS-G of connectivity_generic_bus

The following EXPRESS-G diagram is incomplete and does not necessarily include all the relationships of connectivity_generic_bus. It is included for informative purposes only. In case of any discrepancy with the EXPRESS description, the EXPRESS shall take precedence.

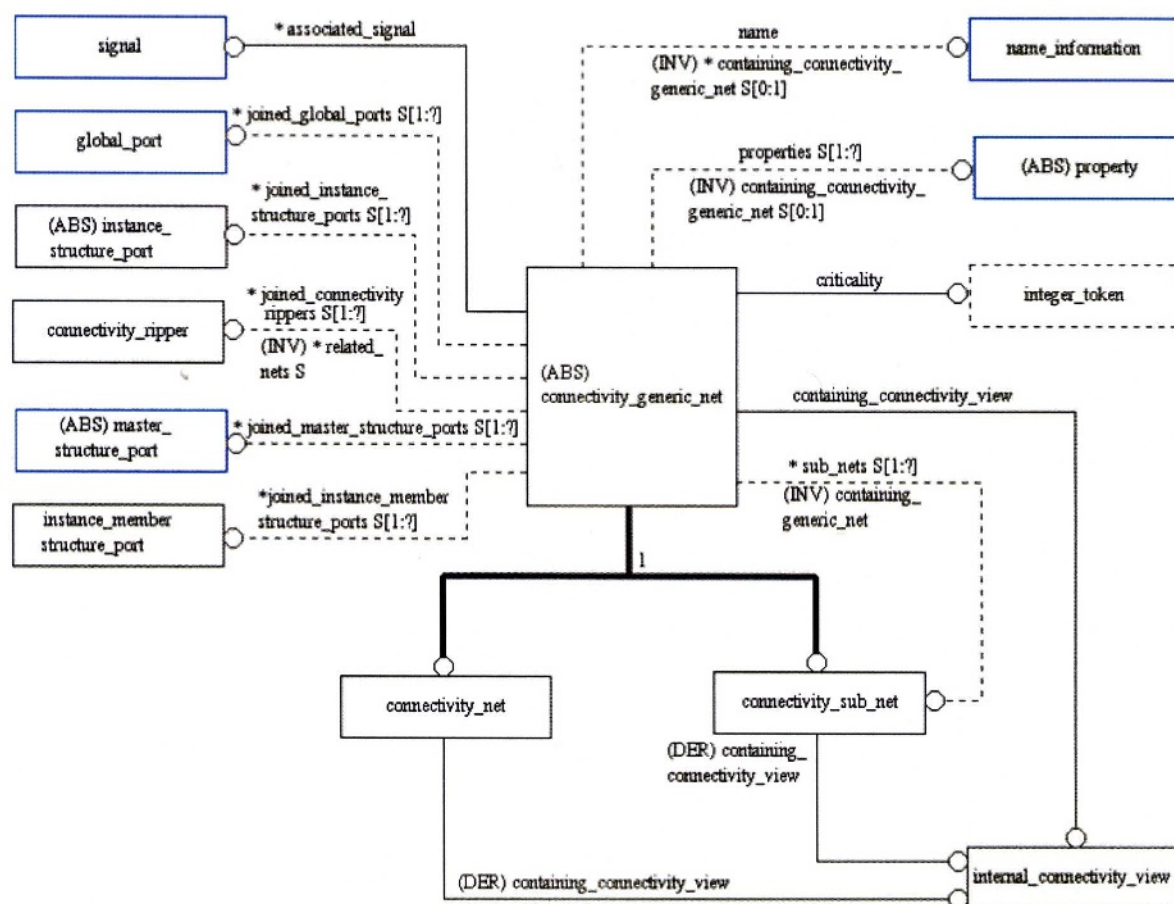
The blue boxes are linked to further EXPRESS-G diagrams. The black boxes are, apart from simple EXPRESS types, linked to the EXPRESS description of the object.



8.7 Partial EXPRESS-G of connectivity_generic_net

The following EXPRESS-G diagram is incomplete and does not necessarily include all the relationships of connectivity_generic_net. It is included for informative purposes only. In case of any discrepancy with the EXPRESS description, the EXPRESS shall take precedence.

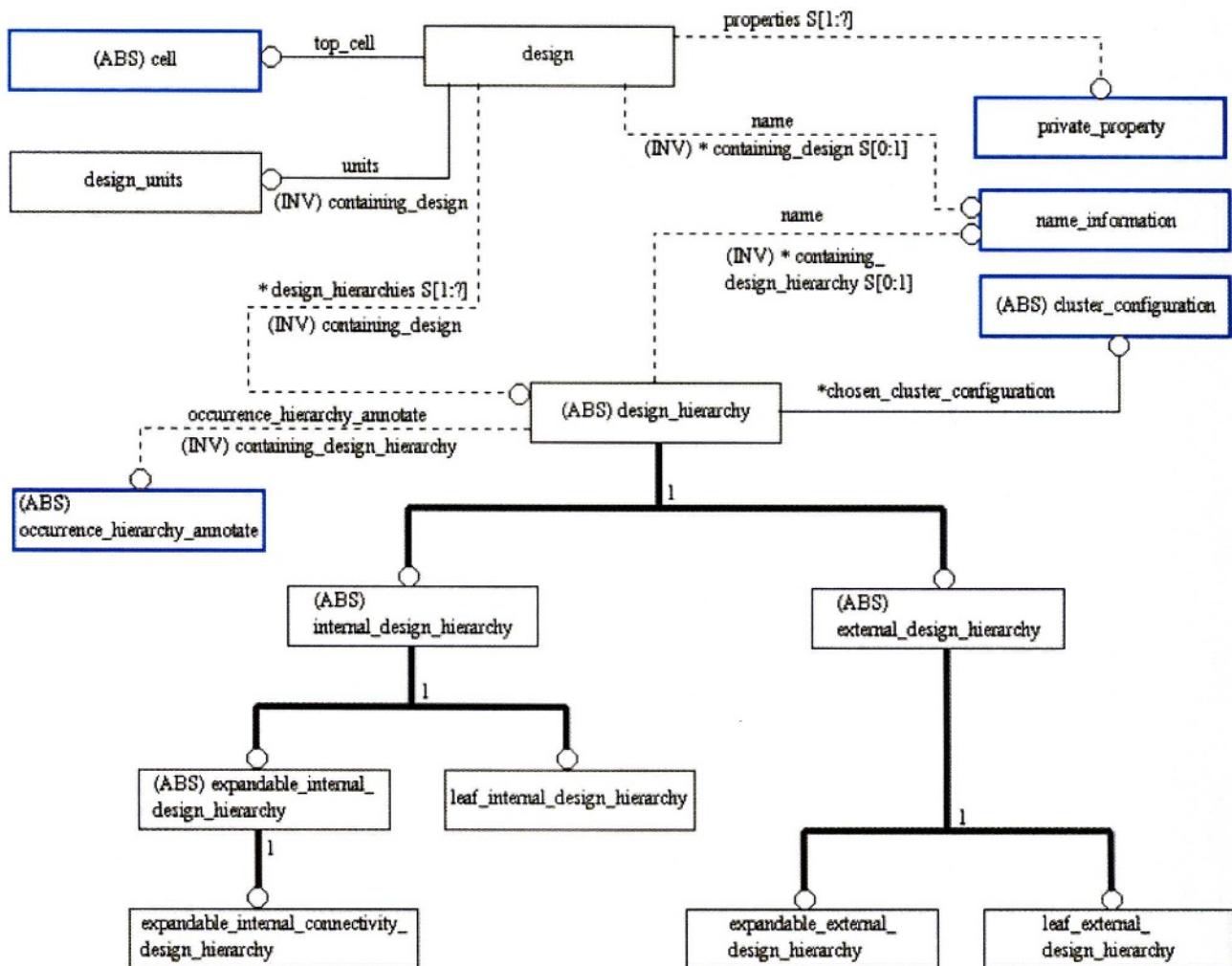
The blue boxes are linked to further EXPRESS-G diagrams. The black boxes are, apart from simple EXPRESS types, linked to the EXPRESS description of the object.



8.8 Partial EXPRESS-G of design

The following EXPRESS-G diagram is incomplete and does not necessarily include all the relationships of design. It is included for informative purposes only. In case of any discrepancy with the EXPRESS description, the EXPRESS shall take precedence.

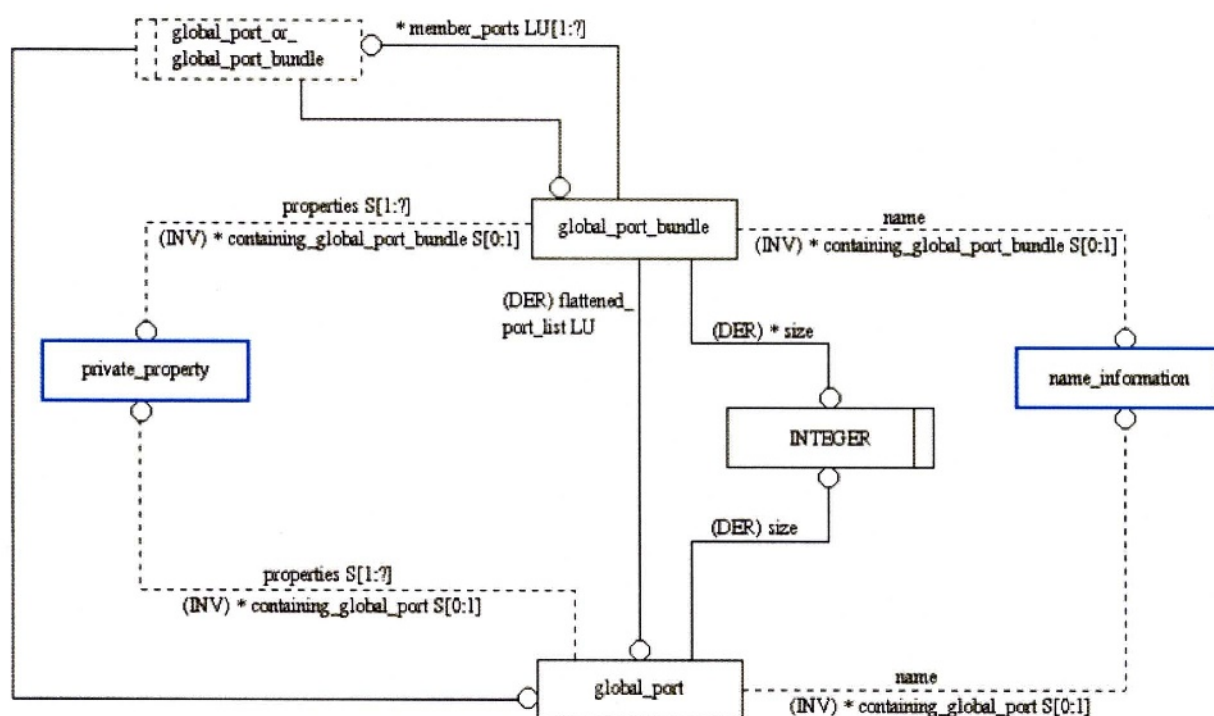
The blue boxes are linked to further EXPRESS-G diagrams. The black boxes are, apart from simple EXPRESS types, linked to the EXPRESS description of the object.



8.9 Partial EXPRESS-G of global_port

The following EXPRESS-G diagram is incomplete and does not necessarily include all the relationships of global_port. It is included for informative purposes only. In case of any discrepancy with the EXPRESS description, the EXPRESS shall take precedence.

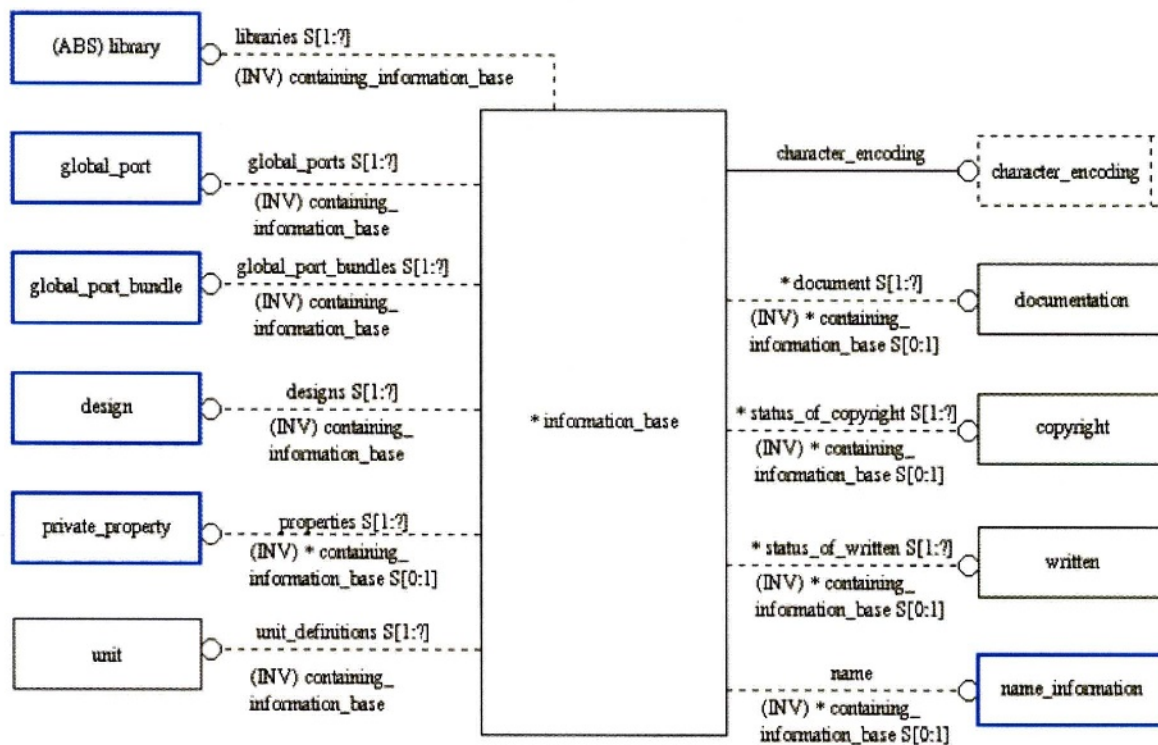
The blue boxes are linked to further EXPRESS-G diagrams. The black boxes are, apart from simple EXPRESS types, linked to the EXPRESS description of the object.



8.10 Partial EXPRESS-G of information_base

The following EXPRESS-G diagram is incomplete and does not necessarily include all the relationships of information_base. It is included for informative purposes only. In case of any discrepancy with the EXPRESS description, the EXPRESS shall take precedence.

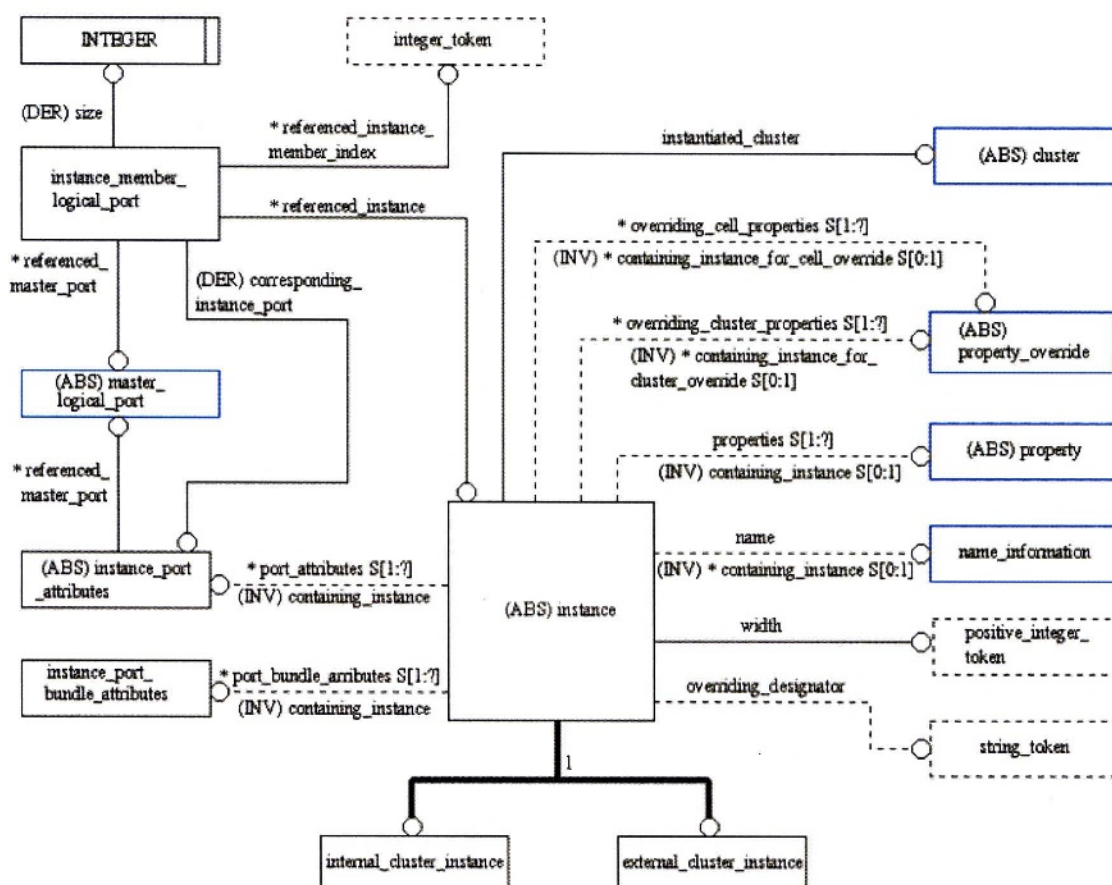
The blue boxes are linked to further EXPRESS-G diagrams. The black boxes are, apart from simple EXPRESS types, linked to the EXPRESS description of the object.



8.11 Partial EXPRESS-G of instance

The following EXPRESS-G diagram is incomplete and does not necessarily include all the relationships of instance. It is included for informative purposes only. In case of any discrepancy with the EXPRESS description, the EXPRESS shall take precedence.

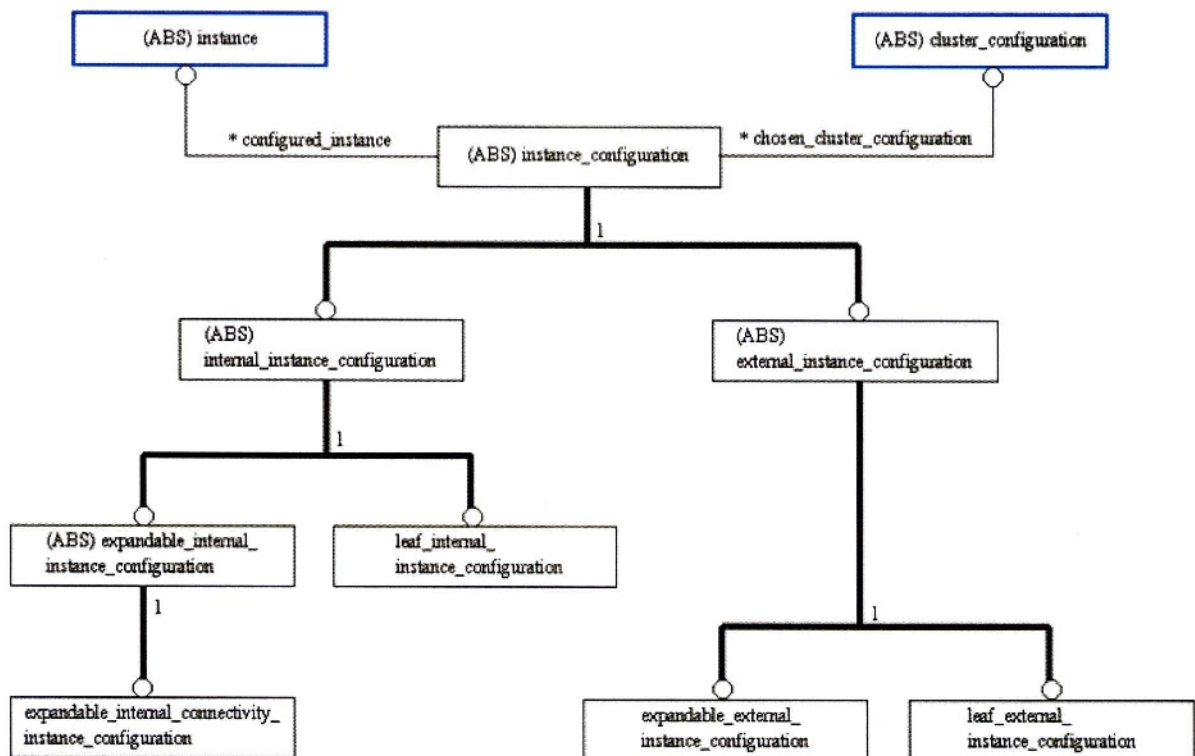
The blue boxes are linked to further EXPRESS-G diagrams. The black boxes are, apart from simple EXPRESS types, linked to the EXPRESS description of the object.



8.12 Partial EXPRESS-G of instance_configuration

The following EXPRESS-G diagram is incomplete and does not necessarily include all the relationships of instance_configuration. It is included for informative purposes only. In case of any discrepancy with the EXPRESS description, the EXPRESS shall take precedence.

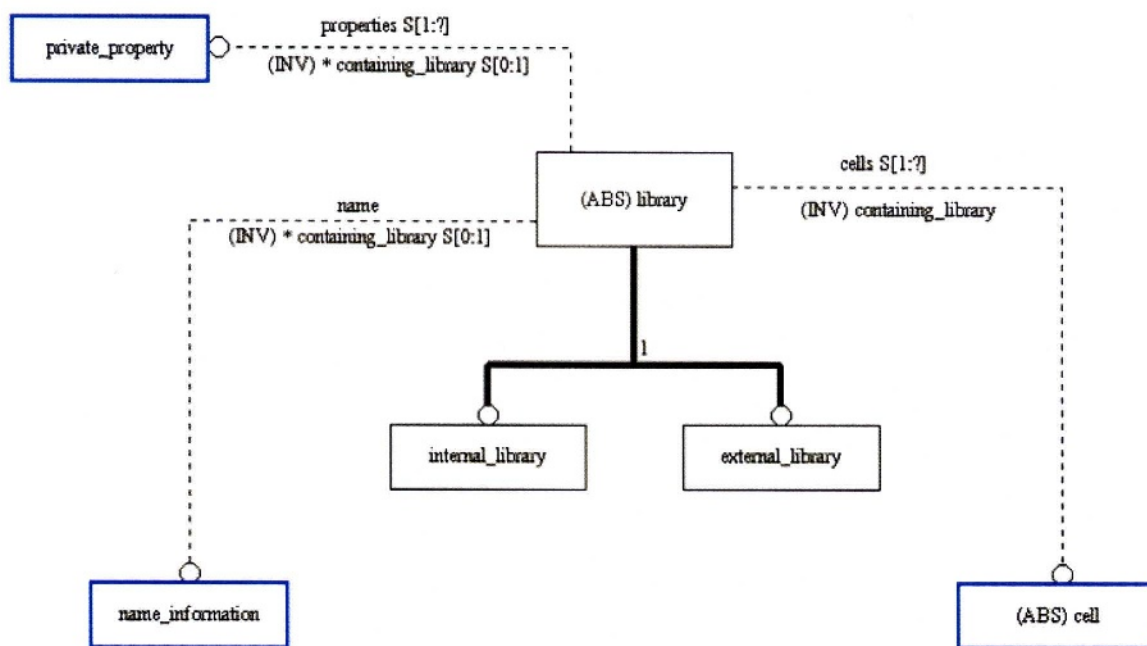
The blue boxes are linked to further EXPRESS-G diagrams. The black boxes are, apart from simple EXPRESS types, linked to the EXPRESS description of the object.



8.13 Partial EXPRESS-G of library

The following EXPRESS-G diagram is incomplete and does not necessarily include all the relationships of library. It is included for informative purposes only. In case of any discrepancy with the EXPRESS description, the EXPRESS shall take precedence.

The blue boxes are linked to further EXPRESS-G diagrams. The black boxes are, apart from simple EXPRESS types, linked to the EXPRESS description of the object.



8.14 Partial EXPRESS-G of master_port_annotate

The following EXPRESS-G diagram is incomplete and does not necessarily include all the relationships of master_port_annotate. It is included for informative purposes only. In case of any discrepancy with the EXPRESS description, the EXPRESS shall take precedence.

The blue boxes are linked to further EXPRESS-G diagrams. The black boxes are, apart from simple EXPRESS types, linked to the EXPRESS description of the object.

