

# **Virtuoso EDIF 200 Reader and Writer User Guide**

**Product Version ICADVM20.1  
October 2020**

© 2020 Cadence Design Systems, Inc. All rights reserved.  
Printed in the United States of America.

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Open SystemC, Open SystemC Initiative, OSCI, SystemC, and SystemC Initiative are trademarks or registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission.

**Trademarks:** Trademarks and service marks of Cadence Design Systems, Inc. contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 800.862.4522.

All other trademarks are the property of their respective holders.

**Restricted Permission:** This publication is protected by copyright law and international treaties and contains trade secrets and proprietary information owned by Cadence. Unauthorized reproduction or distribution of this publication, or any portion of it, may result in civil and criminal penalties. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. Unless otherwise agreed to by Cadence in writing, this statement grants Cadence customers permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used only in accordance with a written agreement between Cadence and its customer.
2. The publication may not be modified in any way.
3. Any authorized copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement.
4. The information contained in this document cannot be used in the development of like products or software, whether for internal or external use, and shall not be used for the benefit of any other party, whether or not for consideration.

**Patents:** The Cadence Products covered in this manual are protected by U.S. Patents 5,790,436; 5,812,431; 5,859,785; 5,949,992; 6,493,849; 6,278,964; 6,300,765; 6,304,097; 6,414,498; 6,560,755; 6,618,837; 6,693,439; 6,826,736; 6,851,097; 6,711,725; 6,832,358; 6,874,133; 6,918,102; 6,954,908; 6,957,400; 7,003,745; 7,003,749.

**Disclaimer:** Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

**Restricted Rights:** Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor

---

# Contents

---

<u>Preface</u>	5
<u>Scope</u>	5
<u>Licensing Requirements</u>	6
<u>Related Documentation</u>	6
<u>Installation, Environment, and Infrastructure</u>	6
<u>Virtuoso Tools</u>	6
<u>Additional Learning Resources</u>	6
<u>Video Library</u>	6
<u>Virtuoso Videos Book</u>	7
<u>Rapid Adoption Kits</u>	7
<u>Help and Support Facilities</u>	7
<u>Customer Support</u>	7
<u>Feedback about Documentation</u>	8
<u>Typographic and Syntax Conventions</u>	9
<b>1</b>	
<b><u>EDIF 200 In</u></b>	<b>11</b>
<u>Working in the Team Design Manager Environment</u>	11
<u>Understanding EDIF</u>	11
<u>Overview of Translation Process</u>	12
<u>EDIF 200 In Support Features</u>	12
<u>EDIF 200 Limitations</u>	13
<u>Using EDIF 200 In</u>	15
<u>Starting EDIF 200 In from a UNIX Prompt</u>	15
<u>Starting EDIF 200 In from the CIW</u>	15
<u>Creating a Template File</u>	17
<u>How EDIF 200 In Translates Data</u>	19
<u>Mapping EDIF Constructs to DFII Objects</u>	20
<u>How EDIF 200 In Translates Constructs</u>	22
<u>How EDIF 200 In Translates Connectivity</u>	27
<u>How EDIF 200 In Translates Design Hierarchy</u>	30

## Virtuoso EDIF 200 Reader and Writer User Guide

---

<u>How EDIF 200 In Translates Parameters</u> .....	30
<u>How EDIF 200 In Translates Mosaics</u> .....	31
<u>How EDIF 200 In Translates Object Attributes</u> .....	32
<u>EDIF 200 In Output Files</u> .....	38
<u>EDIF 200 In Form Fields</u> .....	39

## 2

<u>EDIF 200 Out</u> .....	43
<u>Understanding EDIF</u> .....	44
<u>EDIF 200 Out Overview</u> .....	44
<u>EDIF 200 Limitations</u> .....	44
<u>Using EDIF 200 Out</u> .....	46
<u>Starting EDIF 200 Out from a UNIX Prompt</u> .....	46
<u>Starting EDIF 200 Out from the CIW</u> .....	47
<u>Creating a Template File</u> .....	50
<u>Creating a Hierarchy File</u> .....	53
<u>How EDIF 200 Out Translates Data</u> .....	54
<u>How EDIF 200 Out Translates Cells and Views</u> .....	54
<u>How EDIF 200 Out Translates Constructs</u> .....	55
<u>How EDIF 200 Out Translates Connectivity</u> .....	57
<u>How EDIF 200 Out Translates Groups</u> .....	59
<u>How EDIF 200 Out Translates Labels</u> .....	59
<u>How EDIF 200 Out Translates Layers</u> .....	59
<u>How EDIF 200 Out Translates the Technology File</u> .....	60
<u>How EDIF 200 Out Translates Libraries</u> .....	60
<u>How EDIF 200 Out Translates Mosaics</u> .....	60
<u>How EDIF 200 Out Translates Properties</u> .....	61
<u>How EDIF 200 Out Translates Reference Designators</u> .....	62
<u>How EDIF 200 Out Translates Shapes</u> .....	62
<u>How EDIF 200 Out Produces a Flattened Netlist</u> .....	63
<u>EDIF 200 Out Output Files</u> .....	64
<u>Error Messages</u> .....	64
<u>Translated Design Data File</u> .....	65
<u>EDIF 200 Out Form Fields</u> .....	70

---

# Preface

---

This user guide contains reference information about the following Cadence products:

- EDIF 200 In, which translates files from the EDIF 200 format into the Cadence® Design Framework II (DFII) database format.
- EDIF 200 Out, which translates files from the DFII format into the EDIF 200 format.

This user guide is aimed at developers and designers of integrated circuits and assumes that you are familiar with the EDIF 200 language.

This preface contains the following topics:

- [Scope](#)
- [Licensing Requirements](#)
- [Related Documentation](#)
- [Additional Learning Resources](#)
- [Customer Support](#)
- [Feedback about Documentation](#)
- [Typographic and Syntax Conventions](#)

## Scope

Unless otherwise noted, the functionality described in this guide can be used in both mature node (for example, IC6.1.8) and advanced node and methodologies (for example, ICADVM20.1) releases.

Label	Meaning
(ICADVM20.1 Only)	Features supported only in the ICADVM20.1 advanced nodes and advanced methodologies release.
(IC6.1.8 Only)	Features supported only in mature node releases.

## Licensing Requirements

For information on licensing in the Virtuoso design environment, see [\*Virtuoso Software Licensing and Configuration Guide\*](#).

## Related Documentation

### Installation, Environment, and Infrastructure

- [\*Cadence Installation Guide\*](#)
- [\*Virtuoso Design Environment User Guide\*](#)
- [\*Cadence SKILL Language User Guide\*](#)
- [\*Cadence SKILL Language Reference\*](#)

### Virtuoso Tools

- [\*Virtuoso EDIF 200 Reader and Writer SKILL Reference\*](#)
- [\*Virtuoso Schematic Editor User Guide\*](#)
- [\*Virtuoso NC Verilog Environment User Guide\*](#)

## Additional Learning Resources

### Video Library

The [\*Video Library\*](#) on the Cadence Online Support website provides a comprehensive list of videos on various Cadence products.

To view a list of videos related to a specific product, you can use the *Filter Results* feature available in the pane on the left. For example, click the *Virtuoso Layout Suite* product link to view a list of videos available for the product.

You can also save your product preferences in the Product Selection form, which opens when you click the *Edit* icon located next to *My Products*.

## **Virtuoso Videos Book**

You can access certain videos directly from Cadence Help. To learn more about this feature and to access the list of available videos, see [Virtuoso Videos](#).

## **Rapid Adoption Kits**

Cadence provides a number of [Rapid Adoption Kits](#) that demonstrate how to use Virtuoso applications in your design flows. These kits contain design databases and instructions on how to run the design flow.

To explore the full range of training courses provided by Cadence in your region, visit [Cadence Training](#) or write to [training\\_enroll@cadence.com](mailto:training_enroll@cadence.com).

**Note:** The links in this section open in a separate web browser window when clicked in Cadence Help.

## **Help and Support Facilities**

Virtuoso offers several built-in features to let you access help and support directly from the software.

- The Virtuoso *Help* menu provides consistent help system access across Virtuoso tools and applications. The standard Virtuoso *Help* menu lets you access the most useful help and support resources from the Cadence support and corporate websites directly from the CIW or any Virtuoso application.
- The Virtuoso Welcome Page is a self-help launch pad offering access to a host of useful knowledge resources, including quick links to content available within the Virtuoso installation as well as to other popular online content.

The Welcome Page is displayed by default when you open Cadence Help in standalone mode from a Virtuoso installation. You can also access it at any time by selecting *Help – Virtuoso Documentation Library* from any application window, or by clicking the *Home* button on the Cadence Help toolbar (provided you have not set a custom home page).

For more information, see [Getting Help](#) in *Virtuoso Design Environment User Guide*.

## **Customer Support**

For assistance with Cadence products:

### ■ Contact Cadence Customer Support

Cadence is committed to keeping your design teams productive by providing answers to technical questions and to any queries about the latest software updates and training needs. For more information, visit <https://www.cadence.com/support>.

### ■ Log on to Cadence Online Support

Customers with a maintenance contract with Cadence can obtain the latest information about various tools at <https://support.cadence.com>.

## Feedback about Documentation

You can contact Cadence Customer Support to open a service request if you:

- Find erroneous information in a product manual
- Cannot find in a product manual the information you are looking for
- Face an issue while accessing documentation by using Cadence Help

You can also submit feedback by using the following methods:

- In the Cadence Help window, click the *Feedback* button and follow instructions.
- On the Cadence Online Support [Product Manuals](#) page, select the required product and submit your feedback by using the *Provide Feedback* box.



## Typographic and Syntax Conventions

The following typographic and syntax conventions are used in this manual.

<i>text</i>	Indicates names of manuals, menu commands, buttons, and fields.
text	Indicates text that you must type as presented. Typically used to denote command, function, routine, or argument names that must be typed literally.
<i>z_argument</i>	Indicates text that you must replace with an appropriate argument value. The prefix (in this example, <i>z_</i> ) indicates the data type the argument can accept and must not be typed.
	Separates a choice of options.
{ }	Encloses a list of choices, separated by vertical bars, from which you <b>must</b> choose one.
[ ]	Encloses an optional argument or a list of choices separated by vertical bars, from which you <b>may</b> choose one.
[ ?argName <i>t_arg</i> ]	Denotes a <i>key argument</i> . The question mark and argument name must be typed as they appear in the syntax and must be followed by the required value for that argument.
...	Indicates that you can repeat the previous argument.
	Used with brackets to indicate that you can specify zero or more arguments.
	Used without brackets to indicate that you must specify at least one argument.
, ...	Indicates that multiple arguments must be separated by commas.
=>	Indicates the values returned by a Cadence® SKILL® language function.
/	Separates the values that can be returned by a Cadence SKILL language function.

If a command-line or SKILL expression is too long to fit within the paragraph margins of this document, the remainder of the expression is moved to the next line and indented. In code excerpts, a backslash ( \ ) indicates that the current line continues on to the next line.

# **Virtuoso EDIF 200 Reader and Writer User Guide**

## **Preface**

---

---

## EDIF 200 In

---

EDIF 200 In translates design data files from the EDIF 200 format into the Cadence® Design Framework II (DFII) format.

For information about

- An overview of the EDIF 200 In translation process, see [“Understanding EDIF”](#) on page 11
- Procedures for starting and controlling an EDIF 200 In translation, see [“Using EDIF 200 In”](#) on page 15
- Explanations of how EDIF 200 In translates different kinds of views, see [“How EDIF 200 In Translates Data”](#) on page 19
- An overview of the EDIF 200 In output files, see [“EDIF 200 In Output Files”](#) on page 38
- A description of the EDIF 200 In form fields, see [“EDIF 200 In Form Fields”](#) on page 39

## Working in the Team Design Manager Environment

If you choose to run EDIF 200 In in a team design manager (TDM) workArea, note the following:

- For a new library, EDIF 200 In does not check anything in to the TDM environment.
- For an existing library that is already under TDM control, EDIF 200 In overwrites this library and checks everything in to the TDM environment.

If adding a new cell to an existing library that is already under TDM control, EDIF 200 In automatically checks this cell in to the TDM environment.

## Understanding EDIF

The Electronic Design Interchange Format (EDIF) is a non-proprietary, standard interchange format that uses text to describe electronic design data. To express constructs that represent

# Virtuoso EDIF 200 Reader and Writer User Guide

## EDIF 200 In

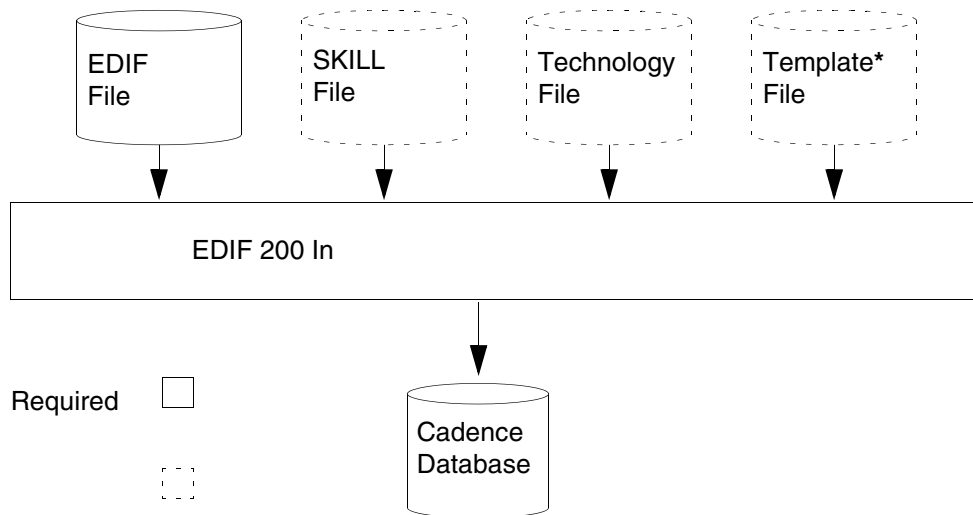
---

electronic design, EDIF 200 uses a syntax similar to LISP, a list-processing programming language.

For more information about the EDIF 200 syntax and the functional and semantic descriptions of each EDIF 200 construct, see the *Electronic Design Interchange Format Version 200, ANSI/EIA 548 Manual*.

## Overview of Translation Process

The Cadence EDIF 200 In translator translates design data from an EDIF 200 format into the DfII format.



\* A template file is required only for batch mode operation.

## EDIF 200 In Support Features

The EDIF 200 In translator supports

- EDIF version 200
- Keyword Level 0
- EDIF constructs
- Legal EDIF files that pass the semantic checker written by University of Manchester, EDIF Technical Centre

The semantic checker generates error and warning messages. Warning messages do not prevent processing.

- Other input files with Component Description Format (CDF) information for any EDIF file generated by Cadence EDIF 200 Out

EDIF 200 In automatically looks for the CDF files in the run directory.

Using this feature, you can bring a Cadence design with CDF information back into the Cadence environment (providing the CDF information was stored in the EDIF format using EDIF 200 Out). The CDF information is reapplied to the design when you run EDIF 200 In.

The EDIF 200 In translator does not support

- Version 110 or any other version of EDIF
- Aliases or keyword macros
- Variables, expressions, or control flow

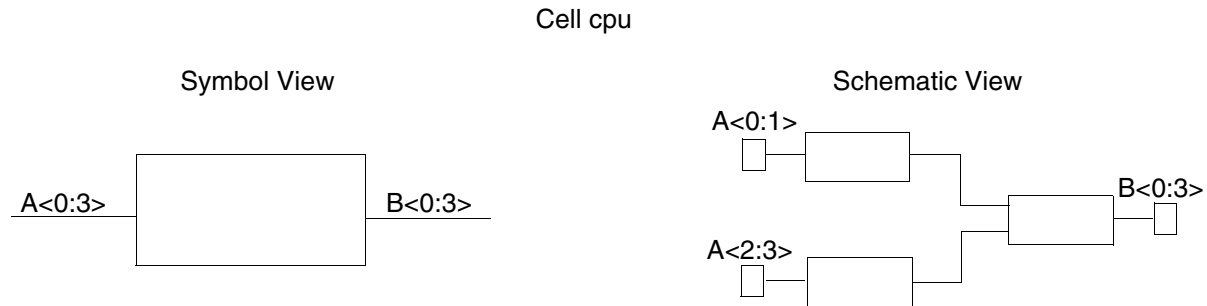
## **EDIF 200 Limitations**

You need to be aware of the following EDIF 200 limitations:

- EDIF 200 does not include an equivalent to the DFII layer-purpose pair. See [“Mapping EDIF Constructs to DFII Objects”](#) on page 20 for more information.
- You cannot describe donuts in EDIF 200.  
Donuts are complex shapes that you construct as composite shapes.
- The EDIF 200 standard does not include the concept of a net that is global to an arbitrary hierarchical depth.
- EDIF 200 does not support the following design styles:
  - ❑ Inconsistent port grouping between the symbol view and the schematic view of a cell (see example 1 below)
  - ❑ Replication of a tapped bit from a bus or bundle (see example 2 below)

## Example 1

This example shows inconsistent port grouping between the symbol view and the schematic view of a cell.

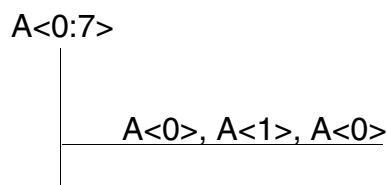


In the example, cell cpu has two interface ports: A<0:3> and B<0:3>. Port A<0:3> is grouped on the symbol of cell cpu. But on the schematic, port A<0:3> is divided into ports A<0:1> and A<2:3>.

In most CAE systems, you establish connectivity by connection and by name, such that ports A<0:1> and A<2:3> are members of port A<0:3>. However, in EDIF 200, ports A<0:1> and A<2:3> cannot be represented because the interface (which defines ports) contains only A<0:3> and B<0:3>.

## Example 2

This example shows replication of a tapped bit from a bus or bundle.



In the example, the bus slice is three bits wide. The first bit is A<0>, the second bit is A<1>, and the third bit is A<0>. This slice cannot be represented in EDIF 200 because it requires A<0> to be represented twice in the port list of a joined statement.

## Using EDIF 200 In

You can start and control EDIF 200 In from either a UNIX prompt or from the Command Interpreter Window (CIW).

### Starting EDIF 200 In from a UNIX Prompt

From a UNIX prompt, you refer to a template file. The template file contains the name of the data file that you want to read in and the preset values for the EDIF 200 In options that you want to apply during the translation process. Template files are described later in this section.

- To refer to a template file from a UNIX prompt, type the following:

```
edifin templateFile
```

- To refer to a template file from a UNIX prompt and use a specific `cds.lib` file, type the following:

```
edifin templateFile -cdslib cdslibFile
```

This command tells the software to use a particular `cds.lib` file instead of searching for the `cds.lib` file in the standard locations. See [Cadence Library Path Editor User Guide](#) for more information about `cds.lib` files.

**Note:** After you translate a schematic, you must use the *Design – Check and Save* command from the schematic window to ensure connectivity. See [Virtuoso Schematic Editor User Guide](#) for more information.

### Starting EDIF 200 In from the CIW

From the CIW, you can access the EDIF 200 In form to specify the options you want to apply to the current design translation.

To access the EDIF 200 In form,

1. Choose *File – Import – EDIF 200*.

## Virtuoso EDIF 200 Reader and Writer User Guide

### EDIF 200 In

The EDIF 200 In form appears.

EDIF 200 In

Template File: 1215-142/tools.lnx86/dfII/samples/xlUI/edifIn.il

Load Save

Run Directory: .

User-Defined SKILL File:

Name Mapping File:

InstName CharMap List:

Technology File:

Input File(s):

Set Graphic To Schematic Symbol: ☒ TRUE ☐ FALSE

Add Net Label To All Segments: ☒ TRUE ☐ FALSE

Use Instance Name For Nets: ☐ TRUE ☒ FALSE

Set Schematic DBUPerUU: ☒ default (160) (for non\_Cadence EDIF)  
☐ read from EDIF (for Cadence EDIF)  
☐ specify value (optional):

Sheet Symbol Library: US\_8ths

Case Sensitivity: ☒ preserve ☐ upper ☐ lower

Keep Original Brackets In Names: ☐ TRUE ☒ FALSE

Select Font Style: stick

Do Not Overwrite Existing Views: ☐ TRUE ☒ FALSE

Do Not Overwrite Existing Schematic Views: ☐ TRUE ☒ FALSE

OK Cancel Defaults Apply Help

**Note:** The `cds.lib` file replaces the search path from earlier versions of EDIF 200 In. The `cds.lib` file identifies the locations of your reference libraries. For more



information, see [Cadence Library Path Editor User Guide](#).

**2.** Complete the *Input File(s)* field.

The input files are the EDIF 200 files to be translated.

**3.** Specify any other options you want to apply during the translation.

Information about the optional fields is provided in the [“EDIF 200 In Form Fields”](#) on page 39.

**4.** Do one of the following:

- ☐ Click *OK* to close the EDIF 200 In form and begin the translation process.
- ☐ Click *Apply* to keep the EDIF 200 In form open and begin the translation process.

EDIF 200 In creates a log file called `edifin.log` in the run directory. The translated library is also created in the run directory. (By default, the *Run Directory* field is set to the current working directory.)

**Note:** If you translated a schematic, you must use the *Design – Check and Save* command from the schematic window to ensure connectivity. See [Virtuoso Schematic Editor User Guide](#) for more information.

## Creating a Template File

A template file is a text file that contains preset values for the options you want to apply during an EDIF 200 In translation.

There are three ways to create a template file:

- Click *Save* on the EDIF 200 In form to save the values you specified on the form. The information is saved to the filename you enter in the *Template File* field.
- Use your local text editor to copy the sample template file to a new file and modify the new file as needed. The sample template file (`edifIn.il`) is located in the `your_install_dir/tools/dfII/samples/xlUI` directory.
- Use your local text editor to create a new template file, like the one shown in the sample below. The sample shows all the EDIF 200 In variable names and the SKILL property list format.

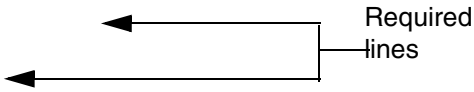
# Virtuoso EDIF 200 Reader and Writer User Guide

## EDIF 200 In

### Sample Template File

The following is a sample EDIF 200 In template file:

```
edifInKeys = list(nil
  'runDirectory "."
  'inFile ""
  'userSkillFile ""
  'mapFile ""
  'charMapForInstName ""
  'techFile ""
  'setSchematicType "TRUE"
  'addNetLabelToAllSegment "TRUE"
  'useInstNameForNet "TRUE"
  'setSchematicDBUPerUU "FALSE"
  'DBUPerUUValue ""
  'sheetSymbolLib "US_8ths"
  'caseSensitivity "preserve"
  'keepOrigBracketsInNames "FALSE"
  'fontStyle "stick"
  'noOverWrite "FALSE"
  'noOverwriteSch "FALSE"
)
```



**Note:** You use the `inFile` field to specify the EDIF 200 files to be translated.

### Mapping Form Fields to Template File Entries

The following table maps the EDIF 200 In form fields to their corresponding template file entries. For descriptions of the form fields, see [“EDIF 200 In Form Fields”](#) on page 39.

### Mapping Form Fields to Template File Entries

EDIF 200 In form field	EDIF 200 In template file entry	Default value
<i>Run Directory</i>	'runDirectory	" . "
<i>User-Defined SKILL File</i>	'userSkillFile	" "
<i>Name Mapping File</i>	'mapFile	" "
<i>InstName CharMap List</i>	'charMapForInstName	" "
<i>Technology File</i>	'techFile	" "
<i>Input File(s)</i>	'inFile	" "
<i>Set Graphic To Schematic Symbol</i>	'setSchematicType	"TRUE"

## Virtuoso EDIF 200 Reader and Writer User Guide

### EDIF 200 In

#### Mapping Form Fields to Template File Entries, *continued*

EDIF 200 In form field	EDIF 200 In template file entry	Default value
<i>Add Net Label To All Segments</i>	'addNetLabelToAllSegment	"TRUE"
<i>Use Instance Name For Nets</i>	'useInstNameForNet	"TRUE"
<i>Set Schematic DBUPerUU</i> with the fields:		
■ <i>default(160)</i>	'setSchematicDBUPerUU	"FALSE"
■ <i>read from Cadence EDIF</i>	'setSchematicDBUPerUU	"TRUE"
■ <i>specify value</i>	'DBUPerUUValue	" "
<i>Sheet Symbol Library</i>	'sheetSymbolLib	" "
<i>Case Sensitivity</i>	'caseSensitivity	"preserve"
<i>Keep Original Brackets In Names</i>	'keepOrigBracketsInNames	"FALSE"
<i>Select Font Style</i>	'fontStyle	"stick"
<i>Do Not Overwrite Existing Views</i>	'noOverWrite	"FALSE"
<i>Do Not Overwrite Existing Schematic Views</i>	'noOverwriteSch	"FALSE"

Any value you enter for 'DBUPerUUValue overrides the value in the 'setSchematicDBUPerUU field. If you use 'DBUPerUUValue, make sure the value you choose is the same value you used to create the EDIF file, and set the 'setSchematicDBUPerUU field to "FALSE". For more information, see ["Set Schematic DBUPerUU."](#)

## How EDIF 200 In Translates Data

Each construct in EDIF 200 is associated with one or more view types. EDIF 200 In supports only constructs associated with *general*, *netlist*, *schematic*, *masklayout*, and *graphic* view types. EDIF 200 In does not support constructs associated with *behavior*, *document*, *logicmodel*, *pcblayout*, *symbolic*, or *stranger* view types.

**Note:** If you are not sure how EDIF 200 In handles a construct, create a sample file and submit it to EDIF 200 In. If EDIF 200 In does not support the construct, the program informs

you. If EDIF 200 In does support it, examine the output library to determine how EDIF 200 In handles the construct.

## Mapping EDIF Constructs to DFII Objects

The following table shows how EDIF 200 constructs are mapped to the DFII objects by EDIF 200 In.



### Caution

***There are important differences between the EDIF 200 constructs and the DFII objects. To prevent a loss of data, study the following table and the information that follows the table before you attempt a data conversion.***

## Mapping EDIF 200 Constructs to DFII Objects

EDIF 200 construct	DFII object
annotate	label
arc	arc
boundingBox	bounding box
cell	cell
circle	ellipse
color	layer-purpose pair color
connectLocation	pin
criticality	criticality
curve	arc
designator	property named <i>refDes</i>
direction	terminal direction
display	<i>theLabel</i>
dot	ellipse
endType	<i>pathStyle</i>
external	library
figure	layer-purpose pair

# Virtuoso EDIF 200 Reader and Writer User Guide

## EDIF 200 In

### Mapping EDIF 200 Constructs to DFII Objects, *continued*

EDIF 200 construct	DFII object
figureGroup	layer-purpose pair definition
includeFigureGroup	layer-purpose pair definition
instance	any instance
instanceMap	property
joined	net-to-terminal definition
library	library
net	net
netBundle	bundle
offPageConnector	terminal
openShape	curve or line
orientation	orient
origin	origin
page	sheet cell view
path	path or line
pathWidth	path width
permutable	groups
polygon	polygon
port	terminal
portBundle	terminal
portImplementation	pin, figure
portInstance	<i>instanceTerminal</i> (not direct mapping)
property	property
protectionFrame	routing barrier
rectangle	rectangle
stringDisplay	text
symbol	view
textHeight	font height

### Mapping EDIF 200 Constructs to DFII Objects, *continued*

EDIF 200 construct	DFII object
<code>transform</code>	<code>transform</code>
<code>view</code>	<code>cellview</code>
<code>visible</code>	<code>layer visibility</code>

## How EDIF 200 In Translates Constructs

The following sections discuss EDIF 200 constructs. There are five types of constructs:

- General constructs
- Netlist view constructs
- Schematic view constructs
- Masklayout view constructs
- Graphic view constructs

### General Constructs

The `general` constructs described in this section are often used or translated differently than as defined by the Electronic Industries Association (EIA) standards.

#### Cell Construct

EDIF 200 In converts EDIF 200 cells to DFII cells.

#### Contents Construct

The `contents` construct defines a detailed implementation of the cellview. Most objects that go into the database of the DFII view are defined in constructs within the EDIF 200 `contents` construct.

#### Direction Construct

The `direction` construct defines whether a port is input, output, or bidirectional. If no direction is defined, EDIF 200 In sets the direction to `unknown`.

#### Display Construct

The `display` construct is supported for displaying text. For example, EDIF 200 In uses the `display` construct, within the context of the `keywordDisplay` construct, to create a label for displaying the cell name as follows:

```
(keywordDisplay CELL
  (display textLayer
    (justify LOWERCENTER)
    (origin (pt 0 -20))
  )
)
```

#### External Construct

EDIF 200 uses the `external` construct to declare libraries that are referenced but not defined in the EDIF file. EDIF 200 In expects to find libraries declared as external to exist on the system and to be accessible through the paths specified in the `cds.lib` file.

EDIF 200 In uses the `cds.lib` file to locate the views, opens each view in the read-only mode, and verifies the existence of the ports declared in the interface section.

External library declarations must be `edifLevel 0`. Parameterization is not supported.

#### Figure and FigureGroup Constructs

EDIF 200 In reads the `figureGroup` construct in the EDIF file and the set of system layers in the technology file to define the layer name and layer-purpose. You must define the layer name and layer-purpose for each shape. If the shape is a pin, the default layer name is *pin* and the layer-purpose is *drawing*. If the shape is a net, the default layer name is *wire* and the layer-purpose is *drawing*. If the shape is a net label, the default layer name is *wire* and the layer-purpose is *label*.

#### Library Construct

EDIF 200 In uses the `library` construct to create the library that contains the cellviews. If a library defined in the EDIF file exists in the DFII environment, EDIF 200 In opens the library in append mode.

#### OpenShape Construct

The `openShape` construct describes a shape that is open.

#### Port Construct

Ports and port bundles defined in the EDIF 200 *interface* section are mapped to DFII terminals. When EDIF 200 In encounters a `port` construct, it generates a terminal. The DFII environment supports only one-dimensional arrayed ports.

#### PortBundle Construct

When EDIF 200 In reads a `portBundle` construct, it generates a terminal name by concatenating the name of each port listed in the `listOfPorts` construct. This list is separated by commas. Consider the following example:

```
(portBundle pbExample
  (listOfPorts
    (port a)
    (port b)
  )
)
```

## Virtuoso EDIF 200 Reader and Writer User Guide

### EDIF 200 In

---

```
(port (array c 3))
)
```

In this example, EDIF 200 In creates a terminal named a,b,c<0:2>.

#### PortImplementation Construct

EDIF 200 In uses the `portImplementation connectlocation` construct to define a pin shape and to set the layer name to *pin* and the layer-purpose to *drawing*. You must set the layer name and layer-purpose if you want to use the schematic editor to edit the design.

#### Scale Construct

In the `numberDefinition` construct, the `scale` construct establishes the ratio of DFII units to user units. The formula is as follows:

$$\text{ScaleRatio} = (\text{edifUnits} / \text{userUnits in meters}) * \text{conversion}$$

The user units conversion is .0254 m/inch for schematics, and  $1.0 \times 10^{-6}$  m/micron for mask layouts. But because netlists are not graphic, distances are meaningless for netlist views. Therefore, the DFII point is equivalent to the EDIF point divided by the ScaleRatio.

For example, if the EDIF file contains

```
(numberDefinition
  (scale 1 (E 254-6) (unit DISTANCE))
)
..
(viewtype schematic)
boundingBox (rectangle (pt 0 0) (pt 80 60))
```

the DFII mapping points are as follows:

ScaleRatio	=	$(1 / 254e^{-6}) / 254e^{-4}$
	=	100
DFII point	=	$(\text{edifpoint} / \text{ScaleRatio})$
	=	$80 / 100$
	=	.8

#### Technology Construct

The technology file information in DFII 4.4 and beyond is not compatible with version 4.3.x.

#### View Construct

EDIF 200 In maps EDIF 200 `view` constructs to DFII `cellview` objects. Then EDIF 200 In attaches properties to the cellviews.



## Netlist View Constructs

The netlist view describes the design connectivity for circuit simulators and layout tools. Because the schematic view contains network topology plus graphic information, `netlist` view constructs are a subset of `schematic` view constructs.

**Note:** You cannot edit a nongraphic netlist view. The schematic editor does not allow you to edit a view type other than *schematic*.

### Contents Construct

For the netlist view type, the `contents` construct defines the connectivity of instances, nets and netBundles, and ports.

### UserUnits Construct

For the netlist view type, EDIF 200 In sets the `userUnits` construct to "inches" and the conversion number to 2.54\*10 m/UU.

## Schematic View Constructs

The schematic view is a logical description of an electronic design. It is a superset of the netlist view because the schematic view specifies all the connectivity information of the design plus graphic information.

The EDIF 200 schematic view can contain symbol libraries and schematics at the block, gate, and transistor level.

### Contents Construct

The `contents` construct of a `schematic` view consists of connectivity and graphics that include instances and nets of standard device symbols and descriptive labels. In addition, the `contents` is subdivided into pages. The existence of `page` constructs in the `contents` construct of a schematic view causes EDIF 200 In to create an indexed schematic.

### Page Construct

EDIF 200 In supports the `page` construct for the schematic view and generates multisheet schematics. EDIF 200 In uses the `offPageConnector` construct in the `contents` construct outside the scope of a page to establish off-page connectors in the DFII database.

### Symbol Construct

Use the `symbol` construct to generate symbols for schematics or symbolic layouts. In a schematic view type, the `symbol` construct defines graphics for a schematic symbol, such as a NAND gate.

A schematic symbol is a standalone entity that the DFI environment places as an instance in a schematic; EDIF 200 In defines a schematic symbol using the `symbol` construct embedded in an `interface` construct, not in the context of a sheet schematic.

#### **Transform Construct**

For schematic views that have geometrical constructs, EDIF 200 In supports the `transform` construct of an instance.

#### **UserUnits Construct**

For the schematic view type, set `userUnits` to "inches" and the conversion number to  $2.54 \times 10^{-2}$  m/UU.

### **Masklayout View Constructs**

The `masklayout` view constructs are discussed below. `masklayout` constructs are often used or translated differently than as defined by the EIA standards.

#### **Contents Construct**

A `masklayout` view can contain connectivity, but the primary information transmitted to the database is the mask artwork defined by the `figure` constructs within the `contents` construct.

#### **Transform Construct**

For `masklayout` views that have geometrical constructs, EDIF 200 In supports the `transform` construct of an instance.

#### **UserUnits Construct**

Set `masklayout userUnits` to "microns" and use a conversion number of  $10^{-6}$  m/UU; for example:

$$\begin{aligned}\text{UserUnitsPerMicron} &= (1 \text{ edif unit} / 1 \times 10^{-9} \text{ m}) * (1 \times 10^{-6}) \text{ m/UU} \\ &= 1000 \text{ UserUnitsPerMicron}\end{aligned}$$

### **Graphic View Constructs**

The `graphic view` construct that EDIF 200 In supports is the `graphic view` type.

#### **User Units**

Set `graphic userUnits` to "microns" and use a conversion number of  $10^{-6}$  m/UU, for example:

$$\text{DBUPerUU} = (1 \text{ edif unit} / 1 \times 10^{-9} \text{ m}) * (1 \times 10^{-6}) \text{ m/UU}$$

= 1000 DBUPerUU

## How EDIF 200 In Translates Connectivity

EDIF 200 In imposes certain constraints because of differences in connectivity rules and models between EDIF 200 and DFII. These constraints and differences are discussed in the following sections.

### Differences in Connectivity Rule

DFII allows one master port for each port joined in a net at a given level of hierarchy. The joining of two or more master ports at the same level constitutes a short.

For example, the following connection is allowed because port P1 at the current level of hierarchy is joined to port P1 on instance I1.

```
(net N1
  (joined (portRef P1) (portRef P1 (instanceRef I1)))
)
```

But the following connection is *not* allowed because both ports (P1 and P2) are at the same level of hierarchy.

```
(net N2
  (joined (portRef P1) (portRef P2))
)
```

### Differences in Connectivity Models

The EDIF 200 connectivity model differs from the DFII connectivity model. EDIF 200 uses *net-based* connectivity; that is, EDIF 200 defines a net and establishes connectivity by stating which ports are connected to the net. The DFII environment supports a *terminal-based* connectivity model.

Consider the following example:

```
(port (array P 2))
...
(instance i1 (viewRef v1 (cellRef c1 (libraryRef l1))))
..
(net a
  (joined
    (portRef (member p 0))
    (portRef p1 (instanceRef i1))
  )
)
(net b
  (joined
```

```
        (portRef (member p 1))  
        (portRef p2 (instanceRef i1))  
    )  
)
```

For a schematic view, EDIF 200 In generates a net named `a`, `b` and connects the `p<0:1>` terminal to it. The `a` net connects to the `p1` port on instance `i1`. The `b` net connects to the `p2` port on instance `i1`.

For a netlist view, EDIF 200 In generates net `a` and net `b`. The `a` net connects to the `p<0>` terminal. The `b` net connects to the `p<1>` terminal.

## Physical Implementation of Nets

EDIF 200 expresses the physical implementation of a net through a `figure` within the context of the `net` construct. EDIF 200 In also supports `commentGraphics` and `property` constructs.

## Critical Nets

EDIF 200 In uses the `criticality` construct to define the relative importance of a net. Placement and routing uses the `criticality` construct to establish net criticality.

## Rippers

When EDIF 200 In translates a cell whose cell type is `RIPPER`, the string property `schType` is attached to the `cellview` and its value is set to `"ripper"`.

EDIF 200 In processes the `interface` section of `RIPPER` cellviews differently from `GENERIC` cellviews. Rather than creating a terminal, EDIF 200 In creates a pin for each port defined in the interface and a terminal for each joined construct declared in the interface.

The pins are attached to the terminals based on the references in the joined constructs. The following EDIF fragment illustrates this concept:

```
(cell rip (cellType RIPPER)  
  (view symbol (viewType SCHEMATIC)  
    (interface  
      (port (array A 512) (direction INOUT))  
      (port (array B 512) (direction INOUT))  
      (port (array C 512) (direction INOUT))  
      (joined  
        (portRef A)  
        (portRef B)  
        (portRef C)) ... )))
```

In this example, EDIF 200 In creates a pin for ports A, B, and C. EDIF 200 In creates a terminal named `ripTerm` when it processes the joined constructs. Because all three ports are referenced in the single joined construct, the pins corresponding to each of these ports are attached to the terminal.

When the terminal and its associated pins are created for a ripper, the width of the pins is not recorded. The widths of the connections between the ports of the ripper are determined when the ripper is instantiated. The connectivity is established when the ripper is used.



#### Caution

**Although EDIF 200 In can accept ripper cell definitions, the connectivity of nets cannot be guaranteed if the nets use rippers that do not conform to the Electronic Industries Association (EIA) recommended structure.**

**Note:** The EDIF 200 In structure for creating rippers is consistent with the recommendations by the EDIF Monograph Series publication, *EDIF Connectivity*, and by the EDIF Schematic Technical Subcommittee in the *Using EDIF 200 for Schematic Transfer* application guide.

When the ripper is used in a net, EDIF 200 In extracts the information from the joined constructs in the nets connected to the ripper. It builds a connection expression for each port of the ripper that is connected.

The connection expression, a string property attached to the instance, denotes the connectivity between the net and a particular ripper cell pin. The property name contains the ripper pin name. The following EDIF fragment contains an instance that refers to the ripper cell defined in the previous example:

```
instance I1 (viewRef symbol (cellRef rip)) ...)
...
(net
(joined
  (portRef D)
  (portRef (member A 0) (instanceRef I1))) ...)
```

When EDIF 200 In processes this net, it extracts the member information for the referenced port instance A. Because the instance is a ripper cell, EDIF 200 In uses the member information to construct a net expression property that attaches to instance I1 (`instanceRef I1`). The name of the net expression property is `edifNetExpr_A`. The value of the property is `"0=0"`.

Both vectors are equivalent and, therefore, a shorter notation can be applied and the value can be set to `"0"` (zero).

The syntax for the net expression properties is as follows:

```
net_expression_name ::= 'edifNetExpr_'ripperPinName
net_expression ::= vector_expression='vector_expression'
```

EDIF 200 In uses a subset of the available vector expressions. For more information about vector expressions, refer to [Virtuoso Schematic Editor User Guide](#).

**Note:** Rippers are not allowed in netlists. Rippers are allowed in schematics, but only as a support mechanism for EDIF. Cadence recommends that you edit any schematics generated from EDIF 200 In that contain rippers. Replace the rippers with wires and labels that represent tap expressions. Refer to [Virtuoso Schematic Editor User Guide](#) for information about tap expressions.

EDIF 200 In supports one-dimensional, two-dimensional, and multidimensional rippers. One-dimensional rippers are connection points between wires and buses of different sizes. Two-dimensional rippers are those cells that have two ports that are not implemented in the same location. Multidimensional rippers are those that rip out several signals from a bus with a single ripper.

**Note:** Different nets cannot join the same port of a ripper cell.

## How EDIF 200 In Translates Design Hierarchy

EDIF 200 In establishes design hierarchy through the `instance` construct. However, EDIF 200 In does not support either the `parameterAssign` construct, which assigns parameters to instances, or the `portInstance` construct, which applies and modifies properties on instance terminals.

## How EDIF 200 In Translates Parameters

You can set the parameters for some Cadence applications by setting user properties in the DFI environment. For example, you can use the graphics editor to set properties on graphics data, which is equivalent to setting parameters for an application that runs in the graphics environment.

The following example shows how you can set properties for *schematic* views. In the example, EDIF 200 In uses the values in the `scale` construct together with the `viewType` to determine scaling.

```
(library L
  (edifLevel 0)
  (technology
    (numberDefinition
      (scale 160 (e 254 -4) (unit DISTANCE))
    )
  )
  (cell C
    (cellType GENERIC)
    (view V
      (viewType SCHEMATIC)
      (interface ...)
```

```
        (property screenGridSpacing
          (number (e 125 -3))
        )
        (property snapSpacing
          (number (e 625 -4))
        )
        (property screenGridMultiple
          (integer 8)
        )
        ...
      )
    )
  )
```

In the preceding example, the `scale` construct says that every 160 distance units in the EDIF file corresponds to  $254 \times 10^{-4}$  meters. The `viewType` is `schematic`. Therefore, EDIF 200 In sets the `userUnit` attribute to "inches" and uses a conversion of  $2.54 \times 10^{-2}$  m/UU. The scaling is calculated as follows:

$$\begin{aligned} \text{DBUPerUU} &= (160 \text{ edif units} / 254 \times 10^{-4} \text{ m}) * (2.54 \times 10^{-2} \text{ m} / \text{UU}) \\ &= 160 \text{ DBUPerUU} \end{aligned}$$

The other properties in the example set the screen grid spacing, snap spacing, and screen grid multiple of the graphics editor. For more information about setting user properties, see [\*Virtuoso Design Environment User Guide\*](#).

## How EDIF 200 In Translates Mosaics

EDIF 200 In translates mosaics only for mask layout. EDIF 200 In supports two-dimensional instance arrays. For example, the following fragment produces a 3 by 5 simple mosaic in the DFII database.

```
(instance (array mosaic1 3 5)
(viewRef V1 (cellRef C1 (libraryRef L1)))
)
```

If you replace the `viewRef` construct in the example with a `viewList` construct (which places instances of different views in the array), EDIF 200 In generates a complex mosaic that does not permit connectivity.

The DFII environment does not support connectivity involving mosaic instances. However, the following one-dimensional arrayed instance generates an iterated instance (not a mosaic instance) in the DFII database. And EDIF 200 In supports connection to ports on iterated instances.

```
(instance (array I2 8)
(6viewRef V2 (cellRef C2 (libraryRef L1)))
)
```

## How EDIF 200 In Translates Object Attributes

Most objects in the DFII environment have associated attributes that are used by one or more applications. Attributes are similar to user-defined properties because they are pieces of information attached to an object. However, properties exist only at your discretion, whereas attributes always exist as a permanent part of the database.

Cadence specifies the attribute type. When the DFII environment creates a database object, it assigns a value to each of the object's attributes. If you do not specify a value, the system chooses a default value. You can change the attribute value at any time.

You can set certain attributes in EDIF 200 In. Use the EDIF 200 `property` construct with an owner "Cadence" qualification to distinguish the attribute from the usual `property` construct.

For example, to set a ground net signal type, use the following construct:

```
(net (rename GND "gnd!")
(joined
  (portRef S (instanceRef &0))
  (portRef GND (instanceRef &2))
)
  (property sigType
    (string "ground")
    (owner "Cadence")
  )
)
```

To set object attributes using owner "Cadence" properties, use certain key words for both the *property* name and the *property* value. The next sections describe which property names and property values to set for figure groups, instances, and nets.



## Figure Groups

A figure group must contain the following property information:

Attribute:	layer number
Property name:	layer
Legal values:	integers between 0 and 127
Attribute:	layer priority
Property name:	priority

The following example shows how you can use EDIF 200 In to translate a figure group:

```
(library L1
  (edifLevel 0)
  (technology
    (figureGroup diffusion
      (property layer
        (integer 52)
        (owner "Cadence")
      )
      (property priority
        (integer 37)
        (owner "Cadence")
      )
    )
  )
)
```

## Instances

An instance must contain the following property information:

Attribute:	instance placement status
Property name:	placementStatus
Legal values:	"unplaced" (default for netlist view) "suggested" "placed" "locked" "firm" (default for masklayout, graphic, and schematic views)

The following example shows how you can use EDIF 200 In to translate an instance:

```
(instance I1
  (viewRef ...)
  (transform ...)
  ...
  (property placementStatus
    (string "suggested")
    (owner "Cadence")
  )
  ...
)
```

## Nets

The three net types are as follows:

- Connection status
- Signal type
- Global

The following three sections describe the property information of each net type.

### ***Net Connection Status***

A net connection status must contain the following property information:

Attribute:	net connection status
Property name:	connStatus
Legal values:	"connected"
	"edgeConnected"
	"connectOutside"
	"needToConnect" (default)

The following example shows how you can use EDIF 200 In to translate net connection status:

```
(net N1
  (joined ...)
  ...
  (property connStatus)
    (string "connectOutside")
    (owner "Cadence")
  )
  ...
)
```

### ***Net Signal Type***

A net signal type must contain the following property information:

Attribute:	signal type
Property name:	sigType
Legal values:	"signal" (default)
	"ground"
	"supply"
	"clock"
	"testLatch"

The following example shows how you can use EDIF 200 In to translate net signal type:

```
(net VDD
  (joined ...)
  ...
)
```

```
(property sigType)
  (string "supply")
  (owner "Cadence")
)
...
```

### ***Net Global Status***

A net global status must contain the following property information:

Attribute:	global signal
Property name:	global
Legal values:	(true)
	(false) (default)

The following example shows how you can use EDIF 200 In to translate net global status:

```
(net VDD
  (joined ...)
  ...
  (property global
    (boolean (true))
    (owner "Cadence")
  )
  ...
)
```

For a schematic view, EDIF 200 In automatically appends an exclamation point (!) to the end of the global signal name. For example, it changes VDD to VDD! to represent a global signal. For a netlist view, EDIF 200 In does not modify the global net name.

### **Net Names**

EDIF 200 In converts curly braces ({ }) and brackets ([ ]) in net names to angle brackets (< >). EDIF 200 In also converts double periods (..) and hyphens (-) in the bit range of net names into colons (:). The table shows how EDIF 200 In expands net names and vector expressions

#### **How EDIF 200 In Expands Net Names and Vector Expressions**

Name	Number of Members	Expanded Form
clk	1	clk

# Virtuoso EDIF 200 Reader and Writer User Guide

## EDIF 200 In

### How EDIF 200 In Expands Net Names and Vector Expressions, *continued*

Name	Number of Members	Expanded Form
data<2>	1	data<2>
<*1>base	1	base
<*2>term	2	term,term
<*2>(a,b),c	5	a,b,a,b,c
<*2>(a,<*2>b)	6	a,b,b,a,b,b
b<0:2>	3	b<0,1,2>
b<0:2:1>	3	b<0,1,2>
b<3:0:2>	2	b<3,1>
b<0:2*2>	6	b<0,0,1,1,2,2>
b<(0:2)*2>	6	b<0,1,2,0,1,2>
b<0,2*2>	3	b<0,2,2>
b<(0,2)*2>	4	b<0,2,0,2>
b<(0,1:3:4*1,2:2>	3	b<0,1,2>
b<0:1,2:2>	3	b<0,1,2>

As shown in the table, the expanded form of a name is semantically equivalent to the original specification. It is created by expanding the range specifications, applying repeat operators, and removing parentheses.

To determine the  $n$ th member name, use a zero-based counting scheme to find the  $n$ th element in the expanded form. For example, member zero of  $\langle *2 \rangle(a, \langle *2 \rangle b)$  is  $a$ , member one is  $b$ , member two is  $b$ , and so on. If the name contains a vector expression, such as  $b\langle 0:1,2:2 \rangle$ , then member zero is called  $b\langle 0 \rangle$ , member one is  $b\langle 1 \rangle$ , and member two is  $b\langle 2 \rangle$ .

**Note:** Net names can have up to 8000 characters only.

## EDIF 200 In Output Files

In addition to a translated design data file, EDIF 200 In produces log messages. The messages are written to either a file or the screen, depending on how you started EDIF 200 In.

- When you use the EDIF 200 In form to start EDIF 200 In, the software writes the log messages to the `CDS.log` file in your home directory and to the `edifin.log` file in your run directory.
- When you start EDIF 200 In by typing the `edifin` command at the UNIX prompt, the software displays the log messages on your screen and writes error and warning messages to `edifin.log` file in your run directory.

You can redirect the messages by piping the messages to a file. To pipe the messages to a file, type the following at the UNIX prompt:

```
edifin templateFile >& file.log
```

where *templateFile* is the name of the template file and *file.log* is the name of the log file you want to create.

## EDIF 200 In Form Fields

The screenshot shows the 'EDIF 200 In' dialog box with the following fields and options:

- Template File:** A text field containing the path '1215-142/tools.lnx86/dfII/samples/xlUI/edifIn.il'. Below it are 'Load' and 'Save' buttons.
- Run Directory:** A text field containing a single period '.'.
- User-Defined SKILL File:** An empty text field.
- Name Mapping File:** An empty text field.
- InstName CharMap List:** An empty text field.
- Technology File:** An empty text field.
- Input File(s):** An empty text field.
- Set Graphic To Schematic Symbol:** Radio buttons for 'TRUE' (selected) and 'FALSE'.
- Add Net Label To All Segments:** Radio buttons for 'TRUE' (selected) and 'FALSE'.
- Use Instance Name For Nets:** Radio buttons for 'TRUE' and 'FALSE' (selected).
- Set Schematic DBUPerUU:** Radio buttons for 'default (160) (for non\_Cadence EDIF)' (selected), 'read from EDIF (for Cadence EDIF)', and 'specify value (optional)' (with an adjacent empty text field).
- Sheet Symbol Library:** A text field containing 'US\_8ths'.
- Case Sensitivity:** Radio buttons for 'preserve' (selected), 'upper', and 'lower'.
- Keep Original Brackets In Names:** Radio buttons for 'TRUE' and 'FALSE' (selected).
- Select Font Style:** A dropdown menu showing 'stick'.
- Do Not Overwrite Existing Views:** Radio buttons for 'TRUE' and 'FALSE' (selected).
- Do Not Overwrite Existing Schematic Views:** Radio buttons for 'TRUE' and 'FALSE' (selected).

At the bottom of the dialog are buttons for 'OK', 'Cancel', 'Defaults', 'Apply', and 'Help'.

- **Template File**— specifies the name of a text file that contains filenames and values for the options you want to apply during an EDIF 200 In translation process.

## Virtuoso EDIF 200 Reader and Writer User Guide

### EDIF 200 In

---

- **Load**—button reads the template file and applies its contents (field names and values) to the EDIF 200 In form.
- **Save**—button creates a template file that contains the filenames and option values you specify in the form. The information is written to the filename you specify in the *Template File* field.
- **Run Directory**—is the directory where the output files and libraries are generated. You must have write permission to this directory. The default is the current working directory from which you started the software.
- **User-Defined SKILL File**—is a file that contains user-defined SKILL routines.
- **Name Mapping File**—lets you specify the file in which EDIF 200 In writes specific name mapping information such as the legal names that EDIF 200 In substitutes for illegal names. If you do not provide a filename in this field, EDIF 200 In does not generate the file.
- **InstName CharMap List**—is a file that contains the user-defined mapping information. The mapping information tells EDIF 200 In how to handle instance names that contain illegal characters.

The format is a list containing mapping pairs enclosed in parentheses. For example, the string “( / | ) (\$ - )” maps the character “/” in an instance name to the character “|” and maps the character “\$” to the character “-”. That is, the first character in the parentheses is the original character and the second character is the character to which the first one is mapped.

- **Technology File** —specifies the name of a technology file. A technology file sets the library configuration. If you do not specify a technology file, EDIF 200 In uses the current library configuration.
- **Input File(s)**—specifies the names of the EDIF 200 files to translate. Specify more than one input file by separating each file with a space. You must specify at least one input file.
- **Set Graphic To Schematic Symbol**—specifies whether EDIF 200 In converts graphic symbols to schematic symbols.
  - ☐ **TRUE**—converts graphic symbols to schematic symbols.
  - ☐ **FALSE**—does not convert graphic symbols to schematic symbols. This is useful for nonschematic transfer.
- **Add Net Label To All Segments**—specifies whether EDIF 200 In adds labels to all net segments without labels.
  - ☐ **TRUE**—adds labels to all net segments without labels.



## Virtuoso EDIF 200 Reader and Writer User Guide

### EDIF 200 In

---

- ☐ *FALSE*—does not add labels to any net segments without labels, except bus signals tapped from a bus.
- *Use Instance Name For Nets*—specifies the net name for an unconnected terminal.
  - ☐ *TRUE*—specifies the net name that EDIF 200 In derives from the corresponding instance name and terminal name. The syntax for the derived net name is `instanceName.portName`
  - ☐ *FALSE*—specifies the default net name that EDIF 200 In generates.
- *Set Schematic DBUPerUU*—specifies how EDIF 200 In sets the value for DBUPerUU (database units per user unit), which is the value for the grid spacing on the schematic.
  - ☐ *default (160) for non\_Cadence EDIF*—specifies that the default value of 160 DBUPerUU is used. Use this setting for third-party EDIF files.
  - ☐ *read from EDIF for Cadence EDIF*—specifies that EDIF 200 In is to take this value from the EDIF file. Use this setting for EDIF files generated by EDIF 200 Out.
  - ☐ *specify value (optional)*—lets you enter a specific value. Any value you enter in this field overrides the two previous choices.
- *Sheet Symbol Library*—specifies the library containing sheet symbols used to generate schematic sheets. The default is the `US_8ths` library.
- *Case Sensitivity*—specifies whether EDIF 200 In changes the case of letters in all the EDIF identifiers; that is, all the names (including cell names, net names, and instance names), renames, and keywords in the EDIF file.
  - ☐ *preserve*—converts EDIF identifiers without changing case. For example, the name `Obj1` remains `Obj1`.
  - ☐ *upper*—converts EDIF identifiers to uppercase. For example, the name `Obj1` becomes `OBJ1`.
  - ☐ *lower*—converts EDIF identifiers to lowercase. For example, the name `Obj1` becomes `obj1`.
- *Keep Original Brackets In Names*—specifies whether the bus notation stays in its original form or is mapped into Cadence bus notation.
  - ☐ *TRUE*—keeps the original bus notation in the EDIF 200 file.
  - ☐ *FALSE*—converts non-angle-bracketed bus notation into the Cadence bus notation, which uses angle brackets (`< >`).

**Note:** If the EDIF file is *schematic view*, the bus notation must be mapped into the Virtuoso® Schematic Composer schematic capture bus notation. That is, if you are

going to use the composer tool for design entry, use *FALSE* to convert bus notation into Cadence bracketed notation. Otherwise, the composer tool does not generate the correct connectivity for the design. Use *TRUE* if the design is going to be worked in another tool.

- *Select Font Style*—sets the font style you want to use in the DFI database. The font choices are *stick*, *euroStyle*, *fixed*, *gothic*, *math*, *roman*, *script*, and *swedish*. The default is *stick*.
- *Do Not Overwrite Existing Views*—specifies whether the existing cellview has to be recreated during EDIF 200 In translation or not.
  - ☐ *TRUE*—keeps the original cellview during EDIF 200 In translation..
  - ☐ *FALSE*—overwrites the original cellview.
- *Do Not Overwrite Existing Schematic Views*—specifies whether the existing schematic cellview has to be re-created while updating the symbol views during EDIF 200 In translation or not.
  - ☐ *TRUE*—keeps the original cellview in the schematic during EDIF 200 In translation.
  - ☐ *FALSE*—overwrites the original cellview in the schematic.

---

## EDIF 200 Out

---

EDIF 200 Out translates design data files from the Cadence® Design Framework II (DFII) format into the EDIF 200 format.

For information about

- An overview of the EDIF 200 Out data translation process, see [“Understanding EDIF”](#) on page 44
- Procedures for starting and controlling an EDIF 200 Out translation, see [“Using EDIF 200 Out”](#) on page 46
- Explanations of how EDIF 200 Out translates different kinds of views, see [“How EDIF 200 Out Translates Data”](#) on page 54
- A sample EDIF 200 Out output file, see [“EDIF 200 Out Output Files”](#) on page 64
- A description of the EDIF 200 Out form fields, see [“EDIF 200 Out Form Fields”](#) on page 70

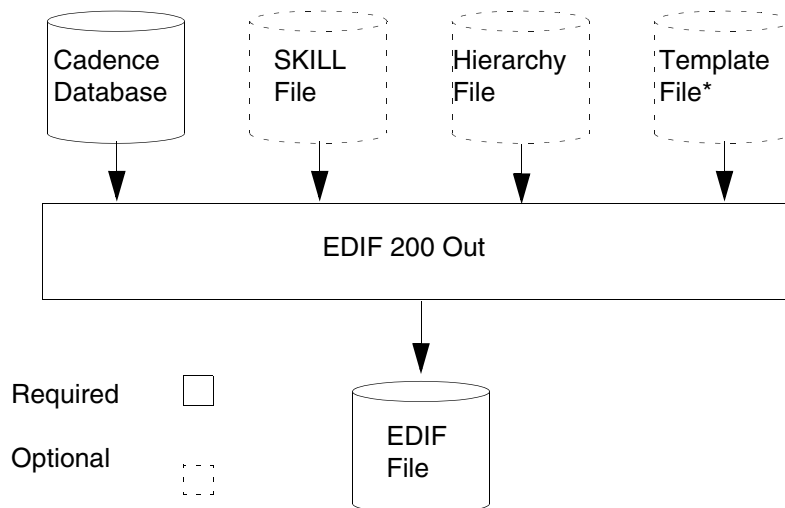
## Understanding EDIF

The Electronic Design Interchange Format (EDIF) is a nonproprietary, standard interchange format that uses text to describe electronic design data. To express constructs that represent the electronic design, EDIF 200 uses a syntax similar to LISP, a list-processing programming language.

For more information about the EDIF 200 syntax and the functional and semantic descriptions of each EDIF 200 construct, see the *Electronic Design Interchange Format Version 200, ANSI/EIA 548* manual.

### EDIF 200 Out Overview

The Cadence EDIF 200 Out translator changes the format of design data from DFII into the EDIF 200 format.



\* A template file is required only for batch mode operation.

### EDIF 200 Limitations

You need to be aware of the following EDIF 200 limitations.

- EDIF 200 does not include an equivalent to the DFII layer-purpose pairs.

You can use the Cadence SKILL language interface to force EDIF 200 Out to create multiple layer-purpose pairs in the EDIF 200 output file. EDIF 200 describes layers using the `figureGroup` construct. You can also use the EDIF 200 `property` construct, but the `property` construct cannot define a layer with more than one purpose. (A layer with more than one purpose violates the EDIF 200 grammar rule stating that names defined in the same name scope must be unique.)

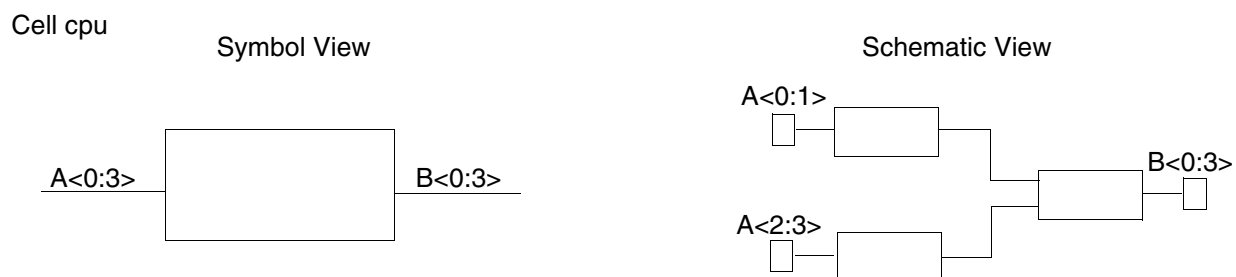
- You cannot describe donuts in EDIF 200.

Donuts are complex shapes that you construct as composite shapes.

- The EDIF 200 standard does not include the concept of a net that is global to an arbitrary hierarchical depth.
- EDIF 200 does not support the following design styles:
  - ❑ Inconsistent port grouping between the symbol view and the schematic view of a cell (see example 1)
  - ❑ Replication of a tapped bit from a bus or bundle (see example 2)
- EDIF 200 Out cannot output the portRef to net construct when numInst exceeds 32767.

### Example 1

This example shows inconsistent port grouping between the symbol view and the schematic view of a cell.

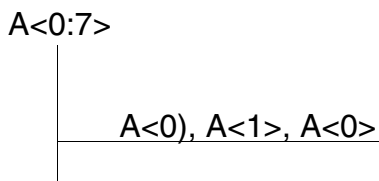


In the example, cell cpu has two interface ports: A<0:3> and B<0:3>. Port A<0:3> is grouped on the symbol of cell cpu. But on the schematic, port A<0:3> is divided into ports: A<0:1> and A<2:3>.

In most CAE systems, you establish connectivity by connection and by name, such that ports A<0:1> and A<2:3> are members of port A<0:3>. However, in EDIF 200, ports A<0:1> and A<2:3> cannot be represented because the interface (which defines ports) contains only A<0:3> and B<0:3>.

## Example 2

This example shows replication of a tapped bit from a bus or bundle.



In the example, the bus slice is three bits wide. The first bit is  $A<0>$ , the second bit is  $A<1>$ , and the third bit is  $A<0>$ . This slice cannot be represented in EDIF 200 because it requires  $A<0>$  to be represented twice in the port list of a joined statement.

## Using EDIF 200 Out

You can start and control EDIF 200 Out from either a UNIX prompt or the Command Interpreter Window (CIW).

### Starting EDIF 200 Out from a UNIX Prompt

From a UNIX prompt, you reference a template file. The template file contains preset values for the EDIF 200 Out options that you want to apply during the translation process. Template files are described later in this section.

- To reference a template file from a UNIX prompt, type the following:

```
edifout templateFile
```

where *templateFile* is the name of the specific template file.

- To reference a template file from a UNIX prompt and use a specific `cds.lib` file, type the following:

```
edifout templateFile -cdslib cdslibFile
```

This command tells the software to use a particular `cds.lib` file (named *cdslibFile*) instead of searching for the `cds.lib` file in the standard locations. For more information about `cds.lib` files, see [Cadence Library Path Editor User Guide](#).

## **Starting EDIF 200 Out from the CIW**

From the CIW, you can access the EDIF 200 Out form to specify the options you want to apply to the current design translation.

To access the EDIF 200 Out form,

1. Choose *File – Export – EDIF 200*.

**Note:** Before you try to translate a schematic, use the *Design – Check and Save* command from the schematic window to ensure connectivity. For more information, see *Virtuoso Schematic Editor User Guide*.

## Virtuoso EDIF 200 Reader and Writer User Guide

### EDIF 200 Out

---

The EDIF 200 Out form appears (top half).

EDIF 200 Out

Template File: i2b/lrx86/tools.lrx86/dfII/samples/xlUI/edif0ut.il

Load Save

Design: Browse

Run Directory: .

Library Name:

Cell Name:

View Name:

External Libraries:

Design Name:

User-Defined SKILL File:

Hierarchy File:

Stop Cell Expansion File:

Output File: edif.out

Technology File: default.tf

Output CDF Data ? ☒ No

OK Cancel Defaults Apply Help



## Virtuoso EDIF 200 Reader and Writer User Guide

### EDIF 200 Out

The EDIF Out form (bottom half).

The screenshot shows the 'EDIF 200 Out' dialog box with the following settings:

- Output CDF Data ? : ☒ No, ☐ Yes, as properties in the outputted EDIF file, ☐ Yes, to separate cdf data files
- ReplaceBundleWithArray : ☒ TRUE, ☐ FALSE
- Output Format : ☒ Schematic, ☐ Netlist
- Generate Scalar EDIF : ☒ FALSE, ☐ TRUE
- Expand All Objects : ☒ Expand All Objects, ☐ Expand All Objects Except Ports
- Inherited Connections : ☒ Use Default Value (Ignore Expressions), ☐ Interpret Expressions
- NetlistTranslationMode : ☒ Hierarchical, ☐ Flat
- Skip Duplicate Instance Name Check : ☐
- Flatten Run Directory :
- Ripper Library Name :
- Ripper Cell Name :
- Ripper View Name :

Buttons at the bottom: OK, Cancel, Defaults, Apply, Help.

**Note:** The `cds.lib` file replaces the library search path from earlier versions of EDIF 200 Out. The `cds.lib` file identifies the locations of your reference libraries. For more information, see [Cadence Library Path Editor User Guide](#).

#### 2. Type in the library, cell, and view names.

The cell and view names are optional. However, if you enter only a library name, the entire library is translated.

**Note:** You can also use the Browser to fill in this part of the form. Click *Browse*, then select the library, cell, and view names from the Browser to fill out the form.

3. Specify any options you want to apply during translation.

4. Do one of the following:

- ☐ Click *OK* to close the EDIF 200 Out form and begin the translation process.
- ☐ Click *Apply* to keep the EDIF 200 Out form open and begin the translation process.

EDIF 200 Out creates a log file called `edifout.log` in the run directory. When the translation is complete, an EDIF 200 file is created with the output filename you specified on the form. (You can specify a path as the output filename to create the output file in a directory other than the run directory.)

## Creating a Template File

A template file is a text file that contains preset values for the options you want to apply during an EDIF 200 Out translation.

There are three ways to create a template file:

- Click *Save* on the EDIF 200 Out form to save the values you specified on the EDIF 200 Out form. The information is saved to the file you enter in the *Template File* field.
- Use your local text editor to copy the sample template file to a new file and modify the new file as needed. The sample template file (`edifOut.il`) is located in the `your_install_dir/tools/dfII/samples/xlUI` directory.
- Use your local text editor to create a new template file, like the one shown in “[Sample Template File](#)” on page 51. The sample shows all the EDIF 200 Out variable names in the SKILL property list format.

## Required Information

An EDIF 200 Out template file must include the name of the DFII library containing the top-level cellview.

Although the following entries are not required, they are recommended:

- Name of the cell that maps to the cellview to be translated
- Name of the view that maps to the cellview to be translated

All other fields and values are optional.

## Sample Template File

The following sample shows a template file without most values. As described above, you can enter the values by editing a file or by saving the information you enter in the EDIF 200 Out form.

```
edifOutKeys = list( nil
  'runDirectory "."
  'library ""
  'cell ""
  'viewName ""
  'externalLibList ""
  'design ""
  'ILFile ""
  'hierarchyFile ""
  'stopCellExpansionFile ""
  'outputFile "edif.out"
  'techFile "default.tf"
  'outputCDFData "FALSE"
  'replaceBundleWithArray "TRUE"
  'netlistOption "TRUE"
  'interpretInhConn "FALSE"
  'scalarOption "FALSE"
  'scalarWithPortsUnexpanded "FALSE"
  'netlistMode "Hierarchical"
  'skipDupInstNameCk "FALSE"
  'flattenDir ""
  'ripperLibraryName ""
  'ripperCellName ""
  'ripperViewName ""
)
```

**Note:** The library name is required. The other fields are optional.

## Mapping Form Fields to Template File Entries

The following table maps the EDIF 200 Out form fields to their template file entries. (Form field descriptions begin in [“EDIF 200 Out Form Fields”](#) on page 70.)

### Mapping Form Fields to Template File Entries

---

EDIF 200 Out form field	EDIF 200 Out template file entry	Default value
<i>Run Directory</i>	'runDirectory	" . "
<i>Library Name</i>	'library	" "

## Virtuoso EDIF 200 Reader and Writer User Guide

### EDIF 200 Out

#### Mapping Form Fields to Template File Entries, *continued*

EDIF 200 Out form field	EDIF 200 Out template file entry	Default value
<i>Cell Name</i>	'cell	" "
<i>View Name</i>	'viewName	" "
<i>External Libraries</i>	'externalLibList	" "
<i>Design Name</i>	'design	" "
<i>IL Function File</i>	'ILFile	" "
<i>Hierarchy File</i>	'hierarchyFile	" "
<i>Stop Cell Expansion File</i>	'stopCellExpansionFile	" "
<i>Output File</i>	'outputFile	"edif.out"
<i>Technology File</i>	'techFile	"default.tf"
<i>Output CDF Data</i>	'outputCDFData	"FALSE"
<i>Replace Bundle With Array</i>	replaceBundleWithArray	"TRUE"
<i>Output Format</i>	'netlistOption	"TRUE"
<i>Schematic</i>		
<i>Netlist</i>		
<i>Generate Scalar EDIF</i>	'scalarOption	"FALSE"
<i>Inherited Connections</i>	'interpretInhConn	"FALSE"
<i>Scalar with Ports Unexpanded</i>	'scalarWithPortsUnexpanded	"FALSE"
<i>Netlist Translation Mode</i>	'netlistMode	"Hierarchical"
<i>Skip Duplicate Instance Name Check</i>	'skipDupInstNameCk	"FALSE"
<i>Flatten Run Directory</i>	'flattenDir	" "
<i>Ripper Library Name</i>	'ripperLibraryName	" "

## Virtuoso EDIF 200 Reader and Writer User Guide

### EDIF 200 Out

#### Mapping Form Fields to Template File Entries, *continued*

EDIF 200 Out form field	EDIF 200 Out template file entry	Default value
<i>Ripper Cell Name</i>	'ripperCellName	" "
<i>Ripper View Name</i>	'ripperViewName	" "

## Creating a Hierarchy File

A hierarchy file specifies the expansion rules of hierarchy that EDIF 200 Out uses. You can use a text editor to create this file. Each line of the hierarchy file contains a symbol name followed by one or more cellview names that tell EDIF 200 Out the order in which to look for cellviews when expanding the hierarchy.

### Hierarchy File Format

The format of the hierarchy file is as follows:

```
symbolViewName ViewRepName1 ViewRepName2
```

### Sample EDIF 200 Out Hierarchy File

```
symbol schematic cmos_sch nmos_sch
symbolNeg schematic cmos_sch nmos_sch
abstract layout symbolic
```

In the sample hierarchy file, the first line tells EDIF 200 Out to first look for `schematic`, then `cmos_sch`, and then `nmos_sch` when expanding the hierarchy of `symbol`.

The second line tells EDIF 200 Out to first look for `schematic`, then `cmos_sch`, and then `nmos_sch` when expanding the hierarchy of `symbolNeg`.

The third line tells EDIF 200 Out to first look for `layout` and then for `symbolic` when expanding the hierarchy of `abstract`.

**Note:** Attach the property `edifStop=on` to a cellview of a symbol to stop EDIF 200 Out at that instance level in the hierarchy and to include only the symbol of that cellview in the EDIF output file. Another way to stop cell expansion is, for each cell where you want expansion to stop, list these cell names in a file. Then specify that file in the *Stop Cell Expansion File* option. For each cell name listed, EDIF 200 Out does not perform expansion at the instance level. For more information, see [“Stop Cell Expansion File”](#) on page 99.

## How EDIF 200 Out Translates Data

Many objects in the DFII database correspond to constructs in the EDIF language. For example, in both EDIF and DFII, a cell can have many different views associated with it. The similarity between EDIF constructs and DFII objects makes it possible to map most EDIF constructs to DFII objects.



***There are important differences between the EDIF 200 constructs and the DFII objects. To prevent a loss of data, study the information that follows before you attempt a data conversion.***

## How EDIF 200 Out Translates Cells and Views

In the DFII database, a cell is a design object that can have a variety of views associated with it. Each view describes the cell from a different perspective. A view can be a *schematic*, a *symbol*, a *mask layout*, or a *simulation model*. The design process that is applied to a cell is dictated by the associated views.

The DFII cellview is the product of the mapping of cells and views in a library. There is a direct correspondence between the cell and cellview objects in the DFII environment and the `cell` and `view` constructs in the EDIF 200 language.

In the EDIF language, every view has an *interface* and a *contents* section. The cell interface section describes how the cell is viewed from outside the context of the cell. For example, the `symbol` construct, which is contained in the interface section of the cell, describes a symbol for a cell. The cell interface section also contains the cell *ports*. The contents section describes the view of the cell from within the context of the cell.

The DFII database separates the symbol definition from the internal definitions of a cell. The EDIF description of an indexed schematic, however, has a full interface and contents section in the resulting EDIF description.

**Note:** The previous discussion does not apply to parameterized views. EDIF 200 Out supports Level 0 only.

The following table shows how EDIF 200 Out maps DFII view types to EDIF view types.

## How EDIF 200 Out Translates Constructs

### Mapping DFII View Types to EDIF 200 View Types

---

DFII View Type	EDIF 200 View Type
graphic	GRAPHIC
masklayout	MASKLAYOUT <sup>1</sup>
symbolicLayout	MASKLAYOUT
schematic	SCHEMATIC
schematicSymbol	SCHEMATIC
netlist	NETLIST <sup>2</sup>
anything else	STRANGER

---

1. The MASKLAYOUT view is not fully supported.

2. You can also generate a NETLIST view by turning on the *Netlist Only* option on the EDIF 200 Out form.

The following table shows how EDIF 200 Out maps DFII objects to EDIF 200 Out constructs.

### Mapping DFII Objects to EDIF 200 Out Constructs

---

DFII Object	EDIF 200 Construct
any instance	instance
bounding box	boundingBox
bundle	netBundle
cell	cell
dot	dot
ellipse	circle
font height	textHeight
instance	viewRef
layer visibility	visible
layer-purpose pair definition	figureGroup, includeFigureGroup
library	external, library

## Virtuoso EDIF 200 Reader and Writer User Guide

### EDIF 200 Out

#### Mapping DFII Objects to EDIF 200 Out Constructs, *continued*

DFII Object	EDIF 200 Construct
line	path
net	net
	<b>Note:</b> Net names can have up to 8000 characters only.
net attributes	property
net-to-terminal definition	joined
path	path
path type	endType
path width	pathWidth
patch cord	ripper
permutable terminals	permutable
pin	portImplementation
polygon	polygon
property	annotate, designator, property
rectangle	rectangle
routing barrier	protectionFrame
scaling properties	scale
shape	figure
terminal	port, portBundle
terminal attributes	property
terminal direction	direction
text	annotate, designator, stringDisplay
text display parameters	display
transform	orientation, origin, transform
view	contents, view, interface



## How EDIF 200 Out Translates Connectivity

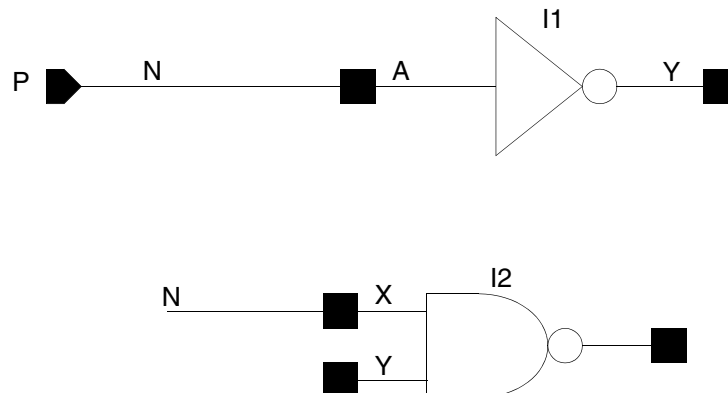
The DFII connectivity model is a subset of the EDIF connectivity model. EDIF 200 Out translates terminals to *ports*, pins to *portImplementations*, and arrayed nets to *netBundles* or net arrays.

When creating a *netlist* view of a multisheet schematic, EDIF 200 Out merges the connectivity of each sheet into one cell within the library for the schematic.

### Connectivity by Name

The terminals and instance terminals of a net in a schematic might not appear connected but are connected by naming each of the parts of the net with the same name, as shown in the following example:

#### Example of Defining Connectivity by Name



In the example, net N has two distinct parts, which EDIF Out translates as follows:

```
(net N
  (joined
    (portRef P)
    (portRef A (instanceRef I1))
    (portRef X (instanceRef I2))
  )
)
```

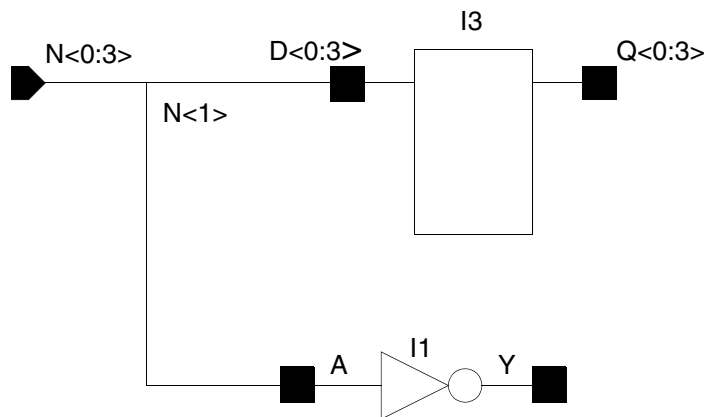
When the connection by name occurs between two different sized nets, such as the connection implied by a bus tap expression, the connection by name is complex. For information on bus tap and connection by name, refer to [Virtuoso Schematic Editor User Guide](#).

## Extracting a Signal

In the following example, the connection by name occurs between a bundle and a net which is created by extracting one signal. In EDIF 200, you extract a signal by using a special type of cell called a *ripper*.

The bundle N<0:3> is a 4-bit bus. The tap expression shows that the member N<1> is being extracted from the bus to form a net separate from the bus (even though both the bus and the net share a common signal).

### Example of Extracting a Signal



EDIF 200 Out produces the following output to describe the connectivity between the bundle N<0:3> and the extracted member net N<1>:

```
((instance splitter 1
(viewRef symbol (cellRef ripper_1 (libraryRef cdsRipLib_1))))
...
(netBundle (rename N_0_TO_3_ "N<0:3>")
(listOfNets
(net (rename N_0_ "N<0>")
(joined
(portRef (member N 0))
(portRef (member D 0) (instanceRef I3))))
(net (rename N_1_ "N<1>")
(joined
(portRef (member N 1))
(portRef (member src 0) (instanceRef splitter_1))
(portRef (member D 1) (instanceRef I3))))
(net (rename N_2_ "N<2>")
(joined
(portRef (member N 2))
(portRef (member D 2) (instanceRef I3))))
(net (rename N_3_ "N<3>")
(joined
(portRef (member N 3))
(portRef (member D 3) (instanceRef I3))))))
```

```
...
(net (rename N_1_ "N<1>")
(joined
(portRef (member dst 0) (instanceRef splitter_1))
(portRef A (instanceRef I1)))) ...
```

EDIF 200 Out inserts the instance `splitter_1` into the design so the connection between the bundle and the extracted net can be made. The ripper shows the exact relationship between the nets in question. The ripper cell provided by Cadence is a dot. EDIF 200 Out writes out a ripper cell to replace the dot. If you do not want to use the ripper cell provided by Cadence, you can specify the name of a default ripper in the template file by noting the library, cell, and view names.

## How EDIF 200 Out Translates Groups

You can use groups in the DFII environment to create relationships between IDs in the same view. These relationships are pertinent only within the DFII environment. EDIF uses arrays or lists to group objects together. No relationship is implied other than membership in an array or list. EDIF 200 Out does not support groups.

## How EDIF 200 Out Translates Labels

In general, labels are translated to an `annotate` construct within a `commentGraphics` construct.

Interpreted labels are handled differently. Interpreted labels display the value of a property. They frequently display reference designators and instance names. When interpreted labels are used on symbols or in pin figures, EDIF 200 Out writes a `keywordDisplay` for the `refDes` property.

## How EDIF 200 Out Translates Layers

Graphic information in views is mapped onto layers, which the technology file specifies as layer-purpose pairs. The technology file is attached to a specific library. Therefore, opening a library also opens a technology file.

Each DFII layer has a name, number, and purpose. Layer-purpose pair information cannot be expressed in EDIF. EDIF 200 Out writes the layer name in a `figureGroup` construct but does not write purpose information in `figureGroups`.

## How EDIF 200 Out Translates the Technology File

The technology file also includes scaling information for the distance units of a view. The system uses this unit property information to convert scaling information into the proper values for translation into EDIF. The EDIF `numberDefinition` construct describes scaling information for various physical units.

The `library` construct (which describes locally defined cells) and the `external` construct (which declares external libraries) both contain the `technology` construct.

## How EDIF 200 Out Translates Libraries

You specify the instance hierarchy in the template file or in the EDIF 200 Out form. EDIF 200 Out scans the instance hierarchy of the top-level cell to generate a library list. The cells in these libraries that are part of the instance hierarchy of the top-level cell are expanded unless the library is specified in the list of external libraries in either the template file or the run form. Libraries that contain expanded cells are written out as `library` constructs. Designated external libraries are written out as `external` constructs.

## How EDIF 200 Out Translates Mosaics

Mosaics represent an array of instances that can have one or more instance masters. There are two kinds of mosaics: *simple* and *complex*. EDIF 200 Out translates only simple mosaics.

A *simple* mosaic has the following traits:

- A regular array—all cells in the same row have the same height, and all cells in the same column have the same width.
- There is only one instance master.
- Every cell has the same orientation.

Any other mosaic is a complex mosaic.

To translate *complex* mosaics, you must first expand the complex mosaic into component instances, then translate the layout. If the layout contains many large and complex mosaics, the resulting EDIF file is large but contains *all* design information.

**Note:** EDIF 200 supports arrays within the `instance` construct. This provides a mapping for simple mosaics into EDIF.

## How EDIF 200 Out Translates Properties

Although both the DFII environment and EDIF support properties, they handle them differently. EDIF 200 Out can translate DFII properties attached to all object types. Almost every EDIF construct can contain EDIF properties.

There are four types of DFII properties:

- Scalar
- Range
- Enumerated
- Time

### Scalar Properties

Scalar properties can have one of eight value types. However, EDIF 200 In and EDIF 200 Out translate only four of those types.

The eight scalar property value types are Boolean, string, integer, floating-point, filename, nlp expression, SKILL expression, and time. The four value types that EDIF In and EDIF 200 Out translate into or out of the DFII environment are Boolean, string, floating-point, and integer.

You can use string values to represent the following value types: nlp expressions, SKILL expressions, and filename. EDIF 200 Out translates these value types into string values.

**Note:** To successfully transfer information, a user-defined property must be understood by both the sender and the receiver of the EDIF file.

### Range Properties

Range properties are expressed as integer, floating-point, or time values. EDIF 200 Out writes out range properties as `miNoMax` values.

Range properties contain upper and lower bounds that define the inclusive range of the property. The following example shows how to translate an integer-valued range property named `prop_Delay` with lower bound 0 (zero), value 33, and upper bound 100:

```
(property prop_Delay (miNoMax (mnm 0 33 100)))
```

## Enumerated Properties

Enumerated properties (`enum`) contain a string value and an enumerated set (list) of strings. Because enumerated properties imply their own type by the list of strings attached to the property, they are written out as string properties without any enumeration.

## Time Properties

Time properties contain a number that represents a calendar time. The number consists of the year, month, day, hour, minute, and second. EDIF uses the `timeStamp` construct to describe time. EDIF 200 Out converts the numerical time data into a string and writes out the property as a string value.

## How EDIF 200 Out Translates Reference Designators

The property `refDes` provides reference designator information to applications. Reference designators are used on instantiated objects, such as symbols and symbol pins, to denote information such as packaging names and pin numbers. EDIF 200 Out does not process the `refDes` property as a regular property but checks for the existence of the `refDes` property in `interface` and `port` constructs. If a `refDes` property exists, EDIF 200 Out writes out a `designator` construct to convey this information.

**Note:** For more information about displaying reference designators, see [“How EDIF 200 Out Translates Labels”](#) on page 59.

## How EDIF 200 Out Translates Shapes

Most shapes in DFII can be mapped directly into EDIF 200 shapes. However, DFII circles, lines, and donuts *cannot* be mapped directly into EDIF 200 shapes.

In the DFII environment, a circle is a special case of an ellipse, and EDIF 200 Out approximates the ellipse as two arcs whose endpoints are the major axes. EDIF 200 Out translates a line into a zero-width path. EDIF 200 Out decomposes a donut into a set of arcs and lines that form two halves of the donut. The coincident edges are along the vertical bisector of the donut.



***EDIF 200 Out does not translate arrow-ended paths.***

## How EDIF 200 Out Produces a Flattened Netlist

EDIF 200 Out translates a design hierarchically; it does not flatten the design directly. To produce a flattened netlist, you must flatten the design using an appropriate tool such as *PRFlatten* and then run EDIF 200 Out on the flattened design database.

**Note:** You must define symbolic pins in the library, otherwise *PRFlatten* might not produce correct results.

To create a flattened netlist in interactive mode, run EDIF 200 in flat mode so it automatically invokes *PRFlatten* before generating the final flattened EDIF netlist. This flat mode does not work in batch mode, so you must have a flat design database before you run `edifout`. Both methods are explained in more detail in the following sections.

## Producing a Flattened Netlist in Interactive Mode

In interactive mode (that is, within the DFII environment), you use the EDIF 200 Out form to control how EDIF 200 Out produces a flattened netlist. When you set the *Netlist Translation Mode* to *Flat*, EDIF 200 Out starts *PRFlatten* automatically. When *PRFlatten* is completed, EDIF 200 Out uses the *PRFlatten* output (the flattened design) as input. When your design data is organized according to DFII conventions, you do not have to provide any extra files to set up the flattening process.

**Note:** When Netlist Translation Mode is Flat, EDIF 200 Out requires one of the following licenses:

- ☐ 945 Virtuoso(R) EDIF 200 Writer
- ☐ 95310 Virtuoso(R) Layout Suite XL (or above for GXL, EXL, and EAD)
- ☐ 95321 Virtuoso(R) Layout Suite - GXL

Unavailability of these licenses results in the failure of EDIF 200 Out process.

When you use the EDIF 200 Out form to flatten the design, the default stop list is “symbol” and the default view list is “schematic symbol.” If the view names in the library or libraries do not correspond with the DFII defaults, you need to prepare a special `.simrc` file for *PRFlatten*. If *PRFlatten* does not find a `.simrc` file, it uses the DFII defaults to set up the run environment.

## Producing a Flattened Netlist in Batch Mode

Before you start an EDIF 200 Out batch run, you must manually run *PRFlatten* to flatten the design. When *PRFlatten* is finished, you can run an EDIF 200 Out batch job on the flattened design.

## EDIF 200 Out Output Files

In addition to a translated design data file (in which graphical input objects are mapped to textual output objects), EDIF 200 Out produces log messages. The messages are written to either a file or the screen, depending on how you started EDIF 200 Out.

- When you use the EDIF 200 Out form to start EDIF 200 Out, the software writes the log messages to the `CDS.log` file in your home directory and the `edifout.log` file in your run directory.
- When you start EDIF 200 Out by typing the `edifout` command at the UNIX prompt, the software displays the log messages on your screen and in the `edifout.log` file.

You can redirect the messages by piping the messages to a file. To pipe the messages to a file, type the following at the UNIX prompt:

```
edifout templateFile >& file.log
```

where *templateFile* is the name of the template file and *file.log* is the name of the log file you want to create.

**Note:** You can optionally create separate output files for CDF data. Refer to the description of the Output CDF Data field at the end of this chapter for more information.

## Error Messages

EDIF 200 Out reports two types of messages in the `edifout.log` file:

- Error messages, which indicate database objects that EDIF 200 Out cannot process
- Warning messages, which indicate recoverable problems

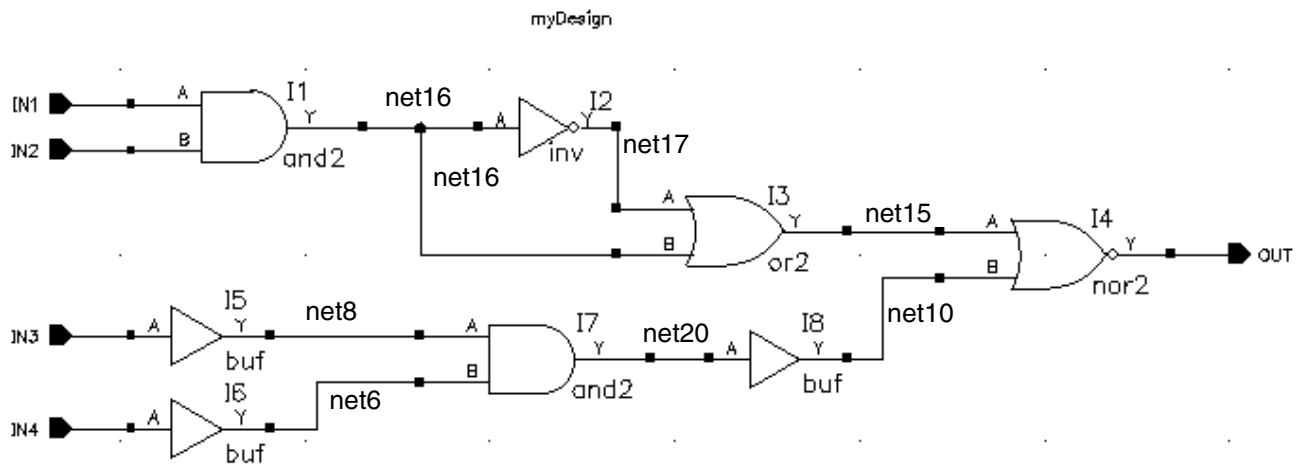
If error or warning messages appear in the `edifout.log` file, check the translated design data file before you use it.



## Translated Design Data File

The following examples show how EDIF 200 Out translates a design. The first example shows a typical schematic. The second example shows the textual output file generated by EDIF 200 Out after it translates the schematic data.

### A Typical Schematic



### Translated Design Data File for the Sample Schematic

```
(edif CADENCE_EDIF
  (edifVersion 2 0 0)
  (edifLevel 0)
  (keywordMap (keywordLevel 0))
  (status
    (written (timeStamp 1995 2 28 3 24 10)
      (data Origin "myMachine")
      (program "edifout" (version "EXP Fri Feb 10 09:55:36
        PST 1995 (cds9481)")))))
  (external sample
    (EDIFLEVEL 0)
    (technology (numberDefinition))
    (cell and2 (cellType GENERIC)
      (view symbol (viewType SCHEMATIC)
        (interface
          (port Y (direction OUTPUT))
          (port B (direction INPUT))
          (port A (direction INPUT)))))
    (cell inv (cellType GENERIC)
      (view symbol (viewType SCHEMATIC)
        (interface
          (port Y (direction OUTPUT))
          (port A (direction INPUT)))))
    (cell or2 (cellType GENERIC)
      (view symbol (viewType SCHEMATIC)
        (interface
```

# Virtuoso EDIF 200 Reader and Writer User Guide

## EDIF 200 Out

---

```
(port Y (direction OUTPUT))
(port B (direction INPUT))
(port A (direction INPUT))))
(cell nor2 (cellType GENERIC)
  (view symbol (viewType SCHEMATIC)
    (interface
      (port Y (direction OUTPUT))
      (port B (direction INPUT))
      (port A (direction INPUT))))))
(cell buffer (cellType GENERIC)
  (view symbol (viewType SCHEMATIC)
    (interface
      (port Y (direction OUTPUT))
      (port A (direction INPUT))))))
(cell buf (cellType GENERIC)
  (view symbol (viewType SCHEMATIC)
    (interface
      (port Y (direction OUTPUT))
      (port A (direction INPUT))))))
(external basic
  (EDIFLEVEL 0)
  (technology (numberDefinition))
  (cell ipin (cellType GENERIC)
    (view symbol (viewType GRAPHIC)
      (interface)))
  (cell opin (cellType GENERIC)
    (view symbol (viewType GRAPHIC)
      (interface)))
(library design
  (edifLevel 0)
  (technology
    (numberDefinition
      (scale 160 (E 25400 -6) (unit DISTANCE)))
    (figureGroup marker
      (color 100 100 0)
      (visible (true))
      (property layerNumber (integer 238))))
  (cell document_exam (cellType GENERIC)
    (view schematic (viewType SCHEMATIC)
      (interface
        (port IN1 (direction INPUT))
        (port IN2 (direction INPUT))
        (port IN3 (direction INPUT))
        (port IN4 (direction INPUT))
        (port OUT (direction OUTPUT))
        (property schGeometryLastRecorded
          (string "Mon Feb 27 19:17:41 1995%10%"))
        (property schGeometryVersion
          (string "sch.ds.gm.1.4"))
        (property lastSchematicExtraction
          (string "Mon Feb 27 19:17:41 1995%10%"
            (owner "Cadence")))
        (property schXtrVersion (string "sch.9.01")
          (owner "Cadence"))
        (property (rename net_35_ "net#")
          (integer 40))
        (property (rename pin_35_ "pin#")
          (integer 6) (owner "Cadence"))
        (property (rename instance_35_ "instance#")
          (integer 14) (owner "Cadence"))
```

## Virtuoso EDIF 200 Reader and Writer User Guide

### EDIF 200 Out

---

```
(property instancesLastChanged
  (string "Mon Feb 27 19:17:40 1995%10%")
  (owner "Cadence"))
(contents
  (page SH1
    (instance I1
      (viewRef symbol (cellRef and2 (libraryRef sample)))
      (transform (origin (pt -310 110)))))
    (instance I2
      (viewRef symbol (cellRef inv (libraryRef sample)))
      (transform (origin (pt -10 110)))))
    (instance I3
      (viewRef symbol (cellRef or2 (libraryRef sample)))
      (transform (origin (pt 110 20)))))
    (instance I4
      (viewRef symbol (cellRef nor2 (libraryRef sample)))
      (transform (origin (pt 390 0)))))
    (instance I5
      (viewRef symbol (cellRef buf (libraryRef sample)))
      (transform (origin (pt -310 -70)))))
    (instance I6
      (viewRef symbol (cellRef buf (libraryRef sample)))
      (transform (origin (pt -310 -150)))))
    (instance I7
      (viewRef symbol (cellRef and2 (libraryRef sample)))
      (transform (origin (pt -60 -90)))))
    (instance I8
      (viewRef symbol (cellRef buffer (libraryRef sample)))
      (transform (origin (pt 190 -90)))))
      (commentGraphics (annotate (stringDisplay "myDesign"
        (display (figureGroupOverride text (textHeight 10))
          (justify LOWERCENTER)
          (orientation R0)
          (origin (pt 70 200)))))))
    (portImplementation OUT
      (connectLocation
        (figure pin (dot (pt 640 0)))))
    (instance I13
      (viewRef symbol (cellRef opin (libraryRef basic)))
      (transform (origin (pt 640 0)))))
    (portImplementation IN4
      (connectLocation
        (figure pin (dot (pt -360 -150)))))
    (instance I12
      (viewRef symbol (cellRef ipin (libraryRef basic)))
      (transform (origin (pt -360 -150)))))
    (portImplementation IN3
      (connectLocation
        (figure pin (dot (pt -360 -70)))))
    (instance I11
      (viewRef symbol (cellRef ipin (libraryRef basic)))
      (transform (origin (pt -360 -70)))))
    (portImplementation IN2
      (connectLocation
        (figure pin (dot (pt -360 90)))))
    (instance I10
      (viewRef symbol (cellRef ipin (libraryRef basic)))
      (transform (origin (pt -360 90)))))
    (portImplementation IN1
      (connectLocation
        (figure pin (dot (pt -360 130)))))
```

## Virtuoso EDIF 200 Reader and Writer User Guide

### EDIF 200 Out

---

```
(instance I9
  (viewRef symbol (cellRef ipin (libraryRef basic)))
  (transform (origin (pt -360 130)))))
(net net6 (joined
  (portRef B (instanceRef I7))
  (portRef Y (instanceRef I6)))
  (figure wire (path (pointList
    (pt -150 -110) (pt -60 -110))))
  (figure wire (path (pointList
    (pt -150 -150) (pt -150 -110))))
  (figure wire (path (pointList
    (pt -190 -150) (pt -150 -150)))))
(net IN1 (joined
  (portRef IN1)
  (portRef A (instanceRef I1)))
  (figure wire (path (pointList
    (pt -360 130) (pt -310 130)))))
(net IN2 (joined
  (portRef IN2)
  (portRef B (instanceRef I1)))
  (figure wire (path (pointList
    (pt -360 90) (pt -310 90)))))
(net OUT (joined
  (portRef OUT)
  (portRef Y (instanceRef I4)))
  (figure wire (path (pointList
    (pt 590 0) (pt 640 0)))))
(net IN3 (joined
  (portRef IN3)
  (portRef A (instanceRef I5)))
  (figure wire (path (pointList
    (pt -360 -70) (pt -310 -70)))))
(net net8 (joined
  (portRef A (instanceRef I7))
  (portRef Y (instanceRef I5)))
  (figure wire (path (pointList
    (pt -190 -70) (pt -60 -70)))))
(net net10 (joined
  (portRef B (instanceRef I4))
  (portRef Y (instanceRef I8)))
  (figure wire (path (pointList
    (pt 340 -20) (pt 390 -20))))
  (figure wire (path (pointList
    (pt 340 -90) (pt 340 -20))))
  (figure wire (path (pointList
    (pt 310 -90) (pt 340 -90)))))
(net net15 (joined
  (portRef Y (instanceRef I3))
  (portRef A (instanceRef I4)))
  (figure wire (path (pointList
    (pt 310 20) (pt 390 20)))))
(net net16 (joined
  (portRef Y (instanceRef I1))
  (portRef A (instanceRef I2))
  (portRef B (instanceRef I3)))
  (figure wire (path (pointList
    (pt -60 0) (pt 110 0))))
  (figure wire (path (pointList
    (pt -60 0) (pt -60 110))))
  (figure wire (path (pointList
    (pt -60 110) (pt -10 110)))))
```

## Virtuoso EDIF 200 Reader and Writer User Guide

### EDIF 200 Out

---

```
(figure wire (path (pointList
  (pt -110 110) (pt -60 110))))))
(net net17 (joined
  (portRef Y (instanceRef I2))
  (portRef A (instanceRef I3)))
(figure wire (path (pointList
  (pt 110 40) (pt 110 110))))))
(net net20 (joined
  (portRef A (instanceRef I8))
  (portRef Y (instanceRef I7)))
(figure wire (path (pointList
  (pt 140 -90) (pt 190 -90))))))
(net IN4 (joined
  (portRef IN4)
  (portRef A (instanceRef I6)))
(figure wire (path (pointList
  (pt -360 -150) (pt -310 -150))))))
)
)
```

## EDIF 200 Out Form Fields

### Top Half

The screenshot shows a dialog box titled "EDIF 200 Out". It contains the following fields and controls:

- Template File:** A text field containing the path `!2b/lnx86/tools.lnx86/dfII/samples/xlUI/edif0ut.il`. Below it are "Load" and "Save" buttons.
- Design:** A text field with a "Browse" button next to it.
- Run Directory:** A text field containing a single period `.`
- Library Name:** An empty text field.
- Cell Name:** An empty text field.
- View Name:** An empty text field.
- External Libraries:** An empty text field.
- Design Name:** An empty text field.
- User-Defined SKILL File:** An empty text field.
- Hierarchy File:** An empty text field.
- Stop Cell Expansion File:** An empty text field.
- Output File:** A text field containing `edif.out`.
- Technology File:** A text field containing `default.tf`.
- Output CDF Data ?** A radio button group with the "No" option selected.

At the bottom of the dialog are five buttons: "OK" (highlighted in red), "Cancel", "Defaults", "Apply", and "Help".

## Virtuoso EDIF 200 Reader and Writer User Guide

### EDIF 200 Out

#### Bottom Half

The screenshot shows the 'EDIF 200 Out' dialog box with the following options:

- Output CDF Data ?**
  - ☒ No
  - ☐ Yes, as properties in the outputted EDIF file
  - ☐ Yes, to separate cdf data files
- ReplaceBundleWithArray** ☒ TRUE ☐ FALSE
- Output Format** ☒ Schematic ☐ Netlist
- Generate Scalar EDIF**
  - ☒ FALSE
  - ☐ TRUE
    - ☒ Expand All Objects
    - ☐ Expand All Objects Except Ports
- Inherited Connections**
  - ☒ Use Default Value (Ignore Expressions)
  - ☐ Interpret Expressions
- NetlistTranslationMode** ☒ Hierarchical ☐ Flat
- Skip Duplicate Instance Name Check** ☐
- Flatten Run Directory**
- Ripper Library Name**
- Ripper Cell Name**
- Ripper View Name**

Buttons at the bottom: **OK** (highlighted in red), Cancel, Defaults, Apply, Help.

**Template File** is the name of a text file that contains preset values for the options you want to apply during an EDIF 200 Out translation process.

**Load** button reads the template file and applies its contents (field names and values) to the EDIF 200 Out form.

**Save** button creates a template file that contains the filenames and option values you specify in the form. The information is written to the filename you specify in the *Template File* field.

**Design Browse** button opens the Browser. The Browser automatically seeds the form with the library, cell, and view names that you select.

**Run Directory** is the directory where output files are created. The default is the current working directory.

**Library Name** is name of the DFI library that contains the cellview that you want to translate into EDIF 200 format.

**Cell Name** is the name of the cell that you want to translate into EDIF 200 format.

**View Name** is the view type of the cell that you want to translate into EDIF 200 format.

**External Libraries** specifies the names of libraries to translate as external constructs. External libraries are not expanded.

**Design Name** is the name you want to give to the EDIF design construct.

**User-Defined SKILL File** is a file that contains user-defined SKILL procedures for EDIF 200 Out to use during translation.

**Hierarchy File** is the name of a text file that contains the names of cellviews that EDIF 200 Out scans as it expands the instance hierarchy of the top-level cellview. See [“Creating a Hierarchy File”](#) on page 53 for more information.

**Stop Cell Expansion File** lets you specify a file that contains one or more cell names. Use a single space to separate each cell name. The file format is:

```
/* Library_Name Cell_Name1 Cell_Name2... for each line*/
sample1 inv and2 nand2
sample2 BUF MUX
```

For each cell specified in this file, EDIF 200 Out stops the expansion at the symbol level. In other words, EDIF 200 Out does not traverse down its schematic contents. This option works the same as attaching the `edifStop=on` property on the symbol view of the cell. See [“Creating a Hierarchy File”](#) on page 53 for more information.

**Output File** is the name you want to assign to the EDIF 200 file. The default is `edif.out`.

**Technology File** is the name of the technology file. A technology file sets the library configuration. If you do not specify a technology file, EDIF 200 Out uses the current library configuration.

**Output CDF Data** dumps the CDF information for your design into separate files and puts a marker in the EDIF 200 Out file that indicates the names of these CDF information files. If you have CDF information at the library level, the CDF file is named `libName_lib.cdf`. If you also have CDF information for a cell, an additional CDF file is created and named



*cellName.cdf*. If you have multiple cells with CDF information, the CDF files are named *cellName1.cdf*, *cellName2.cdf*, and so on. You can have one or any combination of these CDF files. If you select this option for a design with no CDF information, no files are created. The default is off.

**Note:** This option lets you store a Cadence design with CDF data referenced by the EDIF output file. You can later recall the design back into the Cadence environment with EDIF 200 In. The CDF data is reapplied to the design at this time. (The CDF data files must be in the run directory to be reapplied.)

**ReplaceBundleWithArray** specifies whether EDIF 200 Out replaces Virtuoso® Schematic Composer net bundles and port bundles with arrayed nets. The default is *TRUE*.

**TRUE** replaces composer net bundles and port bundles with EDIF net array and port array constructs

**FALSE** does not replace net bundles and port bundles with net array and port array constructs.

**Output Format** specifies which EDIF view (or format) EDIF 200 Out writes. The default is *Schematic*.

**Schematic** lets EDIF 200 Out translate the view to the cellview of the cell you want to translate. If that cellview is not supported, the view type is set to *stranger*. EDIF 200 Out issues an error message if both this option and *Interpret Expressions* of inherited connections are on at the same time. (See *Inherited Connections*, below.)

**Netlist** generates a netlist view only. In other words, graphical information is not translated. This option must be on if you want EDIF 200 Out to expand net expressions and create resolved nets and connectivities. (See *Inherited Connections*, below.)

**Generate Scalar EDIF** expands all the multiple-bit objects to single-bit (scalar) objects and suppresses the generation of ripper cells in the EDIF file. For example, a port named A<0:3> becomes three ports in the output EDIF file: A<0>, A<1>, and A<3>.

**FALSE** generates regular EDIF 200 Out format, which sustains the original width of objects in the design.

**TRUE** automatically turns on the *Netlist* output format option because the scalar option cannot support *Schematic* output format. *TRUE* works with either one of the following options:

**Expand All Objects** expands all objects in the design into single bit.

**Expand All Objects Except Ports** expands all objects in the design except for ports. In other words, this option keeps the original form for ports, but expands all other objects, such as nets and instances.

**Inherited Connections** tells EDIF 200 Out how to handle the inherited connections features in the design. For the usage and mechanism of inherited connections in a design, refer to the *Virtuoso Schematic Editor User Guide*.

**Use Default Value (Ignore Expressions)** lets EDIF 200 Out ignore all the net expressions. In other words, EDIF 200 Out works as if there are no net expressions in the design.

**Note:** The *Use Default Value* option is required for translating a design into an EDIF 200 *Schematic* view because expansion of net expressions is not supported for EDIF 200 *Schematic* view output.

**Interpret Expressions** lets EDIF 200 Out expand net expressions and create resolved nets and connectivities. This option works only with the *Netlist* output format option. EDIF 200 Out issues an error message if both this option and *Schematic* output format are on at the same time.

**Netlist Translation Mode** determines whether the design is flattened before translation. (This option is available when the *Netlist* output format option is on.)

**Hierarchical** produces a hierarchical netlist.

**Flat** produces a flattened netlist.

**Skip Duplicate Instance Name Check** lets you skip a duplicate instance name check. The default is off.

**Flatten Run Directory** is the name of a directory where you want the software to temporarily store the files for *PRFlatten*. The default is the run directory (.).

**Ripper Library Name** is the library that contains the ripper cell. The default is `basic`.

**Ripper Cell Name** is the name of the ripper cell. The default is `patch`.

**Ripper View Name** is the name of the ripper cell cellview. The default is `symbol`.