IC Compiler™ II
Data Model
User Guide

Version O-2018.06, June 2018

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Synopsys® Free and Open-Source Licensing Notices for IC Compiler™ II Products

Contents

Document Overview

This document includes licensing information relating to free and open-source software ("FOSS") included with Synopsys's® IC Compiler II™ products (the "SOFTWARE"). The terms of the applicable FOSS license(s) govern Synopsys's® distribution and your use of the SOFTWARE. Synopsys® and the third-party authors, licensors, and distributors of the SOFTWARE disclaim all warranties and all liability arising from any and all use and distribution of the SOFTWARE. To the extent the FOSS is provided under an agreement with Synopsys® that differs from the applicable FOSS license(s), those terms are offered by Synopsys® alone.

Synopsys® has reproduced below copyright and other licensing notices appearing within the FOSS packages. While Synopsys® seeks to provide complete and accurate copyright and licensing information for each FOSS package, Synopsys® does not represent or warrant that the following information is complete, correct, or error-free. SOFTWARE recipients are encouraged to (a) investigate the identified FOSS packages to confirm the accuracy of the licensing information provided herein and (b) notify Synopsys® of any inaccuracies or errors found in this document so that Synopsys® may update this document accordingly.

Certain FOSS licenses (such as the GNU General Public License, GNU Lesser General Public License, and Eclipse Public License) require that Synopsys® make the source code corresponding to the distributed FOSS binaries available under the terms of the same FOSS license. Recipients who would like to receive a copy of such corresponding source code should submit a request to Synopsys® by post at:

Synopsys
Attn: Open Source Requests
690 East Middlefield Road
Mountain View, CA 94043

Please provide the following information in all submitted FOSS requests:
- the FOSS packages for which you are requesting source code;
- the Synopsys® product (and any available version information) with which the requested FOSS packages are distributed;
- an email address at which Synopsys® may contact you regarding the request (if available); and
- the postal address for delivery of the requested source code.

FOSS Package Notices

GNU Bison 3.0.4

Project homepage: https://www.gnu.org/software/bison/
Project license:

Copyright © 1998-2013 Free Software Foundation, Inc. This file is part of Bison, the GNU Compiler Compiler.

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Preamble
The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.
To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps:(1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS
0. Definitions.
"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.
The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source. The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.
All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.
No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.
You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.
You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a)The work must carry prominent notices stating that you modified it, and giving a relevant date.

b)The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c)You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d)If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.
You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a)Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b)Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c)Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d)Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e)Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.
"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.
You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.
You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.
Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.
A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.
If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.
Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.
The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.
THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.
IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.
If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs
If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>

This program is free software: you can redistribute it and/or modify it under the terms of the
GNU General Public License as published by the Free Software Foundation, either version 3 of
the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without
even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
General Public License for more details.
You should have received a copy of the GNU General Public License along with this program. If
not, see <http://www.gnu.org/licenses/>.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program> Copyright (C) <year> <name of author>
This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software,
and you are welcome to redistribute it under certain conditions; type `show c' for details.
```

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

Flex 2.5.37

Project homepage: http://flex.sourceforge.net/
Project license:

LEF/DEF Reader 5.8-p008

Project homepage: http://projects.si2.org/openeda.si2.org/projects/lefdef
Project license:

1. DEFINITIONS

1.1 "Contribution" means the submission of the Original Program by Cadence and the submission of any modifications, improvements, additions, corrections or other changes to the Program, including source code, object code, application program interface definitions and formats, and documentation, made by any Contributor where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. Each Contributor acknowledges and agrees that no guarantee is provided that a Contribution shall be included within the Program as distributed by any Recipient. A Contribution must be made by a Contributor if; (1) such Contribution is an error correction to the Program, or (2) the Contributor commercially distributes the Program with such Contribution, in which case the Contribution must be made within thirty (30) days of the commercial distribution of the Program. “Commercially distributes” as used herein shall mean distribution of the Program by a Contributor in consideration for revenue received. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software or data distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

1.2 "Contributor" means Cadence and any other entity that distributes the Program.

1.3 "Licensed Patents" mean patent claims licensable by any Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

1.4 "Original Program" means the original version of the software accompanying this Agreement as released by Cadence, including source code, object code, application program interface definitions and formats, and documentation, if any.

1.5 "Program" means the Original Program and Contributions.

1.6 "Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

2.1 Subject to the terms and conditions of this Agreement, each Contributor hereby grants each Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, in source code and object code format.

2.2 Subject to the terms and conditions of this Agreement, each Contributor hereby grants each Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code format along with any documentation. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution.

2.3 Subject to the terms and conditions of this Agreement, Cadence hereby grants each Recipient a non-exclusive, worldwide, royalty-free trademark license, to use the Cadence trademarks “LEF”, or “DEF”, in accordance with Cadence’s marking policies, in connection with the license grants hereunder provided, however, that use of the Cadence trademarks in connection with Recipient’s distribution of the Program is only permitted for the unmodified Program.

2.4 Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

2.5 Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement. Except as expressly stated in Sections 2.1, 2.2 and 2.3 above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

3. DISTRIBUTION REQUIREMENTS

3.1 A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that: (a) it complies with the terms and conditions of this Agreement; and (b) the terms and conditions of its license agreement: (i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose; (ii) effectively excludes on behalf of all Contributors all liability for damages, including, but not limited to, direct, indirect, special, incidental and consequential damages; (iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and (iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

3.2 When the Program is made available in source code form it must be made available under the terms and conditions of this Agreement. A copy of this Agreement must be included with each copy of such Program. A Contributor may not charge a fee for the distribution of the source code of the Program.

3.3 Each Contributor must preserve all copyright and other notices that appear in the Program. In addition, each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

3.4 While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

4. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

5. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

6. GENERAL

6.1 If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

6.2 If Recipient institutes patent litigation against a Contributor with respect to a patent applicable to software (including a cross-claim or counterclaim in a lawsuit), then any patent licenses granted by that Contributor to such Recipient under this Agreement shall terminate as of the date such litigation is filed. If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2.2 shall terminate as of the date such litigation is filed.

6.3 If a Commercial Contributor distributes any Program, which includes such Contributors modifications, improvements, additions, corrections or other changes, and has not made a Contribution of such Program, any licenses granted to such Commercial Contributor under this Agreement shall terminate as of the date of such Program is distributed.

6.4 All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

6.5 Cadence may publish new versions (including revisions) of this Agreement from time to time. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. No one other than Cadence has the right to modify this Agreement.

6.6 This Agreement is governed by the laws of the State of California and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

END OF TERMS AND CONDITIONS

IN WITNESS WHEREOF, THE PARTIES HERETO HAVE ENTERED INTO THIS AGREEMENT AS OF THE EFFECTIVE DATE.
RECIPIENT NAME:

____
Company Name if Corporate License
Signed by: ________________________
Name: ___________________________
Title: ____________________________
Mailing Address: __________________
____
Phone Number: ___________________
e-mail address: ___________________

mcpp 2.7.2
Project homepage: http://sourceforge.net/projects/mcpp/
Project license:

This software including the files in this directory is provided under the following license.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Oasis Wrapper 1.0.0
Project homepage:
Project license:

Threading Building Blocks 4.4.0
Project homepage: https://www.threadingbuildingblocks.org/
Project license: https://www.threadingbuildingblocks.org/licensing

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
   a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
   b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
   c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.
   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
   a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein.
You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.
   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11.BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12.IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs
If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the
GNU General Public License as published by the Free Software Foundation; either version 2 of
the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without
even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if
not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
02110-1301 USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it under certain conditions; type
`show c' for details.
```

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program Gnomovision' (which makes
passes at compilers) written by James Hacker.
<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice
```

TCL/TK 8.5.12

Project homepage: https://www.tcl.tk/
Project license: https://www.tcl.tk/software/tcltk/license.html

zlib 12.5.3

Project homepage: http://www.zlib.net/
Project license: http://www.zlib.net/zlib_license.html

Jean-loup Gailly   Mark Adler
jloup@gzip.org   madler@alumni.caltech.edu

libedit
Project homepage: http://netbsd.org/
Project license:

Contents

Preface

This preface includes the following sections:

- About This User Guide
- Customer Support

About This User Guide

The Synopsys IC Compiler II tool provides a complete netlist-to-GDSII or netlist-to-OASIS® design solution, which combines proprietary design planning, physical synthesis, clock tree synthesis, and routing for logical and physical design implementations throughout the design flow.

This guide describes the creation, access, editing, and storage of design data using the IC Compiler II implementation and library manager tools. For more information about these tools, see the following companion volumes:

- *IC Compiler II Library Preparation User Guide*
- *IC Compiler II Design Planning User Guide*
- *IC Compiler II Implementation User Guide*
- *IC Compiler II Timing Analysis User Guide*
- *IC Compiler II Graphical User Interface User Guide*

Audience

This user guide is for design engineers who use the IC Compiler II library manager and implementation tools to prepare libraries and implement designs.

To use the IC Compiler II tool, you need to be skilled in physical design and synthesis, and be familiar with the following:

- Physical design principles
- The Linux or UNIX operating system
- The tool command language (Tcl)

Related Publications

For additional information about the IC Compiler II tool, see the documentation on the Synopsys SolvNet® online support site at the following address:

https://solvnet.synopsys.com/DocsOnWeb

You might also want to see the documentation for the following related Synopsys products:

- Design Compiler®
- IC Validator
- PrimeTime® Suite

Release Notes

Information about new features, enhancements, changes, known limitations, and resolved Synopsys Technical Action Requests (STARs) is available in the *IC Compiler II Release Notes* on the SolvNet site.

To see the *IC Compiler II Release Notes*,

1. Go to the SolvNet Download Center located at the following address:

   https://solvnet.synopsys.com/DownloadCenter

2. Select IC Compiler II, and then select a release in the list that appears.

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates syntax, such as `write_file`.
`Courier italic`	Indicates a user-defined value in syntax, such as `write_file` *`design_list`*.
`Courier bold`	Indicates user input—text you type verbatim—in examples, such as `prompt>` **`write_file top`**
[ ]	Denotes optional arguments in syntax, such as `write_file [-format` *`fmt`*`]`
...	Indicates that arguments can be repeated as many times as needed, such as *`pin1 pin2 ... pinN`*
|	Indicates a choice among alternatives, such as `low | medium | high`
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl key and pressing C.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

The SolvNet site includes a knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The SolvNet site also gives you access to a wide range of Synopsys online services including software downloads, documentation, and technical support.

To access the SolvNet site, go to the following address:

https://solvnet.synopsys.com

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to sign up for an account.

If you need help using the SolvNet site, click HELP in the top-right menu bar.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a support case to your local support center online by signing in to the SolvNet site at https://solvnet.synopsys.com, clicking Support, and then clicking “Open A Support Case.”
- Send an e-mail message to your local support center.
  - E-mail support_center@synopsys.com from within North America.
  - Find other local support center e-mail addresses at

    https://www.synopsys.com/support/global-support-centers.html
- Telephone your local support center.
  - Call (800) 245-8005 from within North America.
  - Find other local support center telephone numbers at

    https://www.synopsys.com/support/global-support-centers.html

1

Libraries and Blocks

A *block* is a container for physical and functional design data. A *library* is a collection of related blocks, together with technology information that applies to the block collection. A chip design consists of one or more blocks, often stored in different *design libraries*. A design library uses instances of blocks defined in lower-level libraries, called *reference libraries*. A design library can serve as a reference library for another design library.

The following topics describe the usage of libraries and blocks:

- Design Libraries
- Cell Libraries
- Reference Library List
- Loading the Technology Data
- Blocks
- Block Types
- Sparse Libraries
- File Attachments
- Library Packaging

Design Libraries

A *design library* stores all of the information about a design, including the associated technology information and the reference library setup information. In addition to the technology and reference library setup information, a design library contains various versions of the designs stored in the design library. A version of a design is referred to as a *block*. The block name can be the same as or different than the top module name of the design. For example, for a design with a top module named my_design, you could have blocks named my_design_preplace, my_design_postplace, and my_design_postcts.

For each block, the design library can contain one or more of the following views:

- Design view

  The design view contains the layout information for the block.

- Frame view

  The frame view is an abstraction of the layout view that contains only the information needed for placement and routing. The exclusion of unnecessary data reduces the database size and routing time. For information about generating frame views, see the *IC Compiler II Library Preparation User Guide*.

- Outline view

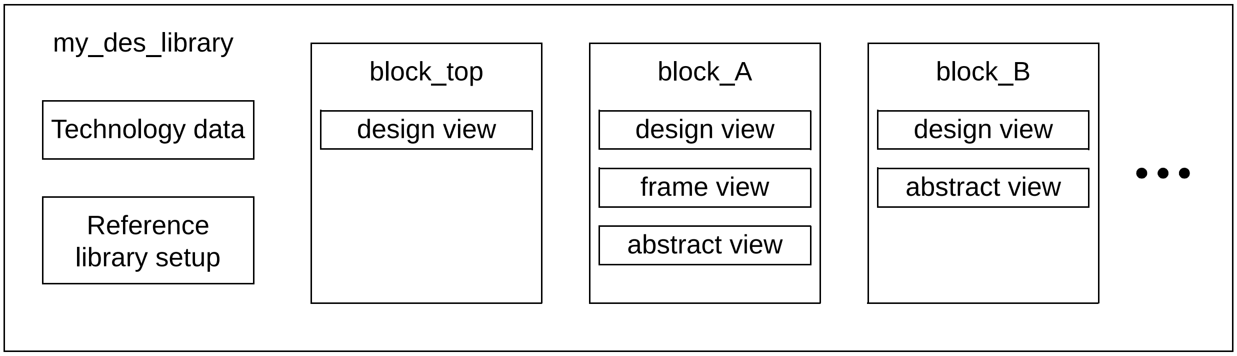
  The outline view is used for floorplan creation for very large hierarchical blocks. It contains the hierarchy information, but no leaf cells or nets. For information about creating and using the outline view, see the *IC Compiler II Design Planning User Guide*.

- Abstract view

  The abstract view is a lightweight representation of the block that contains only the information needed to perform top-level global placement, timing, and other tasks. For information about creating and using the abstract view, see the *IC Compiler II Design Planning User Guide*.

Figure 1-1 shows a high-level view of the contents of a typical design library.

Figure 1-1 Design Library Contents

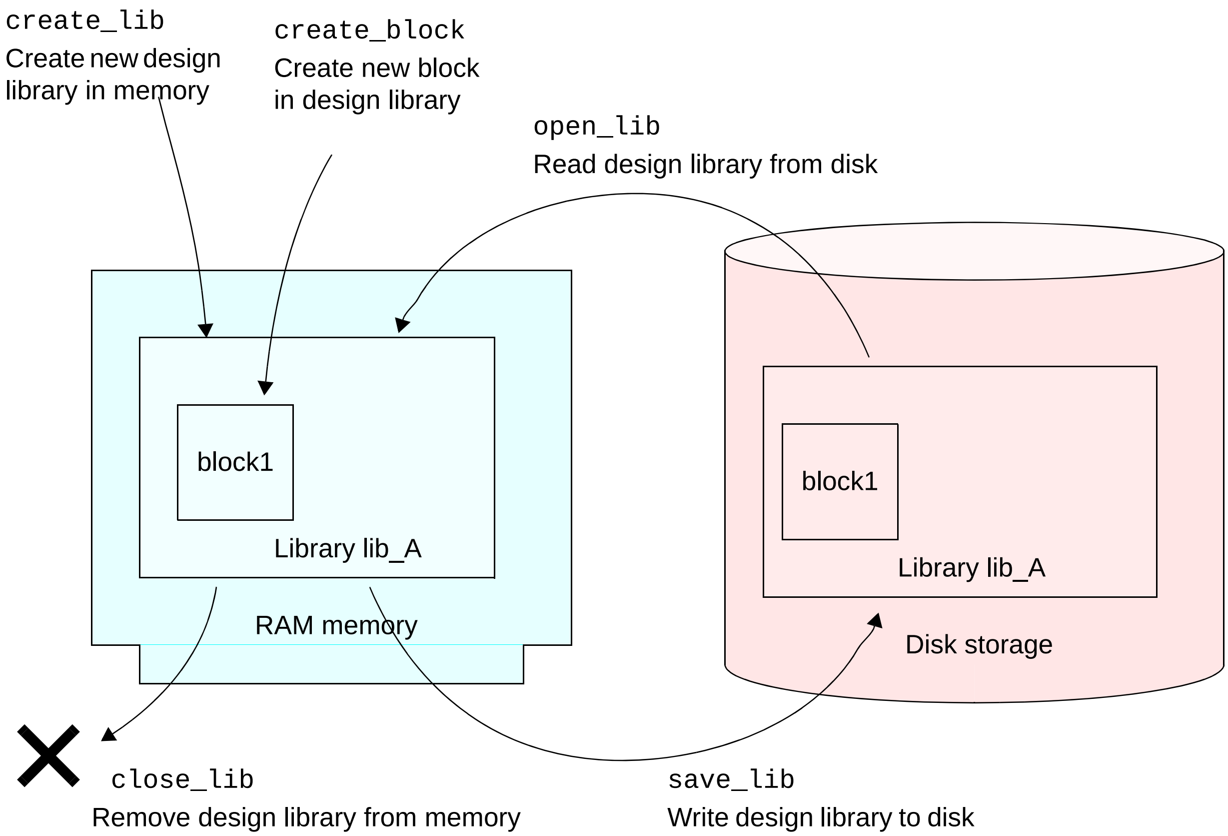


A design library can contain one or more different designs, or a complete hierarchical design that includes all views of the design. You can store the parts of a design in a single design library or in multiple design libraries that share the same technology information.

The *current library* is the default design library affected by library-related commands. By default, the last library opened is the current library. You can explicitly set the current library by using the `current_lib` command.

The `create_lib` command creates a new design library and optionally specifies the associated technology data and reference library setup. You create blocks in a design library by using the `create_block` command. The `save_lib` command saves an open design library to disk and the `open_lib` command opens a saved design library for block access, as shown in the following figure.

Figure 1-2 Creating, Opening, Editing, Saving, and Closing a Library



Each design library has a *reference library list* that points to other libraries containing blocks referenced in the design library. In other words, a block that you edit in the design library contains *instances* of blocks defined in other libraries. The referenced blocks can be leaf-level library blocks in cell libraries or soft macros, intellectual property (IP) blocks, or other types of blocks contained in other design libraries.

The following table briefly describes the commands that operate on design libraries.

Table 1-1 Commands That Operate on Design Libraries

Command	Description
`close_lib`	Closes a library, removing it from memory
`copy_lib`	Copies a design library on disk
`create_lib`	Creates a design library in memory
`current_lib`	Sets or gets the current library
`move_lib`	Moves a library on disk
`open_lib`	Opens a saved library for viewing or editing
`report_lib`	Reports library information
`save_lib`	Saves a design library to disk
`get_libs`	Creates a collection of the libraries loaded in memory
`report_ref_libs`	Reports the reference libraries of a library loaded in memory
`set_ref_libs`	Sets the reference libraries for a library loaded in memory

For more information about working with design libraries, see

- Relative and Absolute Paths to Design Libraries
- Creating a Design Library
- Opening a Design Library
- Setting the Current Design Library
- Querying a Design Library
- Saving a Design Library
- Closing a Design Library
- Design Library Open Count

See Also

- Reference Library List
- Loading the Technology Data
- Blocks
- Library Packaging

Relative and Absolute Paths to Design Libraries

When you create, open, query, save, or close a design library, you can specify an absolute path, a relative path, or no path at all with the library name, as in the following examples.

```
icc2_shell> open_lib lib_A         # No path, opens lib in search_path
...
icc2_shell> open_lib ./lib_A       # Relative path, current directory
...
icc2_shell> open_lib ../lib_A      # Relative path, parent of current dir
...
icc2_shell> open_lib ../../mylibs/lib_A    # Relative up 2 levels, down 1
...
icc2_shell> open_lib /remote/home/mylibs/lib_A    # Absolute path to lib
...
```

This is how the tool finds the specified library:

- No path – The tool looks in the directories specified by the `search_path` variable, in the order that the directories are listed in the variable.
- Relative path – The tool looks in the specified directory relative to the current working directory (`.`) or its parent directory (`..`).
- Absolute path – The tool looks in the specified absolute path, starting with the slash character (`/`).

See Also

- Relative and Absolute Paths to Reference Libraries
- Library Packaging

Creating a Design Library

To create a design library, use the `create_lib` command. When you run this command to create a new design library, you must specify the library name. Slash (/) and colon (:) characters are not allowed in library names.

For example, to create a new design library named my_libA, use the following command:

```
icc2_shell> create_lib my_libA
{my_libA}
```

By default, the `create_lib` command does not save the design library to disk. To save the library to disk when you create it, set the `lib.setting.on_disk_operation` application option to `true` before running the `create_lib` command. You can also save the design library to disk by using the `save_lib` command, as described in Saving a Design Library.

You can optionally specify the following items:

- The reference libraries

  The reference libraries contain the leaf-level blocks such as standard cells and hard macros. To specify the reference library list when you create the design library, use the `-ref_libs` option. For more information, see Specifying a Design Library's Reference Libraries.

- The technology data

  The technology data specifies the process technology information such as measurement units, layers, unit tile dimensions, and routing rules. To provide the technology data when you create the design library, use one of the following methods:

  - Load a technology file by using the `-technology` option. This is the preferred method for providing the technology data.
  - Reference a technology library by using the `-use_technology_lib` option.

  For more information, see Loading the Technology Data.

- The scale factor

  The scale factor specifies the number of database units per micron. The scale factor must be identical for a design library and all of its reference libraries. If you specify reference libraries, their scale factor is the default scale factor for the design library.

  For information about setting the scale factor, see Specifying the Scale Factor.

See Also

- Relative and Absolute Paths to Design Libraries
- Opening a Design Library
- Reference Library List
- Loading the Technology Data
- Reading a Hierarchical Design Into a Single Design Library
- Reading a Hierarchical Design Into Multiple Design Libraries

Specifying the Scale Factor

The scale factor specifies the number of database units per micron. The IC Compiler II tool uses the scale factor to convert floating point numbers into integers when storing data in the cell libraries and design libraries. When the tool saves a floating point number, it multiplies the number by the scale factor and then rounds the number to an integer. When the tool retrieves the number, it divides the number by the scale factor. For example, if the scale factor is 1000, the tool stores a value of 0.0032 as 3 (0.0032 * 1000 = 3.2, which rounds to 3). The tool retrieves this value as 0.003 (3 / 1000).

If the scale factor is not large enough, the precision of specified coordinates might be affected. For example, assume that you create a net shape with the following command:

```
icc2_shell> create_shape -shape_type rect -layer M1 \
   -boundary {{0.500 0.500} {0.5053 0.5053}}
```

With a scale factor of 1000, the bounding box is stored as {500 500} {505 505}, instead of {5000 5000} {5053 5053}. With a scale factor of 10000, the bounding box is stored as {5000 5000} {5053 5053}.

The scale factor for a design library must match the scale factor of the reference libraries associated with the design library and must be a multiple of the length precision of the technology file associated with the design library.

- To determine the scale factor for a reference library, query its `scale_factor` attribute.
- To determine the length precision of the technology file, check its `lengthPrecision` attribute.

If you specify reference libraries for the design library, the default scale factor for the design library is the scale factor of its associated reference libraries. Otherwise, the default scale factor is 10000, which means that each database unit represents 1 Angstrom.

The recommended scale factor is the technology length precision as specified by the `lengthPrecision` attribute in the technology file. To use the technology length precision as the scale factor, set the `lib.setting.use_tech_scale_factor` application option to `true` before running the `create_lib` command.

```
icc2_shell> set_app_options \
   -name lib.setting.use_tech_scale_factor -value true
```

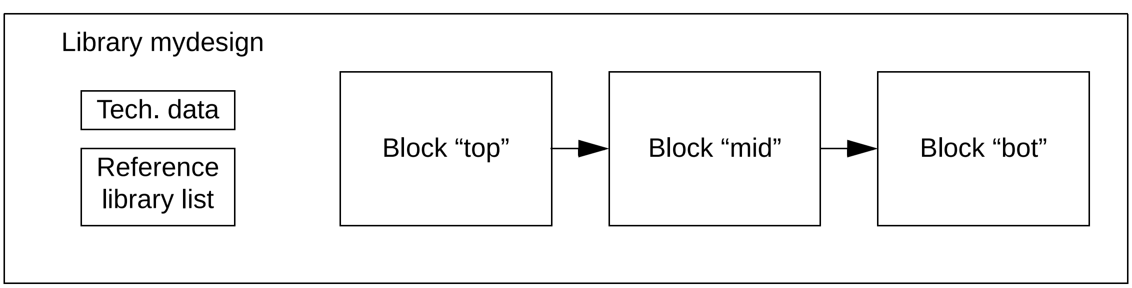
If neither the default scale factor nor the technology length precision is appropriate for your design library, you can explicitly specify the scale factor by using the `-scale_factor` option with the `create_lib` command.

Reading a Hierarchical Design Into a Single Design Library

The following script reads a hierarchical design into a single design library (see Figure 1-3). The input netlists are in Verilog format and the input layout information is in DEF format. You must read in the netlists in hierarchical order, from bottom to top.

```
create_lib mydesign -ref_libs {std_cell_lib.ndm} \
 –technology mytech.tf

read_verilog bot.v       # Implicitly creates new block "bot"
read_def bot.def
...

read_verilog mid.v       # Implicitly creates new block "mid"
read_def mid.def
...

read_verilog top.v       # Implicitly creates new block "top"
read_def top.def
...

save_lib
```

Figure 1-3 Hierarchical Data Stored in a Single Design Library



See Also

- Design Libraries
- Reference Library List

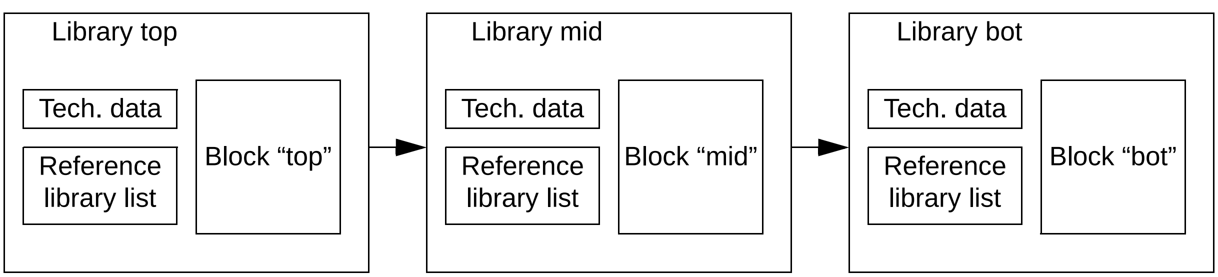
- Specifying a Design Library's Reference Libraries

Reading a Hierarchical Design Into Multiple Design Libraries

The following script reads a hierarchical design into different design libraries for different levels of the design (see Figure 1-4). The input netlists are in Verilog format and the input layout information is in DEF format.

```
# Create library bot and read the design
create_lib bot -ref_libs {std_cell_lib.ndm} \
 -technology mytech.tf
read_verilog bot.v          # Implicitly creates new block "bot"
read_def bot.def
...
save_lib

# Create library mid and read the design
create_lib mid -ref_libs {std_cell_lib.ndm bot} \
 -technology mytech.tf
read_verilog mid.v          # Implicitly creates new block "mid"
read_def mid.def
...
save_lib

# Create library top and read the design
create_lib top -ref_libs {std_cell_lib.ndm bot mid} \
 -technology mytech.tf
read_verilog top.v          # Implicitly creates new block "top"
read_def top.def
...
save_lib
```

Figure 1-4 Hierarchical Design Data Stored in a Multiple Libraries



See Also

- Design Libraries
- Reference Library List
- Specifying a Design Library's Reference Libraries

Opening a Design Library

To open an existing design library saved on disk, use the `open_lib` command:

```
icc2_shell> open_lib my_libA
Information: Loading library file '/usr/lib/StdCells.ndm' (FILE-007)
Information: Loading library file '/usr/lib/RAMs.ndm' (FILE-007)
Information: Loading library file '/usr/lib/PhysicalOnly.ndm' (FILE-007)
{my_libA}
```

The tool opens the specified design library, makes that library the current library, and opens all of its associated reference libraries. Opening a design library means loading it into memory and making its blocks accessible. To reduce memory usage, you can restrict the process, voltage, and temperature combinations loaded into memory from the cell libraries by using the `set_pvt_configuration` command, as described in Loading Timing Information Based on Operating Corners. To enable interpolation during timing analysis and optimization, create scaling groups by using the `define_scaling_lib_group` command, as described in “Creating Scaling Groups” in the *IC Compiler II Timing Analysis User Guide*.

Note:
Before opening a design library, ensure that the `link_library` variable setting is the same as when the design library was created or last opened. Otherwise, the tool will rebuild the cell libraries in its reference library list.

By default, the `open_lib` command opens the design library in read/write mode and the associated reference libraries in read-only mode. You can override the default read/write modes by using the `-read` and `-ref_libs_for_edit` options with the `open_lib` command.

See Also

- Relative and Absolute Paths to Design Libraries
- Creating a Design Library
- Saving a Design Library
- Closing a Design Library
- Reference Library List
- Creating Cell Libraries

Setting the Current Design Library

The *current library* is the default library affected by library-related commands. By default, the library most recently opened is the current library. You can explicitly set any open library to be the current library by using the `current_lib` command:

```
icc2_shell> current_lib my_libA
{my_libA}
```

To determine which library is the current library, use the `current_lib` command by itself:

```
icc2_shell> current_lib
{my_libA}
```

See Also

- Creating a Design Library
- Opening a Design Library
- Querying a Design Library
- Saving a Design Library
- Closing a Design Library

Querying a Design Library

You can get the following types of information about the open libraries:

- The current library – use the `current_lib` command.

  ```
  icc2_shell> current_lib
  {my_design}
  ```

- The libraries loaded in memory – use the `get_libs` command.

  ```
  icc2_shell> open_lib lib_A
  Information: Loading library file '... libA' ...
  Information: Loading library file '... ref_lib1' ...
  Information: Loading library file '... stdhvt.ndm' ...
  {lib_A}

  icc2_shell> open_lib lib_B
  Information: Loading library file '... libB' ...
  Information: Loading library file '... ref_lib2' ...
  Information: Loading library file '... stdhvt.ndm' ...
  {lib_B}

  icc2_shell> get_libs                # Get all open libs
  {lib_A lib_B ref_lib1 ref_lib2 stdhvt.ndm}
  ```

```
icc2_shell> get_libs -explicit      # Get the libs opened explicitly
{lib_A lib_B}

icc2_shell> get_libs -implicit      # Get the libs opened as ref libs
{ref_lib1 ref_lib2 stdhvt.ndm}
```

- The blocks in a library – use the `get_blocks` or `list_blocks` command.

```
icc2_shell> get_blocks -all
{lib_A:block1.design lib_A:block2.design lib_B:blocka.design
lib_B:blockb.design}

icc2_shell> list_blocks
Lib lib_A /path/libs/lib_A tech current
 -> 0 block1.design Apr-20-16:36
 +> 0 block2.design Apr-20-16:38 current
Lib lib_B /path/libs/lib_B tech
 -> 0 blocka.design Apr-18-14:02
 -> 0 blockb.design Apr-18-14:20

icc2_shell> list_blocks lib_B
Lib lib_B /path/libs/lib_B tech
 -> 0 blocka.design Apr-18-14:02
 -> 0 blockb.design Apr-18-14:20
```

The characters at the beginning of each row of the block list indicate the following:

```
 + block open
 - block not open
 > block selected to resolve link references
 * block modified and not yet saved
```

The word `tech` at the end of the `Lib` line indicates that the library has technology data, and the word `current` means it is the current library.

- The associated reference libraries – use the `report_ref_libs` command.

```
icc2_shell> current_lib lib_B
{lib_B}
icc2_shell> report_ref_libs
...
Library: lib_B
...
   Name           Path          Location
---------------------------------------------------
*+ ref_lib1       ../../libs    /path/libs/ref_lib1
*+ ref_lib2       ../../libs    /path/libs/ref_lib2
*+ stdhvt         ../../libs    /path/libs/std/stdhvt.ndm
```

The characters at the beginning of each row of the library list indicate the following:

```
 * library currently open
 + library has technology data
```

```
 - library name and location not available
```

- The associated process technology data – use the `get_techs` command.

```
icc2_shell> get_techs
{tech28nm1p tech13nm2x}
icc2_shell> get_techs -of_objects lib_A
{tech28nm1p}
```

- The parasitic technology (defined by a TLUPlus file) associated with a library – use the `report_lib` command.

```
icc2_shell> report_lib -parasitic_tech my_tech_lib
...
Library: my_tech_lib
...
Full name: /path/tech-info/.../my_tech_lib:my_tech
File name: /path/tech-info/.../my_tech_lib
Design count: 0
No timing data

Parasitic tech data:
-----------------------------------------
 Parasitic tech name:             my_tech_Cmax
 Parasitic itf technology name:   my_tech_Cmax
 Parasitic tech type:             TLUPLUS
 Parasitic source file name:      /path/Tlup/...
 ...
```

See Also

- Relative and Absolute Paths to Design Libraries
- Opening a Design Library
- Setting the Current Design Library
- Saving a Design Library
- Closing a Design Library

Saving a Design Library

When you create a design library or edit its contents, the changes are stored only in memory. To save a design library to disk, use the `save_lib` command.

```
icc2_shell> current_lib
{lib_A}

icc2_shell> save_lib
Saving library 'lib_A'
1

icc2_shell> save_lib lib_B
Saving library 'lib_B'
1

icc2_shell> save_lib -all
Saving all libraries...
5
```

The `save_lib` command saves all blocks in the design library that have been modified and not yet saved. Be sure to save a new or edited library before you close it.

You can save a design library in compressed format to reduce the file size by using one of the following methods:

- To save all open blocks in a library in compressed format, use the `-compress` option with the `save_lib` command.
- To save all design data in the current tool session in compressed format, set the `lib.setting.compress_design_lib` application option to `true`. If you set the application option back to `false` (the default), any design libraries created from that point are saved in uncompressed format. However, previously compressed libraries are preserved in their compressed format.

To save previously compressed libraries to uncompressed libraries, set the `is_compressed` attribute on the library and all of its blocks to `false` before saving them.

Note:
The tool compresses design libraries only. Cell libraries cannot be compressed.

See Also

- Relative and Absolute Paths to Design Libraries
- Creating a Design Library
- Opening a Design Library

- Setting the Current Design Library
- Closing a Design Library

Closing a Design Library

When you no longer need access to data in a library, you can close it. Be sure to save the changes in the library before you close it.

To close a library, use the `close_lib` command.

```
icc2_shell> current_lib
{lib_A}

icc2_shell> close_lib
Closing library 'lib_A'
1

icc2_shell> close_lib lib_B
Closing library 'lib_B'
1

icc2_shell> close_lib -all
Closing all libraries...
1
```

To deliberately close an edited library and discard the changes:

```
icc2_shell> close_lib -force
Closing library 'lib_A'
1
```

Important:

By default, when you close a design library by using the `close_lib` command, the tool does not save the open blocks and does not issue a warning about unsaved design changes. To save new versions for all open blocks in a library, use the `-save_designs` option with the `close_lib` command.

If the library *open count* is 1 or more after you execute the `close_lib` command, the library remains open.

To remove an unwanted design library from disk, first use the `remove_blocks` command to remove all the blocks in the library. Note that removed blocks cannot be recovered. Then use operating system commands to remove the unwanted library directory.

You can save a design library in compressed format before you close it by specifying the `-compress` and `-save_designs` options with the `close_lib` command. The tool compresses design libraries only. Cell libraries cannot be compressed.

See Also

- Relative and Absolute Paths to Design Libraries
- Creating a Design Library
- Opening a Design Library
- Setting the Current Design Library
- Saving a Design Library
- Design Library Open Count

Design Library Open Count

The *open count* of a design library is an integer specifying the number of times the library has been opened. The tool monitors the open count to keep the library open as long as you need access to it, based on a matching number of `open_lib` and `close_lib` commands used on the library.

When you open a closed library or create a new library, its open count is set to 1. Each time the same library is reopened, its open count is incremented by 1:

```
icc2_shell> open_lib lib_C
Information: Loading library file ...
{lib_C}
...
icc2_shell> open_lib lib_C
Information: Incrementing open_count of library 'lib_C' to 2. (LIB-017)
...
icc2_shell> open_lib lib_C
Information: Incrementing open_count of library 'lib_C' to 3. (LIB-017)
...
```

Opening a design library also opens its associated reference libraries, which increments the open count for each reference library as well.

The `close_lib` command decrements a library's open count by 1. If the resulting new count is 1 or more, the library remains open; or if the new count is 0, the library is closed and removed from memory.

```
icc2_shell> close_lib lib_C
Information: Decrementing open_count of library 'lib_C' to 2. (LIB-018)
1
...
icc2_shell> close_lib lib_C
Information: Decrementing open_count of library 'lib_C' to 1. (LIB-018)
1
...
```

```
icc2_shell> close_lib lib_C
Closing library 'lib_C'
1
```

To determine the open count of a library without affecting the count:

```
icc2_shell> get_attribute [get_libs lib_C] open_count
3
```

To close a library irrespective of its open count:

```
icc2_shell> close_lib lib_C -purge
Closing library 'lib_C'
1
```

Using the `save_lib` command does not affect the open count.

Be sure to save a new or edited library before you close it. The tool does not warn you about unsaved data before it closes the library.

See Also

- Creating a Design Library
- Opening a Design Library
- Saving a Design Library
- Closing a Design Library

Cell Libraries

A *cell library* is a unified library that contains the logical and physical information for a specific technology and one or more of its library cells. Each library cell is represented as a single object with multiple views that contain various types of information about the cell.

This type of library is used only as a reference library for design libraries; it does not itself have design-specific data, a reference library list, or lower-level libraries. Reference libraries contain basic leaf-level blocks such as standard cells, I/O pads, and hard macros.

You can create cell libraries by using the library manager tool as described in the *IC Compiler II Library Preparation User Guide* or by providing the source files when you create the design library with the `create_lib` command or update a design library's reference libraries with the `set_ref_libs` command.

When you open a design library, the tool implicitly opens its associated cell libraries. To open a cell library directly, specify the complete library name, including any extension, such as

.ndm. For all other commands that operate on a cell library, specify the library name without the extension. For example,

```
icc2_shell> open_lib /path/nlibs/std32hvt.ndm
Information: Loading library file '/path/nlibs/std32hvt.ndm' (FILE-007)
{std32hvt}
...
icc2_shell> get_libs
{lib_A lib_B std32hvt}
...
icc2_shell> current_lib std32hvt
{std32hvt}
icc2_shell> close_lib std32hvt
Closing library 'std32hvt'
1
```

See Also

- Reference Library List
- Specifying a Design Library’s Reference Libraries
- Creating Cell Libraries
- *IC Compiler II Library Preparation User Guide*
- Library Packaging

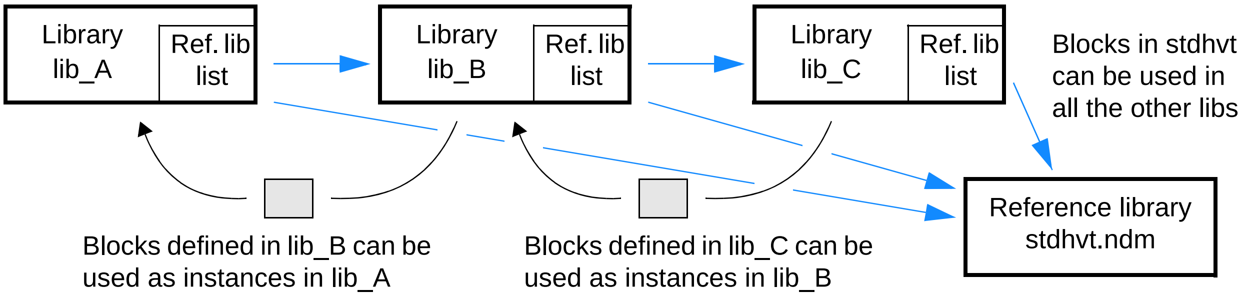
Reference Library List

When you build a design, the building blocks for the design, whether library cells, hard macros, or other blocks, must be in the libraries associated with the design library. These associated libraries can be cell libraries or other design libraries. They are called reference libraries of the design library and are specified in the design library's *reference library list*.

The entries in a reference library list each describe a one-way, one-level relationship between two libraries in a design hierarchy. If you set lib_B as a reference library of lib_A, you can use blocks in lib_B as instances in lib_A, but not the other way around.

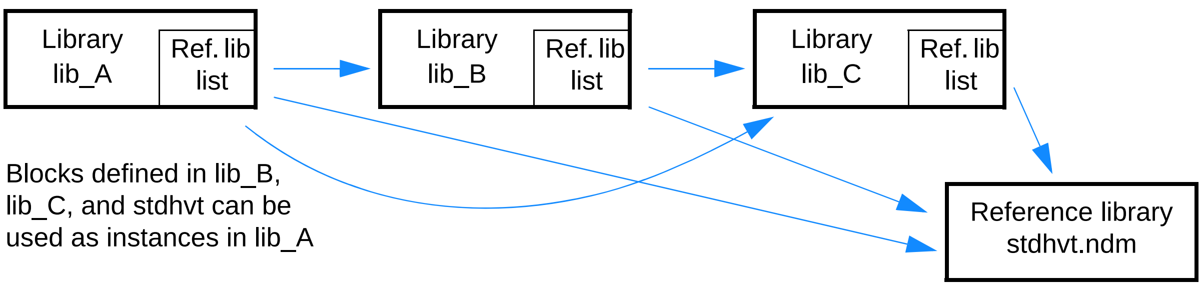
You can set lib_B to be a reference library of lib_A, and set lib_C to be a reference library of lib_B, as shown in the following figure.

Figure 1-5 Reference Library Lists



In that case, you can use blocks from lib_C as instances in lib_B, and use blocks in lib_B as instances in lib_A. However, you cannot directly use blocks in lib_C as instances in lib_A unless you also define a reference library relationship directly between lib_A and lib_C, as shown in the following figure.

Figure 1-6 Reference Library List Spanning Multiple Levels



To avoid naming conflicts between libraries or blocks in different physical hierarchies, set the `use_hier_ref_libs` attribute on the child design libraries. When you set this attribute on a child library, the `link_block` command uses the reference library list of the child library to resolve references for any blocks referenced from that library down the block hierarchy.

In the following example, the tool uses the reference libraries from the sub_block1 and sub_block2 child libraries when linking the top-level block:

```
icc2_shell> set_attribute [get_libs sub_block1] \
     use_hier_ref_libs true
icc2_shell> set_attribute [get_libs sub_block2] \
     use_hier_ref_libs true
icc2_shell> current_block top
icc2_shell> link_block
```

To report the reference libraries of a design library, use the `report_ref_libs` command. For details, see Reporting Reference Libraries.

For more information, see

- Specifying a Design Library's Reference Libraries
- Reporting Reference Libraries
- Rebinding Reference Libraries of a Design Library

See Also

- Design Libraries
- Blocks
- Library Packaging

Specifying a Design Library's Reference Libraries

When you specify the reference libraries for a design library, you can specify

- Existing cell libraries, which can be single-file cell libraries generated with releases earlier than N-2017.09, directory-based cell libraries generated with N-2017.09 or later releases, or a mix of single-file and directory-based cell libraries.
- Physical source files, which can be physical libraries, LEF files, or Milkyway reference libraries

  If you specify physical source files, the tool creates IC Compiler II cell libraries, as described in Creating Cell Libraries, and associates the cell libraries with the design library.
- Other design libraries

You can specify the libraries and source files using relative paths, absolute paths, or no paths. For details, see Relative and Absolute Paths to Reference Libraries.

- To specify the reference libraries when you create a new design library, use the `-ref_libs` option of the `create_lib` command.

  ```
  icc2_shell> create_lib lib_B \
   -ref_libs {../LIBS/lib_C ../STND/stdhvt.ndm} ...
  {lib_B}
  ```

- To specify the reference libraries for an existing design library, open the library and use the set_ref_libs command:

```
icc2_shell> open_lib lib_B
Information: Loading library file '/remote/home/mylibs/lib_B'
{lib_B}
icc2_shell> set_ref_libs -ref_libs {../LIBS/lib_C ../STND/stdhvt.ndm}
../LIBS/lib_C ../STND/stdhvt.ndm
```

  To add reference libraries to the existing reference library list, use the -add option. To remove reference libraries from the existing reference library list, use the -remove option. To replace the reference library list, use the -ref_libs option. You can specify physical source files only with the -ref_libs option.

  If you change the reference libraries associated with a design library, you can update the references of an existing block in the library by using the link_block -rebind command. To control the order in which different lower-level block views are selected for rebinding, use the set_view_switch_list command.

The reference library link order depends on the order and types of files specified in the -ref_libs option. For cell libraries generated from physical source files, the logic library order in the link_library variable determines the cell library link order.

To report the reference libraries that have been set for a design library, use the report_ref_libs command.

See Also

- Design Libraries
- Reference Library List

Relative and Absolute Paths to Reference Libraries

When you specify a reference library or physical source file for a design library using the set_ref_libs or create_lib command, you can specify a simple name with no path, a relative path, or an absolute path, as shown in the following examples.

```
icc2_shell> set_ref_libs -ref_libs stdB.ndm          # Uses search_path

icc2_shell> set_ref_libs -ref_libs mylibs/stdB.ndm   # Uses search_path

icc2_shell> set_ref_libs -ref_libs ./lib_A  # Relative to design library

icc2_shell> set_ref_libs -ref_libs ../lib_A # Relative up one level

icc2_shell> set_ref_libs -ref_libs ../../mylibs/lib_A

icc2_shell> set_ref_libs -ref_libs /remote/home/mylibs/lib_A  # Absolute
```

This is how the tool finds the specified reference library or source file:

- Relative path (recommended) – The tool looks in the specified directory relative to the *design library's* directory (`.`) or the *design library's* parent directory (`..`) . This relative path is stored in the design library, so if you move the library and its reference libraries together to a new location, the tool still automatically finds each reference library located in the same place *relative to the design library*.
- Simple name – The tool looks in the directories specified by the `search_path` variable, in the order that the directories are listed in the variable. This path resolution *binds* the reference library to the design library. If you want to change the bindings based on a new `search_path` variable setting, run the `set_ref_libs` command with the `-rebind` option. For details, see Rebinding Reference Libraries of a Design Library.
- Absolute path – The tool looks in the specified absolute path, starting with the slash character (`/`). This absolute path is stored in the design library, so you can move the design library while keeping its reference libraries in the same absolute locations.

Note:
Moving a design library and its reference libraries using the Library Packaging feature (`write_lib_package` and `read_lib_package`) moves the reference libraries to a new directory under the restored design library directory and automatically rebinds them. For details, see Restoring Data From a Library Package.

See Also

- Specifying a Design Library's Reference Libraries
- Reference Library List

Creating Cell Libraries

You can create cell libraries by using the library manager tool as described in the *IC Compiler II Library Preparation User Guide* or by specifying the physical libraries with the `create_lib` or `set_ref_libs` command.

To create a cell library when you run the `create_lib` or `set_ref_libs` command,

1. Specify the search path for the library source files by setting the `search_path` variable.

```
icc2_shell> set_app_var search_path ". \
   $ADDITIONAL_SEARCH_LOCATIONS $search_path"
```

2. Specify the logic libraries by setting the `link_library` variable.

```
icc2_shell> set_app_var link_library "* $LINK_LIBRARY_FILES"
```

   To specify process labels for the logic libraries, set the `lib.configuration.process_label_mapping` application option. For more information about process labels, see “Identifying the Process Associated With a Logic Library” in the *IC Compiler II Library Preparation User Guide*.

3. Specify the library generation options by setting the `lib.configuration` application options.

   ❍ To specify the output directory for the generated cell libraries, set the `lib.configuration.output_dir` application option.

     If you do not set this application option, the command creates the cell libraries in a directory named CLIBS under the current working directory.

     To minimize the disk space requirement when you are working with a design team, ensure that the entire team uses the same setting for this application option and includes this location in the search path, as specified by the `search_path` variable. In general, the project lead would create the cell libraries in a central location and the rest of the team would use the generated cell libraries.

   ❍ To specify a configuration script with library manager commands used to customize the library configuration process, set the `lib.configuration.default_flow_setup` application option.

     If you set this application option, the `create_lib` command sources this script at the start of the library configuration process.

   ❍ To specify assembly scripts with library manager commands used to assemble the cell libraries, set the `lib.configuration.assembly_scripts` application option.

     This application option maps an assembly script to each physical library specified in the `-ref_libs` option. If you set this application option, the `create_lib` command sources these scripts to generate the cell libraries instead of using the default flow.

   ❍ To display messages issued by the library manager, set the `lib.configuration.display_lm_messages` application option to `true`.

   ❍ If you are using LEF files for the physical source and need to convert the LEF site names to technology file site names, set the `lib.configuration.lef_site_mapping` application option.

   ❍ If you are using Milkyway reference libraries for the physical source, you must specify the IC Compiler executable with the `lib.configuration.icc_shell_exec` application option.

4. Specify the physical source files by using the `-ref_libs` option with the `create_lib` or `set_ref_libs` command.

   You must specify a technology file by using the `-technology` option with the `create_lib` command; otherwise, the tool cannot create the cell libraries. The `set_ref_libs` command gets the technology data from the design library.

For example, to create a design library using the mytech.tf technology file and the physical libraries specified by the $PHYSICAL_LIB_FILES variable, use the following command:

```
icc2_shell> create_lib -technology mytech.tf \
   -ref_libs $PHYSICAL_LIB_FILES mydesign
```

You can specify any combination of physical libraries, LEF files, and Milkyway reference libraries. You can also specify IC Compiler II cell libraries; for the library generation process, these are used only to exclude logic libraries from library generation.

For details about how the `create_lib` or `set_ref_libs` command generates the cell libraries, see Cell Library Generation.

Cell Library Generation

To create the cell libraries, the `create_lib`, `set_ref_libs`, or `open_lib` command

1. Sources the script file specified by the `lib.configuration.default_flow_setup` application option, if any.
2. Assembles the cell libraries by running either the default flow or the scripts specified in the `lib.configuration.assembly_scripts` application option.

   The default flow includes the following steps:

   a. Sets the design mismatch configuration to `auto_fix`.

   ```
   set_current_mismatch_config auto_fix
   ```

   b. Creates a library workspace for the exploration flow.

   ```
   create_workspace -flow exploration myworkspace
   ```

   c. Loads the technology file into the library workspace.

   The technology data source depends on the command that initiates the cell library generation.

   - For the `create_lib` command, the tool loads the technology file specified with the `-technology` option.
   - For the `set_ref_libs` and `open_lib` commands, the tool loads the technology file from the design library.

   d. Loads the logic libraries and physical source files into the library workspace.

   The logic libraries are specified with the `link_library` variable. The physical source files are specified with the `-ref_libs` option.

Note:
If the -ref_libs option includes IC Compiler II cell libraries, the tool loads only the logic libraries that are not used in those cell libraries into the library workspace.

e. Analyzes the library source files by running the group_libs command.

f. Validates and commits the exploration flow library workspace by running the process_workspaces command.

The tool saves the cell libraries in the directory specified by the lib.configuration.output_dir application option.

See Also

- Creating a Design Library
- Specifying a Design Library’s Technology File
- Specifying a Design Library’s Reference Libraries
- *IC Compiler II Library Preparation User Guide*

Updating Cell Libraries for Newer Tool Versions

To update the cell libraries when you move to a newer version of the IC Compiler II tool, run the set_ref_libs command. When you run this command, the tool automatically generates updated cell libraries using the latest library schema.

To update cell libraries with the set_ref_libs command,

1. Specify the library generation options by setting the lib.configuration application options, as described in Creating Cell Libraries.

2. Specify the physical source files by using the set_ref_libs command.

   For example, to update the cell libraries associated with the mydesign design library, use the following command:

   ```
   icc2_shell> open_lib mydesign
   icc2_shell> set_ref_libs -ref_libs $PHYSICAL_LIB_FILES
   ```

   For details about how the set_ref_libs command generates the cell libraries, see Cell Library Generation.

Reporting Reference Libraries

To report the reference libraries of a design library, use the report_ref_libs command:

```
icc2_shell> create_lib lib_A -ref_libs \
  {../libs/SCLL.ndm ../libs/SCHH.ndm ../BLOCKS/MACROS}
```

```
{lib_A}
icc2_shell> report_ref_libs
...
   Name    Path                 Location
   -----------------------------------------------------------------
*+ SCLL    ../libs/SCLL.ndm    /remote/project10/test1/libs/SCLL.ndm
*+ SCHH    ../libs/SCHH.ndm    /remote/project10/test1/libs/SCHH.ndm
*  MACROS  ../BLOCKS/MACROS    /remote/project10/test1/BLOCKS/MACROS
    "*" = Library currently open
    "+" = Library has technology information
```

The report shows the Name, Path (as originally specified), and Location (absolute path) of each reference library in the design library's reference library list.

The entry in the Path column is the simple name, relative path, or absolute path originally specified for the reference library.

See Also

- Specifying a Design Library's Reference Libraries
- Reference Library List

Loading Timing Information Based on Operating Corners

By default, the tool loads the timing information from all logic libraries included in the reference libraries. To include only the timing information that matches the valid operating corners (process, voltage, and temperature values) for your design, define a PVT configuration for the design library. The PVT configuration applies only to the current session and must be specified before you open the design library.

Note:
The PVT configuration is applied when loading the cell libraries. It does not apply to cell libraries that are already open when you run the `set_pvt_configuration` command.

A PVT configuration is a sequence of rules, each of which specifies the process labels, process numbers, voltages, and temperatures to match. If you want all the data that matches a specific set of operating corners, a single rule is sufficient. If you want to match a specific subset of many operating corners, you must specify multiple rules.

To define a PVT configuration rule, use the `set_pvt_configuration` command.

- To create a new rule, use the `-add` option.

  By default, the command uses the rule_*n* naming convention. To specify the rule name, use the `-name` option.

- To modify an existing rule, use the `-rule` option to specify the rule name.

- To add filters to a rule, use one or more of the following options: `-process_labels`, `-process_numbers`, `-voltages`, and `-temperatures`.
- To remove filters from a rule, use the `-clear_filter` option.

For example, to create a PVT configuration rule that loads the timing data only for voltages of 0.65 or 1.32 volts and a temperature of 25 degrees for the mylib design library, use the following commands:

```
icc2_shell> set_pvt_configuration -add \
   -voltages {0.65 1.32} -temperatures {25}
icc2_shell> open_lib mylib
```

Rebinding Reference Libraries of a Design Library

A design library's associated reference libraries contain lower-level blocks used as instances in the design library's blocks. You associate or *bind* a reference library with a design library by using `-ref_libs` option of the `create_lib` command or by using the `set_ref_libs` command. If you specify a simple name for the reference library, the command resolves the path by using the `search_path` variable.

If you make a change that invalidates the reference library list, you need to *rebind* the reference libraries. If necessary, first set the `search_path` variable appropriately, then use the `set_ref_libs -rebind` command:

```
icc2_shell> current_lib
{lib_A}
icc2_shell> set_app_var search_path {. ../MYLIBS ../NLIBS}
. ../MYLIBS ../NLIBS
icc2_shell> set_ref_libs -rebind
../MYLIBS/lib_C ../MYLIBS/lib_D ../NLIBS/stdhvt.ndm}
```

Rebinding a library does not affect the bindings of blocks already existing in the design library. To rebind these blocks using an updated reference library list, use the `link_block` command with the `-rebind` option.

See Also

- Specifying a Design Library's Reference Libraries
- Relative and Absolute Paths to Reference Libraries
- Reporting Reference Libraries
- Reporting Unbound Objects
- Reference Library List

Loading the Technology Data

Each design library needs process technology information such as measurement units, layers, unit tile dimensions, and routing rules. This type of information comes from a technology file. The tool reads the data from the technology file and stores that information in a library. The technology file syntax is described in the *Synopsys Technology File and Routing Rules Reference Manual*.

A design library can get its technology information either directly from a technology file or indirectly through another library called a *technology library*. A technology library contains only technology data and does not contain design data or leaf-level reference library cells. The direct method is the preferred method.

The two methods are summarized in the following figures.

Figure 1-7 Direct Method of Specifying a Library's Technology Data

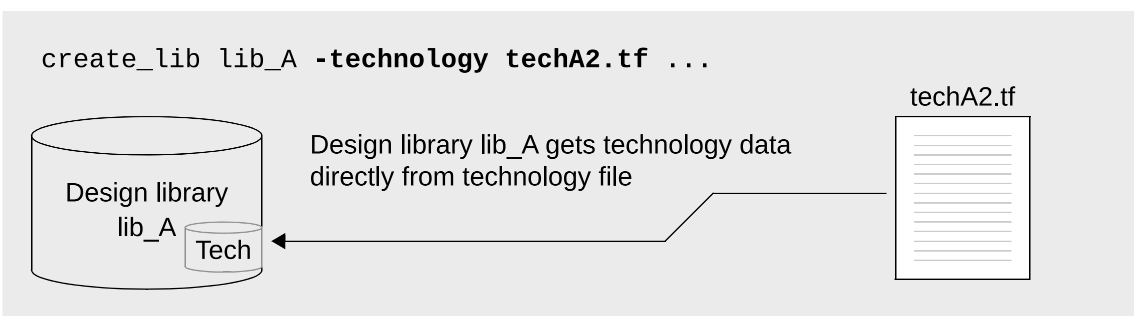
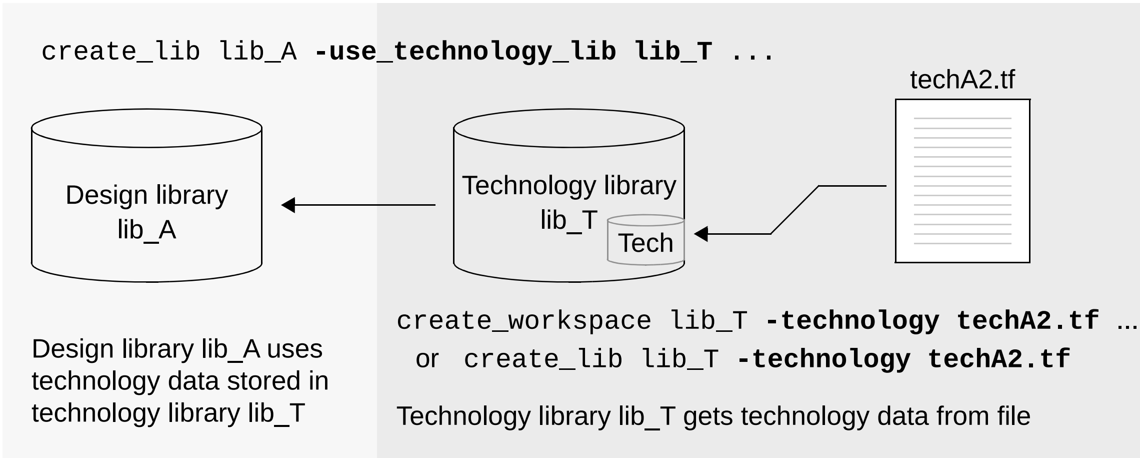


Figure 1-8 Indirect Method of Specifying a Library's Technology Data



The IC Compiler II tool requires the site symmetry and each layer's routing direction and track offset to be defined in the design library or technology library. For details, see "Completing the Technology Data" in the *IC Compiler II Library Preparation User Guide*.

To report the source of the technology data associated with a library, use the `report_ref_libs` command.

You can determine the consequences of loading new technology data into an existing design library by using the `report_tech_diff` command.

For more information, see

- Specifying a Design Library's Technology File
- Specifying a Design Library's Technology Library

See Also

- Reference Library List
- Specifying a Design Library's Reference Libraries

Specifying a Design Library's Technology File

To specify the technology file for a new design library, use the `-technology` option with the `create_lib` command:

```
icc2_shell> create_lib my_lib -technology /usr/TECH/my-tech.tf
Information: Loading technology file '/usr/TECH/my-tech.tf' (FILE-007)
{my_lib}
```

To update the technology file for an existing library, use the `read_tech_file` command:

```
icc2_shell> open_lib my_lib
{my_lib}
icc2_shell> read_tech_file /usr/TECH/my-tech2.tf
Information: Replacing technology file '/usr/TECH/my-tech.tf' with
 '/usr/TECH/my-tech2.tf' (LIB-037)
1
```

In either case, the command reads the technology file and stores the technology data in the library. After that, the library no longer needs or uses the technology file. However, be sure to keep the technology file for reference and for future updates.

You can specify the technology file using an absolute path, relative path, or a simple name. If you use a simple name, the tool looks for the technology file in the directories defined by the `search_path` variable.

When you use the `read_tech_file` command, the data in the new technology file replaces any existing technology data, whether from a reference library, a previous technology file, or specified with Tcl commands. The new technology data must be upwardly compatible with the existing technology data with respect to layer, contact, and unit tile definitions, as explained in Technology File Compatibility.

When you read a technology file, the tool performs syntax and semantic checks on the contents of the technology file. The technology file checker has two modes:

- User mode (the default)

  In this mode, the tool downgrades the message severity for suspected errors for the general user.

- Developer mode

  In this mode, the tool increases the message severity for suspected errors for the technology file developer to correct or waive. To enable this mode, set the `file.tech.library_developer_mode` application option to `true`.

See Also

- Specifying a Design Library's Technology Library
- Loading the Technology Data
- Technology File Compatibility
- Reference Library List

Technology File Compatibility

When you update the technology file for a library using the `read_tech_file` command, the data in the new technology file replaces any existing technology data. The new technology data must be upwardly compatible with the existing technology data with respect to unit tile, layer, and contact definitions:

- Unit tiles – The new technology file must contain at least the same `Tile` definitions, with the same dimensions, as the old technology file.

  The new technology can define additional tiles.

- Layers – The new technology file must contain at least the same layers as the old technology file, in the same order, with the same settings for the layer name and the `layerNumber`, `maskName`, and `dataTypeNumber` attribute settings.

  The new technology can define additional layers, and the design rules for the existing layers need not be the same as in the old technology.

- Contact codes – The new technology file must contain at least the same `ContactCode` definitions as the old technology file, with the same settings for the contact name and the `cutLayer`, `lowerLayer`, and `upperLayer` attribute settings.

  The new technology can define additional contact codes, and the design rules for the existing contact codes need not be the same as in the old technology.

Be sure that the new technology file is compatible. The `read_tech_file` command applies the newly specified technology file without checking for compatibility.

See Also

- Loading the Technology Data
- Specifying a Design Library’s Technology File

Specifying a Design Library’s Technology Library

To specify the technology library for a new design library, use the `-ref_libs` and `-use_technology_lib` options with the `create_lib` command:

```
icc2_shell> create_lib my_lib -ref_libs {lib_T ref_A} \
  -use_technology_lib lib_T ...
```

To update the technology file associated with an existing design library, use the `set_ref_libs` command:

```
icc2_shell> open_lib my_lib
{my_lib}
icc2_shell> set_ref_libs -ref_libs lib_T2 -use_technology_lib lib_T2
lib_T2
```

The technology library specified with the `-use_technology_lib` option must be a library containing technology data previously read into it from a technology file, and it must be a library in the `-ref_libs` list.

Note:
If the `-ref_libs` option contains physical source files, you must specify a technology file, as described in Specifying a Design Library's Technology File, rather than a technology library.

See Also

- Loading the Technology Data
- Reference Library List

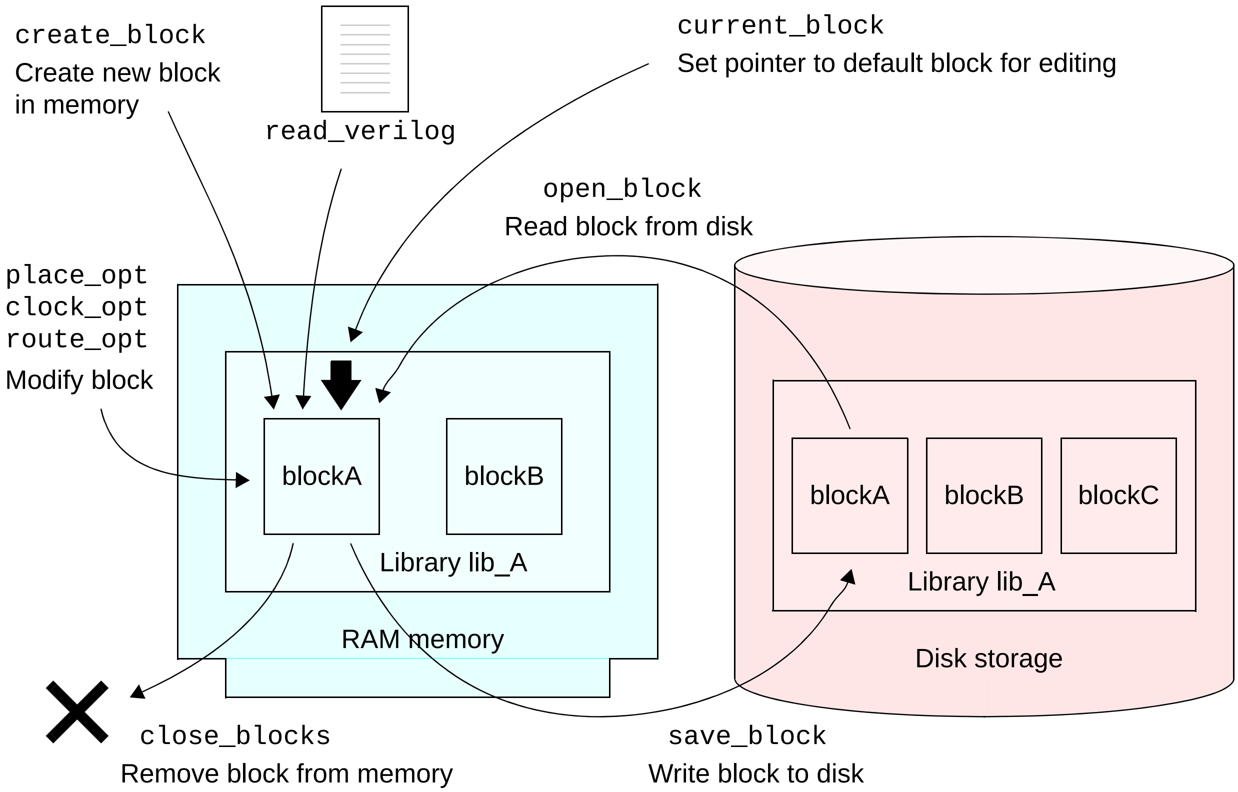
Blocks

A *block* is a container for physical and functional design data. A typical project consists of the following steps to create and edit each block, at each hierarchical level:

1. Read the design netlist using the `read_verilog` command. This implicitly creates a new block, like using the `create_block` command.
2. Read other design information and constraints by using commands such as `load_upf`, `read_sdc`, and `read_def`.
3. Perform physical implementation by using the `place_opt`, `clock_opt`, and `route_opt` commands.
4. Save the block by using the `save_block` command.

You create, query, and edit blocks in memory. To save a new or edited block to disk, use the `save_block` command; or to read a block from disk into memory, use the `open_block` command, as shown in the following figure.

Figure 1-9 Creating, Opening, Editing, Saving, and Closing a Block



A block can have multiple *views*, each view containing an alternative representation of the same design:

- Design view – a complete physical view that contains the full design information of the cell
- Frame view – a limited physical view that contains only the information needed to perform placement and routing
- Abstract view – an interface-only block representation used in complex hierarchical designs
- Outline view – a simplified representation used for floorplanning

The following table briefly describes the commands that operate on blocks.

Table 1-2 Commands That Operate on Blocks

Command	Description
`close_blocks`	Closes a block, removing it from memory
`copy_block`	Copies a block to a new block in the same or different design library
`create_block`	Creates a new block in memory
`current_block`	Sets or gets the current block
`get_blocks`	Creates a collection of the blocks loaded in memory
`link_block`	Resolves the references in a block
`list_blocks`	Lists the blocks (stored on disk) in the specified design libraries
`move_block`	Moves a block to a new block, design library, or view
`open_block`	Opens a saved block for viewing or editing
`rebind_block`	Rebinds the references in a block
`remove_blocks`	Removes a block from memory and disk
`reopen_block`	Changes the open mode (read-only or edit) of an opened block
`save_block`	Saves a block to disk

For more information about blocks, see:

- Block Naming Conventions
- Block Labels
- Block Views
- Creating a Block
- Opening a Block
- Setting the Current Block
- Querying Blocks
- Saving a Block

- Copying a Block
- Block Open Count
- Block Types

See Also

- Design Libraries
- Reference Library List
- Cell Libraries
- Library Cells
- Library Packaging

Block Naming Conventions

You specify a block using the following format:

```
[libName:][blockName][/labelName][.viewName]
```

where *blockName* is required when you use the `create_block` and `open_block` commands.

For example,

```
icc2_shell> open_block my_lib:MUX2/ver1.design
```

The following default naming conventions apply to a specified block:

- Default library: current library
- Default block: current block
- Default label: none
- Default view
  - For the `open_block` and `create_block` commands, the default view is the design view.
  - For the `copy_block`, `move_block`, `remove_blocks`, `rename_block`, and `save_block` commands,
    - If you do not specify the source block, the command performs the task on the view of the current block.

- If you specify the source block without a view, the command performs the task on all available views of the source block.

For example, if the current block is the design view of the `my_lib:MUX2` block, the following commands save only the design view of the block:

```
save_block
save_block MUX2.design
save_block my_lib:MUX2.design
```

The following examples show commands you can use to save all views of the block:

```
save_block MUX2
save_block my_lib:MUX2
save_block MUX2 -as NEW_MUX2
```

The current block is the default block affected by block-related commands. The last block opened is the current block by default. You can explicitly set the current block by using the `current_block` command.

You can create and access a block in the current library using a simple name:

```
create_block MUX2
save_block MUX2
close_blocks MUX2
open_block MUX2
```

If you want to use a more complex block name, append a *label* using a slash character:

```
icc2_shell> create_block MUX3/ver1
```

Block labels are useful for maintaining different block versions in hierarchical designs.

Note that you cannot use a period character to add a block name extension:

```
icc2_shell> create_block MUX3.ver1
Error: Invalid block name 'MUX3.ver1'. (DES-008)
...
```

The period character is reserved for designating the *view* name: .design, .frame, .abstract, or .outline.

For more information about block naming conventions, see

- Block Labels
- Block Views

Block Labels

When you create a block, you can optionally specify a label as part of the block name. You can use labels to make different versions of the same block. Each version is considered a separate block. For a hierarchical design, you can effectively manage different versions of the full hierarchy by using different labels for different design versions.

If used, the label follows a slash character:

```
[libName:][blockName][/labelName][.viewName]
```

For example, to specify “ver1” as a block label:

```
icc2_shell> open_block my_lib:MUX2/ver1.design
```

The default label is an empty string. In other words, there is no default label. You can use labels only for designs in design libraries, not library cells in cell libraries.

To save a block on disk using a new or different label, use the `save_block` command with the `-label` option. You can save different versions of a block while maintaining the current block setting in the tool:

```
current_block blkZ
save_block -label v1
save_block -label v2
current_block
```

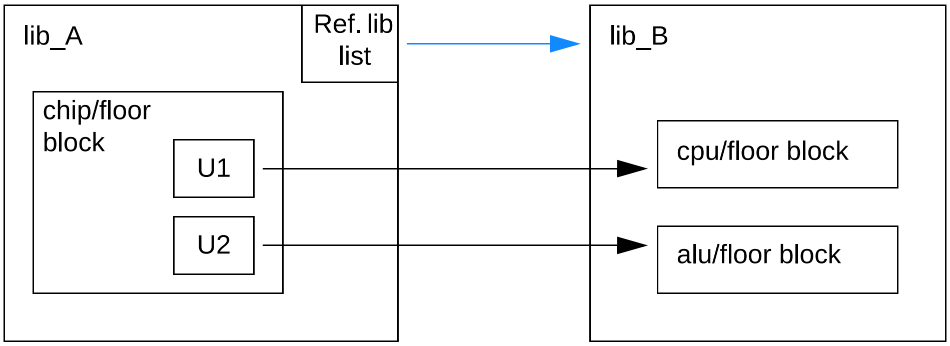
See Also

- Block Views
- Block Naming Conventions

Using Labels in a Hierarchical Design

Suppose that you have a top-level design named chip/floor in a design library named lib_A. The block contains instances U1 and U2, which represent lower-level blocks named cpu/floor and alu/floor in reference library lib_B, as shown in the following figure. All the blocks are at the floorplanned stage.

Figure 1-10 Hierarchical Design Using Block Names With Labels



You decide to proceed to the placement stage, and you want to keep a copy of the design at the floorplanned stage for possible modification later. You save the chip/floor block to a new block named chip/placed by using the `save_block -hierarchical` command:

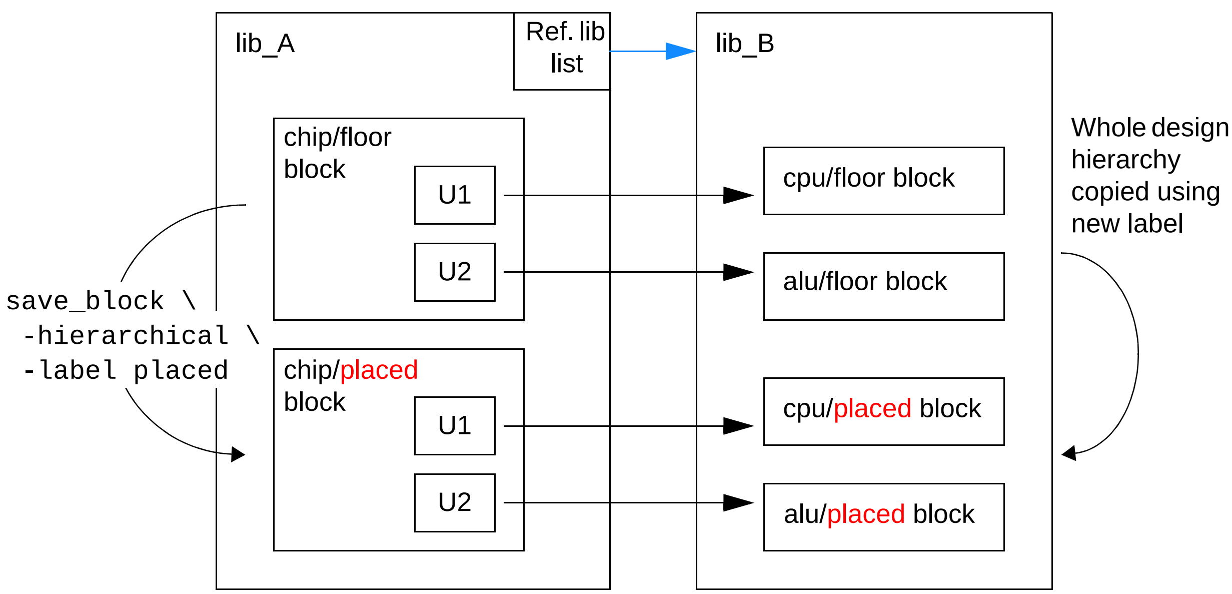
```
icc2_shell> current_block
icc2_shell> save_block -hierarchical -label placed
```

or equivalently:

```
icc2_shell> save_block -hierarchical -as chip/placed
...
```

This effectively copies the chip/floor block to a new block, including its full hierarchy, creating new blocks in both the current library and the lower-level reference library. Then you can edit the new top-level and new lower-level blocks, leaving the original blocks unchanged, as shown in the following figure.

Figure 1-11 Copying a Hierarchical Design With Labels



See Also

- Block Labels

Creating a Label When Reading a Verilog Netlist

When you read a Verilog netlist with the `read_verilog` command, the command creates a new block to contain the netlist. If you want the new block to have a label, you can do either of the following:

- Specify the label name together with the design name:

```
icc2_shell> read_verilog mychip.v -design chip/floor
...
```

- Set the `file.verilog.default_user_label` application option to the desired label name before you read in the Verilog netlist:

```
icc2_shell> set_app_options \
  -name file.verilog.default_user_label -value floor
file.verilog.default_user_label floor
icc2_shell> read_verilog mychip.v -design chip
Information: Reading Verilog into new design 'chip/floor' in lib ...
...
```

See Also

- Block Labels
- Block Naming Conventions

Block Views

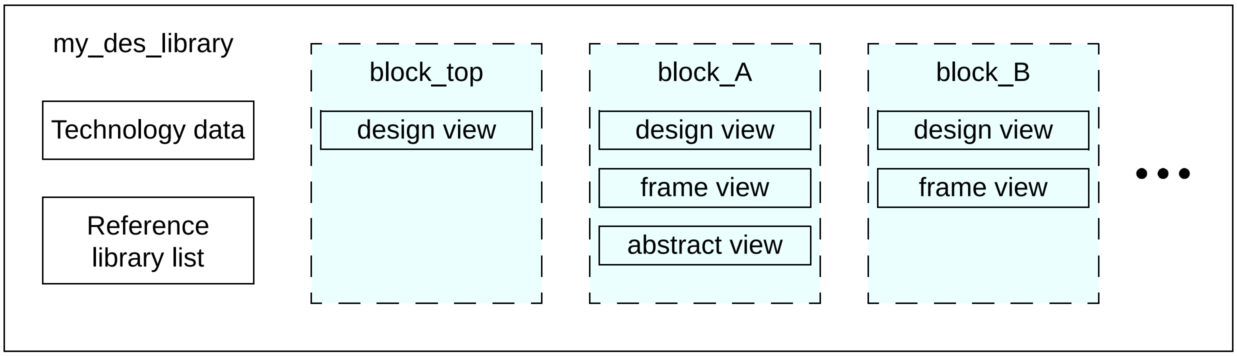
A block can have multiple *views*, with each view containing an alternative representation of the block used for a specific purpose. The IC Compiler II infrastructure supports the following types of views for blocks in design libraries:

- Design view (`.design`) – a complete physical view that contains the full design information of the cell, including placed block instances and routed nets. This is the default view type.
- Frame view (`.frame`) – a limited physical view that contains only the information needed to perform placement of the block as an instance and routing to the ports of the instance: the block outline, pins, via regions, and routing blockages.
- Abstract view (`.abstract`) – a simplified view that contains only the interface information of a subdesign, used for placement and timing analysis at the next higher level of the design. For details, see the *IC Compiler II Implementation User Guide*.

- Outline view (`.outline`) – a simplified view of a large child block that contains only the hierarchy information, without nets or leaf-level library cells, used for floorplan creation. For details, see the *IC Compiler II Design Planning User Guide*.

The following figure shows how the block views are stored in a typical design library.

Figure 1-12 Design Library Contents



The following additional types of views are supported for library-cell blocks in cell libraries:

- Timing view (`.timing`) – a functional description of the timing, logic, and power characteristics of a library cell. The implementation tool uses this information for timing analysis, power analysis, and optimization.
- Layout view (`.layout`) – a physical-only view of the shapes in a library cell, which contains the same information as the GDSII description of the cell, not including connectivity and pin information. The library manager tool uses this view as a data source for generating the design view for a library cell, and the implementation tool uses this view for mask generation.

Note:
The layout view (`.layout`) applies only to leaf-level library cells. This is different from the "layout view" shown in GUI windows when you view a design. The GUI "layout view" shows the full chip layout, which is actually a display of the design view of the block.

When you create, open, save, or close a block, you specify the block view as the last item in the full block name, after the period character:

[*libName:*][*blockName*][*/labelName*][**.*viewName***]

For example, the following command opens the frame view of block NAND2/ver1:

```
icc2_shell> open_block NAND2/ver1.frame
```

If you do not specify a view name,

- The `create_block` and `open_block` commands open the design view of the block.

- The copy_block, move_block, remove_blocks, rename_block, and save_block commands operate on all available views of the block.

The implementation tool resolves the hierarchical references of a design when you run the link_block command or when the tool performs linking implicitly. The process of associating a block to its lower-level reference blocks is called *binding*.

To specify the order in which different views are selected for binding, use the set_view_switch_list command. You can set the view type priority at the block level, at the library level, and globally.

To query the current order of priority, use the get_view_switch_list command. The default order is {abstract design frame outline}.

To rebind an existing block, use the link_block -rebind command:

```
icc2_shell> link_block -rebind
Using libraries: lib_A lib_B tech32lvt
Visiting block lib_A:block1.design
Design 'block1' was successfully linked.
```

See Also

- Block Labels
- Block Types
- Library Cells
- Blocks

Creating a Block

To create a new block for holding design data, use the create_block command:

```
icc2_shell> create_block MUX2
Information: Creating block 'MUX2.design' in library 'my_lib'. (DES-013)
{my_lib:MUX2.design}
```

This command creates the block in memory and sets it as the current block. To save the new block, use the save_block command.

To create a new block that overwrites an existing block of the same name:

```
icc2_shell> create_block -force MUX2
```

The read_verilog command implicitly creates a new block to contain the Verilog netlist read by the command, so there is no need for the create_block command in that situation.

See Also

- Block Naming Conventions
- Opening a Block
- Saving a Block
- Copying a Block
- Closing a Block
- Reading a Verilog Netlist
- Blocks

Opening a Block

To open an existing saved block for viewing or editing, use the `open_block` command:

```
icc2_shell> open_block my_lib:MUX2
Opening block 'my_lib:MUX2.design'
{my_lib:MUX2.design}
```

The tool opens the specified block and sets it as the current block. If you specify the library name with the block name (for example, `my_lib:MUX2`), the tool also opens that library, if it is not already open, and sets it as the current library.

To open a block in read-only mode:

```
icc2_shell> open_block -read my_lib:MUX2
```

See Also

- Block Naming Conventions
- Creating a Block
- Saving a Block
- Copying a Block
- Closing a Block
- Blocks

Setting the Current Block

The *current block* is the default block affected by block-related commands. By default, the block most recently opened is the current block. To explicitly set a block to be the current block, use the `current_block` command:

```
icc2_shell> current_block MUX2
{my_lib:MUX2.design}
```

The tool sets the specified block as the current block. The block must be already open. If you specify the library name together with the block name (for example, `my_lib:MUX2`), the tool also sets that library as the current library.

To determine which block is the current block, use the `current_block` command by itself:

```
icc2_shell> current_block
{my_lib:MUX2.design}
```

See Also

- Block Naming Conventions
- Creating a Block
- Opening a Block
- Querying Blocks
- Saving a Block
- Copying a Block
- Closing a Block
- Blocks

Querying Blocks

You can get the following types of information about the open blocks:

- The current block – use the `current_block` command.

  ```
  icc2_shell> current_block
  {my_lib:MUX2.design}
  ```

- The blocks in the library – use the `list_blocks` command.

  ```
  icc2_shell> list_blocks
  Lib lib_A /path/lib_dir/lib_A current
    +> 0 MUX2.design Apr-30-21:01
    +> 0 MUX3.design Apr-30-21:29 current
  ```

```
Lib lib_B /path/lib_dir/lib_B
  +  0 ADDR1.design Apr-29-20:54
3

icc2_shell> list_blocks lib_B
Lib lib_B /path/lib_dir/lib_B
  +  0 ADDR1.design Apr-29-20:54
1
```

By default, the `list_blocks` command lists all blocks in all open libraries, but not blocks in associated reference libraries. Use the `list_blocks` command options to override the default behavior:

```
icc2_shell> list_blocks libA      # List blocks in library lib_A only
...
icc2_shell> list_blocks -ref_libs     # List blocks in ref libs also
...
icc2_shell> list_blocks -lib_cells     # List lib_cell blocks also
...
```

- Create a block collection – use the `get_blocks` command.

```
icc2_shell> set my_blks1 [get_blocks] # Get open blocks in current lib
{lib_A:MUX2.design lib_A:MUX3.design}

icc2_shell> set my_blks2 [get_blocks -all]  # Get blocks in open libs
{lib_A:MUX2.design lib_A:MUX3.design lib_B:ADDR1.design}

icc2_shell> set my_blks2 [get_blocks *ADDR* -all]  # Search string
{lib_B:ADDR1.design}
```

See Also

- Block Naming Conventions
- Querying a Design Library
- Opening a Block
- Setting the Current Block
- Saving a Block
- Copying a Block
- Closing a Block
- Blocks

Saving a Block

By default, a block you create, edit, or copy exists only in memory and is lost when you end the tool session. To save a block to disk, use the `save_block` command:

```
icc2_shell> save_block MUX2
```

By default, the command saves only the specified block and ignores any lower-level blocks. To also save the lower-level blocks of the specified block, use the `-hierarchical` option:

```
icc2_shell> save_block MUX2 -hierarchical
```

To save more than one block, use the `-blocks` option:

```
icc2_shell> save_block -blocks [get_blocks]
```

Note:
You cannot use the `-as` option with the `-blocks` option.

To save a block with a new name without affecting the original saved block or current block setting, use the `-as` option:

```
icc2_shell> save_block MUX2 -as MUX2B
Information: Saving 'lib_A:MUX2.design' to 'lib_A:MUX2B.design' ...
```

To save a block with a new or different label without affecting the original saved block or current block settings:

```
icc2_shell> save_block MUX2 -label ver1
Information: Saving 'lib_A:MUX2.design' to 'lib_A:MUX2/ver1.design' ...
```

You can save a block in compressed format to reduce the file size by using one of the following methods:

- To compress only the current view of the current block, specify the `-compress` option with the `save_block` command.
- To compress all views of the current block and subblocks, specify the `-hierarchical` and `-compress` options with the `save_block` command.

See Also

- Creating a Block
- Opening a Block
- Copying a Block
- Closing a Block
- Blocks

Copying a Block

To copy a block to a new block in the same library and view or a different library and view, use the `copy_block` command. If you do not explicitly specify a view for the source block and the destination block, all available views of the source block are copied to the destination block, and the design view of the destination block is returned. The block's source library does not need to be open. The tool opens the library if it is not already open.

By default, copied blocks are stored in memory only. To list the blocks stored in memory, run the `list_blocks` command. To save the copied block to disk, set the `design.on_disk_operation` application option to `true` before running the `copy_block` command. The default is `false`.

The following example copies Top block to Top2 block for subsequent block exploration and saves the Top2 block to disk. The destination library and view are taken from the original block:

```
icc2_shell> set_app_options design.on_disk_operation true
icc2_shell> copy_block -from_block Top -to_block Top2
{"lib:Top2.design"}
```

See Also

- Creating a Block
- Opening a Block
- Saving a Block
- Closing a Block
- Blocks

Closing a Block

When you no longer need access to a block, you can close it with the `close_blocks` command:

```
icc2_shell> close_blocks MUX2
Closing block 'lib_A:MUX2.design'
1
```

To automatically save a block before closing it:

```
icc2_shell> close_blocks -save MUX2
...
```

Alternatively, you can first save the block by using the `save_block` command.

To close a block and deliberately discard recent changes to the block:

```
icc2_shell> close_blocks -force MUX2
```

If the block *open count* is 1 or more after you execute the `close_lib` command, the block remains open.

To remove unwanted blocks from disk, use the `remove_blocks` command. Note that removed blocks cannot be recovered.

Do not attempt to add, move, or delete block files directly using operating system commands, as doing so can corrupt the database.

See Also

- Block Naming Conventions
- Creating a Block
- Opening a Block
- Setting the Current Block
- Saving a Block
- Copying a Block
- Block Open Count
- Blocks

Block Open Count

The *open count* of a block is an integer specifying the number of times the block has been opened. The tool monitors the open count to keep the block open as long as you need access to it, based on the number of `open_block` and `close_blocks` commands used on the block. The tool assumes that you want the block to be closed when the number of `close_blocks` commands equals the number of `open_block` commands.

When you use the `create_block` or `open_block` command for the first time, the block's open count is set to 1. Each time the same block is reopened, its open count is incremented by 1:

```
icc2_shell> open_block MUX3
Opening block 'lib2:MUX3.design'
...
icc2_shell> open_block MUX3
Information: Incrementing open_count of block 'lib2:MUX3.design' to 2...
{lib2:MUX3.design}
...
icc2_shell> open_block MUX3
Information: Incrementing open_count of block 'lib2:MUX3.design' to 3...
{lib2:MUX3.design}
...
```

Opening a hierarchical block also opens its associated lower-level blocks, which increments the open count for each of the lower-level blocks as well.

The `close_blocks` command decrements the block's open count by 1. If the new count is 1 or more, the block remains open; or if the new count is 0, the block is closed and removed from memory.

```
icc2_shell> close_blocks MUX3
Information: Decrementing open_count of block 'lib2:MUX3.design' to 2...
...
icc2_shell> close_blocks MUX3
Information: Decrementing open_count of block 'lib2:MUX3.design' to 1...
...
icc2_shell> close_blocks MUX3
Closing block 'lib2:MUX3.design'
1
```

To close a block irrespective of the open count:

```
icc2_shell> close_blocks MUX3 -purge
Closing block 'lib2:MUX3.design'
```

To determine the open count of a block without affecting the count, use the `get_attribute` command:

```
icc2_shell> get_attribute [get_blocks MUX3] open_count
3
```

Using the `save_block` command does not affect the open count.

To determine whether a block is already open:

```
icc2_shell> open_block -check MUX3
Opening block 'my_lib:MUX3.design'
{my_lib:MUX3.design}
```

If the block is already open, it stays open without any change to the block's open count. If the block is not already open, the command returns an error message.

See Also

- Block Naming Conventions
- Creating a Block
- Opening a Block
- Saving a Block
- Copying a Block
- Closing a Block
- Blocks

Block Types

A block contains both physical layout data and functional design information to support synthesis, analysis, and optimization of the physical design.

With respect to the design hierarchy, there are two basic types of blocks: *designs*, which are the top-level and intermediate-level blocks, and *library cells*, which are the leaf-level blocks. Designs are stored in *design libraries*, whereas library cells are stored in *cell libraries*.

A block can have multiple *views*, with each view containing an alternative representation of a design. Each type of view serves a specific purpose.

For more information, see

- Designs
- Library Cells
- Block Labels
- Block Views

See Also

- Design Libraries
- Reference Library List
- Creating a Block
- Opening a Block
- Saving a Block
- Copying a Block
- Closing a Block
- Blocks

Designs

A *design* is a representation of a circuit that consists of block instances and connections. A typical way to build a design is to read a Verilog netlist into a new block and have the IC Compiler II tool perform placement and routing based on the netlist contents.

The lower-level blocks used to build a design can be library cells, macros (hierarchical cells that represent lower-level designs), or a combination of both types of cells. A design can itself be used as a macro instance inside another design at a higher level of hierarchy.

See Also

- Design Libraries
- Reference Library List
- Specifying a Design Library's Reference Libraries
- Block Labels
- Blocks

Library Cells

A *library cell* (lib_cell object) is a representation of a leaf-level function such as logic gate or I/O pad. A library cell model includes physical information such as layer shapes, pins, and routing blockages; and functional information such as logic, timing, and power parameters. Library cells are stored in *cell libraries*.

The following commands operate on library cells:

- `get_lib_cells` – Creates a collection of library cells
- `report_lib_cells` – Generates a report on specified library cells
- `set_lib_cell_purpose` – Specifies how a collection of library cells can be used, either including or excluding purposes such as clock tree synthesis, optimization, or hold fixing

To list or query library cells and their attributes:

```
icc2_shell> list_blocks -lib_cells
...
icc2_shell> get_blocks -of_objects [get_libs stdhvt] -lib_cells
...
icc2_shell> list_attributes -application -class lib_cell
...
icc2_shell> get_attribute -objects [get_lib_cells stdhvt/DFF1] -name area
7.8785
icc2_shell> report_attributes -application -class lib_cell \
  [get_lib_cells stdhvt/DFF1A]
...
```

A library cell model typically consists of different block *views* used for different purposes:

- *design* view for storing the physical and functional details
- *frame* view for placement and routing
- *timing* view for storing the timing, power, and noise characteristics of the cell
- *layout* view for library cell preparation and mask generation, containing only the geometric data obtained from a GDSII file

For information about preparing library cells and cell libraries, see the *IC Compiler II Library Preparation User Guide*.

See Also

- Design Libraries
- Reference Library List
- Cell Libraries

- Block Views
- Blocks

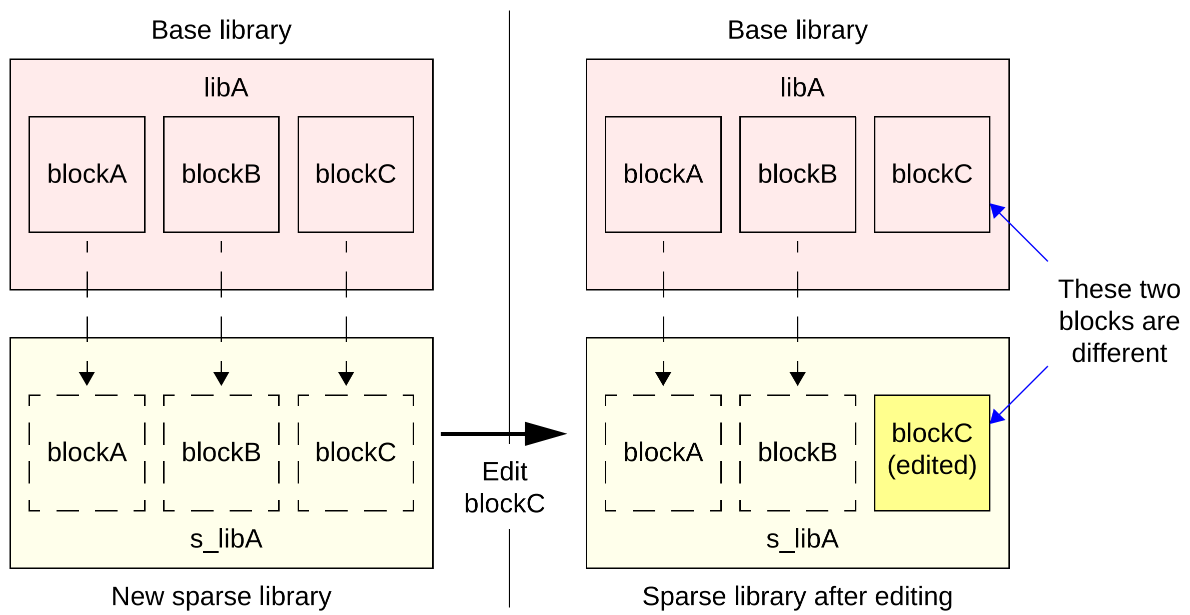
Sparse Libraries

When multiple users or projects share a common set of design data, you can use *sparse libraries* to efficiently share the common data. A single central *base library* contains the stable data shared between different users or projects, while multiple sparse libraries build upon that data by modifying the blocks in the base library or by adding new blocks.

You create a sparse library by using the `create_lib` command with the `-base_lib` option to specify the related base library. The sparse library initially uses the technology data, reference library list, and blocks of the base library. After you create a sparse library, you can modify its contents without affecting the base library.

The following figure shows how the tool maintains the contents of a sparse library.

Figure 1-13 Base Library and Sparse Library Relationships

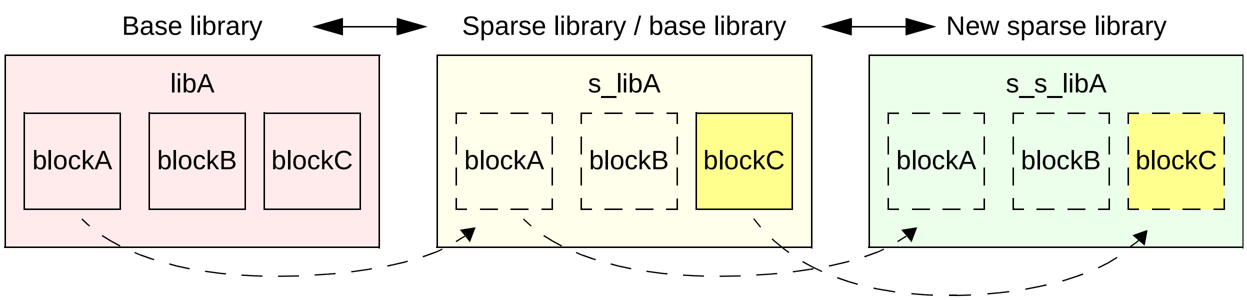


When you create a new sparse library, it points to the contents in the related base library, so its contents are exactly the same as the base library. When you edit a block in the sparse library, such as blockC in the figure, that block becomes different from the one in the base library, so that block is saved locally in the sparse library directory.

Multiple users can create multiple sparse libraries associated with the same base library. These users concurrently share the same data in the base library, without any chance of corrupting the shared database. All changes they make to their sparse libraries affect only their own libraries, not the base library.

The base library associated with a sparse library can itself be a sparse library, as shown the following figure. Library s_libA is a sparse library to libA but also a base library to the new sparse library s_s_libA. To find the blocks of the sparse library s_s_libA, the tool traverses the chain of sparse-base libraries until it finds a locally saved block.

Figure 1-14 Sparse Library Used as a Base for Another Sparse Library



For more information, see

- Creating a Sparse Library
- Remote and Local Blocks
- Reverting Blocks in a Sparse Library
- Synchronizing a Sparse Library to a Base Library
- Synchronizing a Base Library to a Sparse Library

Creating a Sparse Library

To create a new sparse library, use the `create_lib` command with the `-base_lib` option to specify the related base library:

```
icc2_shell> create_lib -base_lib libA s_libA
...
Information: Creating Sparse View library 's_libA' with base library
 'libA'. (NDM-103)
...
{s_libA}
```

The base library can be either open or closed. If closed, it must be accessible in the paths specified by the `search_path` variable. If you specify a simple name, the command uses the `search_path` variable to search for the base library and uses the first one that it finds. The base library can itself be a sparse library.

The new sparse library initially contains the same blocks, reference library list, and technology data as its base library. When you add, edit, or remove blocks, the tool stores the changes in the local library directory.

To save a sparse library, use the `save_lib` command, just like any library:

```
icc2_shell> save_lib
Saving library 's_libA'
1
```

To determine whether a library is sparse, query its `is_sparse` attribute:

```
icc2_shell> get_attribute -name is_sparse [get_libs libA]
false
icc2_shell> get_attribute -name is_sparse [get_libs s_libA]
true
```

To determine a sparse library's base library, query its `base_lib` attribute:

```
icc2_shell> get_attribute -name base_lib [get_libs s_libA]
libA
```

See Also

- Sparse Libraries
- Remote and Local Blocks
- Synchronizing a Sparse Library to a Base Library
- Synchronizing a Base Library to a Sparse Library

Remote and Local Blocks

You can use commands such as `create_block`, `open_block`, `save_block`, `copy_block`, and `reopen_block` to edit the contents of a sparse library, just like any library.

A block saved and maintained in the base library called is a *remote block*, whereas a block saved and maintained in the sparse library directory is a called a *local block*. Opening or copying a remote block in a sparse library is an in-memory operation. When you save the block, the block becomes local and is saved in the local sparse library directory. The local block is maintained separately from the remote block.

If you save a hierarchical remote block, that block and its lower-level blocks are all made local and saved in the local sparse library directory.

To determine whether an open block is remote or local, query the block's `is_remote` attribute:

```
icc2_shell> open_block BLK1   # Open remote block stored in base library
...
```

```
icc2_shell> create_block BLK5   # Create local block in sparse library
...
icc2_shell> get_attribute -name is_remote [get_blocks BLK1]
true
icc2_shell> get_attribute -name is_remote [get_blocks BLK5]
false
...
icc2_shell> save_block BLK1              # Saving BLK1 makes it local
...
icc2_shell> get_attribute -name is_remote [get_blocks BLK1]
false
```

To create a collection of all local blocks in the current library:

```
icc2_shell> get_blocks -filter "is_remote == false"
{BLK1 BLK5}
```

To create a collection of all open sparse libraries:

```
icc2_shell> get_libs -filter "is_sparse == true"
{s_libA x_libA y_libA}
```

Commands that operate on blocks in a sparse library only affect that library; they have no effect on blocks in the base library. For example, if you remove a remote block from a sparse library using the `remove_blocks` command, that block ceases to exist in the sparse library and can no longer be opened there, even though the same block still exists in the base library.

See Also

- Sparse Libraries
- Creating a Sparse Library
- Reverting Blocks in a Sparse Library
- Synchronizing a Sparse Library to a Base Library
- Synchronizing a Base Library to a Sparse Library

Reverting Blocks in a Sparse Library

To *revert* a block means to discard the local block saved in a sparse directory and go back to using the identically named block in the base library. To revert one or more blocks, use the `revert_blocks` command. This command works only when the blocks are closed, as shown in the following example.

```
icc2_shell> create_lib -base_lib libA s_libA
...
Information: Creating Sparse View library 's_libA' with base library
```

```
 'libA'. (NDM-103)
...
icc2_shell> open_block BLK1
Information: Updating library catalog of Sparse View library 's_libA'
...
Information: Adding block 'BLK1.design' to Sparse View library
 's_libA'. (NDM-105)
...
icc2_shell> save_block BLK1
...
icc2_shell> get_attribute -name is_remote [get_blocks BLK1]
false
icc2_shell> close_blocks BLK1        # Must close block before reverting
...
icc2_shell> revert_blocks BLK1
...
Information: Reverted design 's_libA:BLK1.design' and removed it ...
1
icc2_shell> open_block BLK1  # Reopen remote block stored in base library
...
icc2_shell> get_attribute -name is_remote [get_blocks BLK1]
true
```

Note that *reverting* is different from *removing* a block. Removing a block discards the local block and also makes the original remote block inaccessible.

If you accidentally remove a block when you meant only to revert it, you can recover the block from the base library by rebasing the sparse library with the `set_base_lib` command, without affecting other local blocks in the sparse library. For details, see Synchronizing a Sparse Library to a Base Library.

See Also

- Sparse Libraries
- Creating a Sparse Library
- Remote and Local Blocks
- Synchronizing a Sparse Library to a Base Library
- Synchronizing a Base Library to a Sparse Library

Synchronizing a Sparse Library to a Base Library

In some situations, you might need to *rebase* a sparse library, which means to update the association between the sparse library and its base library without affecting the local blocks in the sparse library. For example,

- The base library has been renamed or moved.

- The technology data, reference library list, or blocks in the base library have been changed.
- You removed (instead of reverted) a block from the sparse library, and you need to recover the original block from the base library.
- You want to associate a sparse library with a different base library.

To rebase a sparse library, first close all the open blocks in the library, then use the `set_base_lib` command to specify the new or modified base library:

```
icc2_shell> current_lib s_libA             # Existing sparse library
{s_libA}
icc2_shell> close_blocks *                 # Close all open blocks
...
icc2_shell> set_base_lib -library s_libA -base_lib libB
...
Information: Rebasing Sparse View library 's_libA' to base library
 'libB'. (NDM-103)
...
Information: Updating library catalog of Sparse View library 's_libA'
 from base library 'libB'. (NDM-104)
...
 -- marking removal of design: s_libA:BLK1.design missing in base-lib:
 'libB'
 -- marking addition of design: libB:BLKZ.design
Information: Removing block 's_libA:new_block.design' from Sparse View
 library 's_libA'. (NDM-106)
Information: Adding block 'BLKZ.design' to Sparse View library
 's_libA'. (NDM-105)
```

Rebasing a sparse library updates the contents of the library based on the contents of the new or updated base library. Remote blocks no longer existing in the base library are removed from the sparse library, and any new remote blocks are added to the sparse library. Local blocks in the sparse library are not affected, irrespective of the new base library contents.

You can specify the path to the base library as a simple name, a relative path, or an absolute path. If you specify a simple name, the command uses the `search_path` variable to search for the base library and uses the first one that it finds.

To query the path to the base library from a sparse library, look at the sparse library's `base_lib_path` attribute:

```
icc2_shell> get_attribute -name base_lib_path [get_libs s_libA]
/remote/path/user/icc2data/tmp/libA
```

Setting the `base_lib_path` attribute using the `set_attribute` command has the same effect as using the `set_base_lib` command.

See Also

- Synchronizing a Base Library to a Sparse Library

Synchronizing a Base Library to a Sparse Library

A base library typically contains stable data that can be shared among multiple sparse libraries, and the related sparse libraries contain only blocks that differ from the common base. However, in some situations, you might want to move data from a sparse library to the base library so that the other sparse libraries can have access to that data.

The IC Compiler II tool does not provide commands specifically for updating a base library with data from a sparse library. However, you can manually update the base library with data taken from a sparse library.

For example, suppose you have a base library named libA containing a block BLK1, and a sparse library named s_libA containing a newer version of BLK1. To update the base library with the newer version of BLK1, use the following script to copy the newer version back to the base library.

```
open_lib s_libA                          # Open sparse library
open_lib -edit libA                      # Open base library for editing
copy_block -from_block s_libA:BLK1 -to_block libA:BLK1 # Copy block
save_lib libA                            # Save modified base library
```

See Also

- Synchronizing a Sparse Library to a Base Library

File Attachments

You can attach any type of file to a block or library. The attached file becomes part of the block or library. You can use this feature to store any important information related to the block or library such as design specifications, Tcl scripts, or signoff documents.

The following commands support the file attachment feature:

- `add_attachment` – Attaches a file to a specified block or library; copies the file into the block or library.
- `report_attachments` – Reports the attachments on a block or library.
- `open_attachment` – Opens a file handle for the attachment, like the Tcl `open` command, allowing you to read or restore the file.
- `remove_attachments` – Removes one or more attached files from a block or library.

For details, see the man pages for the `add_attachment`, `report_attachments`, `open_attachment`, and `remove_attachments` commands.

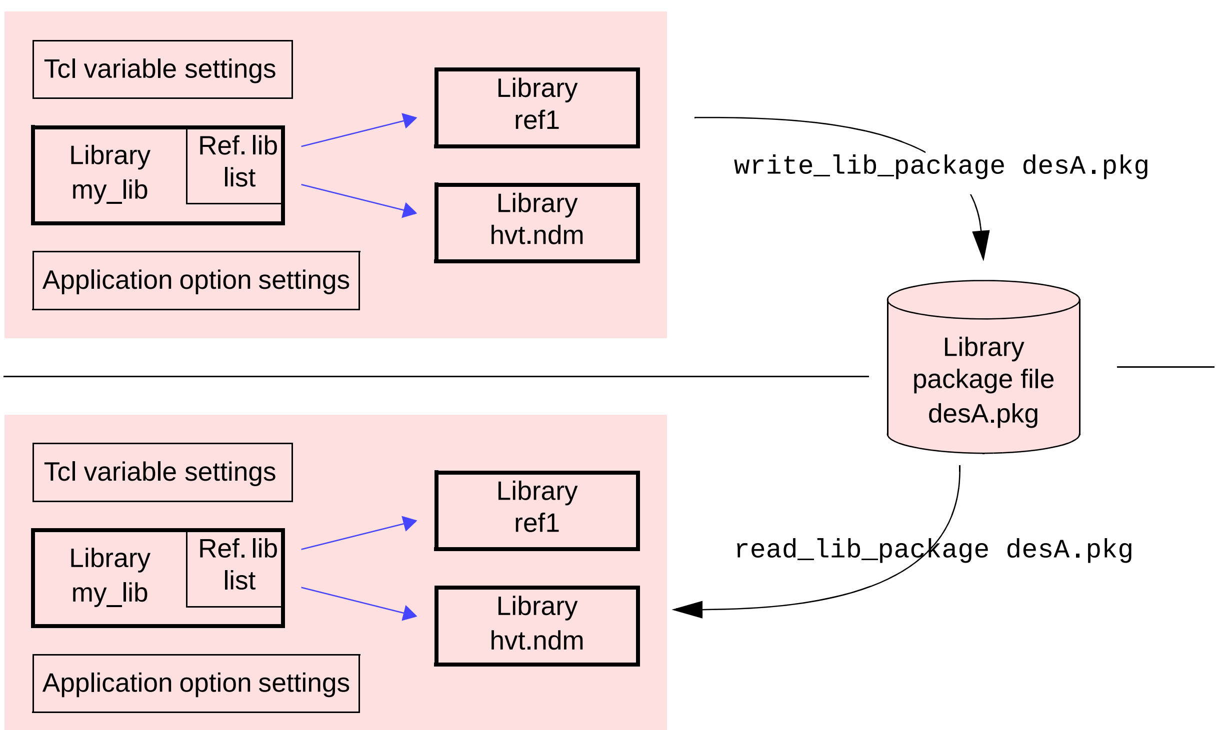
Library Packaging

To transfer or archive a chip design, you can put all of the library data and environment settings into a single file called a *library package*:

- The `write_lib_package` command puts the library data and environment settings into a compressed library package file.
- The `read_lib_package` command restores the original data and environment settings from the library package file.

The following figure shows how to create and restore a library package.

Figure 1-15 Creating a Library Package and Restoring the Library Data



For details, see

- Creating a Library Package
- Restoring Data From a Library Package

Creating a Library Package

To create a library package, use the `write_lib_package` command:

```
icc2_shell> current_lib                    # Checks current library
{my_lib}
icc2_shell> write_lib_package desA.pkg  # Creates package for current lib
1
```

By default, the `write_lib_package` command creates a library package file that contains the current design library, its reference libraries, and environment settings such as Tcl variables, application options, and the current block.

The command automatically compresses the library package, reducing disk usage. You can disable compression by setting the `lib.setting.compress_lib_package` application option to `false` before running the `write_lib_package` command.

The `write_lib_package` command has options to

- Create a package for a library other than the current library (`-library`)
- Include only a specified subset of the design library blocks in the package (`-blocks`)
- Exclude specified reference libraries from the package (`-exclude_ref_libs`)
- Report the command progress and application options being saved (`-verbose`)

The command writes the library package file to the current working directory. The file name can have any extension, such as .pkg, or no extension at all.

See Also

- Library Packaging
- Restoring Data From a Library Package
- Design Libraries
- Blocks

Restoring Data From a Library Package

To restore the data stored in a library package, use the `read_lib_package` command:

```
icc2_shell> read_lib_package /path/pkgs/desA.pkg
Information: Loading library file '/path/working2/my_lib' (FILE-007)
Information: Loading library file '/path/working2/my_lib/reflibs/ref1'
Information: Loading library file '/path/working2/my_lib/reflibs/hvt.ndm'
Opening block 'my_lib:top.design'
1
```

```
icc2_shell> current_lib
{my_lib}
icc2_shell> report_ref_libs
...
   Name  Path                          Location
   ------------------------------------------
*  ref1  my_lib/reflibs/ref1          /path/working2/my_lib/reflibs/ref1
*+ hvt   my_lib/reflibs/hvt.ndm       /path/working2/my_lib/reflibs/hvt.ndm
    "*" = Library currently open
    "+" = Library has technology information
1
```

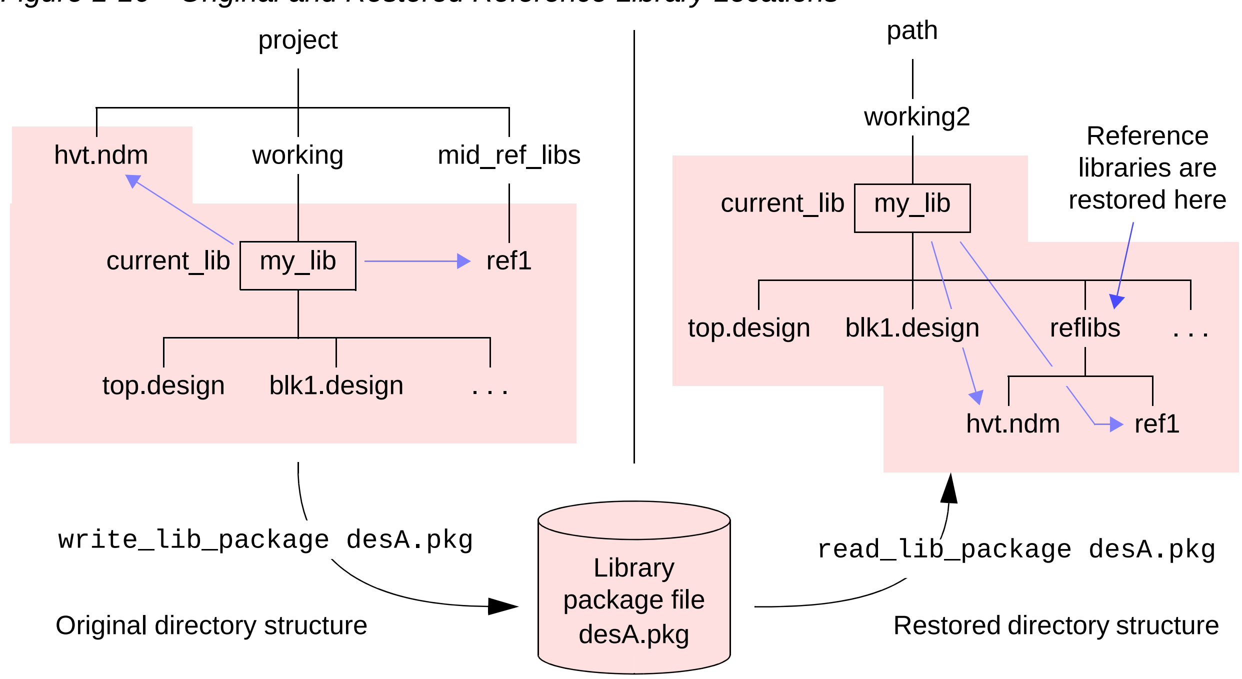
By default, the `read_lib_package` command restores the library to the current working directory, unpacks and restores the reference libraries, and invokes the saved environment settings such as Tcl variables, application options, open blocks, and current block.

The `read_lib_package` command has options to:

- Specify a destination directory for writing the library (`-destination`)
- Overwrite an existing library in the destination directory (`-overwrite`)

The unpacking of the libraries in the reference library list does not preserve the original absolute or relative locations of those libraries. Instead, the tool restores all of the reference libraries to a single subdirectory named “reflibs” under the directory where the design library is restored, as shown in the following figure.

Figure 1-16 Original and Restored Reference Library Locations



In the restored design library, the library list correctly points to the new locations of the restored reference libraries. You can begin working on the restored design immediately, without concern for the changed reference library locations.

See Also

- Library Packaging
- Creating a Library Package
- Design Libraries
- Blocks

2

Data Import and Export

When you work with a design in the IC Compiler II tool, you open and edit the design view of a block stored in a design library.

You can read and save the design data as described in the following topics:

- Reading a Verilog Netlist
- Allowing Incomplete or Inconsistent Design Data
- Saving a Design in ASCII Format
- Writing a Design in GDSII or OASIS Stream Format
- Reading and Writing LEF Data
- Reading and Writing DEF Data
- IC Compiler II Layer Mapping File
- IC Compiler Layer Mapping File Syntax
- Changing Object Names

Reading a Verilog Netlist

The starting point for physical implementation is a design netlist in the Verilog hardware description language. The IC Compiler II tool reads one or more structural gate-level Verilog netlists and uses this information as the basis for placement and routing.

To read a Verilog netlist into the implementation tool,

1. Read the Verilog netlist files for the design by using the `read_verilog` command:

   ```
   icc2_shell> read_verilog ../my_data/my_des.v
   ```

   The command implicitly creates a new block named *top_module_name*.design and reads the Verilog netlist data into the block.

   The command identifies the top-level module as the only module in the Verilog file not used as an instance by any other modules. If the file has multiple modules not used as instances, you need to identify the one that is the top-level module:

   ```
   icc2_shell> read_verilog -top my_top ../my_data/my_des.v
   ```

   To explicitly specify a new block name other than *top_module_name*.design:

   ```
   icc2_shell> read_verilog -top my_top -design desA ../my_data/my_des.v
   ```

2. If you are not using a UPF file to specify the power infrastructure, and if you want to explicitly specify the power and ground supply port and net names (the default names are VDD and VSS), specify the names as application options. For example,

   ```
   set_app_options -name mv.pg.default_power_supply_port_name -value VD1
   set_app_options -name mv.pg.default_power_supply_net_name -value VD1
   set_app_options -name mv.pg.default_ground_supply_port_name -value VS1
   set_app_options -name mv.pg.default_ground_supply_net_name -value VS1
   ```

3. Link the block by using the `link_block` command:

   ```
   icc2_shell> link_block
   ...
   Design 'desA' was successfully linked
   1
   ```

   The linking process resolves the cell references by searching the reference libraries in the order specified by the design library's reference library list.

To save the block to disk, use the `save_block` command.

To write out the Verilog netlist for the block, use the `write_verilog` command.

See Also

- Creating a Design Library
- Opening a Design Library
- Saving a Block
- Saving a Design in ASCII Format
- Changing Object Names for Verilog Output

Allowing Incomplete or Inconsistent Design Data

Updating a design can result in data mismatch. For example, the number of pins on a block might not match its representation at the top-level. By default, the tool does not link any block with incomplete or mismatched data.

To enable the tool to successfully link a block even if it has incomplete or inconsistent data, set a *mismatch configuration* for the block. This allows the tool to either ignore or fix different types of mismatching data and continue the linking process. Blocks linked with mismatching data can be used only for preroute feasibility analysis.

You set the mismatch configuration by using the `set_current_mismatch_config` command, as described in Setting a Mismatch Configuration. These are the predefined mismatch configuration types:

- `default` (default)

  This configuration does not allow any mismatching data.

- `auto_fix`

  This configuration allows the linker to repair all the mismatch types shown in Table 2-1.

Alternatively, you can define your own mismatch configuration using the `create_mismatch_config` command. The tool either ignores or repairs each of the mismatch types shown in Table 2-1, as specified by the `create_mismatch_config` command. For more information, see Creating a Custom Mismatch Configuration.

Table 2-1 Mismatch Configuration Error Types

Error type	Description
bus_bit_naming Supported repairs:	A bus is bit-blasted in the reference cell, but is defined as a bus in a different module.
bus_bit_blast_naming	Reconstructs the buses using the style specified in the link.bit_blast_naming_style application option.
missing_logical_reference Supported repairs:	The netlist contains an unresolved reference.
create_blackbox	Creates a physical black box for the cell, which enables the missing cell to be placed in the floorplan and retained.
create_empty_logic_module	Creates an empty logic module for the cell; you can create a physical black box for this module and apply a timing model to it.
record_unbound_instance	Records the unresolved reference, but does not perform any repairs.
missing_port Supported repairs:	The number of ports differs between the netlist and the reference library.
create_missing_port_mapping	Creates a placeholder port on the cell with a missing port, either the library cell or the cell instance in the netlist.
port_name_case Supported repairs:	Pin name mismatch between the netlist and the reference library due to case-sensitivity.
record_port_name_case_insensitivity	Records the mismatch but does not perform any repairs.
reference_name_case Supported repairs:	Cell name mismatch between the netlist and the reference library due to case-sensitivity.
record_cell_name_case_insensitivity	Records the mismatch but does not perform any repairs.

To see the mismatches identified and fixed by the tool during block linking, use the report_design_mismatch command after linking is complete. For more information, see Reporting Mismatch Configurations.

Creating a Custom Mismatch Configuration

To create a custom mismatch configuration,

1. Create the configuration by using the `create_mismatch_config` command.

   By default, all the error types are treated as error conditions. To specify a different mismatch configuration as the basis for the new user-defined configuration, use the `-ref_config` option.

2. Specify how to handle a specific error type by setting its `action` attribute for the configuration.

   Specify one of the following values for the `action` attribute: `error`, `ignore`, or `repair`. The initial setting of this attribute comes from the reference configuration used to create the user-defined configuration.

3. If you set the `action` attribute to `repair`, specify the repair process to use by setting the `current_repair` attribute of the error type for the configuration.

   To determine the available repairs for an error type, query the `available_repairs` attribute of the error type. Table 2-1 describes many of the error types and their available repairs.

For example, the following commands create a user-defined mismatch configuration named user_def that repairs unresolved references by creating physical black box cells:

```
icc2_shell> create_mismatch_config user_def
icc2_shell> set_attribute [get_mismatch_type missing_logical_reference] \
   action(user_def) repair
icc2_shell> set_attribute [get_mismatch_type missing_logical_reference] \
   current_repair(user_def) create_blackbox
```

See Also

- Allowing Incomplete or Inconsistent Design Data
- Setting a Mismatch Configuration
- Reporting Mismatch Configurations

Setting a Mismatch Configuration

By default, the tool performs checks for mismatched data using the `default` mismatch configuration, which does not allow any mismatches. To set a different mismatch configuration, use the `set_current_mismatch_config` command.

For example, to set the mismatch configuration to the predefined auto-fix configuration, use the following command:

```
icc2_shell> set_current_mismatch_config auto_fix
1
```

This configuration allows the tool to repair all the mismatch types shown in Table 2-1.

To see all available mismatch configurations, use the `report_mismatch_configs -all` command.

See Also

- Allowing Incomplete or Inconsistent Design Data
- Creating a Custom Mismatch Configuration
- Reporting Mismatch Configurations

Reporting Mismatch Configurations

You can report mismatch configurations by using the commands described in the following table:

Report	Command
Report the name of the current mismatch configuration, such as `default` or `auto_fix`	`get_current_mismatch_config`
Report the details of the current mismatch configuration	`report_mismatch_configs`
Report all available mismatch configurations	`report_mismatch_configs -all`
Restrict the report to specific mismatch configurations	`report_mismatch_configs -config_list` *list*
Report the mismatches identified and fixed by the tool during block linking	`report_design_mismatch`

In the following example, the tool reports the details of the current mismatch configuration, including the mismatch error types and repair strategies:

```
icc2_shell> report_mismatch_configs

...

Config : default
------------------------------------------------------------------------------------
Mismatch-Type             Action  ...     Repair Strategy  Available Strategies
------------------------------------------------------------------------------------
...
missing_logical_reference error   ...     record_unbound_instance
                                          {create_blackbox create_empty_logic_module }
missing_port              error   ...     {delete_extra_port }
port_name_case            error   ...     record_port_name_case_insensitivity
                                          {record_port_name_case_insensitivity }
reference_name_case       error   ...     record_cell_name_case_insensitivity
                                          {record_cell_name_case_insensitivity }
...
bus_bit_naming            error   ...     bus_bit_blast_naming
                                          {bus_bit_blast_naming }
1
```

See Also

- Allowing Incomplete or Inconsistent Design Data
- Setting a Mismatch Configuration
- Creating a Custom Mismatch Configuration

Saving a Design in ASCII Format

In addition to saving a design as a block in a design library, you can also save the design as a set of ASCII text files for transfer to third-party tools. These are the types of ASCII design data files and the commands to generate them:

- Verilog netlist

```
icc2_shell> write_verilog -hierarchy all my_design.v
...
```

- DEF file (Design Exchange Format physical design description)

```
icc2_shell> write_def my_design.def
...
```

- UPF file (IEEE 1801 Unified Power Format power intent specification file)

```
icc2_shell> save_upf my_design.upf
...
```

- SDC files (Synopsys Design Constraints files)

  ```
  icc2_shell> write_sdc -output my_design.sdc
  ...
  ```

  You can also use the `write_script` command to write a script that contains more complete design and constraint setup information than what is written to the SDC file.

Writing a Design in GDSII or OASIS Stream Format

You can write out the physical data of a design in GDSII or OASIS stream format for tapeout or for exporting the data to third-party tools. The `write_gds` and `write_oasis` commands are available in both the implementation tool (icc2_shell) and library manager tool (icc2_lm_shell).

To write out the current block in GDSII format:

```
icc2_shell> write_gds my_design.gds
```

To write out a specified block in GDSII format:

```
icc2_shell> write_gds -library libA -design blockB my_design.gds
```

To write out the current block in GDSII format and map the IC Compiler II layers to GDSII layers as specified in a layer mapping file:

```
icc2_shell> write_gds -layer_map tech12.map my_design.gds
```

To write out the entire hierarchical design in GDSII format, including the current block and all of its lower-level blocks:

```
icc2_shell> write_gds -hierarchy all my_design.gds
```

To write out the design up to a specific level of hierarchy, use the `-child_depth` option with the `write_gds` or `write_oasis` command. In the following example, only the top block is written out in GDSII format:

```
icc2_shell> write_gds -child_depth 0 my_design.gds
```

By default, the `write_gds` or `write_oasis` command writes out the current block of the current library and maintains the original layer numbers as it converts the IC Compiler II data to stream format. The `write_gds` and `write_oasis` command options let you specify:

- The name of the library, design, and view to be written out
- The mapping of IC Compiler II database layers to stream-format layers
- The mapping of net, instance, and pin names in the conversion to stream format
- The manner of writing out pin objects (text or geometry)

- The handling of mask-shifted multiple-patterning data
- The layers, hierarchical blocks, and fill data to be included in the stream output
- The merging of design data with existing stream files

The `write_gds` and `write_oasis` commands have many more options not described here. For details, see the man page for the command. For information about the mapping of IC Compiler II layers to stream-format layers, see IC Compiler II Layer Mapping File.

Reducing the GDSII and OASIS File Size When Writing PG Vias

You can reduce the file size when you write out GDSII and OASIS files by compressing individual PG vias into via arrays, instead of creating a record for each via in the stream file:

- To compress PG vias into via arrays when generating GDSII layout files, set the `file.gds.pg_via_arrays` application option to `true` before running the `write_gds` command:

  ```
  icc2_shell> set_app_options -name file.gds.pg_via_arrays -value true
  icc2_shell> write_gds my_design.gds
  ```

- To compress PG vias into via arrays when generating OASIS layout files, set the `file.oasis.pg_via_arrays` application option to `true` before running the `write_oasis` command:

  ```
  icc2_shell> set_app_options -name file.oasis.pg_via_arrays -value true
  icc2_shell> write_oasis my_design.oasis
  ```

Both application options are `false` by default. This feature supports PG vias of the following types only: simple vias, custom vias, and simple array vias.

To write out the via properties when you compress PG vias into via arrays, use the `-via_property` option when you run the `write_gds` or `write_oasis` command. If you read the GDSII or OASIS file that includes the via properties back into the IC Compiler II tool, you can create a custom via by specifying the `-via_property` option with the `read_gds` or `read_oasis` command and using the same `-via_property` value that you used to write out the via properties.

Reading and Writing LEF Data

You can import and export library data in Library Exchange Format (LEF) so that you can use IC Compiler II data with external tools. The LEF format defines the elements of an IC process technology and associated library of cell models, including layer, via, placement site type, and macro cell definitions.

Exporting a Cell Library to LEF

To export a complete cell library named S32hvt to a LEF file, use the following `write_lef` command:

```
icc2_shell> write_lef -library S32hvt S32hvt.lef
```

The generated LEF file contains technology information and physical data for all cells in the S32hvt library.

Exporting a Block to LEF

To export a block named chipABC to a LEF file, use the following `write_lef` command:

```
icc2_shell> write_lef -design chipABC chipABC.lef
```

The generated LEF file contains the technology information of the design library that contains the specified block. If the specified block is a frame view, the command writes the cell LEF. If the specified block is a design view, the command writes the cell LEF for all reference designs instantiated in the block as frame views.

When you export data to LEF, you can optionally specify which section types and `PROPERTY` statements to write out, and whether to write out macros. For details, see `write_lef` man page.

Importing LEF Library Data

In the library manager tool, you can read library data in LEF format by using the `read_lef` command. For example,

```
icc2_lm_shell> read_lef S32std.lef -library my_std_lib
```

The `read_lef` command also has options to convert site names, to control the merging of data into existing cell libraries, to specify which section types and `PROPERTY` statements to use, and to specify whether to read macros. For details, see the *IC Compiler II Library Preparation User Guide* and the `read_lef` man page.

Reading and Writing DEF Data

The Design Exchange Format (DEF) defines the elements of an IC design that are related to the physical layout, including the placement and routing information, design netlist, and design constraints. It contains the design-specific information for a circuit and is a representation of the design at any point during the layout process. You can import and export library data in DEF format to use IC Compiler II data with external tools.

Exporting a Block to DEF

To write out the current block to a DEF file, including the physical layout, netlist, and design constraints, use the `write_def` command. If the current block is not linked, it is automatically linked and then written out.

The `write_def` command supports multithreading and uses the number of cores specified by the `set_host_options -max_cores` command. For information about multithreading, see “Enabling Multicore Processing” in the *IC Compiler II Implementation User Guide*.

The following example writes a compressed DEF file using 1000 units per micron:

```
icc2_shell> write_def -compress gzip -units 1000 top.def
```

The following example writes a DEF file that includes the floorplan data and excludes all other optional constructs and object types:

```
icc2_shell> write_def -include {rows_tracks bounds \
     cells blockages} fp.def
```

The following example writes a DEF file that includes the selected object:

```
icc2_shell> write_def -objects [get_site_rows unit_row_*]  row.def
```

The following example writes a DEF file that includes specific cell types with a specific physical status:

```
icc2_shell> write_def -cell_types {corner end_cap pad macro lib_cell} \
     -include_physical_status {placed} cell.def
```

The following example writes a DEF file that includes routed signal nets or clock nets only:

```
icc2_shell> write_def -routed_nets -net_types {signal clock} net.def
```

The following example writes design information across all levels of the physical hierarchy to a DEF file:

```
icc2_shell> write_def -traverse_physical_hierarchy hier.def
```

The following example specifies pairs of site names to match the floorplan’s site name with the site name in the technology data:

```
icc2_shell> write_def -convert_sites { {CORE unit} \
     {unitGA unitGA} } top.def
```

The following example writes a DEF file that includes the via definitions defined in the technology file:

```
icc2_shell> write_def -include_tech_via_definitions top.def
```

Importing DEF Data

To read design information from DEF files into the current block or a specified block, use the `read_def` command:

```
icc2_shell> read_def my_def.def
```

If you use relative path names to specify the DEF files, the command locates the files based on the `search_path` variable.

The following example reads new cells into the design:

```
icc2_shell> read_def -add_def_only_objects cells top.def
```

The following example reads only cells, nondefault routing rules, and nets from the DEF file:

```
icc2_shell> read_def -include {cells routing_rules nets} top.def
```

The following example reads a full-chip DEF file into a hierarchical design:

```
icc2_shell> read_def -traverse_physical_hierarchy hier.def
```

The `read_def` command can also remove all existing physical annotations from the current block before reading the DEF files and specify the mapping of DEF site names. For details, see the `read_def` man page.

Application Options

The following table describes the application options you can use to import and export DEF data. The application option settings are global.

Table 2-2 Application Options in the DEF Interface

Application option	Type	Default	Description
`file.def.append_shape_and_terminal`	Boolean	`true`	Controls whether the `read_def` command appends net wiring shapes and terminals
`file.def.fill_purpose_map`	list of strings	{}	Controls how the DEF fill shape use values are mapped to IC Compiler II shape purposes
`file.def.maintain_via_ladders`	Boolean	`false`	Controls whether the `write_def` and `read_def` commands write and read additional information about via ladders

Table 2-2 Application Options in the DEF Interface (Continued)

Application option	Type	Default	Description
`file.def.non_default_width_wiring_to_net`	Boolean	`false`	Controls whether the `write_def` command writes nets with a nondefault width
`file.def.pg_via_arrays`	Boolean	`true`	Controls whether the `write_def` command writes all regularly spaced vias on power and ground nets in compact via array format
`file.def.scan_cells_in_netlist_order`	Boolean	`false`	Controls whether the `write_def` command outputs floating scan cells in the netlist order
`file.def.set_pg_mask_fixed`	Boolean	`false`	Controls whether the `read_def` command sets all power and ground shape and via masks as fixed
`file.def.support_property_definitions`	Boolean	`false`	Controls whether the `read_def` and `write_def` commands preserve the DEF property definitions and values
`file.def.wrong_way_wiring_to_special_net`	Boolean	`false`	Controls whether the `write_def` command writes wiring with wrong-way default widths to the DEF SPECIALNETS section instead of the NETS section

IC Compiler II Layer Mapping File

The `write_gds`, `read_gds`, `write_oasis`, and `read_oasis` commands convert physical design data between a stream format and the IC Compiler II database format. By default, these commands preserve the layer numbers during the conversion process.

To change the layer numbers or other properties during the conversion process,

- Create a *layer mapping file* to specify the corresponding properties of the layers in stream format and the IC Compiler II database.
- Use the `-layer_map` option of the `write_gds`, `read_gds`, `write_oasis`, or `read_oasis` command to specify the name of the mapping file.

Each line in the layer mapping file specifies a layer in the IC Compiler II technology file and corresponding layer in the GDSII or OASIS file, as shown in the following examples.

Table 2-3 Layer Mapping File Entry Examples

Line in layer mapping file	Mapping effects for GDSII/OASIS output and input
`56 10`	The `write_gds` or `write_oasis` command converts all geometries on layer 56 in the IC Compiler II database to layer 10 in the stream file. The `read_gds/read_oasis` command converts all geometries on layer 10 in the stream file to layer 56 in the IC Compiler II database.
`data 56:4:power 10:2`	The `write_gds` or `write_oasis` command converts all geometric (nontext) power-related data on layer 56, purpose 4, in the IC Compiler II database to layer 10, data type 2 in the stream file. The `read_gds` or `read_oasis` command ignores the `data` keyword. It converts all data on layer 10, data type 2 in the stream file, to layer 56, purpose 4 in the IC Compiler II database.

The following layer mapping file shows some more of the available mapping features.

Example 2-1 Layer Mapping File Example

```
; Disable new layer creation
read_always false
; Map layer 10 with data type 2 in stream file to power net shapes on
; layer 56 with a purpose of 4 in the IC Compiler II database
data 56:4:power 10:2
; Map layers 31 and 32 with data type 0 in stream file to layer 31 with
; a purpose of 0 in the IC Compiler II database
data 31:0 31:0
data 31:0 32:0
```

For backward compatibility with the IC Compiler tool, you can use a layer mapping file written for the IC Compiler Milkyway database format. For details, see IC Compiler Layer Mapping File Syntax.

IC Compiler II Layer Mapping Syntax

Each line in the layer mapping file specifies a layer in the IC Compiler II technology file and the corresponding layer in the GDSII or OASIS stream file using the following syntax:

```
[object_type] tf_layer[:tf_purpose][:use_type][:mask_type]
      stream_layer[:stream_data_type]
```

Each line can contain the following items:

- *object_type* – The types of objects mapped by this rule when writing the stream file

  Valid values are `data` (all nontext objects), `text`, and `all`. The default is `all`.
- *tf_layer* – The layer number in the technology file, a required item
- *tf_purpose* – The purpose number in the technology file

  Valid values are either an integer or the `drawing` keyword.
- *use_type* – The IC Compiler II usage of the geometries in the design

  Valid values are `power`, `ground`, `signal`, `clock`, `boundary`, `hard_placement_blockage`, `soft_placement_blockage`, `routing_blockage`, `area_fill`, and `track`.
- *mask_type* – The multiple-patterning mask constraint of the geometries

  Valid values for metal layers are `mask_one`, `mask_two`, `mask_three`, and `mask_same`. Via layers support the additional values `MASK_FOUR` through `MASK_FIFTEEN`.
- *stream_layer* – The layer number in the stream file, a required item
- *stream_data_type* – The layer data type number in the stream file

Any text in a line that comes after a semicolon character (`;`) is considered a comment and has no effect on layer mapping.

Layer Mapping File Statements

The layer mapping file supports the following optional statements to control the behavior of the `write_gds`, `read_gds`, `write_oasis`, and `read_oasis` commands:

```
read_always true|false [-mapped_only] [-ignore_missing_layers]
blockage_as_zero_spacing true|false
cell_prop_attribute attribute_value
net_prop_attribute attribute_value
pin_prop_attribute attribute_value
```

These statements control the handling of extra or missing layer data, the interpretation of blockage layers, and the optional storage and retrieval of cell, net, and pin names in the GDSII or OASIS data file. Table 2-4 and Table 2-5 describe how these statements affect the behavior of the `write_gds`, `read_gds`, `write_oasis`, and `read_oasis` commands.

Table 2-4 Layer Mapping File Statements to Control the Reading of Layer Data

Statement	Description
`read_always true`	The `read_gds` or `read_oasis` command loads all layers defined in the external data file. For any layers not defined in the technology file, the command creates new layers. This is the default behavior.
`read_always false`	The `read_gds` or `read_oasis` command fails and issues an error message if the external data file contains layers not defined in the technology file.
`read_always false -mapped_only`	The `read_gds` or `read_oasis` command reads only those layers specified in the layer mapping file and ignores all other layers.
`read_always false -ignore_missing_layers`	The `read_gds` or `read_oasis` command reads only those layers defined in the technology file and ignores all other layers.
`blockage_as_zero_spacing false`	The `read_gds` or `read_oasis` command does not mark blockages read from the external data file as zero-spacing blockages, so the blockages follow normal spacing rules. This is the default behavior.
`blockage_as_zero_spacing true`	The `read_gds` or `read_oasis` command marks blockages read from the external data file as zero-spacing blockages.

Table 2-5 Layer Mapping File Statements to Maintain Cell, Net, and Pin Names

Statement	Description
`cell_prop_attribute` *`attribute_value`*	The `write_gds` or `write_oasis` command uses the IC Compiler II cell instance name to set the name associated with the specified cell attribute number in the stream file. Using this feature preserves the cell names when you read the stream data back into the IC Compiler II database in the library manager tool.
`net_prop_attribute` *`attribute_value`*	The `write_gds` or `write_oasis` command writes out each net name as a property attribute of the specified value for each geometric shape in the stream file. Using this feature preserves the net names associated with shapes in the design when you read the stream data back into the IC Compiler II database, reducing the need for connectivity tracing.
`pin_prop_attribute` *`attribute_value`*	The `write_gds` or `write_oasis` command writes out each pin name as a property attribute of the specified value for each pin in the stream file. It also writes out the pin name as text on the same layer and datatype as the pin geometry. Using this feature preserves the pin names when you read the stream data back into the IC Compiler II database, reducing the need for connectivity tracing.

Guidelines for Writing and Reading GDSII and OASIS

The following usage guidelines apply to the layer mapping file and the usage of the `write_gds`, `read_gds`, `write_oasis`, and `read_oasis` commands.

- If conflicting mapping rules are specified in the layer mapping file, the last instance in the file is used and the earlier ones are ignored.
- You can use the asterisk character (`*`) to represent all layer numbers, all purpose numbers, all usage types, or all mask types.
- You can use either the layer mapping file or the write/read stream command options to specify the property attribute numbers used for maintaining the cell, net, and pin names in the stream file. If you use both methods, the layer mapping file definitions have priority.

  For example, consider the following lines in a layer mapping file:

  ```
  ; layer mapping file object property options
  cell_prop_attribute 1 ; read/write cell names as property attribute 1
  net_prop_attribute 2  ; read/write net names as property attribute 2
  pin_prop_attribute 3  ; read/write pin names as property attribute 3
  ```

These lines override the corresponding options in the `write_gds`, `read_gds`, `write_oasis`, and `read_oasis` commands, such as the following:

```
icc2_shell> write_gds -layer_map my_layers.map \
 -instance_property 4 -net_property 5 -pin_property 6 my_design.gds
```

- By default, the `write_gds` or `write_oasis` command uses the cut data type numbers defined in the technology file. For example,

```
Layer "VIA2"
  cutTblSize      = 3
  cutNameTbl      = (V1SM, V1BAR, V1LRG)    # cut names
  cutWidthTbl     = (0.05, 0.05, 0.10)      # cut width
  cutHeightTbl    = (0.05, 0.10, 0.10)      # cut height
  cutDataTypeTbl  = (5, 10, 15)             # cut data types
  ...
```

  In this example, the `write_gds` or `write_oasis` command maps the V1SM, V1BAR, and V1LRG cuts to data types 5, 10, and 15, respectively. If you do not want this mapping, use the `-ignore_cut_datatype_tbl_mapping` option with the `write_gds` or `write_oasis` command.

- By default, the `write_gds` or `write_oasis` command converts all layer data types to 0 for layers not listed in the layer mapping file, or when no layer mapping file is used. To retain the layer data type numbers in these situations, use the `-keep_data_type` option with the `write_gds` or `write_oasis` command.

- By default, the `write_gds` or `write_oasis` command writes out text objects in the R0 orientation. To modify the orientation of pin text according to the route access direction, do the following:

```
icc2_shell> set_app_options \
  –name "file.gds.rotate_pin_text_by_access_direction" -value true
icc2_shell> set_app_options \
  –name "file.oasis.rotate_pin_text_by_access_direction" -value true
```

- By default, the `write_gds` or `write_oasis` command writes out contact via names using the prefix string “`$$`”. To specify a different prefix string, do the following:

```
icc2_shell> set_app_options \
  –name "file.gds.contact_prefix" -value "MY_PREFIX"
icc2_shell> set_app_options \
  –name "file.oasis.contact_prefix" -value "MY_PREFIX"
```

- Different fill cells with the same name can exist in different subblocks. By default, the `write_gds` and `write_oasis` commands append the name of the top-level block to fill cell names to avoid a name conflict.

  To prevent the `write_gds` command from appending the top-level block name to a fill cell, set the `file.gds.prefix_for_fill` application option to `false`:

```
icc2_shell> set_app_option -name file.gds.prefix_for_fill -value false
```

- You can save a layer mapping file in a design library by using the `set_layer_map_file` command. In that case, the saved file is used as the default mapping file when you perform layout validation with the IC Validator tool or parasitic extraction with the StarRC tool. For details, see the man page for the command.

For information about other options for writing or reading stream files and using the layer mapping file, see the applicable man page.

IC Compiler Layer Mapping File Syntax

For backward compatibility with the IC Compiler tool, you can use a layer mapping file written for the IC Compiler Milkyway database format. For example, for the `write_gds` command, specify the layer mapping file format in one of the following ways:

```
icc2_shell> write_gds -layer_map map_file_name \
 -layer_map_format icc_default ...

icc2_shell> write_gds -layer_map map_file_name \
  -layer_map_format icc_extended ...
```

The `icc_default` setting uses a layer mapping file written for a Milkyway library in default layer mode, which uses layer numbers 1 through 255. The `icc_extended` setting uses a layer mapping file written for a Milkyway library in extended layer mode, which uses layer numbers 1 through 4095.

Each line in the layer mapping file shows a Milkyway object type, Milkyway layer, and resulting layer in the output stream. This is the general syntax:

```
MilkywayObjType[MilkywayNetType][PinCode]
     MilkywayLayer[:MilkywayDataType] StreamLayer[:StreamDataType]
```

The first character in the line is the code for the type of Milkyway object to be translated. Use A for all, T for text, or D for data. The optional second character specifies the net type, such as S for signal, P for power, or G for ground. An optional third character specifies the pin code. The Milkyway layer is a name or an integer. The stream layers and data types are integers.

By default, net text and pin text are mapped into the same layer as the associated net or pin. However, you can specify a different layer for net text or pin text in the layer mapping file.

The following example demonstrates the stream-out layer mapping syntax.

```
T METAL:2 3:5   ; converts text on Milkyway layer
                ; METAL data Type 2 to GDSII Stream
                ; layer #3 data type 5
                ; Note: If you stream back into
                ; Milkyway database without using
                ; the layer map file, this will
```

```
                ; switch your text on layer METAL
                ; to layer METAL5

TS 16 31:6      ; converts text associated with
                ; signal nets on Milkyway layer #16
                ; to GDSII Stream layer #31 and
                ; data type 6

TP METAL3 16    ; converts text associated with
                ; power on Milkyway layer METAL3
                ; to GDSII Stream layer #16

TG 28 16        ; converts text associated with
                ; ground on Milkyway layer #28
                ; to GDSII Stream layer #16

D METAL2 45     ; converts data on Milkyway layer
                ; METAL2 to GDSII Stream layer #45

DS 45:2 18:5    ; converts data associated with
                ; signal nets on Milkyway layer #45
                ; data Type 2 to GDSII Stream layer #18
                ; data Type 5

A METAL6 3:4    ; converts text and data on
                ; Milkyway Layer METAL6 to GDSII
                ; Stream layer #3 and datatype 4

A METAL4 -      ; minus sign (hyphen) prevents transfer of all
                ; text and data on Milkyway layer METAL4
```

You can add a comment to the file by inserting a semicolon. Text is ignored from the semicolon to the end of the line. A hyphen character in the stream layer position prevents the transfer of all text and data in the specified Milkyway object and layer.

A colon is the delimiter between the stream layer and the optional stream data type. For backward compatibility, you can use a space character alone or a colon followed by white space as the delimiter.

Table 2-6 describes the items used in each line of the layer mapping file.

Table 2-6 IC Compiler Layer Mapping File Syntax

Variable	Description
MilkywayObjType	The code for the type of object in the data to be translated: A for all types T for text D for data

Table 2-6 IC Compiler Layer Mapping File Syntax (Continued)

Variable	Description
`MilkywayNetType`	The code for the net type of the object in the data to be translated: S for signal P for power G for ground C for clock U for Up conduction layer (upper layer in a via). Example: metal2 in an m1/m2 via D for Down conduction layer (lower layer in a via). Example: metal1 in an m1/m2 via X for power and ground wires or contacts with a “signal” or “tie-off” routing type A for all net types Note that U and D are used only for via and flattened via objects. They override any other net type set for the layer when the layer is in a via. If `MilkywayNetType` is omitted, the tool maps all Milkyway data of the specified object type to the specified stream layer. If `MilkywayNetType` is included, the tool examines every object of the object type to determine its net type and maps it to the layer you specify. If an object has no net or a net has no type, the tool assumes that it is a signal net. If a layer file contains contradictory lines, the last line overrides any previous line. For example, if an earlier line of a layer file specifies mapping for a particular object or net type and a later line specifies mapping for the same object type but no net type, the tool translates all data of the specified object type as defined by the later line.
`PinCode`	Optional. Can be used when only the Pin geometry or Terminal (top-level pin) geometry in the Milkyway database needs to be translated: T for terminal geometries of the current design P for terminal geometries of the current design, as well as the physical pin geometries of the child cells, depending on the child depth option. Note: PinCode P and T only work when *MilkywayNetType* is set to A. Otherwise, the PinCode rule is ignored.
`MilkywayLayer`	Specifies the name or number, from 1 to 255 (1 to 4095 in extended layer mode), of a layer to be translated and defined in the technology file.

Table 2-6 IC Compiler Layer Mapping File Syntax (Continued)

Variable	Description
MilkywayDataType	The data type of the Milkyway data to be translated: an integer from 0 to 4095.
StreamLayer	Specifies the number of the stream layer to which the objects of `MilkywayObjType` on `MilkywayLayer` are translated. Use an integer from 0 to 32767 or use a hyphen character (-) to ignore the Milkyway object and layer during translation.
StreamDataType	Specifies the number of the stream data type to which objects of `MilkywayObjType` and `MilkywayNetType`, if given, on `MilkywayLayer` are translated. Use an integer from 0 to 32767.

The `write_gds` or `write_oasis` command writes geometries into reserved layers by mapping the tool layer numbers according to the following table.

Table 2-7 Milkyway Layer Numbers and Blockage Usage

Milkyway layer number (default)	Milkyway layer number (extended)	Blockage layer
212	4025	polyBlockage
216	4029	metal4Blockage
217	4030	via3Blockage
218	4031	metal1Blockage
219	4032	metal2Blockage
220	4033	metal3Blockage
223	4036	polyContBlockage
224	4037	via1Blockage
225	4038	via2Blockage

Changing Object Names

The IC Compiler II tool follows certain naming conventions for ports, cells, and nets. When you export a design to a new format such as Verilog, DEF, LEF, or GDSII, you might need to have the object names follow naming conventions of the external tool that are different from IC Compiler II naming conventions.

Before you export a design, you can use the `change_names` command to change the names of cells, nets, and ports so that the new names follow the required naming conventions. The IC Compiler II tool has a built-in capability to convert object names to conform to Verilog requirements. You can also create your own naming conversion rules.

See Also

- Changing Object Names Using Custom Rules
- Changing Object Names for Verilog Output

Changing Object Names Using Custom Rules

Using the `define_name_rules` and `change_names` commands, you can change the names of ports, cells, and nets in the design to conform to specified naming conventions. The name-changing process can perform the following types of changes:

- Force the name of any net connected to a port to match the port name
- Force all port, cell, and net names to be unique across all objects
- Change all occurrences of a given string in object names to a new string
- Restrict the first character, last character, or all characters of object names to a given list of characters
- Prohibit the usage of a given list of characters as the first character, last character, or all characters of object names
- Prohibit the usage of a given list of words as object names
- Restrict the number of characters in each name to a specified maximum limit

The following diagram shows the typical flow for changing the names of objects in the design.

Figure 2-1 Typical Flow for Changing Object Names

`define_name_rules` *`rule_name`* `...`
Define the desired name conversion rule

↓

`report_name_rules` *`rule_name`*
Report name of newly defined conversion rule (optional)

↓

`report_names -rules` *`rule_name`*
Report the names in the design that will be changed (optional)

↓

`change_names -rules` *`rule_name`* `-hierarchy`
Change the names in the design according to the rule

↓

`report_ports, report_cells, report_nets, write_*`
Report the new names; write out the design using new names

The following example changes each occurrence of the string “address” or “data” to “ADR” or “D”, respectively, in all port names.

1. Define the name conversion rule.

```
icc2_shell> define_name_rules UCports -type port \
  -map {{"address","ADR"},{"data","D"}}
1
```

   This defines a rule called “UCports” (uppercase ports), which maps the string “address” to “ADR” and the string “data” to “D” in port names.

2. (Optional) Report the rule just defined and verify its behavior.

```
icc2_shell> report_name_rules UCports
...
Rules Name: UCports
    ...
```

```
    Object Max  Repl Rem
    Type   Len  Char Chrs Prefix Allowed Chars
    ------------------------------------------------------------
    Port   none '_'  no   P      Mapping String: "{"address","ADR"},
                                                  {"data","D"}"
    Cell   none '_'  no   U
    Net    none '_'  no   N
1
```

3. (Optional) Report the names that would be changed by applying the rule, without actually making the changes, and verify that the proposed changes are correct.

```
icc2_shell> report_names -rules UCports
...
        Design          Type     Object           New Name
        ------------------------------------------------------------
        my_design_mp    port     address[27]      ADR[27]
        my_design_mp    port     address[26]      ADR[26]
        my_design_mp    port     address[25]      ADR[25]
        ...
        my_design_mp    port     data[31]         D[31]
        my_design_mp    port     data[30]         D[30]
        my_design_mp    port     data[29]         D[29]
        ...
```

4. Change the names in the design according to the rule.

```
icc2_shell> change_names -rules UCports -hierarchy
 Information: Using name rules 'UCports'.
 Information: 62 objects (60 ports, 2 bus ports, 0 cells & 0 nets)
  changed in design 'my_design_mp'.
```

To apply the name changes across multiple levels of the physical hierarchy, use the `-include_sub_blocks` option with the `-hierarchy` option:

```
icc2_shell> change_names -rules UCports -hierarchy -include_sub_blocks
...
```

The tool determines the physical hierarchy by using the bound view, which must be the design view. If the bound view is not a design view, the `change_names` command issues an error message. When the `change_names` command modifies names in a physical subblock, it updates the bound view; you must re-create any other views of the physical subblock, such as the frame and abstract views.

5. (Optional) Report the object names and verify that they have been changed correctly.

```
icc2_shell> report_ports
...
                      Pin Cap                     Wire Cap
                   Min          Max          Min          Max
Port      Dir    rise/fall    rise/fall     rise/fall    rise/fall
--------------------------------------------------------------------
ADR[0]    out    0.00/0.00    0.00/0.00     0.00/0.00    0.00/0.00
```

```
ADR[10]    out    0.00/0.00    0.00/0.00    0.00/0.00    0.00/0.00
ADR[11]    out    0.00/0.00    0.00/0.00    0.00/0.00    0.00/0.00
...
D[0]       inout  0.00/0.00    0.00/0.00    0.00/0.00    0.00/0.00
D[10]      inout  0.00/0.00    0.00/0.00    0.00/0.00    0.00/0.00
D[11]      inout  0.00/0.00    0.00/0.00    0.00/0.00    0.00/0.00
...
```

When you save or export the design, the design data reflects the new names.

To report the names of ports, cells, and nets across all levels of the physical hierarchy, use the `-include_sub_blocks` option when you run the `report_names -hierarchy` command.

For information about the various rules you can use to change port, cell, and net names, see the man page for the `define_name_rules` command.

See Also

- Changing Object Names for Verilog Output

Changing Object Names for Verilog Output

Before you export a design in Verilog format by using the `write_verilog` command, use the `change_names` command to change all the object names to conform to Verilog conventions:

```
icc2_shell> current_block
{my_lib:my_design/done.design}

icc2_shell> change_names -rules verilog -hierarchy
 Information: Using name rules 'verilog'.
 Information: 321 objects (0 ports, 0 bus ports, 0 cells & 321 nets)
   changed in design 'my_design'.
1
icc2_shell> write_verilog my_design.v
1
```

To see the details of the Verilog naming rules, use the `report_name_rules` command:

```
icc2_shell> report_name_rules verilog
...
Rules Name: verilog
    Collapse name space: true
    Equal port and net names: true
    Equal inout port and net names: true
    Check internal net name: true
    Target bus naming style: %s[%d]
    Remove irregular port bus: true
    Check bus indexing: true
```

```
    Check bus indexing use type info: false
    Remove port bus: false
    Case insensitive: false
    Add dummy nets: false
    Reserved words: not defined
...
```

For a full explanation of each type of name change, see the man page for the `define_name_rules` command.

See Also

- Changing Object Names Using Custom Rules

3

Working With Design Data

When you work with a design in the IC Compiler II tool, you open and edit the design view of a block stored in a design library.

You can query and edit the design data as described in the following topics:

- Application Options
- Working With Objects
- Working With Collections
- Bus and Name Expansion
- Design Hierarchy
- Technology Data Access
- Reporting Design Information
- Polygon Manipulation
- Undoing and Redoing Changes to the Design
- Schema Versions

Application Options

You set the tool's *application options* by using the `set_app_options` command. These options control various aspects of tool behavior. For example, to set the maximum allowed coarse placement density for the current block:

```
icc2_shell> set_app_options -name place.coarse.max_density -value 0.6
place.coarse.max_density 0.6
```

Application options use the following naming convention:

category[.*subcategory*].*option_name*

where *category* (such as `place`) is the tool feature affected by the option and *subcategory* (such as `coarse`) is a refinement of the feature affected by the option.

To get a list of all available application options, use the `get_app_options` command:

```
icc2_shell> get_app_options
shell.common.tmp_dir_path shell.common.enable_line_editing
shell.common.line_editing_mode shell.common.collection_result_...
```

To restrict the list of reported application options, provide a pattern string:

```
icc2_shell> get_app_options place.*
place.coarse.channel_detect_mode place.coarse.congestion_analysis_effort
...
icc2_shell> get_app_options *lib*
design.edit_read_only_libs link.user_units_from_first_library ...
...
```

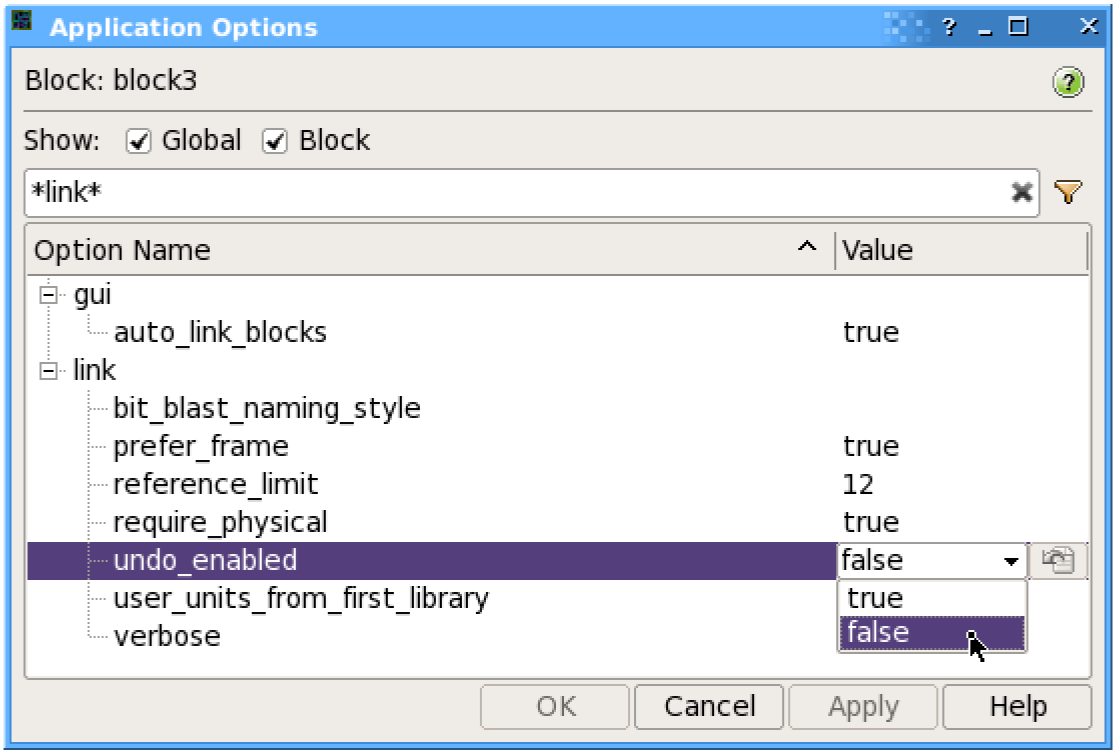
Each application option applies either globally or to a specific block:

- Global application options apply everywhere within the current session.
- Block-level application options apply only to the block on which they are set. Before you can set a block's application options, the block must be open.

You can determine whether an application option is global or block-level by using the `report_app_options` command.

To use the GUI to view, search, and edit application options, choose File > Application Options. This opens the Application Options dialog box, shown in the following figure.

Figure 3-1 Application Options Dialog Box



In the Application Options dialog box, to view the man page for the highlighted application option, click the question mark icon in the upper-right corner.

For more information about application options, see

- Setting Application Options
- Querying Application Options
- Saving Application Options
- Help on Application Options
- Man Pages for Application Options
- User Default for Application Options
- Resetting Application Options

Setting Application Options

To set an application option, use the `set_app_options` command:

```
icc2_shell> set_app_options -name link.verbose -value true
link.verbose true
```

To set a block-level application option on a block other than the current block:

```
icc2_shell> set_app_options -block des_A -name link.verbose -value true
link.verbose true
```

To set multiple application options in a single command:

```
icc2_shell> set_app_options -list \
  {link.verbose true link.reference_limit 8}
link.reference_limit 8 link.verbose true
```

Alternatively, if the options belong to the same category, you can do this:

```
icc2_shell> set_app_options -category link \
 -list {verbose true reference_limit 8}
link.reference_limit 8 link.verbose true
```

See Also

- Application Options
- Querying Application Options
- Saving Application Options
- User Default for Application Options
- Resetting Application Options

Querying Application Options

To get an application option setting, use the `get_app_option_value` command:

```
icc2_shell> get_app_option_value -name link.verbose
true
```

To get a block-level application option setting for a block other than the current block, use the `-block` option.

To get the default setting for an application option, use the `-system_default` option:

```
icc2_shell> get_app_option_value -name link.verbose -system_default
false
```

To get all the attributes of an application option, use the `-details` option:

```
icc2_shell> get_app_option_value -name link.verbose -details
name link.verbose value_type bool default_value false help {Verbose
messages while linking} status basic minimum_value {} maximum_value {}
allowed_values {} is_obsolete false is_global_only false is_read_only
false is_persistent true is_subscripted false is_design_cell_scoped true
is_lib_cell_scoped false
```

To report multiple application option settings, only the global options, or only the user-set options, use the `report_app_options` command:

```
icc2_shell> report_app_options link*
...
Name                        Type          Value  ... default    Scope
--------------------------- ------------- ------ ----------- --------
link.bit_blast_naming_style list_of string {}     ...   ...     block
link.prefer_frame           bool          --     ...   ...     global
link.undo_enabled           bool          --     ...   ...     global
...

icc2_shell> report_app_options link* -global
...
Name                        Type          Value  ...  System-default
--------------------------- ------------- ------ -------------------
link.prefer_frame           bool          --     ...  true
link.undo_enabled           bool          --     ...  true
...

icc2_shell> report_app_options -non_default     # Report user-set options
...
Name                    Type   Value    ...  System-default  Scope
----------------------- ------ -------  ---  --------------- -------
place.coarse.max_density float  0.70     ...  0               block
----------------------- ------ -------  ---  --------------- -------
1

icc2_shell> report_app_options                  # Report all options
...
```

See Also

- Application Options
- Setting Application Options
- Saving Application Options
- Help on Application Options
- Man Pages for Application Options
- User Default for Application Options

- Resetting Application Options

Saving Application Options

Each application option applies either globally or to a specific block. To save application option settings, use the following guidelines:

- Global application options apply everywhere within the current session. They are not stored and are not carried over from one session to the next. To retain these option settings, you can set them in your .synopsys_icc2.setup file.
- Block-level application options apply only to the block on which they are set. They are saved with the block in the design library and are persistent across tool sessions except when they are set by the `set_app_options -as_user_default` command. In this case, they are treated like global application options and are not persistent across tool sessions.
- To determine whether an application option was set using the `-as_user_default` option, run the `report_app_options` command and check the value in the user-default column.
- Use the `write_script` command to save the current application option settings in a script file.

See Also

- Application Options
- Setting Application Options
- Querying Application Options
- Help on Application Options
- Man Pages for Application Options
- User Default for Application Options
- Resetting Application Options

Help on Application Options

You can use the `help_app_options` command to get help on using application options:

```
icc2_shell> help_app_options
Categories of the registered application options are:
  abstract
  clock_opt.congestion
  clock_opt.flow
  clock_opt.hold
  ...
  link
  ...

icc2_shell> help_app_options -category link
link.bit_blast_naming_style        # naming style for bit blasted ...
link.prefer_frame                  # Prefer blocks with FRAME view ...
link.reference_limit               # Maximum number of unresolved ...
link.require_physical              # Require sub-blocks to have ...
link.undo_enabled                  # Allow link_design command to be ...
link.user_units_from_first_library # Set user units to the units of ...
link.verbose                       # Verbose messages while linking

icc2_shell> help_app_options -category link -scope block
link.bit_blast_naming_style        # naming style for bit blasted ...
link.reference_limit               # Maximum number of unresolved ...
link.verbose                       # Verbose messages while linking

icc2_shell> help_app_options link.undo_enabled
link.undo_enabled              # Allow link_block command to be undoable

icc2_shell> help_app_options link.undo_enabled -verbose
name                           link.undo_enabled
value_type                     bool
default_value                  false
help                           Allow link_block command to be undoable
...                            ...
is_read_only                   false
is_persistent                  false
is_subscripted                 false
is_design_cell_scoped          false
is_lib_cell_scoped             false
```

See Also

- Application Options
- Setting Application Options
- Querying Application Options
- Saving Application Options

- Man Pages for Application Options

Man Pages for Application Options

To view the man page for a specific application option, use the `man` command:

```
icc2_shell> man link.reference_limit
...

NAME
       link.reference_limit
              Specifies the maximum number of detailed unresolved
              reference errors display for each block when linking ...
TYPE
        integer

DEFAULT
        12

DESCRIPTION
        In many cases, many link errors have the same root cause ...
        ...
SEE ALSO
        link_block(2)
        get_app_option_value(2)
        ...
```

To view the man page for all application options under a top-level category, use the `man` command to view the man page for *category*`_options`:

```
icc2_shell> man link_options
   ...
DESCRIPTION
       ...
link.bit_blast_naming_style
    ...
link.prefer_frame
    ...
link.reference_limit
    ...
...
```

See Also

- Application Options
- Setting Application Options
- Querying Application Options

- Saving Application Options
- Help on Application Options

User Default for Application Options

You can optionally set a *user default* for an application option. The user default has higher priority than the *system default* but lower priority than an explicit setting for the option. The user default applies only to the current session and is not saved.

For example, the system default for the `place.coarse.max_density` application option is 0.0, so that value applies to all blocks by default. To set the user default to a different value, use the `set_app_options` command with the `-as_user_default` option:

```
icc2_shell> set_app_options -as_user_default \
  -name place.coarse.max_density -value 0.65
place.coarse.max_density 0.65

icc2_shell> report_app_options place.coarse.max_density
...
Name                     Type   Value  User-default  System-default  ...
------------------------ ------ ------ ------------- --------------- ---
place.coarse.max_density float  --     0.65          0               ...
------------------------ ------ ------ ------------- --------------- ---
```

The new default, 0.65, applies to all blocks in the current session that do not have an explicit value set.

You can override the user default for a particular block by setting the `place.coarse.max_density` application option to the desired value for the block:

```
icc2_shell> current_block
{lib_A:block3.design}
icc2_shell> set_app_options -name place.coarse.max_density -value 0.70
place.coarse.max_density 0.7

icc2_shell> report_app_options place.coarse.max_density
...
Name                     Type   Value  User-default  System-default  ...
------------------------ ------ ------ ------------- --------------- ---
place.coarse.max_density float  0.70   0.65          0               ...
------------------------ ------ ------ ------------- --------------- ---
```

The order of priority is “Value” first, then “User-default,” and “System-default” last.

See Also

- Application Options
- Setting Application Options

- Resetting Application Options

Resetting Application Options

To reset an application option so that it has no explicitly assigned value, use the `reset_app_options` command:

```
icc2_shell> reset_app_options place.coarse.max_density
1
icc2_shell> report_app_options place.coarse.max_density
...
Name                     Type   Value  User-default  System-default  ...
------------------------ ------ ------ ------------- --------------- ---
place.coarse.max_density float  --     0.65          0               ...
------------------------ ------ ------ ------------- --------------- ---
```

The `reset_app_options` command removes the “Value” setting and allows the “User-default” setting to apply (if any), or the “System-default” to apply if there is no user default.

To reset a user default setting, use the `reset_app_options` command with the `-user_default` option:

```
icc2_shell> reset_app_options place.coarse.max_density -user_default
1
icc2_shell> report_app_options place.coarse.max_density
...
Name                     Type   Value  User-default  System-default  ...
------------------------ ------ ------ ------------- --------------- ---
place.coarse.max_density float  --     --            0               ...
------------------------ ------ ------ ------------- --------------- ---
```

See Also

- Application Options
- Setting Application Options
- User Default for Application Options

Working With Objects

The IC Compiler II infrastructure supports a full set of design-related objects and Tcl commands to create, query, modify, and remove these objects, including the following types:

- Logical objects such as cells, ports, and nets
- Timing data objects such as modes and corners
- Physical objects such as shapes, vias, tracks, and blockages
- Cell library objects such as libraries, library cells, and library pins

To get the full list of object types, use the following command:

```
icc2_shell> get_defined_attributes -return_classes
block bound bound_shape budget_clock budget_path_type ...
bundle_segment bundle cell clock clock_balance_group clock_group ...
track utilization_config via via_def via_region via_rule ...
...
```

The tool supports commands to create, get, and remove specific types of objects, such as:

```
create_block    get_blocks     remove_blocks
create_net      get_nets       remove_nets
create_tech     get_techs      remove_tech
```

You can also operate on objects of mixed types in a collection by using commands such as the following:

- `move_objects` – move objects by a specified amount or to a specified location
- `flip_objects` – flip objects around a specified axis
- `rotate_objects` – rotate objects by a specified angle or to a specified orientation
- `remove_objects` – remove objects from the design

Some types of objects have special-purpose commands that apply only to specific object types, such as `report_lib`, `add_to_bound`, `set_lib_cell_purpose`, and `connect_net`.

You can perform tasks such as resizing a cell. For example, to return a collection of equivalent library cells for a specific cell or library cell, use the `get_alternative_lib_cells` command. You can then use the collection to replace or resize the cell. The `size_cell` command allows you to change the drive strength of a leaf cell by linking it to a new library cell that has the required properties.

For a design with multiply instantiated modules, edit the module by using the `edit_module` command, as shown:

```
icc2_shell> edit_module [get_module add_block] { \
   set_reference [get_cells U25] -to_block AND2 -pin_rebind none \
   set_reference [get_cells U61] -to_block XOR2 -pin_rebind none}
```

See Also

- Querying Common Design Objects
- Object Attributes
- Querying Objects
- Removing Objects
- Working With Collections

Querying Common Design Objects

Table 3-1 lists some common design objects and the commands to query them. These objects are provided as examples; there are many more objects not shown in the table. For a complete list of object types, use the `get_defined_attributes -return_classes` command.

Table 3-1 Commands to Query Some Common Design Objects

Object	Command	Description
Blockages	`get_pin_blockages`	Returns a collection of pin blockages.
	`report_pin_blockages`	Displays information about the pin blockages.
	`get_placement_blockages`	Returns a collection of placement blockages
	`get_routing_blockages`	Returns a collection of routing blockages
Bounds	`get_bounds`	Returns a collection of placement bounds.
	`get_bound_shapes`	Returns a collection of shapes associated with placement bounds.
	`report_bounds`	Displays information about the placement bounds.

Table 3-1 Commands to Query Some Common Design Objects (Continued)

Object	Command	Description
Cells	`get_cells`	Returns a collection of cell instances. By default, the tool uses the logic hierarchy to resolve cell names. To use the physical hierarchy to resolve cell names, use the `-physical_context` option.
	`report_cells`	Displays information about the cell instances.
Clocks	`get_clocks`	Returns a collection of clocks
	`get_clock_groups`	Returns a collection of clock groups.
	`get_generated_clocks`	Returns a collection of generated clocks.
	`report_clocks`	Reports information about clocks.
Core area	`get_core_area`	Returns a collection that contains the core area.
Guides	`get_io_guides`	Returns a collection of I/O guides.
	`report_io_guides`	Displays information about the I/O guides.
	`get_pin_guides`	Returns a collection of pin guides.
	`report_pin_guides`	Displays information about the pin guides.
I/O rings	`get_io_rings`	Returns a collection of I/O rings.
	`report_io_rings`	Displays information about I/O rings.
Library cell	`get_lib_cells`	Creates a collection of library cells from the reference libraries loaded in memory.
	`report_lib_cells`	Reports information about library cells.
	`get_alternative_lib_cells`	Returns a collection of equivalent library cells for the specified cell or library cell.
Modules	`get_modules`	Returns a collection of logic hierarchies in memory, as defined in the Verilog netlist.
Nets	`get_nets`	Returns a collection of logical nets. To return a collection of physical nets, use the `-physical_context` option.
	`get_shapes`	Returns a collection of shapes associated with nets.
	`report_nets`	Displays information about the nets.

Table 3-1 Commands to Query Some Common Design Objects (Continued)

Object	Command	Description
Net buses	`get_net_buses`	Returns a collection of net buses from the current block.
	`report_net_buses`	Displays net bus information within the current block.
Pins	`get_pins`	Returns a collection of logical pins. To return a collection of physical pins, use the `-physical_context` option.
Ports	`get_ports`	Returns a collection of logical ports. To return a collection of physical ports, use the `-physical_context` option.
	`report_ports`	Displays information about the ports.
Port buses	`get_port_buses`	Returns a collection of port buses from the current block.
	`report_port_buses`	Displays port bus information within the current block.
Power domains	`get_power_domains`	Returns a collection of power domains.
	`get_domain_elements`	Returns a collection of elements associated with power domains.
	`report_power_domains`	Displays information about the power domains.
Power regions	`get_pg_regions`	Returns a collection of power and ground (PG) regions.
	`report_pg_regions`	Displays information about the PG regions.
Site rows	`get_site_rows`	Returns a collection of site rows.
Supply nets	`get_supply_nets`	Returns a collection of supply nets.
	`get_related_supply_nets`	Returns a collection of related supply nets.
	`report_supply_nets`	Displays information about the supply nets.
Supply ports	`get_supply_ports`	Returns a collection of supply ports.
	`report_supply_ports`	Displays information about the supply ports.

Table 3-1 Commands to Query Some Common Design Objects (Continued)

Object	Command	Description
Terminals	`get_terminals`	Returns a collection of terminals.
Tracks	`get_tracks`	Returns a collection of tracks.
	`report_tracks`	Reports the routing tracks for a specified layer or for all layers.
Vias	`get_vias`	Returns a collection of vias.
Voltage areas	`get_voltage_areas`	Returns a collection of voltage areas.
	`get_voltage_area_shapes`	Returns a collection of shapes associated with voltage areas.
	`report_voltage_areas`	Displays information about the voltage areas.

Editing Common Design Objects

Table 3-1 lists some common design objects and the commands to edit them. These objects are provided as examples; there are many more objects not shown in the table. For a complete list of object types, use the `get_defined_attributes -return_classes` command.

Table 3-2 Commands to Work With Some Common Design Objects

Object	Command	Description
Blockages	`create_pin_blockage`	Creates a pin blockage in the current design.
	`remove_pin_blockages`	Removes pin blockages from the current design.
Cells	`create_cell`	Creates a new cell.
	`remove_cells`	Removes specific cell instances.
	`size_cell`	Rebinds leaf cells to a new library cell.
	`change_link`	Changes the reference of an existing cell.

Table 3-2 Commands to Work With Some Common Design Objects (Continued)

Object	Command	Description
Clocks	`create_clock`	Creates a clock object.
	`remove_clocks`	Removes one or more clocks from the current design.
	`set_clock_groups`	Specifies clock groups that are mutually exclusive or asynchronous with each other.
	`remove_clock_groups`	Removes specific exclusive or asynchronous clock groups from the current design.
Nets	`create_net`	Creates a new net.
	`connect_net`	Connects a net to a pin or port.
	`disconnect_net`	Disconnects a net from a pin or port.
	`remove_nets`	Removes an existing net.
Ports	`create_port`	Creates a new port.
	`remove_ports`	Removes ports from the current design.

Object Attributes

Each object type has a list of predefined (also known as application-defined) attributes. You can query these attributes to get information about the design. In some cases, you can modify the attribute settings to control the design implementation.

To list the attributes associated with an object type, use the `list_attributes` command. Specify the `-application` option to report the predefined object attributes. By default, the `list_attributes` command reports only user-defined attributes. The following example lists the predefined attributes for nets:

```
icc2_shell> list_attributes -application -class net
...
Properties:
    A - Application-defined
    U - User-defined
    I - Importable from design/library (for user-defined)
    S - Settable
    B - Subscripted

Attribute Name            Object     Type       Properties  Constraints
-----------------------------------------------------------------------
antenna_rule_name         net        string     A,S
base_name                 net        string     A
bbox                      net        rect       A
dont_touch                net        boolean    A,S
```

```
full_name                  net        string     A
has_topology               net        boolean    A
...
```

You can also report the attributes associated with an object type by using the `get_defined_attributes` command:

```
icc2_shell> get_defined_attributes -application -class net
antenna_rule_name base_name bbox dont_touch full_name is_bbt_object
is_global is_ideal is_in_bundle is_physical is_rdl is_shadow
is_tie_high_net is_tie_low_net max_layer max_layer_mode ...
```

For descriptions of the attributes, see the *object_name*`_attributes` man page for each object type.

Settable Attributes

You can set a “settable” attribute for an object by using the `set_attribute` command. For example,

```
icc2_shell> set_attribute -objects [get_nets net16] -name dont_touch \
  -value true
{{net16}}
icc2_shell> get_attribute -objects [get_nets net16] -name dont_touch
true
```

An attribute that is not “settable” is read-only; you cannot change it directly by using the `set_attribute` command. For example, the `bbox` (bounding box) attribute of a net is not settable. However, when you edit or optimize a net, the tool automatically updates the `bbox` attribute to reflect the net’s bounding box.

Subscripted Attributes

Subscripted attributes are attributes that have multiple values, each of which is associated with a specific key of the attribute. The key represents the available modes or subtypes of the attribute. A subscripted attribute has the following syntax:

attribute(*key*)

For example, the `current_repair` attribute for a mismatch configuration is a subscripted attribute whose key is the name of a mismatch configuration. It can be one of the two predefined mismatch configurations, `auto_fix` or `default`, or the name of a user-defined mismatch configuration.

The following example queries the `current_repair` attribute for the `auto_fix` mismatch configuration:

```
icc2_shell> get_attribute [get_mismatch_types lib_missing_logical_port] \
     current_repair(auto_fix)
create_placeholder_logic_lib_port
```

Cascaded Attributes

For collection-type attributes that point to other design objects, you can query the attributes of that object directly from the current object by using *cascaded attributes*. A cascaded attribute has the following syntax, where the child attribute is an attribute of the single collection object returned by the parent attribute:

```
parent_attribute.child_attribute
```

In other words,

```
get_attribute [get_attribute $object parent_attribute] child_attribute
```

can be simplified to

```
get_attribute $object parent_attribute.child_attribute
```

You can cascade any number of attributes, such as the three in the following example:

```
icc2_shell> get_attribute [get_vias VIA_S1] owner.top_block.top_module
{ORCA}
```

For more information about using cascaded attributes, see Querying Cascaded Attributes.

Reporting Attributes

To query the attributes of an object in the design, use the `report_attributes` command:

```
icc2_shell> report_attributes -application -class net [get_nets clkm]
...
Design         Object      Type        Attribute Name       Value
--------------------------------------------------------------------
my_design_mp   clkm        string      antenna_rule_name
my_design_mp   clkm        string      base_name            clkm
my_design_mp   clkm        rect        bbox                 {357.580 ...
my_design_mp   clkm        boolean     dont_touch           false
my_design_mp   clkm        string      full_name            clkm
my_design_mp   clkm        string      net_type             signal
my_design_mp   clkm        int         number_of_pins       5
...
```

To query just a single attribute of an object, use the `get_attribute` command:

```
icc2_shell> get_attribute -objects [get_nets clkm] -name net_type
signal
icc2_shell> get_attribute -objects [get_nets clkm] -name number_of_pins
5
```

The `-objects` argument can be either a collection or a string. For example,

```
icc2_shell> get_attribute -objects [get_nets n123] -name max_layer
M5

icc2_shell> get_attribute -objects n123 -name max_layer
```

```
M5
```

When you specify the object as a string, the command searches for object types in a specific order. For details, see the man page for the `shell.common.implicit_find_mode` application option. To restrict a search to a specific object type, use the `-class` option. For example,

```
icc2_shell> get_attribute -class net -objects a12 -name dont_touch
false
```

Querying Objects

You can get information about objects in the design several different ways. In the GUI display, when you hover over an object, a ToolTip displays basic information about that object. For more information about that object, right-click it and choose Properties.

At the shell prompt, you can use the `report_*` commands (for example, `report_nets`) to report some types of objects, or the `get_*` commands (for example, `get_nets`) to create a collection of objects. You can use a collection as input to another command, or set a variable to a collection for future use:

```
icc2_shell> get_nets test*
{test_mode test_se test_si test_si_1 ...}

icc2_shell> set my_test_nets [get_nets test*]
{test_mode test_se test_si test_si_1 ...}

icc2_shell> report_nets [get_nets test*]
...
Net                 Fanout   Fanin   Pins
----------------------------------------------
test_mode                1       1      2
test_se                  1       1      2
test_s1                  1       1      2
test_s_1                 1       1      2
...

icc2_shell> report_nets $my_test_nets
... (generates the same report) ...
```

For more information about collections, see the `collections` man page.

For more information about using the `get_*` commands, see

- Query by Location
- Query by Association
- Query in Physical Context
- Querying Cascaded Attributes

Query by Location

To search for objects at a specific location or within a specified rectangular region, use one of the following methods:

- Use the `-at`, `-intersect`, `-touching`, or `-within` options with the `get_*` command.

  Use this method when you are looking for objects of a single type. Figure 3-2 shows the behavior of these options. The behavior is the same as in the IC Compiler tool.

- Use the `get_objects_by_location` command.

  Use this method when you are looking for objects of more than one type. Figure 3-3 shows the default behavior of the `-at`, `-intersect`, `-touching`, and `-within` options with the `get_objects_by_location` command. To change the behavior of these options to match the IC Compiler tool behavior and the `get_*` behavior, set the `design.enable_icc_region_query` application option to `true`.

Figure 3-2 get_* by Location Behavior

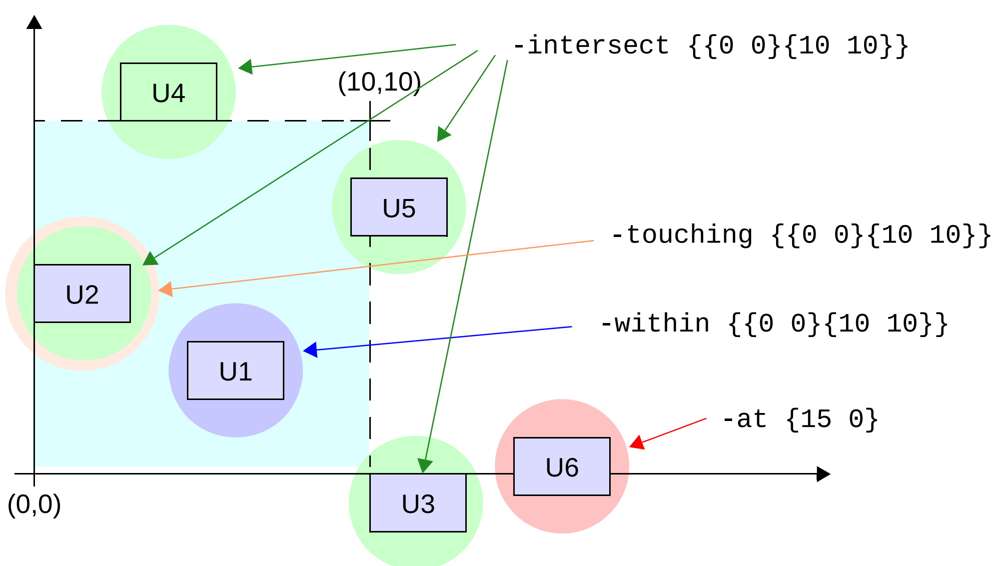
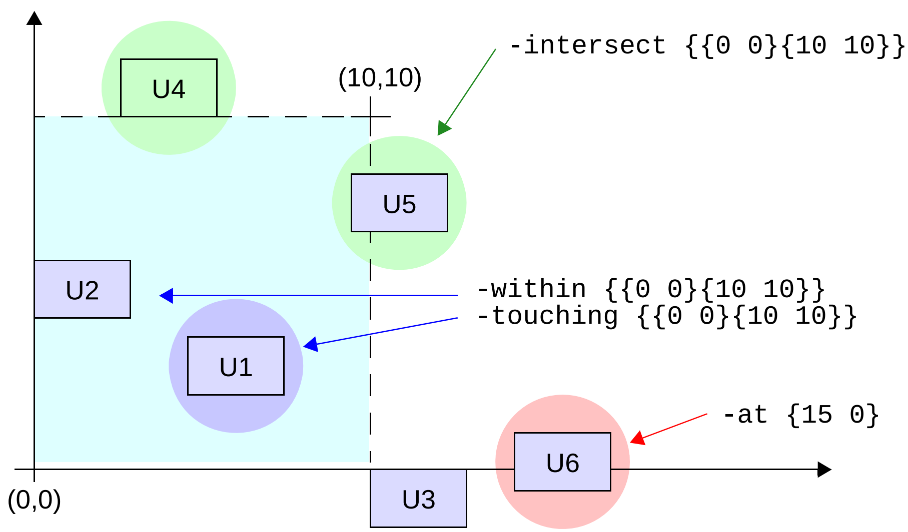


Figure 3-3 get_objects_by_location Behavior



When you search within a box using the `-within` option, the `get_*` commands get only the objects entirely within the box. By default, the `get_objects_by_location` command gets those objects entirely within the box and those that touch the boundary from inside.

When you search for objects that touch a box using the `-touching` option, the `get_*` commands get only the objects that touch the boundary from inside. By default, the `get_objects_by_location` command gets the objects entirely within the box and those that touch the boundary from inside.

When you search for objects that intersect a box using the `-intersect` option, the `get_*` commands get the objects that touch or cross the box boundary. By default, the `get_objects_by_location` command gets only the objects that cross the box boundary; those that only touch the boundary from inside or outside are not included.

When you specify an *x y* point using the `-at` option, the `get_*` commands and the `get_objects_by_location` command get all objects at the specified point.

For example, to get all objects at (15,0):

```
icc2_shell> get_objects_by_location -at {15 0}
{U6}
```

To get all nets and cells at (0,0):

```
icc2_shell> get_objects_by_location -at {0 0} -classes {net cell} \
Warning: Nothing implicitly matched '*' (SEL-003)
```

To get all cells within the box defined by (0,0) and (10,10):

```
icc2_shell> get_cells -within {{0 0} {10 10}}
{U1}
```

```
icc2_shell> get_objects_by_location -within {{0 0} {10 10}} \
  -classes cell
{U1 U2}
```

To get all cells that intersect the boundary of the box defined by (0,0) and (10,10):

```
icc2_shell> get_cells -intersect {{0 0} {10 10}}
{U2 U3 U4 U5}

icc2_shell> get_objects_by_location -intersect {{0 0} {10 10}} \
  -classes cell
{U4 U5}
```

Query by Association

When you use a `get_*` command, in addition to finding objects by name, you can find objects associated with or connected to specific objects by using the `-of_objects` option. For example, to get the cells connected to a given net:

```
icc2_shell> get_cells -of_objects [get_nets net32]
{U83 U75 U18}
```

or to get the nets connected to a given cell:

```
icc2_shell> get_nets -of_objects [get_cells U83]
{net24 net32 net91}
```

You can use the `get_cells` command with the `-of_objects` option to get the cells associated with other cells or with objects such as pins, nets, bounds, I/O guides, I/O rings, voltage areas, voltage area shapes, edit groups, or power strategies.

Query in Physical Context

By default, a `get_*` command searches for objects in the logical context, using the design netlist. To search for objects in the physical context instead, use the `-physical_context` option.

For example, the following command gets all nets in the *logical context* (nets in the design netlist):

```
icc2_shell> get_nets
{net24 net32 ... }
```

The following command gets all nets in the *physical context* (physical nets in the design view):

```
icc2_shell> get_nets -physical_context
{U18/sr0 U18/um18 ... }
```

When you search for an object by name without the `-physical_context` option, the command matches the specified pattern with the *base name* of the object. Conversely, when you use the `-physical_context` option, the command matches the specified pattern with the *full name* of the object in the physical hierarchy:

```
icc2_shell> get_cells -physical_context *I_SDRAM_IF*
  {I_ORCA_TOP/I_SDRAM_IF/sd_mux_dq_out_11 …}
```

The `-physical_context` option is available in commands that get logical objects, such as `get_cells`, `get_pins`, `get_ports`, and `get_nets`; and the `report_hierarchy` command.

When you use the `-physical_context` option, a subblock is treated as a single physical cell, so you cannot use this option from the top-level block to search for logical objects in subblocks, except for a command in the following form:

```
icc2_shell> get_cells –physical_context –of_objects subblock1
...
```

In this case, the command returns all the physical cells of subblock1.

Querying Cascaded Attributes

For collection-type attributes that point to other design objects, you can query the attributes of that object directly from the current object by using cascaded attributes. For example, the `owner` attribute of a via points to the via's owner net. You can query an attribute of the owner net directly from the via object with the `get_attribute` command by cascading the `net` attribute with the via's `owner` attribute.

To get the owner net of via VIA_S1:

```
icc2_shell> get_attribute -objects [get_vias VIA_S1] -name owner
{ISTACK/net12}
```

To get the number of wires in the via owner net:

```
icc2_shell> get_attribute -objects [get_vias VIA_S1] \
  -name owner.number_of_wires
3
```

To get all the vias belonging to the owner net:

```
icc2_shell> get_vias -filter owner.full_name==ISTACK/net12
{VIA_S1 VIA_S2 VIA_S3 VIA_S4}
```

You can cascade any number of attributes, such as the three in the following example:

```
icc2_shell> get_attribute [get_vias VIA_S1] owner.top_block.top_module
{ORCA}
```

Removing Objects

To remove an object from a block, use a `remove_*` command, such as `remove_cells`:

```
icc2_shell> get_cells U11*
{U111 U112 U113 U114 U115}
icc2_shell> remove_cells [get_cells U11*]
5
```

To remove objects belonging to different classes, use the `remove_objects` command:

```
icc2_shell> remove_objects ADDR*
28
```

To protect objects from accidental removal, set their `physical_status` attribute to `locked`. To override this protection, use the `-force` option to remove the object:

```
icc2_shell> get_attribute -objects [get_cells U201] -name physical_status
placed

icc2_shell> set_attribute -objects [get_cells U201] \
  -name physical_status -value locked
{U201}

icc2_shell> remove_cells [get_cells U201]
Error: One or more objects has locked status. (NDMUI-251)

icc2_shell> remove_cells -force [get_cells U201]
1
```

If you remove objects that are included in an existing collection, those objects are removed from the collection as well as from the block.

Working With Collections

A *collection* is a set of design objects such as cells, nets, or libraries. You create, view, and manipulate collections by using commands provided specifically for working with collections. A regular collection contains only one object type, whereas a *group collection* contains multiple object types.

Typically, you create collections with the get_* and all_* commands. For example, to create a collection that contains the cells with instance names that begin with *o* and reference an FD2 library cell, use the following command:

```
icc2_shell> get_cells {o*} -filter {ref_name == FD2}
{o_reg1 o_reg2 o_reg3 o_reg4}
```

Although the returned result looks like a list, it is not. A collection is referenced by a *collection handle*, which is simply a string value that the tool associates with the collection's internal data structure. When the tool returns a collection during an interactive session, for convenience, it shows the collection contents instead of the collection handle. Collections returned by commands during script execution are not printed.

Most command arguments that accept design objects support collections. You can use the get_* commands with the `-of_objects` option to return a collection of objects or a collection of group objects. For example, the following command returns the nets that belong to the MY_GRP group:

```
icc2_shell> get_nets -of_objects [get_groups MY_GRP]
```

The tool has commands to create, get, add, remove, and report groups, such as:

- `create_group`—Creates a group in the current block
- `add_to_group`—Adds objects to a group in the current block
- `remove_from_group`—Removes objects from the group in the current block
- `report_group`—Reports groups in the current block
- `remove_groups`—Removes groups from the current block
- `get_groups`—Creates a collection by selecting groups from the current block

See Also

- *Using Tcl With Synopsys Tools* for detailed information about working with collections
- Working With Objects

Bus and Name Expansion

The tool supports expansion and subscripting of bus names and other object names, making it easier to query bus elements and nets, ports, and pins that share a base name. This feature applies to the `get_nets`, `get_pins`, and `get_ports` commands, as well as commands that accept a collection of nets, ports, or pins.

The examples in the following table demonstrate the name expansion syntax.

Table 3-3 Bus and Name Expansion Examples

Command	Result	Comments
`get_nets a(0:3)`	`{a0 a1 a2 a3}`	Name expansion: A digit followed immediately by a colon (:) and another digit, in parentheses “()” specifies a name expansion range

Table 3-3 Bus and Name Expansion Examples (Continued)

Command	Result	Comments
`get_nets {a[0:3]}`	`{a[0] a[1] a[2] a[3]}`	Bus subscripting: A digit followed immediately by a colon (:) and another digit, in square braces “[ ]” specifies a bus subscripting range
`get_nets {a[0:1,6:8]}`	`{a[0] a[1] a[6] a[7] a[8]}`	Bus subscripting: Two ranges separated by a comma and enclosed by square braces “[ ]”
`get_nets {a[0:4:2]}`	`{a[0] a[2] a[4]}`	Bus subscripting: From 0 to 4, step by 2, enclosed by square braces “[ ]”
`get_nets a(0:4:2)`	`{a0 a2 a4}`	Name expansion: From 0 to 4, step by 2, enclosed by parentheses “( )”
`get_nets {a(0:2)b[0:4:2,3]}`	`{a0b[0] a0b[2] a0b[4] a0b[3] a1b[0] a1b[2] a1b[4] a1b[3] a2b[0] a2b[2] a2b[4] a2b[3]}`	Combined usage of name expansion and bus subscripting

For bus subscripting, the default delimiter characters are square braces “[ ]”. To change to one of the other allowed delimiter character sets, “{ }” “( )” or “< >”, set the `design.bus_delimiters` application option:

```
icc2_shell> set_app_options -name design.bus_delimiters -value {<>}
design.bus_delimiters <>
```

For net, port, and pin subscripting, the default delimiter characters are parentheses “( )”. To change the default, set the `design.name_expansion_delimiters` application option.

Note:
You cannot combine subscripting with wildcards or with the `-regexp`, `-nocase`, or `-exact` options in the `get_nets`, `get_pins`, and `get_ports` commands.

Design Hierarchy

By default, the IC Compiler II tool implements a hierarchical design as physically flat by placing all leaf-level cells, from all hierarchical levels, anywhere in the available chip area.

To implement a lower-level module as a physically distinct block, “commit” that module by using the `commit_block` command. For example,

```
icc2_shell> commit_block u0_1
```

This example creates a separate physical hierarchical block for the logical module named u0_1. The cells in module u0_1 are placed only in this physical block, and this block only contains cells from the module u0_1.

The hierarchy browser in the GUI (View > Hierarchy Browser) lets you visually navigate the logic hierarchy. When you select an instance in the hierarchy browser, the physical block associated with that instance (if any) is highlighted in the physical layout view, as shown in the following figures.

Figure 3-4 Hierarchy Browser

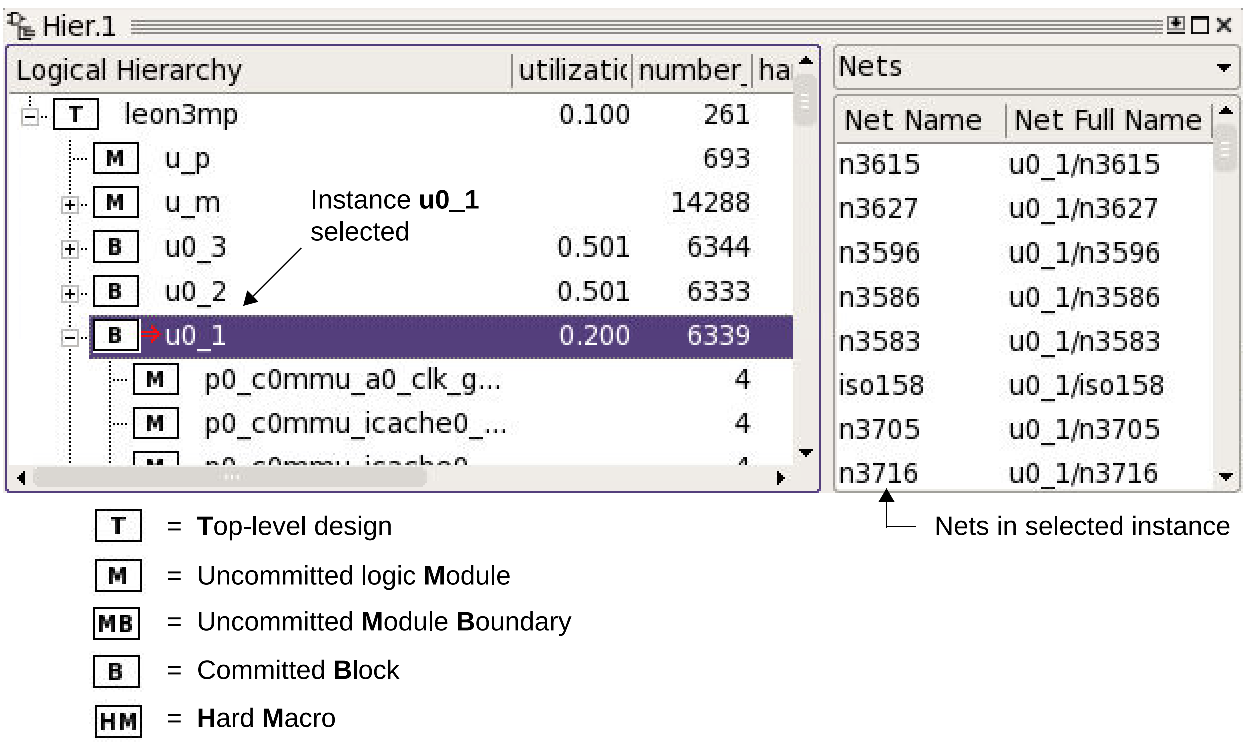
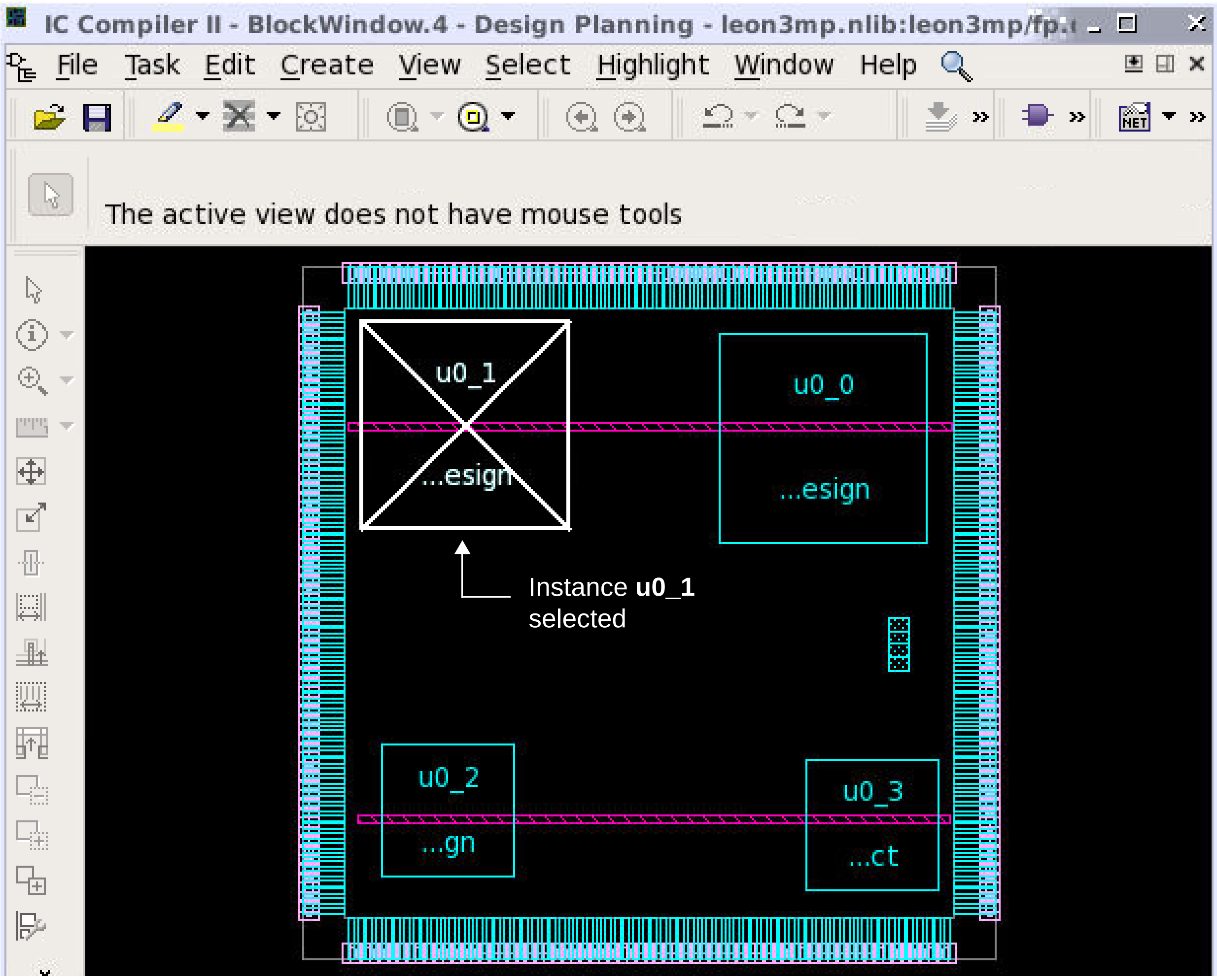


Figure 3-5 Physical Layout View



The `current_block` and `current_design` commands provide two different ways to set the current block in the IC Compiler II tool. The `current_block` command sets the current block by specifying the block name, whereas the `current_design` command sets the current block by specifying a hierarchical module name in the logical netlist.

In addition to the current block, there is also a current working instance, which is a hierarchical cell in the netlist of the current block. You can use the `current_instance` command to traverse the logic hierarchy and set the current instance, much like you use the `cd` command in Linux to traverse a file hierarchy.

To report the full hierarchy of the design, use the `report_hierarchy` command. Use the `-physical_context` option to report only the physical (not logical) cells in the hierarchy, and use the `-no_leaf` option to exclude the leaf-level cells and get a more concise report.

See Also

- current_block
- current_design
- current_instance
- report_hierarchy
- Hierarchical Query Using get_* Commands

current_block

The `current_block` command takes a block as an argument, such as an object returned by the `get_blocks` command. When the `current_block` command is used with an argument, it sets the current block to the specified block. When used without an argument, it returns the current block.

```
icc2_shell> current_block design3mp
{mylibA:design3mp.design}

icc2_shell> current_block
{mylibA:design3mp.design}
```

current_design

The `current_design` command takes a design object as an argument, such as an object returned by the `get_designs` command. In this context, a *design* is a hierarchical module in the logical netlist. When the `current_design` command is used with an argument, it sets the current block to the block containing the specified design. When used without an argument, it returns the current block, just like the `current_block` command.

```
icc2_shell> get_designs
{design3mp mul32_infer0 design3s design3s_2 design3s_2 design3X}

icc2_shell> current_design design3X
{mylibB:design3X.design}

icc2_shell> current_design
{mylibB:design3X.design}
```

current_instance

The current working instance is a hierarchical cell of the current block. The default current instance is the top-level module of the current block. You can use the `current_instance` command to traverse the logic hierarchy and set the current instance much like you use the `cd` command in Linux to traverse a file hierarchy.

```
icc2_shell> current_block
{mylibA:design3mp.design}

icc2_shell> current_instance
Current instance is the top-level module of design 'design3mp'

icc2_shell> get_cells -filter "is_hierarchical==true"
{u0_0 u0_1 u0_2 u0_3 u_m u_p}

icc2_shell> current_instance u0_0
u0_0
icc2_shell> current_instance p0_iu0
u0_0/p0_iu0
icc2_shell> current_instance .
u0_0/p0_iu0
icc2_shell> current_instance ..
u0_0
icc2_shell> current_instance ../u0_1/p0_mul0
u0_1/p0_mul0
icc2_shell> current_instance
Current instance is the top-level module of design 'design3mp'
```

report_hierarchy

The `report_hierarchy` command reports the hierarchy of a block. By default, it generates a report on the current block showing the full physical and logical reference hierarchy down to the leaf module level.

To get a more concise report, use the `-physical_context` option to report only the physical (not logical) cells in the hierarchy or the `-no_leaf` option to exclude the leaf-level cells, or both. For example,

```
icc2_shell> report_hierarchy -physical_context -no_leaf
...
mylibA:design3mp/fp.design
  design3s:design3s/fp.design
     mul32_infer0:mul32_infer0/fp.design
  design3s:design3s_2/fp.design
  design3s:design3s_3/fp.abstract
1
```

```
icc2_shell> report_hierarchy -physical_context
...
mylibA:design3mp/fp.design
  B4ISH1025_NS                saed32io_wb
  D4I1025_NS                  saed32io_wb
  ISH1025_NS                  saed32io_wb
  ...
  design3s:design3s/fp.design
     AND2X1_RVT               saed32rvt
     AND2X2_RVT               saed32rvt
     AND3X1_RVT               saed32rvt
     ...
     mul32_infer0:mul32_infer0/fp.design
        ...
  design3s:design3s_2/fp.design
     ISOLORX1_HVT             saed32hvt
     LSDNSSX1_HVT             saed32hvt_dlvl
     OAI222X1_RVT             saed32rvt
  design3s:design3s_3/fp.abstract
1

icc2_shell> report_hierarchy
...
mylibA:design3mp/fp.design
  LSUPX1_HVT                     saed32hvt_ulvl
  NBUFFX16_LVT                   saed32lvt
  design3s:design3s/fp.design
     ISOLORAOX1_HVT              saed32hvt
     SNPS_CLOCK_GATE_HIGH_div32_7
     SNPS_CLOCK_GATE_HIGH_div32_8
     ...
     (very long report shows all leaf-level logical and physical cells)
     ...
```

By default, multiple occurrences of a block or module are combined into a single report for all instances of the same block unless the occurrences are bound to different views. To expand the report to show the hierarchy of all instances, use the `-hierarchical` option of the command.

For more information, see the man page for the `report_hierarchy` command.

Hierarchical Query Using get_* Commands

The IC Compiler II tool offers commands to query the cells, nets, pins, and ports in the design database:

```
get_cells
get_nets
get_pins
get_ports
```

Each command creates a collection of objects in the design that meet the criteria specified by the command options. You can use a collection as input to another command, or set a variable to the collection for future use.

The command options let you restrict the scope of the query by object name, hierarchical level, physical context, and object attributes. For example, for the `get_cells` command:

```
get_cells [patterns]
  [-hierarchial]
  [-physical_context]
  [-of_objects objects]
  [-filter expression]
  ...
```

```
prompt> get_cells      # get all cells at the current level of hierarchy

prompt> get_cells *    # get all cells at the current level of hierarchy

prompt> get_cells BLKA/*  # get all cells of BLKA

prompt> get_cells R* \    # get cells named R* at
 -hierarchical            # all levels of hierarchy

prompt> get_cells R* \                # get cells named R* in the flat
 -physical_context                    # physical context

prompt> get_cells \                   # get all cells in the flat
 -physical_context \                  # physical context that own
 -of_objects [get_nets n253]          # physical pins of net n253

prompt> get_cells R* \                # get cells named R* at all levels
 -hierarchical \                      # of hierarchy; include only the
 -filter defined(parent_block_cell)   # blocks that have a parent
                                      # (only the cells of subblocks)
```

Query Using Only a Search String

When you query the design for objects using only a search string and no other options, the command searches for objects that have the specified name. The name string can be plain or hierarchical:

```
prompt> get_cells R*        # get cells named R* at current level

prompt> get_cells REG1/R*   # get cells named REG1/R*
                            # (same as cells named R* in REG1)
```

The asterisk wildcard character (`*`) does *not* match with the hierarchy separator character, the forward slash (`/`). Therefore, a search string without any forward slash characters occurs only at the current level and does not traverse the hierarchy.

Note that the slash character is sometimes used as part of a cell name and not as a hierarchy separator, as demonstrated in the following example.

```
prompt> get_cells I_P
{I_P}

prompt> get_cells I_P*      # "*" does not match hierarchy separator,
{I_P}                       # search current level only

prompt> get_cells I_P/*U3   # search under cell I_P, one level only
{I_P/U3 I_P/m2/U3}
```

Why did the last search return `I_P/m2/U3`, which appears to be *two* levels below `I_P`? The answer is that the cell is directly under `I_P` and is literally named “`m2/U3`” with the slash character inside the cell name. You can confirm this by querying the cell attributes:

```
prompt> get_attribute [get_cells I_P/*U3] -name parent_cell
{I_P I_P}

prompt> get_attribute [get_cells I_P/*U3] -name name
U3 m2/U3
```

Query With the -hierarchical Option

When you query the design for objects with the `-hierarchical` option, the command searches for objects throughout the logical hierarchy of the design:

```
prompt> get_cells R* -hierarchical
{BLK1/REG43 BLK2/RXF38 BLK2/ACE1/RG95}
```

The command searches for an object of the exact name at each level of hierarchy, like the `find` command in Linux. Therefore, the search string must be a simple name; it *cannot* use hierarchy separator characters:

```
prompt> get_cells BLK1/R* -hierarchical
Warning: No cell objects matched 'BLK1/R*' (SEL-004)
Error: Nothing matched for collection (SEL-005)
```

If you include a slash character in the search string while using the `-hierarchical` option, the command searches for objects that literally have the slash character as part of the object name; it does not search a lower level of hierarchy.

Query With the -physical_context Option

When you query the design for objects with the `-physical_context` option, the command searches for objects in the flattened physical netlist for the design:

```
prompt> get_cells *RLB_27 -physical_context
{I_BLK1/I_REG19/RLB_27}
```

Note that an asterisk wildcard character (`*`) in a search string matches with the forward slash character, interpreting the slash as part of the object name.

In the flattened physical netlist, all cells are considered leaf cells. This is the netlist seen by the router. The full names of the cells can contain slash characters that reflect the hierarchy of the design, but the physical netlist itself is treated as entirely flat and the slash characters are treated as ordinary characters used in object names.

Because the netlist is considered flat, it is not necessary (and not recommended) to use the `-hierarchical` option together with the `-physical_context` option.

Query With the -of_objects Option

When you query the design for objects with the `-of_objects` option, the command searches for objects that are associated with the listed objects:

```
prompt> get_cells I_P/*                           # get cells
{I_P/U11 I_P/U13 I_P/U161 I_P/U14}

prompt> get_pins -of_objects [get_cells I_P/U13]  # get pins of cell
{I_P/U13/A1 I_P/U13/A2 I_P/U13/Y I_P/U13/VDD I_P/U13/VSS}

prompt> get_nets -of_objects [get_cells I_P/U13]  # get nets connected
{I_P/n287 I_P/n17 I_P/n26 I_P/VDD I_P/VSS}        # to cell

prompt> get_cells -of_objects [get_nets I_P/n17]  # get cells connected
{I_P/U161 I_P/U14 I_P/m2/U13}                     # to net
```

You cannot specify a search string while also using the `-of_objects` option, but you can filter the results with the `-filter` option.

Query With the -filter Option

In a `get_*` command, you can use the `-filter` option to restrict the generated collection to only the objects that meet some filtering condition specified as a Boolean expression.

For example, the following command gets all cells named `B*` but then only allows the cells that have their `ref_name` attribute set to the string `FD2` to be added to the collection:

```
prompt> get_cells "B*" -filter "ref_name == FD2"
{"B_reg1", "B_reg2", "B_reg3", "B_reg4"}
```

The following hierarchy-related attributes can help you find the logical or physical owner of an object:

- `parent_cell` – The *logical* cell that contains the object. For a physical object, this is the cell of the parent block. For a purely logical object, this is different from the cell of the parent block.

- `parent_block` – The *physical* block (libname:block/label.design) that contains the object. The parent block of a block is the same block itself.
- `parent_block_cell` – The first cell that is a block (libname:block/label.design) found while traversing the parent cells up the hierarchy. When the object is associated with a logic hierarchy, the result is different from the parent cell.
- `top_block` – The top-level block (libname:block/label.design) in the current block hierarchy.

For example,

```
prompt> get_cells -hierarchical  # get objects in all levels of
                                 # hierarchy, both logical and physical

# To create a collection of physical cells in a subblock
prompt> get_cells –physical_context \   # get physical cells of object
  –of_objects subblock1                 # named "subclock1"

prompt> get_cells -hierarchical \       # get objects from the top level
  –filter undefined(parent_block_cell)  # (those that have no parent)

prompt> get_cells -hierarchical \       # get cells inside all subblocks
  –filter defined(parent_block_cell)    # (those that have a parent)
```

Technology Data Access

You can get, report, and modify some technology data using Tcl commands. This is the data specified in a technology file and read into a design library by using the `read_tech_file` command or by using the `-technology` option of the `create_lib` command.

Table 3-4 shows the technology objects that you can access from the command line and the commands used to query or modify the objects. For all of these objects, you can use the `report_attributes -class` *`object_class`* command to query the object's attributes. If an object has settable attributes, you can modify the attributes by using the `set_attribute` command.

Table 3-4 Technology Objects and Access Commands

Object class	Technology file section	Object access commands
tech	Technology	create_tech get_techs remove_tech
site_def	Tile	create_site_def get_site_defs remove_site_defs report_site_defs
layer	Layer	create_layer get_layers
purpose	LayerDataType	create_purpose get_purposes
via_def	ContactCode	create_via_def get_via_defs remove_via_defs report_via_defs
design_rule	DesignRule	get_design_rules report_design_rules
pr_rule	PRRule	create_pr_rule get_pr_rules remove_pr_rules report_pr_rules
density_rule	DensityRule	create_density_rule get_density_rules remove_density_rules
via_rule	ViaRule	create_via_rule get_via_rules remove_via_rules report_via_rules

The tool saves changes to the technology file stored in the design library, but it does not save changes to technology data stored in reference libraries, as described in the following table.

Method for storing technology data	Changes saved in the design library?
Technology data stored in cell library referenced by design library	No, available in current session only
Technology data stored in technology library referenced by design library	No, available in current session only
Technology file loaded into design library	Yes

To write out the modified technology data into a technology file, use the `write_tech_file` command. However, the technology file syntax does not support the `symmetry` and `is_default` attributes of site_def objects or the `routing_direction` and `track_offset` attributes of layer objects, so these attributes are not written to the technology file. For more information about preparing the site definition and routing layer technology data, see “Completing the Technology Data” in the *IC Compiler II Library Preparation User Guide*.

You can use the `create_routing_rule` command to create nondefault routing rules and the `set_routing_rule` command to apply these rules to specific layers or nets. These nondefault routing rules do not change the default rules defined in the technology file and are not written out by the `write_tech_file` command.

The following session example shows how to query and modify the site definitions in the design library’s technology data.

```
icc2_shell> list_attributes -application -class site_def
...
Attribute Name      Object     Type       Properties  ...
---------------------------------------------------------
full_name           site_def   string     A,S
height              site_def   distance   A,S
is_default          site_def   boolean    A,S
symmetry            site_def   string     A,S
...
icc2_shell> report_site_defs
...
       Width  Height  Type  Is_default  Tech     Symmetry   Legal_orienta
-------------------------------------------------------------------------
unit1  0.15   1.67    core  default     saed32   --         R0

icc2_shell> report_attributes -application -class site_def \
  [get_site_defs]
...
```

```
Design     Object   Type        Attribute Name        Value
--------------------------------------------------------------
design3mp  unit1    string      full_name              unit1
design3mp  unit1    distance    height                 1.6720
design3mp  unit1    boolean     is_default             true
design3mp  unit1    string      symmetry
...
icc2_shell> set_attribute -objects [get_site_defs] -name symmetry \
  -value {Y}
{unit}
icc2_shell> get_attribute -objects [get_site_defs] -name symmetry
Y
icc2_shell> create_site_def -name unit2 -width 0.1520 \
  -height 3.3440 -symmetry Y
{unit2}
icc2_shell> get_site_defs
{unit1 unit2}
icc2_shell> report_attributes -application -class site_def \
  [get_site_defs unit2]
...
Design     Object   Type        Attribute Name        Value
----------------------------------------------------------------
design3mp  unit1    string      full_name              unit2
design3mp  unit1    distance    height                 3.3440
design3mp  unit1    boolean     is_default             false
design3mp  unit1    string      symmetry              Y
...
```

See Also

- Loading the Technology Data

Reporting Design Information

A block is a container for physical and functional design data. The current block is the default block affected by block-related commands. You can set or report the current block by using the `current_block` command or set the current block for a specified top-level module by using the `current_design` command.

To report information about the design data stored in the current block, use the following commands:

- `report_design`

  By default, this command generates a summary report on the design contents such as the number of cells, hierarchical levels, chip area, and total route length. For details, see Reporting Design Contents.

- `report_unbound`

  This command checks the design for objects that are not linked to a reference library (unbound objects) and reports unbound objects as error or warning messages. For details, see Reporting Unbound Objects.

- `check_physical_constraints`

  This command checks the design for issues related to physical constraints that could lead to problems at later stages of the design flow, and reports these issues as error, warning, and information messages. The checked conditions include basic floorplan data, site rows, move bounds, group bounds, layers, tracks, macros, other cell instances, and placement blockages. For details, see Reporting Physical Constraints.

Reporting Design Contents

By default, using the `report_design` command without options generates a summary report on the design contents such as the number of cells, hierarchical levels, chip area, and total route length. For a more detailed report, specify the report type in the command:

```
icc2_shell> report_design -library
...
icc2_shell> report_design -netlist
...
icc2_shell> report_design -netlist -hierarchical
...
icc2_shell> report_design -floorplan
...
icc2_shell> report_design -routing
...
```

The `report_design` command generates a detailed report as shown in the following example:

```
icc2_shell> report_design -netlist
...
------------------------------------------------------------------------
                            NETLIST INFORMATION
------------------------------------------------------------------------
CELL INSTANCE INFORMATION
------------------------------------------------------------------------
Cell Instance Type          Count % of         Area   % of siteAreaPerSite
                                  total               total
------------------------------------------------------------------------
TOTAL LEAF CELLS            12973  100  9360583.005   100 unit:36911736
  Standard cells            12704   97    61696.760     0 unit:242763
  Filler cells                  0    0        0.000     0
  Diode cells                   0    0        0.000     0
  Module cells                  4    0  6117000.323    65 unit:24069033
    Soft macro cells            4    0  6117000.323    65 unit:24069033
```

```
  Hard macro cells              4    0    49885.923     0 unit:197220
  ...

REFERENCE DESIGN INFORMATION
------------------------------------------------------------------------
Number of reference designs used:88
------------------------------------------------------------------------
Name           Type          Count     Width    Height        Area
------------------------------------------------------------------------
SDFFX1_RVT     lib_cell       1729      5.17      1.67       8.641   ...
NBUFFX32_RVT   lib_cell       1727      6.38      1.67      10.674   ...
AO22X1_RVT     lib_cell       1364      1.52      1.67       2.541   ...
...

NET INFORMATION
------------------------------------------------------------------
NetType                 Count FloatingNets        Vias Nets/Cells
------------------------------------------------------------------
Total                   59276        45346      113837      4.569
Signal                  59276        45346      113837      4.569
Power                       0            0           0      0.000
...

NET FANOUT AND PIN COUNT INFORMATION
---------------------------------------------------
Fanout        Netcount     netPinCount     NetCount
---------------------------------------------------
<2               54241     <2                 45346
2                 2452     2                   9123
3                  979     3                   2224
4                  559     4                    979
5                  243     5                    559
...

PORT AND PIN INFORMATION
-----------------------------------------------------------------
Type         Total     Input    Output     Inout      3-st    ...
-----------------------------------------------------------------
Total       103285     86052     18407      1272       484    ...
Macro          600       344       256         0       256    ...
Ports          261       160       228       127         0    ...
...
```

For more information about the `report_design` command, see the man page. Additional usage information is available in the *IC Compiler II Implementation User Guide* and *IC Compiler II Design Planning User Guide*.

Reporting Unbound Objects

You can check for objects that are not linked to a reference library (unbound objects) by using the `report_unbound` command. When you run the `report_unbound` command, the IC Compiler II tool reports unbound cell instances, site rows, site arrays, and vias with a warning or error message:

```
icc2_shell> report_unbound -verbose
Error: Via definition 'VIA23' is missing. 'VIA_S_3 'and ‘2' other via(s)
which reference this via definition are unbound. (CHUNB-007)
```

By default, the `report_unbound` command check objects at the top level. To report unbound objects in other levels of the physical hierarchy, specify the `-hierarchical` option.

To link an unbound object to a reference library, use the following guidelines:

- Unbound cell instances

  Check your reference library settings, add a reference library as needed with the `set_ref_libs -add` command, and rebind the block with the `link_block -force -rebind` command.

- Unbound site rows or site arrays

  Check the technology file for a site definition; if there is no site definition, create a site definition by using the `create_site_def` command.

- Unbound vias

  Check the via definitions in the current block or the technology file; if the via definition is missing, create a via definition by using the `create_via_def` command.

You can also use the `check_design` command to report unbound objects, as shown in the following example:

```
icc2_shell> check_design -checks unbound
```

The `check_design` report provides the same results as the `report_unbound` report without options.

See Also

- Specifying a Design Library's Reference Libraries
- Reporting Reference Libraries
- Rebinding Reference Libraries of a Design Library
- Reference Library List

Reporting Physical Constraints

You can check a design's physical constraints by using the `check_physical_constraints` command. The command checks the design for issues related to physical constraints that could lead to problems at later stages of the design flow, and reports these issues as error, warning, and information messages.

The checked conditions include

- Basic floorplan data
- Site rows
- Move bounds
- Group bounds
- Layers
- Tracks
- Macros and other cell instances
- Placement blockages

To check the physical constraints of the design, run the `check_physical_constraints` command:

```
icc2_shell> check_physical_constraints
...
Warning: The spacing of layer 'VTL_N' is greater than the difference of
 the pitch and width of the layer. (DCHK-105)
Warning: The spacing of layer 'PO' is greater than the difference of the
 pitch and width of the layer. (DCHK-105)
Information: The layer 'M8' does not contain any PG shapes. (DCHK-104)
Information: The layer 'M9' does not contain any PG shapes. (DCHK-104)
Warning: The orientation of cell instance 'tapfiller_TAPCELLBWP16P90_0'
 does not match any legal orientations. (DCHK-103)
Warning: The orientation of cell instance 'tapfiller_TAPCELLBWP16P90_1'
 does not match any legal orientations. (DCHK-103)
1
```

For information about the issues the tool identifies and how to fix them, see the man page for the corresponding message ID number.

In addition to generating a report, the `check_physical_constraints` command generates an enhanced messaging system (EMS) database that you can view by using the message browser in the IC Compiler II GUI. Create the EMS database by running the `create_ems_database` command before you run the `check_physical_constraints` command:

```
icc2_shell> create_ems_database check_hier.ems
icc2_shell> check_physical_constraints
icc2_shell> save_ems_database
```

In the IC Compiler II GUI message browser, you can sort, filter, and link the messages to the corresponding man page. For information about viewing the EMS database in the message browser, see the *IC Compiler II Implementation User Guide*.

Polygon Manipulation

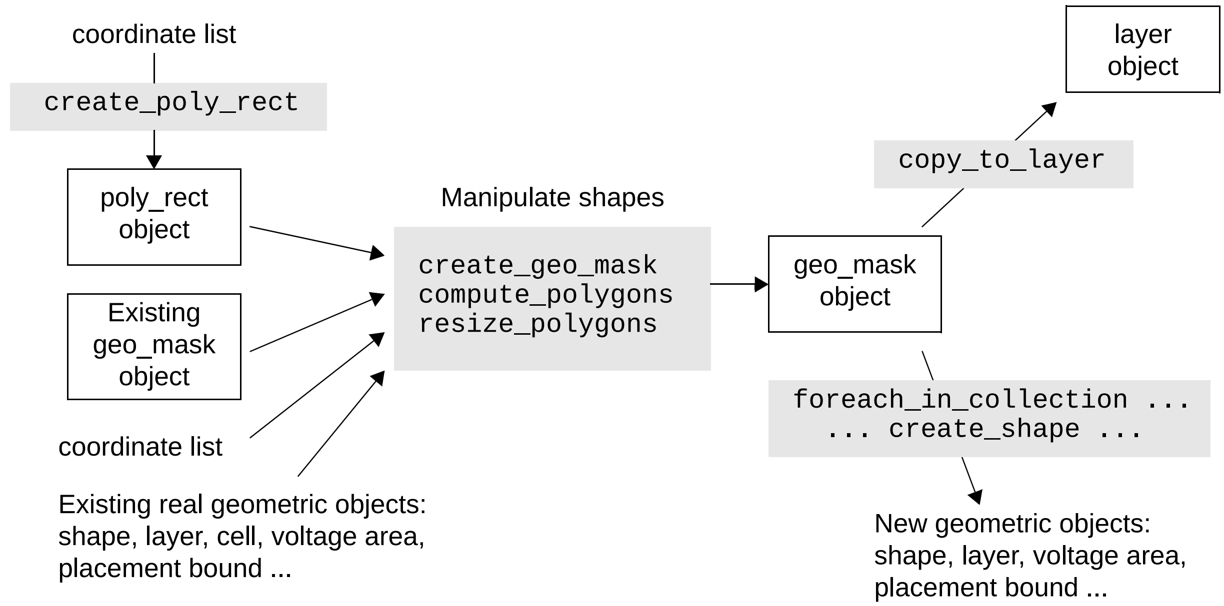
To create and edit polygon shapes, you can use two kinds of abstract geometric objects:

- poly_rect – a single geometric shape that consists of a set of coordinate points
- geo_mask – a collection of poly_rect objects

The poly_rect and geo_mask type objects are abstract and have no direct functional purpose in a block. To create a functional shape such as a mask layer, blockage, or keepout region, you need to convert poly_rect or geo_mask objects into real objects.

The following figure shows the flow to create, edit, and convert abstract shapes to make real shapes.

Figure 3-6 How to Manipulate Geometric Objects



The commands in the following table are available to create, manipulate, and convert polygon shapes.

Table 3-5 Polygon Manipulation Commands

Command	Description
`compute_area`	Calculates the area of a geometric region
`compute_polygons`	Performs a Boolean operation between two shapes to create a geo_mask object
`copy_to_layer`	Copies a geo_mask object to create a collection of shapes in a mask layer
`create_geo_mask`	Copies geometric objects to create a new geo_mask object
`create_poly_rect`	Creates a collection of poly_rect objects from a list of coordinates and optionally associates a layer with the result
`resize_polygons`	Pushes the edges of specified polygons inward or outward, creating a new geo_mask object
`split_polygons`	Decomposes a geometric region into rectangles or trapezoids, creating a collection of geo_mask or poly_rect objects

For details, see

- Creating poly_rect and geo_mask Objects
- Converting geo_mask Objects Into Functional Shapes

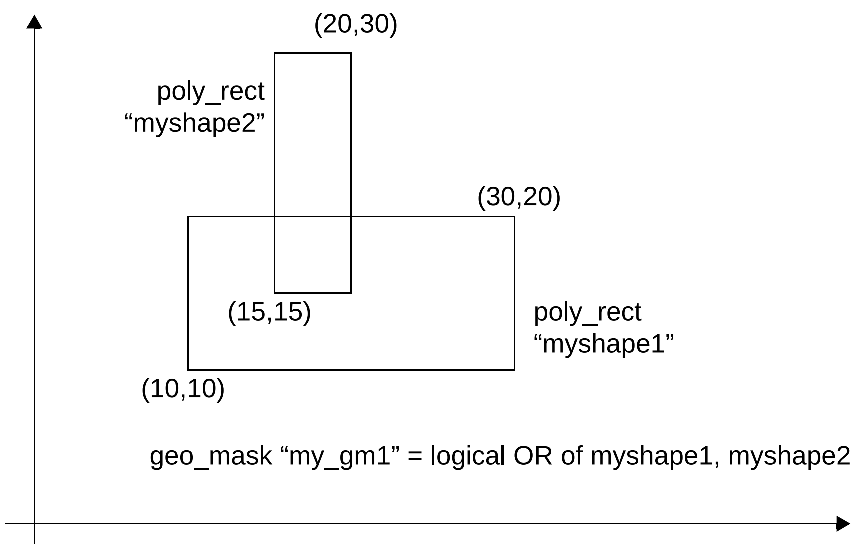
Creating poly_rect and geo_mask Objects

To create a poly_rect object, use the `create_poly_rect` command and specify the vertex coordinates: two points for a rectangle or three or more points for a polygon of any shape.

To create a geo_mask object, you can copy existing geometric shapes with the `create_geo_mask` command or perform operations on existing shapes with the `compute_polygons` or `resize_polygons` command.

In the following example, you create two overlapping rectangular poly_rect objects, perform a logical OR of these shapes to create a new geo_mask object, and copy the geo_mask object to create a rectilinear shape on the M2 layer.

Figure 3-7 Creating a geo_mask Object From Two poly_rect Objects

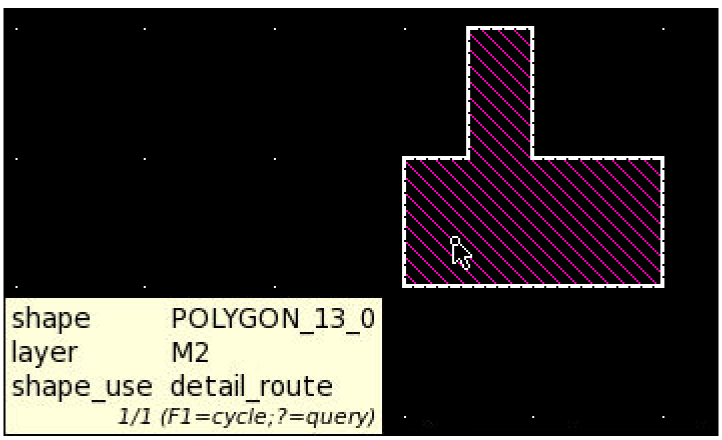



```
icc2_shell> current_block
{libA:block1.design}
icc2_shell> set myshape1 [create_poly_rect -boundary {{10 10} {30 20}}]
{{{10 10} {30 20}}}
icc2_shell> set myshape2 [create_poly_rect -boundary {{15 15} {20 30}}]
{{{15 15} {20 40}}}
icc2_shell> set my_gm1 [compute_polygons \
  -objects1 $myshape1 -objects2 $myshape2 -operation OR]
...
```

```
icc2_shell> report_attributes -application -class geo_mask $my_gm1
...
Design         Object         Type        Attribute Name    Value
----------------------------------------------------------------------
block1         geo_mask       boolean     is_empty          false
block1         geo_mask       string      object_class      geo_mask
block1         geo_mask       collection  poly_rects        {20 40} ...
block1         geo_mask       int         shape_count       1
1
icc2_shell> copy_to_layer -layer M2 -geo_masks $my_gm1
{POLYGON_13_0}
icc2_shell> save_block
...
```

These commands create a rectilinear shape on the M2 layer, which you can display in the GUI as shown in the following figure.

Figure 3-8 geo_mask Object Copied to M2 Layer Shape



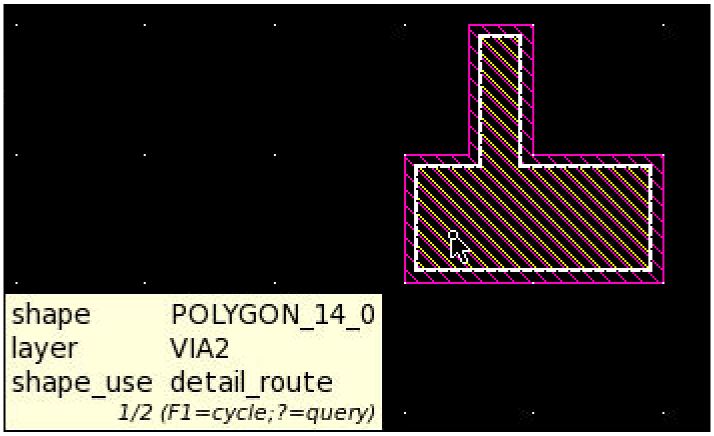
Alternatively, you can use the `compute_polygons` command directly with coordinate lists:

```
icc2_shell> set my_gm1 [compute_polygons \
  -objects1 {{10 10} {30 20}} -objects2 {{15 15} {20 30}} -operation OR]
...
icc2_shell> copy_to_layer -layer M2 -geo_masks $my_gm1
{POLYGON_13_0}
```

Another polygon manipulation command is the `resize_polygons` command, which pushes the edges of specified polygons inward or outward, creating a new geo_mask object. For example, the following script takes the existing shapes on the M2 layer, pushes the edges inward by 1.0 unit to make a new geo_mask object, and then converts the geo_mask object into a new shape on the VIA2 layer.

```
set sh2 [get_shapes -of_objects M2]
set gm2 [resize_polygons -objects $sh2 -size -1]
copy_to_layer -layer VIA2 -geo_masks $gm2
```

Figure 3-9 M3 Layer Shape Created by Resizing



See Also

- Polygon Manipulation
- Converting geo_mask Objects Into Functional Shapes

Converting geo_mask Objects Into Functional Shapes

The poly_rect and geo_mask type objects are abstract and have no direct functional purpose in a block. To create a functional shape such as a mask layer, blockage, or keepout region, you need to convert poly_rect or geo_mask objects into real objects by using a command such as `copy_to_layer`, `create_placement_blockage`, or `create_shape`.

The following example creates a geo_mask object and copies it to a shape on the M2 layer.

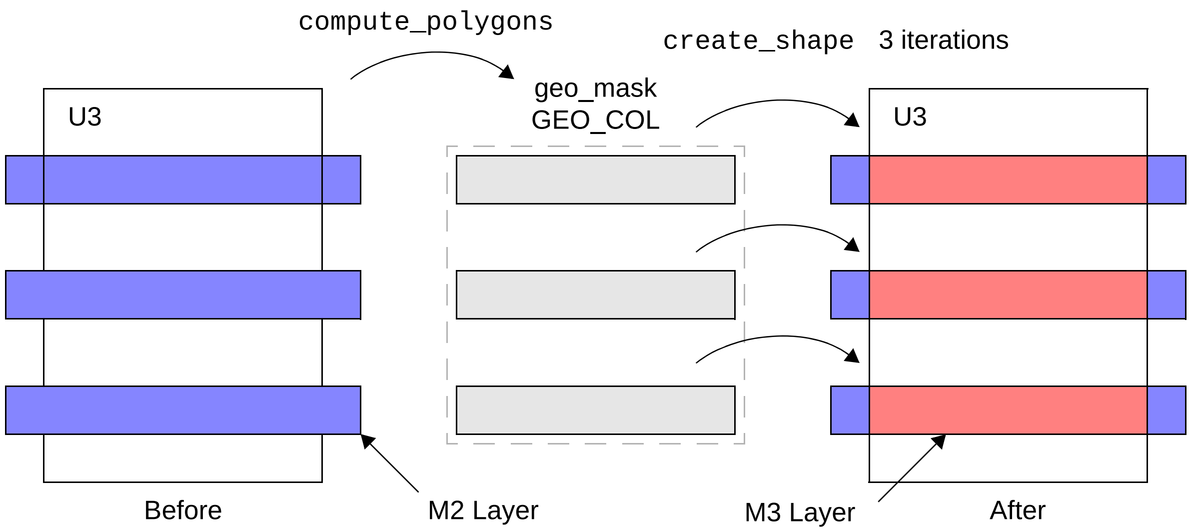
```
icc2_shell> set pr2 [create_poly_rect -boundary {{10 10} {20 30}}]
...
icc2_shell> set gm2 [create_geo_mask -objects $pr2]
...
icc2_shell> copy_to_layer -layer M2 -geo_masks $gm2
{RECT_13_0}
```

A geo_mask object is a collection of poly_rect objects. If the geo_mask object contains multiple distinct shapes, you can iterate through the collection by using the `foreach_in_collection` command, and convert each poly_rect object to a real object.

For example, the following script creates a geo_mask object named GEO_COL, which contains a set of shapes generated by the overlap of cell U3 with all shapes on layer M2. The script then iterates through the GEO_COL collection and converts each shape to a new shape on the M3 layer, as shown in Figure 3-10.

```
set SM [get_cells U3]
set GEO_COL [compute_polygons –operation AND \
  -objects1 $SM -objects2 [get_layers M2] ]

foreach_in_collection PR \
  [get_attribute $GEO_COL poly_rects] {
     set PRPL [get_attribute $PR point_list]
     create_shape –shape_type polygon -layer M3 -boundary $PRPL
}
```

Figure 3-10 Iterating Through a geo_mask Collection to Make New Shapes



See Also

- Polygon Manipulation
- Creating poly_rect and geo_mask Objects

Undoing and Redoing Changes to the Design

You can undo (reverse) or redo (reapply) many of the operations performed in the tool.

To undo the most recent change, use the `undo` command:

```
icc2_shell> create_block BLK1
...
icc2_shell> create_block BLK2
...
```

```
icc2_shell> list_blocks
Lib lib_A /path/libs/lib_A current
  *>  0 BLK1.design May-01-20:28
  *>  0 BLK2.design May-01-20:28 current
2
icc2_shell> undo
1
icc2_shell> list_blocks
Lib lib_A /path/libs/lib_A current
  *>  0 BLK1.design May-01-20:28 current
1
icc2_shell> undo
1
icc2_shell> list_blocks
Lib lib_A /path/libs/lib_A current
0
```

To redo the most recent operation reversed by an `undo` command, use the `redo` command:

```
icc2_shell> redo
1
icc2_shell> list_blocks
Lib lib_A /path/libs/lib_A current
  *>  0 BLK1.design May-01-20:28
1
icc2_shell> redo
1
icc2_shell> list_blocks
Lib lib_A /path/libs/lib_A current
  *>  0 BLK1.design May-01-20:28 current
  *>  0 BLK2.design May-01-20:28
2
```

You can only undo commands that change the design database, such as `create_block`. You cannot undo commands that merely change the tool context, such as `current_design`. Also, some types of design changes cannot be reversed.

To determine whether you can undo the effects of a particular command:

```
icc2_shell> get_undo_info -command create_block
Information: The command 'create_block' is undoable. (UNDO-014)
1
icc2_shell> get_undo_info -command current_lib
Information: The command 'current_lib' is independent of the undo system.
(UNDO-013)
2
```

For more information, see

- Undoing or Redoing Multiple Changes
- Disabling or Limiting the Undo Feature

Undoing or Redoing Multiple Changes

To undo or redo multiple changes in a single command, use the `-levels` option:

```
icc2_shell> undo -levels 3   # Reverse the 3 most recent changes
3

icc2_shell> redo -levels 2   # Reapply the 2 most recent reversed changes
2
```

To redo all recent changes and restore the design to the most recent available state:

```
icc2_shell> redo -all
5
```

The number returned by this command (5 in this example) indicates the number of changes redone.

The tool maintains an internal *undo list* to keep track of the most recent design changes that can be reversed. If you plan to make multiple changes that you might want to undo later, you can set an *undo marker* in the list, and later undo all of changes back to the point at which you set the marker:

```
icc2_shell> create_block block1
...
icc2_shell> create_undo_marker my_marker
...
icc2_shell> create_block block2
...
icc2_shell> create_block block3
...
icc2_shell> undo -marker my_marker
2
icc2_shell> list_blocks
Lib lib_B /path/libs/lib_B current
  *>  0 block1.design May-02-21:08 current
1
```

Conversely, if you plan to undo multiple recent changes that you might want to redo later, you can set an undo marker, and later redo all of the changes forward to the point at which you set the marker:

```
icc2_shell> create_block block1
...
icc2_shell> create_block block2
...
icc2_shell> create_block block3
...
icc2_shell> create_undo_marker mrkA
...
icc2_shell> undo
...
```

```
icc2_shell> undo
...
icc2_shell> list_blocks
Lib lib_A /path/libs/lib_A current
  *>  0 block1.design May-02-21:18 current
1
icc2_shell> redo -marker mrkA
2
icc2_shell> list_blocks
Lib lib_A /path/libs/lib_A current
  *>  0 block1.design May-02-21:18 current
  *>  0 block2.design May-02-21:18
  *>  0 block3.design May-02-21:18
3
```

Note:

If you run a command that is not reversible, the tool clears the undo list and you can no longer reverse previous operations.

To get information about the current undo list:

```
icc2_shell> get_undo_info
is_enabled true is_undoable true undo_cmd_name "create_block" is_redoable
false redo_cmd_name "" depth 4 max_depth 100 memory 104747 max_memory
1000000000
```

Disabling or Limiting the Undo Feature

The undo feature uses runtime and memory to maintain the design history. If you do not need this feature, you can disable it:

```
icc2_shell> set_app_options -name shell.undo.enabled false
shell.undo.enabled false
```

To disable the undo feature for one or more commands or to execute a block of commands as a single “undoable” unit, use the `eval_with_undo` command.

By default, the tool stores no more than 100 changes and uses no more than 1GB of memory to maintain the history of changes for the undo command. You can optionally decrease these limits to reduce runtime and memory usage:

```
icc2_shell> set_app_options -name shell.undo.max_levels -value 50
shell.undo.max_levels 50
icc2_shell> set_app_options -name shell.undo.max_memory -value 500000000
shell.undo.max_memory 500000000
```

Conversely, you can increase these limits to handle a larger undo history.

Schema Versions

The IC Compiler II database infrastructure is periodically updated to support new database features. Each new version of the database is called a "schema" and each schema has a version number. When you save a design library with the `save_lib` command or save a cell library with the `commit_workspace` command, the library information is stored under the tool's current schema.

To get a report of tool and schema version numbers and tool compatibility, use the `report_versions` command:

```
icc2_shell> report_versions
J-2014.06
  ICC2 schema revision: 1.0
...

L-2016.03
   ICC2 schema revision: 1.165
   Compatibility:
        - ICV K-2015.12-SP2
        - PrimeRail K-2015.12-SP2

   Includes Support for:
        - Conn view for rail analysis
...
```

To determine the schema number of a library, open the library and use the `get_attribute` command to report the library's `read_from_schema_version` attribute:

```
icc2_shell> get_attribute -objects [current_lib] \
  -name read_from_schema_version
1.165
icc2_shell> get_attribute -objects [get_libs tech32hvt] \
  -name read_from_schema_version
1.063
```

A schema number such as "1.165" consists of a major version number before the decimal point ("1") and a minor version number after the decimal point ("165").

IC Compiler II tools are backward-compatible, but not directly forward-compatible. In other words, you can read in an older library with a newer tool, which updates the library from the older schema to the current one. However, you cannot always directly read a newer library with an older tool.

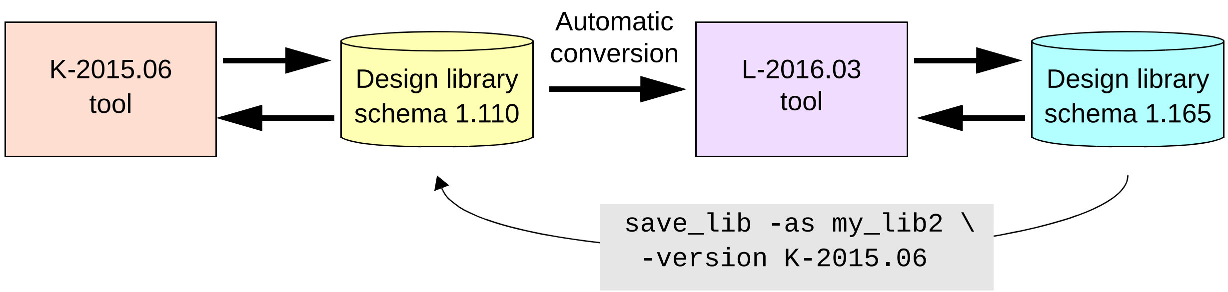
Service pack releases of the tools (such as L-2016.03-SP2) are generally both forward and backward compatible with each other and with the baseline release (such as L-2016.03).

When you save a design library, the tool saves it under the current schema by default. To make a newer library compatible with an older tool, open the library in the newer tool and then save it under a new name by using the `save_lib` command with the `-version` option:

```
icc2_shell> current_lib
{my_lib}
icc2_shell> save_lib -as my_lib2 -version K-2015.06
...
```

Then the new library can be opened by the older tool, as shown in the following figure.

Figure 3-11 IC Compiler II Database Schema Compatibility



If your newer design library has many blocks but you only need a few blocks in the older format, to reduce the conversion time and save disk space, copy the few needed blocks into a separate new design library, and then save the smaller design library in the older format.