

# **Allegro®**

## **PCB Editor User Guide:**

### **Defining and Developing Libraries**

Series XL and GXL

**Product Version 16.2**  
**November 2008**

© 1991–2008 Cadence Design Systems, Inc. All rights reserved.

Portions © Apache Software Foundation, Sun Microsystems, Free Software Foundation, Inc., Regents of the University of California, Massachusetts Institute of Technology, University of Florida. Used by permission. Printed in the United States of America.

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

Allegro PCB Editor contains technology licensed from, and copyrighted by: Apache Software Foundation, 1901 Munsey Drive Forest Hill, MD 21050, USA © 2000-2005, Apache Software Foundation. Sun Microsystems, 4150 Network Circle, Santa Clara, CA 95054 USA © 1994-2007, Sun Microsystems, Inc. Free Software Foundation, 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA © 1989, 1991, Free Software Foundation, Inc. Regents of the University of California, Sun Microsystems, Inc., Scriptics Corporation, © 2001, Regents of the University of California. Daniel Stenberg, © 1996 - 2006, Daniel Stenberg. UMFPACK © 2005, Timothy A. Davis, University of Florida, (davis@cise.ulf.edu). Ken Martin, Will Schroeder, Bill Lorensen © 1993-2002, Ken Martin, Will Schroeder, Bill Lorensen. Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, Massachusetts, USA © 2003, the Board of Trustees of Massachusetts Institute of Technology. All rights reserved.

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

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

All other trademarks are the property of their respective holders.

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

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

**Patents:** Allegro PCB Editor, described in this document, is protected by U.S. Patents 5,481,695; 5,510,998; 5,550,748; 5,590,049; 5,625,565; 5,715,408; 6,516,447; 6,594,799; 6,851,094; 7,017,137; 7,143,341; 7,168,041.

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

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

# Contents

---

<u>Preface</u> .....	1
<u>The Library Development Process Model</u> .....	2
<u>Library Development Tasks</u> .....	2
<u>Library Development Tools</u> .....	3
 <u>1</u>	
<u>Library Development Process</u> .....	1
<u>Library Development Tasks</u> .....	3
<u>Creating Libraries</u> .....	3
<u>Library Files</u> .....	4
<u>Library Development Tools</u> .....	5
 <u>2</u>	
<u>Library Padstacks</u> .....	7
<u>Library and Layout Padstacks</u> .....	8
<u>Library padstacks</u> .....	8
<u>Layout padstacks</u> .....	9
<u>Padstacks and Pins</u> .....	9
<u>Standard Pad Shapes</u> .....	10
<u>Photoplot Pad Data</u> .....	11
<u>Defining Library Padstacks</u> .....	12
<u>Preparing to Define Padstacks</u> .....	12
<u>Using the Padstack Designer</u> .....	13
<u>Recording a Padstack Script</u> .....	17
<u>Updating Layout Padstacks</u> .....	17
<u>Creating a Padstack List File</u> .....	17
<u>Reviewing the Refresh Padstack Log File</u> .....	18
<u>Custom Pad Shape Symbols</u> .....	18
<u>How Custom Pad Shape Symbols Work</u> .....	18
<u>Characteristics of Custom Pad Shapes</u> .....	20

---

<u>Updating a Library Padstack in a Symbol</u> .....	20
<u>Updating a Layout Padstack</u> .....	20
<u>Suppressing Unused Padstacks</u> .....	21
<u>Purging Unused Padstacks</u> .....	22

### 3

<u>Working with Symbols</u> .....	23
<u>Working with the symbol mode</u> .....	23
<u>Symbol Types</u> .....	23
<u>Symbol and Drawing Files</u> .....	25
<u>Legal Classes for Symbol Types</u> .....	27
<u>Creating a Symbol</u> .....	27
<u>Adding Areas</u> .....	28
<u>Adding Pins</u> .....	29
<u>ETCH and Vias in Symbols</u> .....	29
<u>Creating Package Symbols</u> .....	30
<u>Package Symbol Elements</u> .....	30
<u>Prerequisites for Creating a Package Symbol</u> .....	32
<u>Guidelines for Creating Package Symbols</u> .....	33
<u>Defining a Package Symbol</u> .....	33
<u>Defining Symbol Heights</u> .....	34
<u>Defining Component Heights with Properties</u> .....	37
<u>Creating Mechanical Symbols</u> .....	41
<u>Types of Mechanical Symbols</u> .....	41
<u>Guidelines for Creating Mechanical Symbols</u> .....	42
<u>Creating a Board Outline</u> .....	43
<u>Creating Format Symbols</u> .....	43
<u>Library Format Symbols</u> .....	43
<u>Guidelines for Creating Format Symbols</u> .....	44
<u>Defining a Format Symbol</u> .....	45
<u>Creating Flash Symbols</u> .....	45
<u>Choosing a Design Methodology for Negative Planes</u> .....	46
<u>Converting Flash Symbols When Migrating Databases</u> .....	47
<u>Flash Symbols in Padstack Designer</u> .....	47
<u>Treatment of Nonconforming Symbols</u> .....	48

## Allegro PCB Editor User Guide: Defining and Developing Libraries

---

<u>MDA Format Output Files</u> .....	48
<u>Defining a Flash Symbol</u> .....	49
<u>Updating Symbols</u> .....	49
<u>Creating a Symbol List File</u> .....	49
<u>Reviewing the Refresh Log File</u> .....	50

### 4

## Checking Symbols Automatically .....

<u>Overview</u> .....	51
<u>Configuring the check symbol Command</u> .....	51
<u>Globals File</u> .....	51
<u>Rule Table File</u> .....	52
<u>Developing Symbol Check Rules</u> .....	54
<u>Installing Custom Rules</u> .....	57
<u>Predefined Rules</u> .....	58
<u>REFDES Checks</u> .....	58
<u>COMPONENT VALUE Checks</u> .....	59
<u>DEVICE TYPE Checks</u> .....	61
<u>TOLERANCE Checks</u> .....	62
<u>USER PART NUMBER Checks</u> .....	63
<u>GEOMETRY Checks</u> .....	64
<u>Reports</u> .....	65
<u>Global Variables and Values</u> .....	66

### 5

## Preparing Device Files .....

<u>Device File Records</u> .....	72
<u>Device File Format</u> .....	73
<u>Syntax and Field Descriptions</u> .....	75
<u>Guidelines for Creating a Device File</u> .....	80
<u>Creating a Device File</u> .....	80
<u>Specifying Multiple Functions in a Device File</u> .....	81
<u>Specifying Definition Properties in a Device File</u> .....	83
<u>Checking a Device File</u> .....	83
<u>Reviewing the dev_check.log File</u> .....	83

## 6

<u>Using Technology and Parameter Files</u> .....	89
<u>Working with Tech Files</u> .....	89
<u>Accessing Tech Files</u> .....	90
<u>Exporting Tech Files</u> .....	90
<u>Importing Tech Files</u> .....	90
<u>Comparing Tech Files to Designs</u> .....	91
<u>Upreving Tech Files</u> .....	91
<u>Locking Constraint Sets</u> .....	92
<u>Technology Constraints File</u> .....	93
<u>Working with Parameter Files</u> .....	94
<u>Parameter File Syntax</u> .....	94
<u>Accessing Parameter Files</u> .....	94
<u>Exporting Parameter Files</u> .....	94
<u>Importing Parameter Files</u> .....	95

## 7

<u>Generating Allegro PCB Editor Libraries</u> .....	97
<u>Creating Libraries from Existing Designs</u> .....	99
<u>Creating Device and Symbol Files with Batch Commands</u> .....	99
<u>Creating a Clipboard Library</u> .....	99
<u>Setting the CLIPPATH environment</u> .....	100

## 8

<u>Package Symbol Library</u> .....	101
<u>Capacitors</u> .....	102
<u>cap300</u> .....	102
<u>cap400</u> .....	102
<u>cap600</u> .....	103
<u>dipcap</u> .....	103
<u>smdcap</u> .....	103
<u>cap196</u> .....	104
<u>cap1000</u> .....	104
<u>cap1500</u> .....	104

## Allegro PCB Editor User Guide: Defining and Developing Libraries

---

<u>capck05</u>	105
<u>capck06</u>	105
<u>capck60</u>	106
<u>capck62</u>	106
<u>case17-02</u>	107
<u>ck12-10pf</u>	107
<u>ck13-10pf</u>	108
<u>ck14-10pf</u>	108
<u>ck15-10pf</u>	108
<u>ck16-10pf</u>	109
<u>ck17-10pf</u>	109
<u>cy10</u>	109
<u>cy15</u>	109
<u>cy20</u>	110
<u>Resistors</u>	110
<u>res400</u>	110
<u>res500</u>	110
<u>res1000</u>	111
<u>res800</u>	111
<u>resadj</u>	112
<u>smdres</u>	112
<u>SIPs</u>	112
<u>sip6</u>	112
<u>sip8</u>	113
<u>sip10</u>	113
<u>sip12</u>	114
<u>sip30</u>	115
<u>Connectors</u>	116
<u>conn6</u>	116
<u>conn9</u>	116
<u>conn10</u>	117
<u>conn20</u>	118
<u>conn26</u>	119
<u>conn50</u>	119
<u>multiconn30</u>	120
<u>multicon43</u>	120

## Allegro PCB Editor User Guide: Defining and Developing Libraries

---

<u>ibmconn</u>	120
<u>db9</u>	121
<u>db15</u>	121
<u>db25</u>	122
<u>eurocon</u>	123
<u>Crystals</u>	124
<u>crys11mhz</u>	124
<u>crys14</u>	124
<u>Diodes</u>	125
<u>do5</u>	125
<u>do13</u>	126
<u>do35</u>	127
<u>do41</u>	128
<u>dio400</u>	129
<u>dio500</u>	129
<u>Potentiometer</u>	130
<u>pot</u>	130
<u>Test Point</u>	130
<u>tp</u>	130
<u>Switches</u>	131
<u>dipswitch</u>	131
<u>switch</u>	131
<u>DIPs</u>	132
<u>dip32 6</u>	132
<u>dip4 3</u>	133
<u>dip6 3</u>	133
<u>dip8 3</u>	133
<u>dip10 3</u>	134
<u>dip14 3</u>	134
<u>dip16 3</u>	135
<u>dip18 3</u>	135
<u>dip18 4</u>	136
<u>dip20 3</u>	136
<u>dip20 4</u>	137
<u>dip20 6</u>	137
<u>dip22 3</u>	138



## Allegro PCB Editor User Guide: Defining and Developing Libraries

---

<u>dip22_4</u>	138
<u>dip22_6</u>	139
<u>dip24_3</u>	139
<u>dip24_4</u>	140
<u>dip24_6</u>	140
<u>dip26_3</u>	141
<u>dip28_3</u>	141
<u>dip28_6</u>	142
<u>dip40_6</u>	143
<u>dip48_6</u>	144
<u>dip52_6</u>	145
<u>dip64_6</u>	146
<u>dip68_6</u>	147
<u>Jumpers</u>	148
<u>jumper1</u>	148
<u>jumper2</u>	148
<u>jumper3</u>	148
<u>jumper4</u>	149
<u>jumper5</u>	149
<u>jumper8</u>	149
<u>jumper14</u>	150
<u>jumper16</u>	150
<u>Pin Grid Arrays</u>	151
<u>pga68</u>	151
<u>pga84</u>	151
<u>pga100</u>	152
<u>pga101</u>	153
<u>pga120</u>	154
<u>pga124</u>	155
<u>pga132</u>	156
<u>pga132_ci</u>	157
<u>pga132-x</u>	158
<u>pga133</u>	159
<u>pga156</u>	160
<u>pga156_x</u>	161
<u>pga172</u>	162

## Allegro PCB Editor User Guide: Defining and Developing Libraries

---

<u>pga176</u> .....	163
<u>PLCCs</u> .....	164
<u>picc18</u> .....	164
<u>plcc20</u> .....	165
<u>plcc28</u> .....	165
<u>picc32</u> .....	166
<u>plcc44</u> .....	167
<u>plcc48</u> .....	168
<u>picc52</u> .....	169
<u>picc68</u> .....	170
<u>picc84</u> .....	171
<u>icc20</u> .....	172
<u>icc24</u> .....	173
<u>icc28</u> .....	174
<u>icc24</u> .....	175
<u>icc48</u> .....	176
<u>iccs18</u> .....	177
<u>iccs22</u> .....	178
<u>iccs24</u> .....	179
<u>iccs28</u> .....	180
<u>iccs32</u> .....	181
<u>SOICs</u> .....	182
<u>soic8</u> .....	182
<u>soic14</u> .....	182
<u>soic16</u> .....	183
<u>soic16w</u> .....	183
<u>soic20</u> .....	184
<u>soic20w</u> .....	185
<u>soic24</u> .....	186
<u>soic24w</u> .....	187
<u>soic28w</u> .....	187
<u>sol120</u> .....	188
<u>Transistors</u> .....	189
<u>sot23</u> .....	189
<u>sot89</u> .....	190
<u>to3</u> .....	191

## Allegro PCB Editor User Guide: Defining and Developing Libraries

---

<u>to5</u>	191
<u>to12</u>	192
<u>to18</u>	192
<u>to39</u>	192
<u>to46</u>	193
<u>to52</u>	193
<u>to92</u>	193
<u>to107</u>	194
<u>to126</u>	194
<u>to126h</u>	195
<u>to126v</u>	196
<u>to204aa</u>	197
<u>to220abh</u>	197
<u>to220abv</u>	198
<u>to220h</u>	199
<u>to220v</u>	199
<u>Flat Packs</u>	200
<u>flat14</u>	200
<u>flat16</u>	201
<u>flat18</u>	202
<u>flat20</u>	203
<u>flat24</u>	204
<u>flat28</u>	205
<u>cpfp68</u>	206
<u>quadflat24</u>	207
<u>ZIPs</u>	208
<u>zip16</u>	208
<u>zip20</u>	209

## 9

<u>Mechanical Symbol Library</u>	211
----------------------------------	-----

<u>Card Outlines</u>	211
<u>ibm</u>	211
<u>multibus</u>	212
<u>euros</u>	212

## Allegro PCB Editor User Guide: Defining and Developing Libraries

---

<u>eurod</u>	213
<u>Mounting Holes</u>	213
<u>mtq125</u>	213
<u>mtq156</u>	214
<u>mtq250</u>	214

## 10

<u>Format Symbol Library</u>	215
<u>Target</u>	215
<u>target</u>	215
<u>Drawing Formats</u>	216
<u>asizev</u>	216
<u>asizeh</u>	216
<u>bsize</u>	217
<u>csize</u>	217
<u>dsize</u>	218
<u>esize</u>	218

# Preface

---

The first step in a typical PCB physical design process is creating libraries, which are a collection of graphic symbols that represent packages, mechanical elements, drawing formats, and custom pads and padstacks. Library development is the process of updating the libraries of padstacks and symbols that are part of the installation of Allegro PCB Editor.

The library consists of padstack, symbol, and device files in the library directory:

```
<install_dir>/share/pcb/pcb_lib
```

The library directory contains two subdirectories:

```
symbols
devices
```

The `symbols` subdirectory contains:

- Padstack files that define physical pad sizes and shapes. The characteristics of each padstack are contained in a `.pad` file, which contains:

- ☐ Padstack name
- ☐ Top , bottom , and internal pad data
- ☐ Solder paste and film mask data
- ☐ Drill hole information

- Symbol files for:

- ☐ Packages that are physical representation of components, such as dual in-line packages, resistors, capacitors, and edge connectors

Package symbols have a reference designator label and at least one pin with a pin number.

- ☐ Mechanical elements that are usually non-electrical, such as board outlines, mounting holes, plating bars, and card ejectors

Mechanical-only fixtures with drill holes have pins with no pin numbers. Mechanical symbols do not have a reference designator label.

- ❑ Drawing formats of standard drawing sizes that include borders, title blocks, notes, revision blocks, and other types of drawing information

The `devices` subdirectory contains:

- Device files that define logic information for certain component types

The library consists of several types of files, each identified by different file extensions:

Padstacks	.pad
Custom pad shapes	.dra, .ssm
Package symbols	.dra, .psm
Mechanical symbols	.dra, .bsm
Format symbols	.dra, .osm
Flash symbols	.dra, .fsm
Devices	.txt

## The Library Development Process Model

- Update the libraries
  - ❑ Add new library padstacks and symbols
  - ❑ Add unique (that is, custom) pads used in padstack definitions that require a unique pad shape instead of one of the standard geometries
  - ❑ Edit padstacks and symbols
- Create libraries containing
  - ❑ Project-specific symbols
  - ❑ Device files containing logic information for any unique package symbols used in designs
  - ❑ Technology files consisting of specific design rules and constraints that can be applied to other designs

## Library Development Tasks

When creating libraries for Allegro PCB Editor:

- Check and review the supplied library.

- Create padstack library.

You must define padstacks before you create package symbols because each pin in a package symbol must have an associated padstack.

- Create package symbol .psm library.
- Create mechanical symbols (.bsm).
- Create format symbols (.osm).
- Create and check device files.
- Store the library files in the appropriate directories

## Library Development Tools

Allegro PCB Editor supplies the following tools for creating, editing, and updating library components:

- Padstack Designer

The Padstack Designer lets you create and edit padstacks and save them to a design, to a library, or to both at once. You can run the Padstack Designer as a stand-alone tool or in conjunction with the *Tools – Padstack – Modify Library Padstack* (`editpadlib`) command in the Allegro PCB Editor user interface.

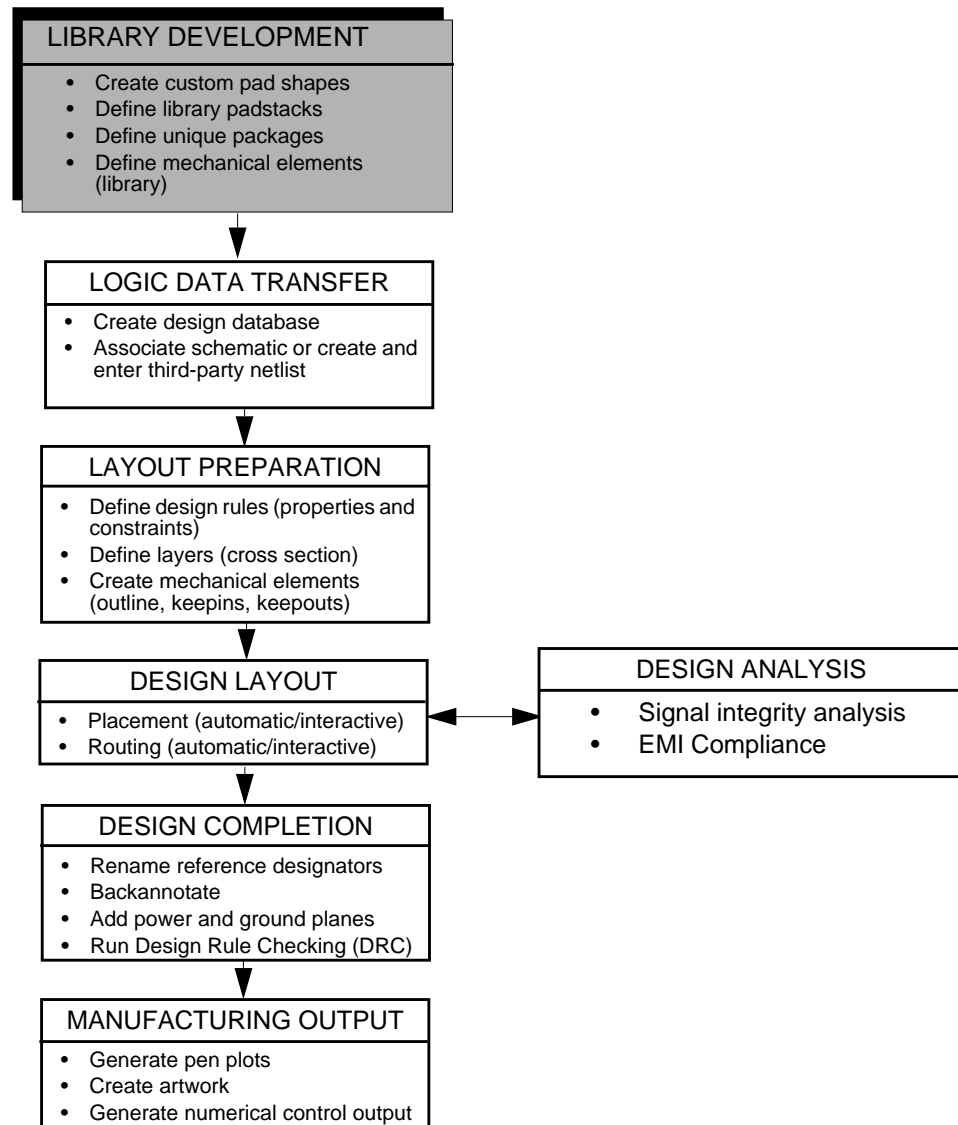
- Symbol Editor

Allegro PCB Editor lets you create and/or edit symbols from one of five symbol modes:

- ☐ Package (or Package Wizard)
- ☐ Mechanical
- ☐ Format
- ☐ Shape
- ☐ Flash

Library padstack definition and symbol creation occur at the beginning of the design flow, as shown in the following figure.

### Library Development in a Design Flow



Allegro PCB Editor lets you define new libraries based on the libraries that are associated with existing designs. This feature is useful for:

- Creating a library for a design that originates from a different CAE or physical layout system, for which the original library is not provided



- Creating a library that is compatible with the padstacks and symbols that are already in a design, such as when the design is from an earlier library revision
- Recovering libraries that may have been lost, by obtaining the library data from a design that contains the correct library data
- Creating a clipboard library that contains elements from designs or symbol drawings

You can create libraries from existing Allegro PCB Editor designs by extracting:

- Device files
- Padstacks
- Symbols

You do this by using the `dlib` command to obtain device files, padstack definitions and symbol definitions from an existing layout, or by running the `create_devices` and `create_sym` batch programs to obtain device files and symbol definitions from an existing layout.

**Note:** Creating new libraries based on existing designs occurs late in a design flow or following its completion.

# **Allegro PCB Editor User Guide: Defining and Developing Libraries**

## Preface

---

---

# Library Development Process

---

The first step in a typical PCB physical design process is to create libraries, which are collections of graphic symbols representing packages, mechanical elements, drawing formats, and custom pads and padstacks.

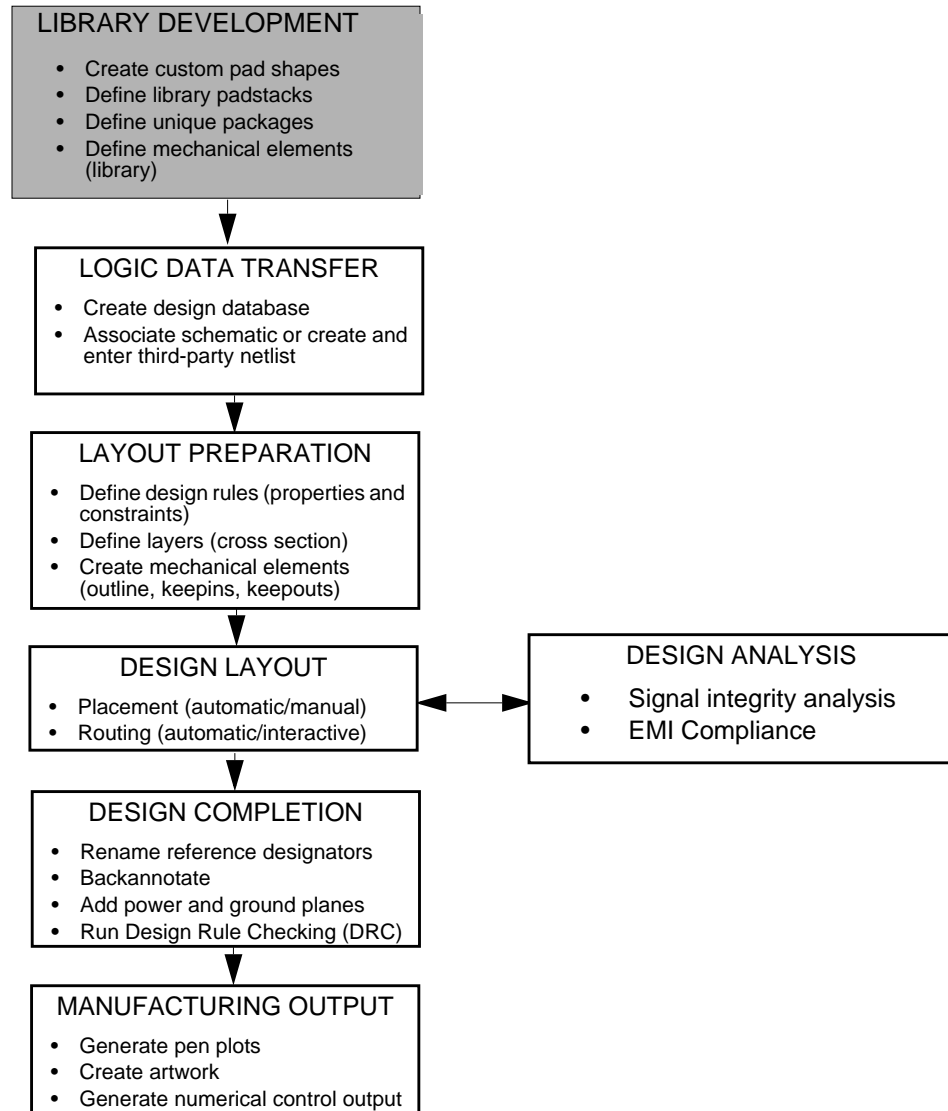
To develop libraries, you update the libraries of padstacks and symbols that are installed with Allegro PCB Editor. Library padstack definition and symbol creation occur at the beginning of the design flow, shown in Figure [1-1](#).

# Allegro PCB Editor User Guide: Defining and Developing Libraries

## Library Development Process

---

**Figure 1-1 Library Development in a Design Flow**



## Library Development Tasks

The library development process consists of these tasks:

- Update the libraries provided by Allegro PCB Editor.
  - ❑ Review the libraries
  - ❑ Add new library padstacks and symbols
  - ❑ Add unique (that is, custom) pads used in padstack definitions that require a unique pad shape instead of one of the standard geometries
  - ❑ Edit padstacks and symbols
- Create libraries containing:
  - ❑ Project-specific symbols

You must define padstacks before you create package symbols because each pin in a package symbol must have an associated padstack.
  - ❑ Device files containing logic information for any unique package symbols used in designs
  - ❑ Technology files consisting of specific design rules and constraints that can be applied to other designs

## Creating Libraries

Later in a design flow or following its completion, you can define new libraries based on the libraries that are associated with existing Allegro PCB Editor designs. This feature is useful for:

- Creating a library for a design that originates from a different CAE or physical layout system, for which the original library is not provided
- Creating a library that is compatible with the padstacks and symbols that are already in a design, such as when the design is from an earlier library revision
- Recovering libraries that may have been lost, by obtaining the library data from a design that contains the correct library data
- Creating a clipboard library that contains elements from designs or symbol drawings

You can create libraries from existing Allegro PCB Editor designs by extracting:

- Device files
- Padstacks
- Symbols

Choose *File – Export – Libraries* (dlib command) to obtain device files, padstack definitions and symbol definitions from an existing layout, or by running the create\_devices and (*File – Create Symbol*) create\_sym batch commands to obtain device files and symbol definitions from an existing layout. The *Allegro PCB and Package Physical Layout Command Reference* describes these commands.

## Library Files

The Allegro PCB Editor library consists of padstack, symbol, and device files supplied in the library directory:

```
<install_dir>/share/lib/pcb/pcb_lib
```

The library directory contains two subdirectories: `devices` and `symbols`.

The `devices` subdirectory contains device files that define logic information for certain component types.

The `symbols` subdirectory contains:

- Padstack files that define physical pad sizes and shapes. The characteristics of each padstack are contained in a `.pad` file, which contains:
  - ☐ Padstack name
  - ☐ Top, bottom, and internal pad data
  - ☐ Solder paste and film mask data
  - ☐ Drill hole information
- Symbol files for:
  - ☐ Packages that are physical representation of components, such as dual in-line packages, resistors, capacitors, and edge connectors

Package symbols have a reference designator label and at least one pin with a pin number.

## Allegro PCB Editor User Guide: Defining and Developing Libraries

### Library Development Process

---

- ❑ Mechanical elements that are usually non-electrical, such as board outlines, mounting holes, plating bars, and card ejectors

Mechanical—only fixtures with drill holes have pins with no pin numbers. Mechanical symbols do not have a reference designator label.

- ❑ Drawing formats of standard drawing sizes that include borders, title blocks, notes, revision blocks, and other types of drawing information

The Allegro PCB Editor library consists of several types of files, each identified by different file extensions:

File Type	Extension
Padstacks	.pad
Custom pad shapes	.dra, .ssm
Package symbols	.dra, .psm
Mechanical symbols	.dra, .bsm
Format symbols	.dra, .osm
Flash symbols	.dra, .fsm
Devices	.txt

## Library Development Tools

Allegro PCB Editor supplies the following tools for creating, editing, and updating library components:

### ■ Padstack Designer

The Padstack Designer lets you create and edit padstacks and save them to design, to a library, or to both at once. You can run the Padstack Designer as a standalone tool or choose *Tools – Padstack – Modify Design Padstack* ([padeditdb](#) command) and *Tools – Padstack – Modify Library Padstack* ([padeditlib](#) command) as you modify padstacks.

### ■ Symbol Editor

Allegro PCB Editor lets you create and edit symbols from one of five symbol modes:

- ❑ Package (or Package Wizard)
- ❑ Mechanical

## **Allegro PCB Editor User Guide: Defining and Developing Libraries**

### **Library Development Process**

---

- ☐ Format
- ☐ Shape
- ☐ Flash



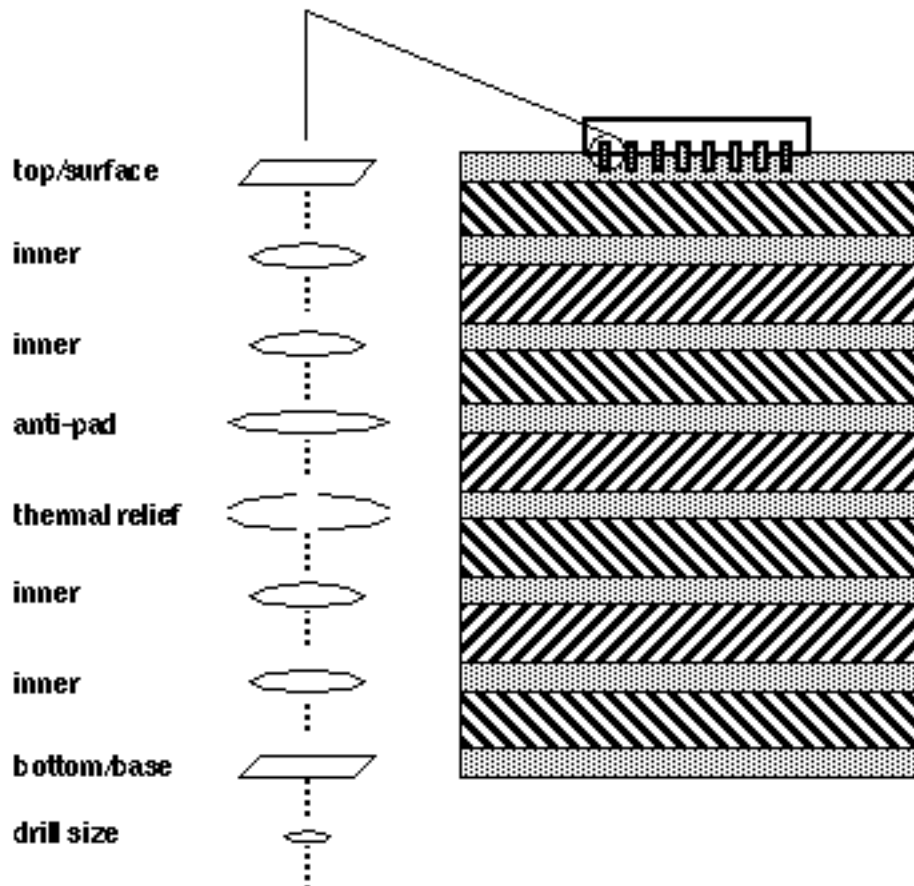
---

## Library Padstacks

---

Each symbol pin in a design must have a padstack associated with it. The padstack describes how the symbol pin connects to each layer in the design. You define new library padstacks using the *Padstack Designer*, which you open using the `pad_designer` command, described in the *Allegro PCB and Package Physical Layout Command Reference*.

### Padstack Definition Expanded on an Eight-Layer Board



A padstack is a file that contains the following information for each layer:

- Pad size and shape
- Drill size and drill display figure

A padstack also describes the following information for the TOP and BOTTOM layers:

- Soldermask
- Pastemask
- Filmmask

A padstack can contain up to sixteen user-defined mask layers. A padstack can also contain Numerical Control (NC) drill data, which Allegro PCB Editor uses to create drill drawings, described in the *Preparing Manufacturing Data* user guide in your documentation set.

## Library and Layout Padstacks

Allegro PCB Editor categorizes padstacks as follows:

### Library padstacks

Library padstacks are padstack definitions contained in the Allegro PCB Editor library or a user-defined library directory.

Allegro PCB Editor supplies a set of standard padstacks in the Allegro PCB Editor library. You can create new library padstacks with the Padstack Designer.

Library padstacks are generic in that they define pad data for beginning (TOP) and ending (BOTTOM) layers. In addition, library padstacks have one default set of pad data that can apply to all internal layers.

You can also add other layers to a library padstack definition. Allegro PCB Editor uses the pad data for these layers only when those same layers are defined in the layout.

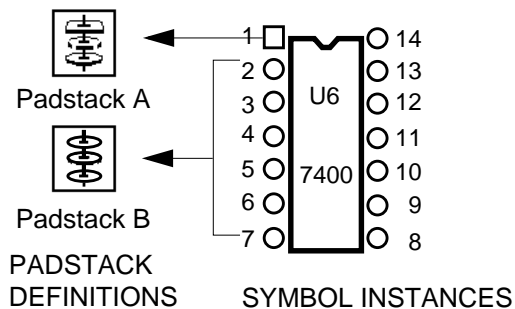
The first time Allegro PCB Editor uses a library padstack in a design, Allegro PCB Editor copies the generic pad data for internal layers from the library to all internal layers defined for the design. Also, as you add new ETCH subclasses to a design, Allegro PCB Editor adds the generic pad data for internal layers to each padstack for each new layer.

## Layout padstacks

A layout padstack is a padstack definition associated with a pin or via in a design.

Once a design contains padstack definitions, those padstacks are considered layout padstacks. Choose *Tools – Padstack – Modify Design Padstack* ([padeeditdb](#) command) to define different sets of pad data for internal layers in layout padstacks. The menu item and command are described in the *Allegro PCB and Package Physical Layout Command Reference*.

Pins share layout padstack definitions. All pins with the same padstack name refer to the same padstack definition stored in the layout, as the following figure shows:



## Padstacks and Pins

In Allegro PCB Editor, each pin in a symbol must have an associated padstack name. When you create a symbol, you add pins to the package symbol drawing. As you add each pin, Allegro PCB Editor finds the library padstack, copies the padstack definition into the symbol drawing, and displays the padstack graphics. For this reason, you must define library padstacks before creating package symbols.





When you create a symbol, Allegro PCB Editor stores the padstack name for each pin in the library symbol, but not the actual padstack data. When you add a symbol to a design for the first time, Allegro PCB Editor copies the padstack data from the padstack library and the symbol data from the symbol library.

Allegro PCB Editor locates the padstack library by using the padstack library path (PADPATH) and the package symbol library by using the symbol library path (PSMPATH) set in the global or local environment file.

Once a padstack has at least one instance in a design, Allegro PCB Editor makes all subsequent references to that padstack in the design and not the padstack in the library.


## Standard Pad Shapes

Allegro PCB Editor displays pads using standard geometric shapes. On the Layers tab of the Padstack Designer, you can choose from the pad types illustrated below in photoplots.

Pad Type	Description	Graphic Displayed
Regular	<p>Positive pads (black) with a regular shape (circle, square, rectangle, oblong, octagon), flashed on positive layers only. Through hole pads require regular pad geometries be defined for every board layer. Surface mount pads require only Top and related Top mask information. Non-standard shaped pads are available as pad shapes and as pad flashes. If you are using your own custom shape, choose shape, which is used for any definition that is not a circle, a square, a rectangle, oblong or an octagon. A Shape symbol for the geometry of the pad must be created manually using the Symbol Editor.</p> <ul style="list-style-type: none"> <li>■ Null</li> <li>■ Circle</li> <li>■ Square</li> <li>■ Rectangle</li> <li>■ Oblong</li> <li>■ Octagon</li> </ul>	
Thermal relief	<p>Used instead of regular pads to reduce the amount of heat absorbed through connect lines or shapes during the manufacturing process. Thermal reliefs can be:</p> <ul style="list-style-type: none"> <li>■ A negative pad on a positive copper area (shape). The thermal relief may be plotted as a regular pad flash (circle) with two lines.</li> <li>■ A positive pad on an embedded metal layer that distributes a voltage, such as power or ground.</li> </ul>	 
Anti-pads	Negative pads (clear, surrounded by black), usually a circle, to prevent connection of a pin to an embedded metal layer.	

## Allegro PCB Editor User Guide: Defining and Developing Libraries

### Library Padstacks

Pad Type	Description	Graphic Displayed
Shapes	Custom pads created as symbols and added to a padstack either upon initial creation or when editing a design.	
Flash	User-defined name of aperture for Gerber flashing of unique pad shapes. See the <i>Defining and Developing Libraries</i> user guide in your documentation set, for more information on conversion methodologies. You can use the Symbol Editor's Add – Flash command to aid you in creating the negative thermal relief flash. You specify the inner and outer diameter sizes, the spoke width, the number of spokes, and the spoke angle. The center dot section can be used to create a filled circle that will graphically locate the center point of the flash. A thermal relief is created as a series of filled shapes located on the class Etch, subclass Top. You do not have to use the Add – Flash command when creating your thermal relief. You can manually draw any number, size and shape of filled shapes. Be sure to create all graphics on the class Etch, subclass Top.	

Each pin can have any pad type (regular, thermal relief, anti-pads, and custom shapes) defined on each ETCH layer of the design. For negative artwork layers, Allegro PCB Editor uses thermal reliefs and anti-pads. For positive artwork layers, Allegro PCB Editor uses just three regular pads. This means a pin can be put anywhere on the design, and Allegro PCB Editor photoplots the correct pad, whether the location is in an open area or inside a filled shape.

### Photoplot Pad Data

When you create plots, Allegro PCB Editor uses the padstack information to write the photoplot pad data for a pin on a particular layer. Allegro PCB Editor determines the pad types for photoplotting as follows:

#### How Allegro PCB Editor Determines Pad Types for Photoplotting

If the Pin Is...	Uses Pad Type...
Not in a filled area on a layer being photoplotted, such as a shape or rectangle	Regular pad on positive artwork

## How Allegro PCB Editor Determines Pad Types for Photoplotting

---

If the Pin Is...	Uses Pad Type...
Inside a filled area on a layer being photoplotted and the pin shares the same net as the filled area	A thermal relief pad created by positive artwork, or a thermal relief pad drawn by positive artwork
Inside a filled area on a layer being photoplotted, but the pin does not share the same net as the filled area	An anti-pad created by positive and negative artwork

---

**Note:** You can enable the suppression of unconnected internal pads during plotting in the Padstack Designer.

## Defining Library Padstacks

You define new library padstacks using the [Padstack Designer](#), which you open using the [pad\\_designer](#) command, described in the *Allegro PCB and Package Physical Layout Command Reference*. Building new library padstacks comprises:

1. [Preparing to Define Padstacks](#), described below.
2. [Using the Padstack Designer](#) to create and save library padstacks.

### Preparing to Define Padstacks

Before you define new library padstacks:

- Check manufacturer specification sheets and verify design requirements.
- Gather the dimensions, physical data, logical data, manufacturing requirements, and documentation requirements for the padstacks you are defining.
- Check that the padstacks are not already in the Allegro PCB Editor library. To display a list of available library padstack definitions, choose *File – Open* in the Padstack Designer.

Use the following table as a guide to pad selection:

Pad Type	Interpretation
h109p	hole; 109 Mils; plated

## Allegro PCB Editor User Guide: Defining and Developing Libraries

### Library Padstacks

---

Pad Type	Interpretation
h109u	hole; 109 Mils; unplated
m43b	multi-bus connector; 43 Mils; BOTTOM mount
m43t	multi-bus connector; 43 Mils; TOP mount
p50c32	pad; 50 Mils; circle shape; 32 Mil drill size
p50s30	pad; 50 Mils; square shape; 30 Mil drill size
pga	pin grid array
s25_48	surface mount pad; 25 x 48 Mils pad size; TOP mount
s25_48b	surface mount pad; 25 x 48 Mils pad size; BOTTOM mount
sq55	pad; square shape; 55 Mils
via	---

- Determine photoplot requirements such as flash names, NC Drill data, and offset requirements.
- Create any custom (unique) pad shapes by choosing *Shape – Add Polygon* ([shape add](#) command), *Shape – Add Rectangle* ([shape add rect](#) command) or *Shape – Add Circle* ([shape add circle](#) command). See the *Preparing for Layout* user guide in your documentation set for additional information on using shapes.

## Using the Padstack Designer

After you have prepared all the data necessary to define padstacks, open the Padstack Designer to create padstacks and save them to a library. Use the [pad designer](#) command, described in the *Allegro PCB and Package Physical Layout Command Reference*.

### Types of Padstacks You Can Create

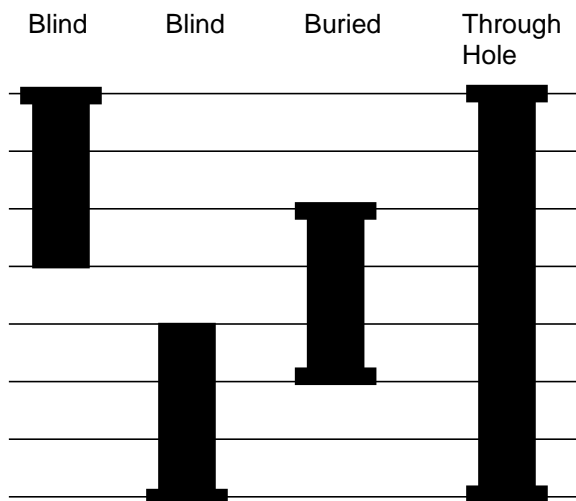
You can create different types of padstack in the Padstack Designer. The padstack type is determined by the Padstack Designer based upon the current pad geometries. As you make edits to pad geometries, the type is continuously evaluated and updated in the Padstack Designer.

There are five padstack types that are defined in the Padstack Designer:

- *Through* refers to a padstack with pads on all ETCH layers.
- *Blind/Buried* indicates the padstack is blind (between the surface and internal layers) or buried (between internal layers).

You can also specify Microvia, an option for Blind/Buried Vias, for use in high density interconnect boards and packages. Once placed, design rules examine touch (via tangency) and coincidence (via stacking) of the padstack's pads and other design objects.

- *Single* indicates the padstack has pads on only one ETCH layer.
- *Mask* refers to a padstack that only contains pad definitions for the mask layers and no drill hole is defined.
- *Undefined* refers to a padstack where neither pads nor drill hole is defined.





### Internal Layers

You can choose to enable the suppression of all unconnected pads on internal layers, as defined on the *Layers* tab, by selecting *Allow Suppression of Unconnected Internal Pads* on the *Parameters* tab. If this option is not selected, you cannot suppress any pads during photoplotting.

### Units

You should choose the units and accuracy to suit your design environment, because all fields that use measurement refer to this value. Remember that these units are converted to board units and accuracy, which may result in rounding when you apply the units to a design. For more procedural information about units, refer to the [Padstack Designer](#), described in the *Allegro PCB and Package Physical Layout Command Reference*.

### Drill Information

Specify your drill hole and drill symbol information before generating drill drawings. The Multiple drill section of the Padstack Designer includes a *Staggered* option so that you can apply separate values to the X,Y Clearance fields.

The drill figure and characters do not display when a padstack initially loads to a design. Slots are represented as objects in the Allegro PCB Editor database. Choose from either an oval or rectangular option. A slot figure and its respective x and y dimension are proportionally scaled to the actual slot size. A slot report, generated using the Tools - Reports menu command, lists slot type, location, and rotation in CSV/HTML format. NC Route, if executed, detects slots on the board and writes them out to the `<design>.rou` file.

To associate positive and negative drill tolerance to any plated or non-plated slot or circle drill, enter values directly to the padstack symbol. Tolerance information appears in the `.drl` file. The *Drill Character Field* associated with the drill/slot symbol has been increased from one to three places.

Non-standard drill options for circle drills include *Laser*, *Plasma*, *Punch*, *Wet/dry etching*, *Photo Imaging*, *Conductive Ink Formation*, or *Other*. This effectively separates non-standard drills into a separate `.drl` file, making it easier for the fabricator to process incoming data from OEMs. For more procedural information about drill-related fields, refer to the [Padstack Designer](#), described in the *Allegro PCB and Package Physical Layout Command Reference*.

## Reports

A series of reports are available in Padstack Designer to help you evaluate padstack information. *Reports – Padstack Summary* displays a summary of the current padstack data. *Reports – Show Instances* Displays all the pins and vias that use the current padstack. *Reports – Library Drill Report* generates a read-only padstack spreadsheet and associated drill information based on the PADPATH setting. For more procedural information about padstack-related reports, refer to the [Padstack Designer](#), described in the *Allegro PCB and Package Physical Layout Command Reference*.

## Layers Tab

The Layers tab of the Padstack Designer defines both the number of layers in the padstack and the individual layer type. When you define a padstack for your library, you only see the layers that you define in the table, but when you open a padstack from the design, you see all design layers related to that padstack.

The top section displays all padstack ETCH layers with the corresponding *Regular Pad*, *Thermal Relief*, and *Anti Pad* definitions. Mask layers are also displayed here, but mask layers are restricted to Regular Pad definitions only. A tabular format represents padstack layers. In library padstacks, you can change the padstack name in the Layer column of any highlighted layer. You cannot change the layer name in a layout padstack. You can edit any of the white fields and Insert, and Delete, Copy, and Paste layer definitions.

The top section also contains the "Single Layer Mode" checkbox. Selecting this checkbox results in the display of only the first etch layer with pads defined on it. Unselecting this checkbox will restore the full etch layer set. For designs with many etch layers, selecting this checkbox allows access to the mask layers without scrolling down through the etch layers.

### Important

Saving or updating a padstack in Single mode will delete pad definitions on all etch layers other than the single one displayed. If there are pads that will be deleted, a confirmer is issued to allow the user to cancel the operation.

The bottom section displays the pad definition areas for *Regular Pad*, *Thermal Relief*, and *Anti Pad*. You do not need to fill in all of the options to save the padstack. If you want to save a padstack file, you must define a pad for at least one layer. For more procedural information about defining layers, refer to the [Padstack Designer](#), described in the *Allegro PCB and Package Physical Layout Command Reference*.

## Editing padstacks

Once a design contains padstack definitions, those padstacks are considered layout padstacks. To modify padstacks within the design, use *Tools – Padstack – Modify Design Padstack* (padeditdb command). Use *Tools – Padstack – Modify Library Padstack* (padeditlib command) to modify a padstack from your library.

Refer to the *Allegro PCB and Package Physical Layout Command Reference* for procedural information on modifying padstacks.

## Recording a Padstack Script

If you have padstacks that share similar specifications, you can automate the process of entering padstack data by choosing *File – Script* in the Padstack Designer.

The script feature lets you record the entries you make on the Padstack Designer in a script file. When you want to define new padstacks that share similar specifications, you can replay the script file and edit the new padstacks as necessary (to assign new padstack names, make a few changes to the padstack specifications, and so on).

## Updating Layout Padstacks

To ensure that a design uses the latest version of the padstacks in the library, Allegro PCB Editor lets you refresh layout padstacks. You can do this interactively by choosing *Tools – Padstack – Refresh* (refresh padstack command) or by running the refresh padstack batch command described in the *Allegro PCB and Package Physical Layout Command Reference*.

## Creating a Padstack List File

When you refresh padstacks, you can either refresh all the padstacks in a design or just the ones in a padstack list file.

List files are ASCII text files that end in the `.lst` extension and contain the names of padstacks that can be updated. Use a text editor to create the padstack list file, and place the file in the current working directory.

The following file format conventions apply:

- Provide only one padstack name on each line.

- Use either uppercase or lowercase letters. Allegro PCB Editor always stores padstack names in uppercase but can read mixed case in this file.
- Do not include leading or trailing white space. Allegro PCB Editor removes it if it appears in the file.

### Sample Padstack List File

This is a sample padstack list file:

```
pad93cir58d  
soj  
via  
smd25-94pf
```

### Reviewing the Refresh\_Padstack Log File

The `refresh_padstack.log` file records the refresh padstack processing.

## Custom Pad Shape Symbols

You can create custom pad symbols that have a non-standard geometric shape. You can then place these custom pads in the symbol library and add them to any padstack, either when you create the padstack or when you edit the padstack in a design.

As with all pads, you can control the display of custom pads by choosing *Display – Status* (`status` command) and selecting *Filled pads and cline endcaps* if using UNIX, or *Filled pads* on Windows to toggle custom pads from solid filled shapes to outlines.

### How Custom Pad Shape Symbols Work

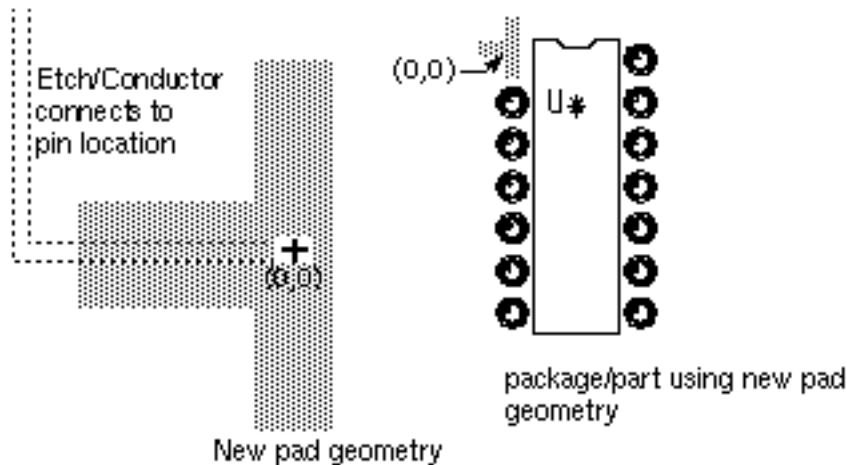
You create a custom pad shape symbol by choosing *Shape – Add Polygon* (`shape add` command), *Shape – Add Rectangle* (`shape add rect` command) or *Shape – Add Circle* (`shape add circle` command) in Allegro PCB Editor. A pad shape becomes a symbol (`.ssm`) file. As with any other symbol, the (0,0) point of the symbol drawing becomes the symbol origin.

## Allegro PCB Editor User Guide: Defining and Developing Libraries

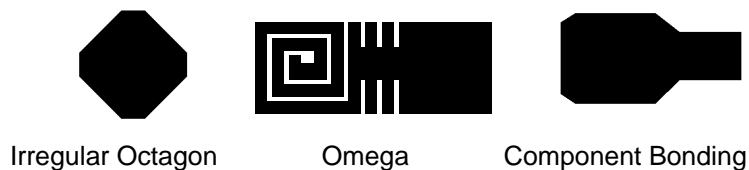
### Library Padstacks

---

When you use the pad shape in a design, the shape is placed as a pad, and the 0,0 location of the shape drawing is centered on the pin or via, as follows:

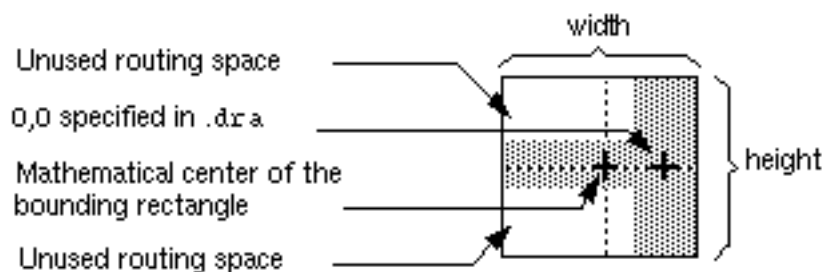


The following are sample custom pad shapes you can define:



The size of most custom pads consists of the extents of the shape. Allegro PCB Editor calculates the offset values (the difference between the mathematical center of the bounding box and the 0,0 location specified in the drawing) and automatically offsets the pad when placing the shape.

For example:



The router cannot route through white space areas. DRC, penplots, and artwork process the actual geometry of the shape.

## Characteristics of Custom Pad Shapes

These are the characteristics of custom pad shapes:

- You can draw a custom pad shape on any ETCH subclass because the padstack controls the actual class/subclass.
- A custom pad shape should consist of only one shape; do not put multiple shapes in the pad shape drawing.
- Custom pad shapes cannot have voids.
- Custom pad shapes are always filled solid.

## Updating a Library Padstack in a Symbol

If you update a library padstack and you want it to be used for pins in a package or mechanical symbol that uses the padstack, you must update the padstack in the symbol.

## Updating a Layout Padstack

You can update padstack definitions in symbols that populate a layout in these ways:

- Edit the padstack definition by choosing *Tools – Padstack – Modify Library Padstack* (padeditlib command), described in the *Allegro PCB and Package Physical Layout Command Reference*.
- Replace all occurrences of a padstack definition in designated symbols in a layout by choosing *Tools – Padstack – Replace* (replace padstack command), described in the *Allegro PCB and Package Physical Layout Command Reference*. For example, you may want to update a class of package symbols in a layout with the updated library padstack definitions.

## Suppressing Unused Padstacks

Suppresses inner-layer pins or vias that are unconnected to a cline or plane on layers you specify using *Setup – Unused Pads Suppression* (unused pads command). Pads on the top or bottom layer are never considered unused, even if they lack connections, nor are outer layers of a blind/buried via instance, which are preserved. Mechanical pins are ignored by pad suppression as are pins and vias with Fixed internal layer padstacks in the Padstack Editor. Pads are added when a connection occurs to a pin or via, and removed when the connection is deleted. Pads cannot be suppressed on any layer that requires negative artwork, as the pad is required to create a negative artwork void.

When all pads of a pin or via padstack are suppressed for the currently visible etch layers of the display, any hole associated with the padstack displays by default using the currently defined color, even if the display of plated and/or non-plated holes is disabled on the *Display* tab of the *Design Parameter Editor*. This occurs by default when the enabling of *Dynamic unused pads suppression* also enables *Display padless holes* by default. If this default display of the hole when pads are suppressed on all visible etch layers of the display is not desired, *Display padless holes* must be explicitly disabled.

To prevent the pads of certain symbols, nets, pins, or vias from being removed, use the `UNUSED_PADS_IGNORE` property. The property can be applied to the following elements:

- Net: All pads on all pins and vias on the specified net are retained.
- Pin: All pads on the specific pin are retained.
- Via: All pads on the specific via are retained.
- Symbol: All pads of all pins and vias on the specific symbol are retained.

The *Suppress Unconnected Pads* option available with the *Manufacture – Artwork* (film param command), can also be used on a per artwork film basis. This option is greyed out if you launch the Artwork form after enabling pad suppression using *Setup – Unused Pads Suppression*. When you enable dynamic suppression for the entire design using *Setup – Unused Pads Suppression*, it takes precedence, even if *Suppress Unconnected Pads* option on the Artwork form is disabled.

## Purging Unused Padstacks

Allegro PCB Editor lets you remove unused padstacks from the list of available padstacks for a design. You can purge all unused padstacks or all derived padstacks by choosing *Tools – Padstack – Modify Library Padstack* (padeditdb command). The menu item and command are described in the *Allegro PCB and Package Physical Layout Command Reference*.

Derived padstacks are those you create by modifying existing padstacks. You may have unused derived padstacks if you restore derived pads to their original state. The derived padstack names are listed as available padstacks in the *Padstack Selection* dialog box until you purge them.



---

## Working with Symbols

---

This chapter provides information on package, mechanical, and format symbols. As with padstack files, Allegro PCB Editor supplies a symbol library containing symbols that you can copy into designs or that you can modify for specific needs. You can also create custom symbols. Creating symbols, while not tied directly to any specific part of a design flow, normally occurs at the beginning of the design process.

### Working with the symbol mode

You work in the symbol mode to create and modify symbols. You can enter symbol editing mode in one of the following ways:

- Choose *File – Open* (open command) and choose a filename with a .dra extension.
- Choose *File – New* and choose one of the symbol drawing types from the *Drawing Type* list box in the *New Drawing* dialog box.

The Allegro PCB Editor workspace then changes from layout to symbol mode. The Symbol mode contains a subset of the layout mode's commands, and also has commands used exclusively with symbols.

### Symbol Types

Allegro PCB Editor symbols are categorized by type. All symbols are derived from a drawing file that is later converted into the appropriate symbol type. The following symbol types are available in Allegro PCB Editor:

- **Package**

Package symbols are the physical representation of electronic devices such as dual in-line packages (DIPs), edge connectors, resistors, capacitors, and transistors. Each package symbol contains pins (with pads and possible drill holes) that act as attachment points for connecting etch added during interactive or automatic routing.

- **Mechanical**

Mechanical elements in a design include items such as tooling holes, outlines, and card ejectors. Since these elements are non-electrical, they contain no reference designators or pin numbers. Mechanical elements can also include keepin or keepout areas for routing, packages, and vias.

- Flash

Flash symbols are pads that you create for photoplotting using standard apertures.

- Format

Format symbols consist of manufacturing drawing formats such as sheet drawings in various sizes, and corporate legends and logos.

- Padstacks

Padstacks are also symbols; however, padstacks have their own editor you use to create and modify padstacks.

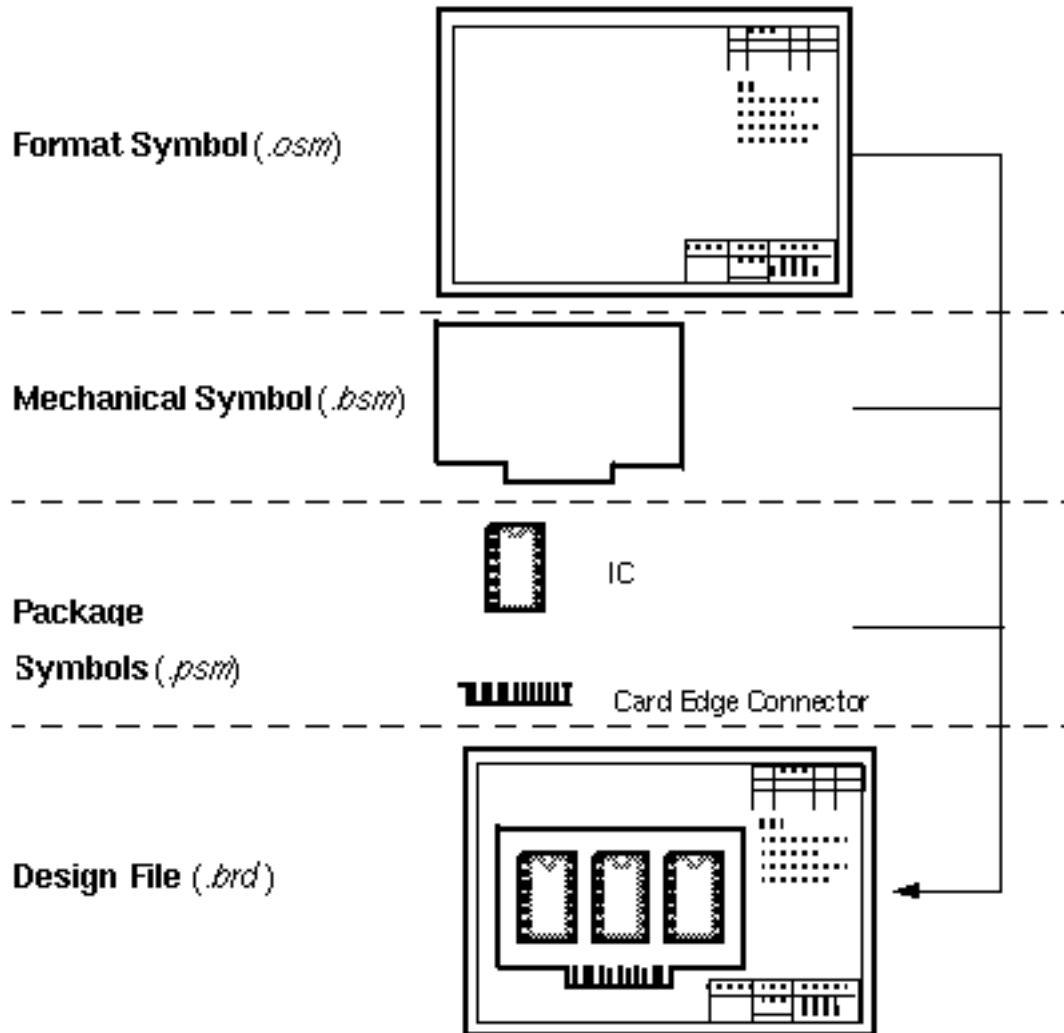
- Shapes

Shapes are another type of symbol; however, you create them by choosing *Shape – Add Polygon* (shape add command), *Shape – Add Rectangle* (shape add rect command) or *Shape – Add Circle* (shape add circle command) and then edit by choosing *Shape – Select Shape or Void* (shape select command). The symbol mode does not support dynamic shapes. To edit traditional static shapes or voids, use commands available from the *Shape – Manual – Void* menu (shape void polygon, shape void circle, shape void rectangle, shape void copy, shape void move, or shape void delete commands).

See the *Preparing for Layout* user guide in your documentation set for additional information on using shapes.

See Figure 3-1 for an example of packagepart, mechanical, and format symbols.

**Figure 3-1 Symbols and their Relation to the Design File**



## Symbol and Drawing Files

A symbol in Allegro PCB Editor comprises two files:

- The drawing file ( *.dra* )
- The symbol file ( *.\*sm* )

The drawing file is the result of all the commands that you choose from the *Add* and *Edit* menus of the symbol mode. Additionally, you choose commands from the Layout menu to add

## Allegro PCB Editor User Guide: Defining and Developing Libraries

### Working with Symbols

---

a layer of electrical characteristics to the drawing, in the form of connections, pins, labels, and constraints.

When you first create a new drawing file for a symbol, you specify its type (package, mechanical, and so forth). After editing the file, you convert it into a symbol with one of these extensions:

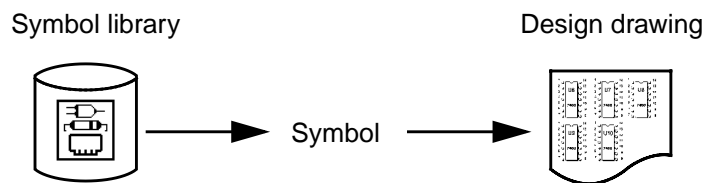
Symbol	Extension
Package	.psm
Mechanical	.bsm
Format	.osm
Shape	.ssm
Flash	.fsm

Allegro PCB Editor automatically creates a symbol (\*.sm file) every time you save a drawing (.dra) when you are in the symbol mode. You no longer need to compile the symbol and save the drawing in two separate steps.

Set the environment variable `no_symbol_onsave` to restore the legacy behavior and allow Allegro PCB Editor to compile the symbol and save the drawing in two steps. For additional information about setting environment variables, see [envved](#).

### The Symbol Library

After creating a symbol, save it in a library to reuse it.



Once you load a library symbol into a design, Allegro PCB Editor uses that symbol definition for future instances of that symbol. If that symbol does not exist in a design, Allegro PCB Editor looks for it in the library.

## Legal Classes for Symbol Types

The following are legal classes for symbols:

	<b>Package</b>	<b>Mechanical</b>	<b>Shape</b>	<b>Flash</b>	<b>Format</b>
BOARD_GEOMETRY	yes	yes	no	no	yes
COMPONENT_VALUE	yes	yes	no	no	no
DEVICE_TYPE	yes	yes	no	no	no
DRAWING_FORMAT	no	no	no	no	yes
ETCH	yes	yes	yes	yes	no
MANUFACTURING	yes	yes	no	no	yes
PACKAGE_GEOMETRY	yes	yes	no	no	no
PACKAGE_KEEPPIN	no	yes	no	no	no
PACKAGE_KEEPOUT	no	yes	no	no	no
PIN	yes	yes	no	no	no
REFERENCE_DESIGNATOR	yes	yes	no	no	no
ROUTE_KEEPPIN	no	yes	no	no	no
ROUTE_KEEPOUT	yes	yes	no	no	no
TOLERANCE	yes	yes	no	no	no
USER_PART_NUMBER	yes	yes	no	no	no
VIA_CLASS	yes	yes	no	no	no
VIA_KEEPOUT	yes	yes	no	no	no

## Creating a Symbol

Open a new file of the desired symbol type (*Package*, *Mechanical*, *Format*, *Shape*, or *Flash*). Details for creating each type of symbol appear in the rest of this chapter.

**Note:** The Symbol mode inherits the dimensions of the associated design. You can specify different drawing parameters by choosing *Setup – Drawing Size* (drawing param command), described in the *Allegro PCB and Package Physical Layout Command Reference*.

When you have created the symbol, choose *File – Create Symbol* (create symbol command) in Symbol mode to convert the active drawing into a symbol. This compiles the

artwork of a drawing, together with any electrical attributes that you specified into a binary file. The menu item and command are described in the *Allegro PCB and Package Physical Layout Command Reference*.

## Adding Areas

Areas for symbols include route, via, and place keepouts and keepins, as well as place-bound rectangles, used to delineate a package boundary that specifies different height constraints of the placement area beneath components.

Keepouts demarcate an area on the symbol in which to avoid placing etch, vias, or other symbols. Keepins demarcate an area on the symbol in which to guide (restrict) the placement of etch, vias, or other symbols. Menu items and commands for areas include:

- *Setup – Areas – Route Keepin* (keepin router command)
- *Setup – Areas – Route Keepout* (keepout router command)
- *Setup – Areas – Package Keepin* (keepin package command)
- *Setup – Areas – Package Keepout* (keepout package command)
- *Setup – Areas – Via Keepout* (keepout via command)
- *Setup – Areas – Probe Keepout* (keepout probe command)
- *Setup – Areas – Package Boundary* (package bound command in the symbol mode to create place-bound rectangles)

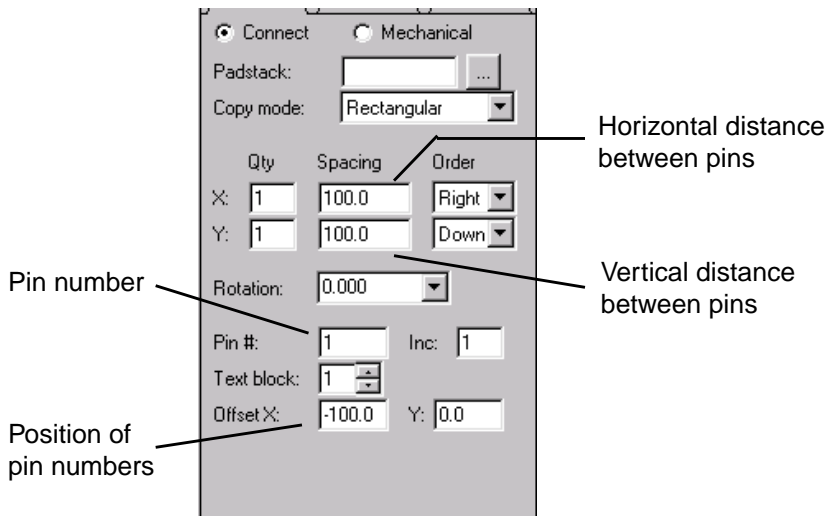
These menu items and commands are described in the *Allegro PCB and Package Physical Layout Command Reference*.

**Note:** Allegro PCB Editor automatically chooses the correct class/subclass based on the command that you choose.

## Adding Pins

You can add pins to package and mechanical symbols. Package symbol pins have pin numbers; mechanical symbol pins do not.

From the symbol mode, choose *Layout – Pins* (add pin command) to initiate pin placement via parameters that you specify in the Options tab.



## ETCH and Vias in Symbols

You can add shapes, etch, and vias to a symbol drawing. When you add a symbol to a design, any etch and via elements in the symbol are added as well. However, you should consider the following when adding symbol elements to a design:

- To delete a layer from a design that has package symbols with an item, such as etch, defined on that layer (subclass), delete the items defined on that layer before deleting the layer.
- Edit items not associated with a symbol either separately or move them through a group window operation.
- Etch text/lines, circles, arcs (unlike connect lines, these elements can never be connected to or associated with any nets) remain associated with their symbols.

You can always move, copy, or delete etch text/lines, circles, and arcs with the symbol. Etch text/lines, circles, and arcs are intended as constructs for etch labeling, as an alternative to silkscreen.

**Note:** When used for electrical connectivity, these constructs may lead to problems with

the ratsnest display and DRC.

- The pin padstack layers remain part of the pin, and the pin always remains a part of the symbol.
- If a user-definable subclass exists in the symbol, you do not need to define the subclass in the design in order for the symbol to be added to the design.

When you add the symbol to the design, the subclass is automatically added to the design.

- You can create mirrored pairs of user-defined subclasses.

The graphics in these subclasses are interchanged whenever the symbol is mirrored.

These subclass pairs must be in the same class and are identified with identical names, except for the words TOP and BOTTOM. For example, a PACKAGE subclass named SPECIAL\_DATA\_TOP is moved to SPECIAL\_DATA\_BOTTOM. Likewise, the graphics in SPECIAL\_DATA\_BOTTOM are moved to SPECIAL\_DATA\_TOP.

## Creating Package Symbols

Package symbols physically represent electronic devices in a design. They must correspond to the logic data loaded in the design database—pin numbers must match, and reference designators must exist. They can also carry manufacturing data, such as company part numbers, component values, and tolerances.

**Note:** Chapter 8, “[Package Symbol Library](#),” shows the package symbols available in the Allegro PCB Editor library.

## Package Symbol Elements

Packages must have the following elements:

- At least one pin

Pins are connect points with associated pad geometry and pin number. Pins cannot be edited (moved or spun) unless the UNFIXED\_PINS property has been attached to the symbol or the design in which the symbol resides.

- Component outline

A component outline is a geometric representation of the component and can consist of lines, arcs, circles, text, and notes.

- Reference designator



A reference designator is text that identifies a device in a design.

- At least one place-bound rectangle (user-defined or system default)

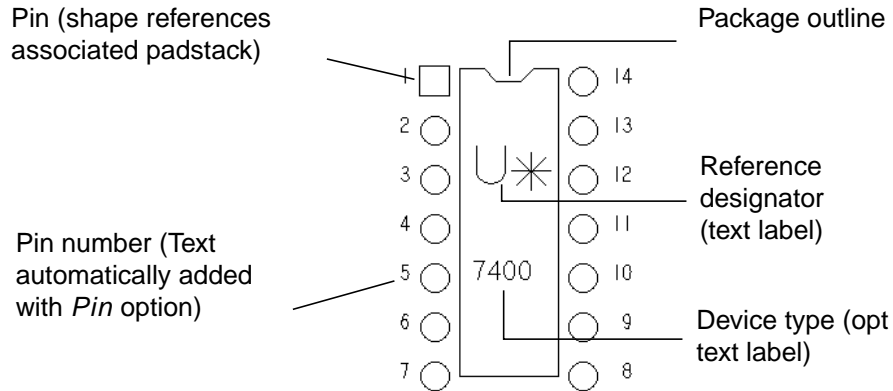
Place-bound rectangles are filled rectangles that define the package part boundary and govern placement restrictions. Placement tools use these rectangles for overlapping and mechanical restraints. DRC also uses them to check for violations of package-to-package keepin areas and keepout areas.

The following is a list of other elements that you can add to a package symbol during the symbol building process:

- Device type text (for the component device type)
- Component value (text for the component value)
- Tolerance (text for the component tolerance)
- Component height (text for the physical height of the component)
- Silkscreen information (text and lines for information on silkscreen layer of drawing)
- User part number (text for the package part number)
- Route keepout shape(s) identifying areas where etch is not allowed
- Via keepout shape(s) identifying via keepout areas
- Etch (etch lines, arcs, rectangles, shapes, and/or text added to the symbol)
- Vias

Figure [3-2](#) shows a DIP14 package symbol drawing, which identifies the class of each element in the drawing. This drawing does not contain all possible package classes.

**Figure 3-2 DIP 14 Package Symbol Drawing**



## Prerequisites for Creating a Package Symbol

Check the symbol library to see if the package symbol exists. If you need to create a package symbol, you should adhere to your company's standards for naming conventions, pad sizes, component origins and orientation, as well as any specific manufacturing criteria such as silkscreen and auto-insertion requirements. For each component, you should have the following physical data:

- Component body size
- Component origin
- Drill size
- Insertion machine clearances
- Pad size
- Pin-to-pin spacing

Create the library padstacks associated with the pins in the components using the Padstack Designer, which you open using the pad designer command, described in the *Allegro PCB and Package Physical Layout Command Reference*.

## Guidelines for Creating Package Symbols

When you create a package symbol:

- Build package symbols on the TOP of the drawing.

Choose *Edit – Mirror* (mirror command) to place symbols on the bottom side of the design during placement. which causes symbols to be mirrored as they are added to a design.

- Set up the text for titling and marking up the symbol.

Choose *Setup – Design Parameters* (prmed command) and choose the *Text* tab of the Design Parameter Editor to specify in the *Text Setup* dialog box the characteristics of each type of text (called text blocks) you can use in the design. Then choose one of these text blocks as the text default along with how text will be justified and the size of markers.

- If you are using automatic placement, you use the same origin in all the package symbols.

The symbol origin determines how the package symbol is physically located in the design. For example, if the origin of the symbol is the center of the component, the pick location tells Allegro PCB Editor to place the component center on that grid point. On the other hand, if the symbol origin is pin 1 of the component, the component is positioned so that pin 1 is at the location picked. Automatic placement uses the same origin to place all components during a placement session.

All commands listed here are described in the *Allegro PCB and Package Physical Layout Command Reference*.

## Defining a Package Symbol

You can create a package symbol in any of these ways:

- Choose *File – New* and choose Package Symbol wizard or run the package\_symbol\_wizard command to create a basic symbol using the wizard. Details about the wizard and instructions for creating a symbol this way appear in the command's description in the *Allegro PCB and Package Physical Layout Command Reference*.
- Manually, as described in Creating a Package Symbol Manually in the *Allegro PCB and Package Physical Layout Command Reference*.
- In batch mode, as described for the create\_sym batch command in the *Allegro PCB and Package Physical Layout Command Reference*.

## Defining Symbol Heights

Height restrictions allow you to limit the boundaries of a package and define the dimensions of its keepout areas.

- You define the 3-D geometry of package boundaries using place-bound rectangles. You can use multiple place-bound rectangles to define different height constraints that describe placement space available under component bodies. You might need multiple rectangles to better define complex, asymmetrical symbols or to maximize placement space.
- A keepout is a restricted area. Height keepout areas allow package symbols whose height is below a minimum or above a maximum to be placed in that area. For example, if the height range of a keepout is 500 to infinity, package symbols whose height is less than or equal to 500 can be placed in it.

### Package Height

The following describes package height values for a package keepout area and a place-bound rectangle.

Package keepout area:

- ❑ If a package keepout has no height value, then it means that the value of PACKAGE\_HEIGHT\_MIN is 0 and the value of PACKAGE\_HEIGHT\_MAX is INFINITY. Packages with heights in this range (the default setting) produce a DRC error.
- ❑ If a package keepout has only a minimum height value of MIN, this means that the value of PACKAGE\_HEIGHT\_MIN is MIN and the value of PACKAGE\_HEIGHT\_MAX is INFINITY and that the keepout flags a DRC if the package height falls within this 3-D space (MIN to INFINITY).
- ❑ If a package keepout has only a maximum height value of MAX, this means that PACKAGE\_HEIGHT\_MIN is 0 and PACKAGE\_HEIGHT\_MAX is MAX and that the keepout flags a DRC if the package height falls within this 3-D space (0 to MAX).
- ❑ If a package keepout has both MIN and MAX height values, then it means that PACKAGE\_HEIGHT\_MIN equals MIN and PACKAGE\_HEIGHT\_MAX equals MAX and that the keepout flags a DRC if the package height falls within this 3-D space (MIN to MAX).

Place-bound rectangle:

- ❑ If a place-bound rectangle has a height value of MAX, this means that values of PACKAGE\_HEIGHT\_MIN is 0 and PACKAGE\_HEIGHT\_MAX is MAX and that the package occupies the 3-D space (0 to MAX).
- ❑ If a place-bound rectangle has both MIN and MAX height values, this means that it occupies the space between the value of MIN for PACKAGE\_HEIGHT\_MIN and the value of MAX for PACKAGE\_HEIGHT\_MAX, which is in the 3-D space (MIN to MAX).

Typically when you specify MIN and MAX values for a place-bound rectangle, the symbol includes multiple place-bound rectangles with different restrictions.

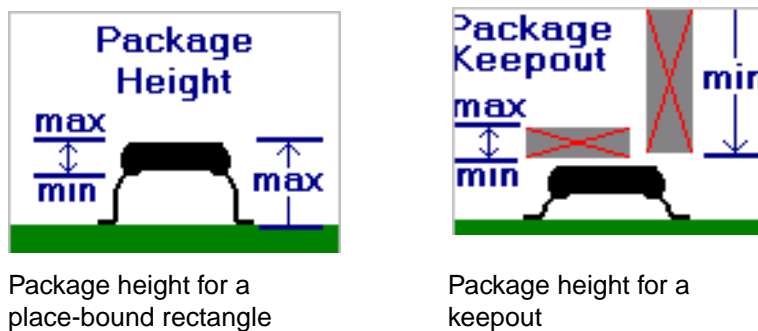
### Package Height Example

For example, if the package keepout PACKAGE\_HEIGHT\_MAX is 1.7 MM and for the place bound PACKAGE\_HEIGHT\_MAX is 0.61 MM, then the exclusion area for the keepout is (0,1.7 MM) and the occupied area for the package is (0, 0.61 MM). Therefore, the package invades the exclusion area of the keepout and thus generates a DRC

Figure 3-3 distinguishes between package height and package keepout height, as displayed in graphics that appear in the *Options* tab when you run the `package_height` command from the layout or symbol modes.

**Note:** The objects of measurement depend on the chosen class.

**Figure 3-3 Graphic Representations of `package_height` Commands**



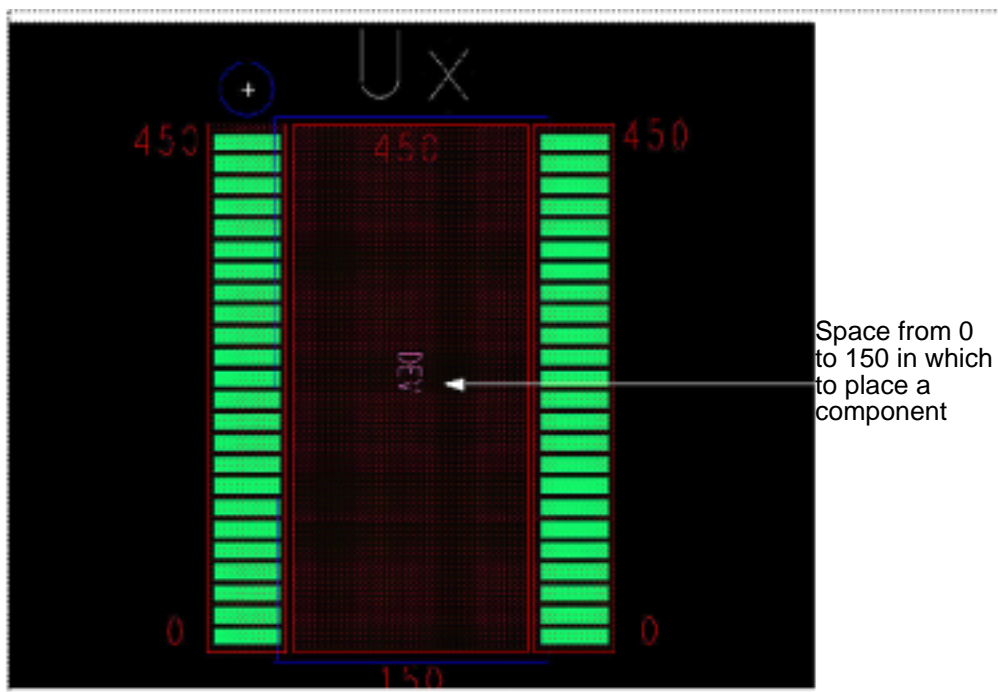
You can use the `package_height` command to update height restrictions on existing objects or to add new areas. The command defaults to updating heights on existing areas but, by choosing either *Add Rectangle* or *Add Shape* from the pop-up menu, you can add new areas. Choosing these options has the same effect as running the `add rect` or the `add fshape`, `shape add`, and `add xshape` commands, after which you are prompted to add the appropriate height property.

**Note:** When drawing a component (other than a die) the Cadence 3D Viewer uses the place bound (or assembly bound) to draw the symbol's outline and the minimum and maximum height properties to give the component thickness above the substrate.

### Multiple Place-bound Rectangles

Figure 3-4 shows how you use three place-bound rectangles (shown in grey) to define the height of a symbol. Two place-bound rectangles define the height parameters (0 to 450) of the pins. A third place-bound rectangle defines the height range of the body of the component (150 to 450). Since the body begins only at 150, there is space under this component from 0 to 150 where a smaller component can be placed.

**Figure 3-4 Component Symbol with Three Place-Bound Rectangles**



Symbol height is the z-dimension (depth) of the package symbol. By default, Allegro PCB Editor sets the maximum height for all unlabeled package symbols to 150 mils.

To ensure that the unit measurements assigned to symbols do not conflict with the unit measurements on the design or module, package symbol height in Allegro PCB Editor versions 14.0 and later is stored in the database as representations of two properties, PACKAGE\_HEIGHT\_MIN and PACKAGE\_HEIGHT\_MAX. You can attach these properties to frectangle and shape objects on class/subclass:

- PACKAGE\_KEEPOUT/ANY (ALL, TOP, or BOTTOM)
- PACKAGE\_GEOMETRY/PLACE\_BOUND\_BOTTOM
- PACKAGE\_GEOMETRY/PLACE\_BOUND\_TOP

If you attempt to place objects containing the package height properties on other class/subclass layers, an error message appears in the status window.

### **Important**

Previous to version 14.0, package height constraints were stored as attached text, causing problems when changing design units and loading symbols whose height constraint design units were dissimilar from those of the rest of the design.

Associating package heights with properties eliminates such problems by updating the package height values when the overall design units are changed and by converting the units of symbols you load to those of the design.

The manner in which package height restrictions are handled in versions prior to 14.0 databases (that is, by way of attached text) are converted to the new method (property definitions) when you:

- Open a pre-14.0 database in 14.0 (and subsequent versions)
- Place a symbol in a 14.0 or later database
- Uprev the database with the uprev batch command, described in the *Allegro PCB and Package Physical Layout Command Reference*.

## **Defining Component Heights with Properties**

The PACKAGE\_HEIGHT\_MIN, PACKAGE\_HEIGHT\_MAX, and HEIGHT properties are methods to define component heights, using either a symbol-driven or a Part Table File (PTF) package height approach. Both methods may be employed in the same design.

### **Symbol-Driven Package Height Model**

When multiple heights within the same symbol are required, assign the PACKAGE\_HEIGHT\_MIN and PACKAGE\_HEIGHT\_MAX properties to component definitions or to package keepout areas and place-bound shapes on the PLACE\_BOUND\_TOP or PLACE\_BOUND\_BOTTOM subclasses. To use this symbol (.psm) driven package height model, choose either *Setup – Areas – Package Height* (package\_height command) or *Edit – Properties* (property\_edit command).

A package symbol can have multiple place-bound shapes, and each of them can have a unique PACKAGE\_HEIGHT\_MIN or PACKAGE\_HEIGHT\_MAX value. A MAX value defines the height of the package, and the MIN value defines a clearance under a part of the package. Allegro also audits the PACKAGE\_HEIGHT\_MIN and MAX values when determining if component-to-component or component-to-keepout violations exist.

#### **If a property assigned to place-bound is: then...**

PACKAGE_HEIGHT_MAX	the symbol fills that area between the board surface and the maximum height specified in the property value
PACKAGE_HEIGHT_MIN	the symbol fills the area above the minimum height specified in the property value, and a clearance exists between the board surface and the minimum value.
PACKAGE_HEIGHT_MIN and PACKAGE_HEIGHT_MAX	the symbol fills that area above the minimum value and below the maximum value.

When evaluating the PACKAGE\_HEIGHT\_MIN or PACKAGE\_HEIGHT\_MAX properties, the first value found applies in the following order of precedence:

- Place-bound shape or rectangle
- Component definition
- Board default package height (MAX value only)

#### **PTF Package Height Model**

The HEIGHT property also controls package height and can be sourced from the Allegro Design Entry HDL Part Table File (PTF). For discrete parts, whose physical footprints are identical except for height variations due to multiple manufacturers, use the PTF package height model, which minimizes design disruption as front-end librarians may already be using this property for IDF support.

When creating the physical footprint, ensure that no PACKAGE\_HEIGHT\_MAX property is assigned to place-bound shapes. Only those symbols whose height is driven from the schematic require this change. (Any existing HEIGHT properties assigned to package symbols take precedence.)



## Allegro PCB Editor User Guide: Defining and Developing Libraries

### Working with Symbols

---

To allow the DRC system to use the component-definition HEIGHT property driven from the PTF, choose *File – Import– Logic (netin* command) to map the component-definition HEIGHT property currently used by the IDF interface to the PACKAGE\_HEIGHT\_MAX property on the component definition.

Values for the HEIGHT and the PACKAGE\_HEIGHT\_MAX properties may specified in any legal length unit: The HEIGHT property value is maintained in user units in the database; the PACKAGE\_HEIGHT\_MAX value is converted to the current board design units.

Because the HEIGHT property is defined as a component property in Allegro it may be passed forward to Allegro from an Allegro Design Entry HDL netlist. Its value cannot be changed in the Allegro database as it is device and netlist driven.

Define the HEIGHT property in one or more of the following locations. When the design is packaged, Packager XL applies the first HEIGHT value found in the following order of precedence.

- as a body property in the symbol definition
- in the part table as either a key or injected property
- the `chips.prt` file as a body property

However, the component may have only one HEIGHT property value. If the component's actual height is irregular, the varying heights of its profile cannot be described using a HEIGHT property, and component-to-component or component-to-package-keepout DRC audits ignore the HEIGHT property's value.

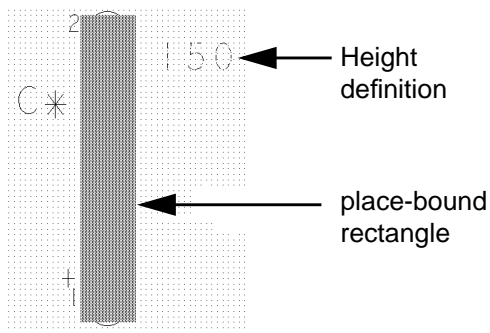
The `idf_out` utility passes a component's HEIGHT value when both of the following conditions are met:

- No PACKAGE\_HEIGHT\_MIN and/or PACKAGE\_HEIGHT\_MAX properties are attached to the place-bound shapes defined in the component instance's symbol
- The environment variable `idf_ignore_comp_height` is not set

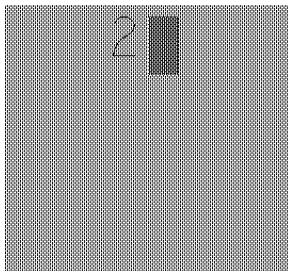
### Adding Symbol Height Text to a Place-Bound Rectangle

You can attach a symbol height label to a place-bound rectangle that specifies a height range, which consists of a maximum and minimum height value that defines the z-dimension (depth) of the package.

If you attach only one height value, Allegro PCB Editor assumes the specified value is the maximum height value and 0 is the minimum. Symbol height is only used with package symbols.



You can choose a location right on the rectangle, as shown below.



Location picks are shown in the status bar. You are automatically put into the attach text function mode, as shown in the status area. Allegro PCB Editor prompts you to enter a text string.

**Note:** If you want to change the symbol height text for the package bound after the component is placed, choose *Edit – Text* (text edit command), described in the *Allegro PCB and Package Physical Layout Command Reference*.

## Creating Mechanical Symbols

Mechanical symbols physically represent mechanical elements in a design.

**Note:** Chapter 9, “Mechanical Symbol Library,” shows the mechanical symbols in the Allegro PCB Editor library.

### Types of Mechanical Symbols

A mechanical symbol can be one of the following:

- An element relating to manufacturing, such as
  - ☐ Design outline
  - ☐ Tooling corners
  - ☐ Plating bars
  - ☐ Mounting holes
  - ☐ Dimensions
  - ☐ Areas (shapes) to be photoplotted
  - ☐ Areas not to be glossed
- Keepin and keepout areas
  - ☐ Package
  - ☐ Package
  - ☐ Route keepin
  - ☐ Route keepout
  - ☐ Via keepout
- Pins without pin numbers (for mounting holes)
- Non-connecting etch elements, such as logos on ETCH layers

You can create each mechanical element as a separate symbol and store the symbol in a library for use in different designs. You use *File – Create Symbol* ([create symbol](#) command) in the symbol mode to convert a mechanical symbol drawing .dra into a mechanical symbol binary file .bsm.

## Guidelines for Creating Mechanical Symbols

When you create a mechanical symbol, follow these guidelines:

- Create mechanical elements on their related classes:

BOARD	Includes elements relating to manufacturing: design outline, tooling corners, plating bar, dimensions, targets, and mounting holes. You create various geometries by placing basic element types, lines, arcs, text, rectangles, and shapes, into subclasses of the appropriate name.
ETCH	Non-connecting etch elements only. For example, to add a company logo on an etch layer.
PACKAGE	A single unfilled shape inside which you can place package symbols.
PACKAGE ETCH	Any number of filled shapes. Each shape defines an area where no package symbols may be placed.
ROUTE KEEPIN	A single unfilled shape inside which etch you can route.
ROUTE KEEPOUT	Shape identifying areas where etch may not be placed.
VIA KEEPOUT	Shape identifying areas where vias may not be placed.
PINS	Identifies pins that do not have pin numbers. These pins are used to define mounting holes and registration holes in the layout geometry.
MANUFACTURING	Shapes identifying areas of the layout to be photoplotted; shapes identifying areas of the layout not to gloss.

- Start a mechanical symbol drawing (an outline, for example) with some mechanical element, such as a tooling hole, in the lower left corner, at 0,0.

This helps maintain the relationship between the symbol drawing origin and the mechanical drawing when placing the mechanical symbol in another design.

- If you have mechanical elements (such as design outlines) that are non-standard and unique to a particular design, and have no mounting holes (pins), create such mechanical elements as part of the design .brd file, by choosing *Add – Line* (add line) or *Add – Rectangle* (add rect) commands, described in the *Allegro PCB and Package Physical Layout Command Reference*.

## Creating a Board Outline

Before creating a board outline, refer to [Guidelines for Creating Mechanical Symbols](#). For procedural information, see *File – Create Symbol* ([create symbol](#) command) in the *Allegro PCB and Package Physical Layout Command Reference*.

## Creating Format Symbols

With Allegro PCB Editor, you can create various format elements in format symbol drawings for standard drawing sizes. You can edit or customize these format symbol elements for unique situations. You can also store these format symbols in a library so that you can easily access standard ones for designs without having to re-create these symbols for each design.

**Note:** Chapter 10, “[Format Symbol Library](#),” shows the format symbols in the Allegro PCB Editor library.

## Library Format Symbols

Format symbols in the Allegro PCB Editor library are standard drawing formats that contain the following:

- Borders (outline)
- Title blocks
- Revision blocks

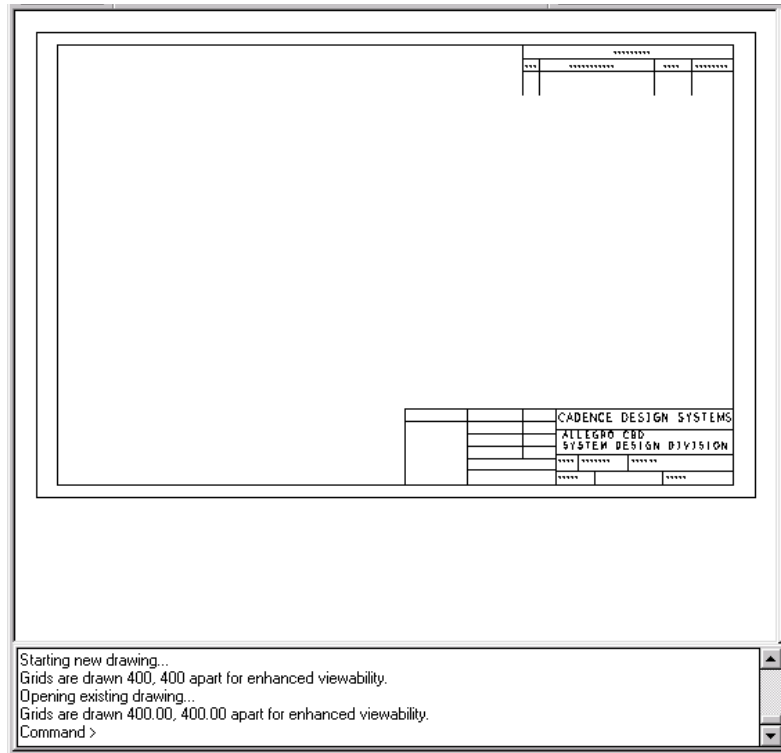
The Allegro PCB Editor library contains a separate format symbol for each standard drawing size. Format symbols have an `.osm` file name extension.

# Allegro PCB Editor User Guide: Defining and Developing Libraries

## Working with Symbols

---

The following sample format symbol shows the format elements included in a format symbol. This format has a border, a revision block, and a title block for a B size drawing.



## Guidelines for Creating Format Symbols

When you create format symbols that can be used for a variety of designs:

- Create one symbol for each drawing size.

Format drawings are not scalable, so plan to create one for each drawing size used by your company. When you create a new drawing, Allegro PCB Editor offers the four standard sizes—A, B, C, and D—as options.

- Create format elements on their related subclasses.
- Include all the format elements that might be needed for the design, then use the visibility controls for each subclass to determine which elements to display.
- Before you convert the drawing to a symbol, verify the origin of the symbol.

Choose an origin that facilitates placing the format into the design drawing.

Use the message bars at the bottom of the Allegro PCB Editor window to see what text has been typed in, the x,y location of the cursor in the active window, or what command is currently active.

## Defining a Format Symbol

Before creating a format symbol, refer to [Guidelines for Creating Format Symbols](#). For procedural information, see *File – Create Symbol* (`create symbol` command) in the *Allegro PCB and Package Physical Layout Command Reference*.

## Creating Flash Symbols

Flash symbols are pads that you create for photoplotting using standard apertures. For raster formats (Gerber RS-274X, Barco DPF, and McDonald Dettwiler MDA), you can include definitions for all apertures in the artwork file. For standard apertures and pads-as-shapes, you can derive the aperture definition from the Allegro PCB Editor design.

For aperture flashes, however, the design does not contain information about the geometry required for flashes referenced in the padstack data if the design has not been flash-converted from pre-14.0 versions of Allegro PCB Editor (as described in [Choosing a Design Methodology for Negative Planes](#)). In designs that have not been flash-converted, Allegro PCB Editor displays only a circle with a cross-wire for the flash.

In new or flash-converted designs, information is stored as flash (`.fsm`) symbols. Designs created in versions of Allegro PCB Editor 14.0 (and later) or flash-converted from earlier designs contain geometry information on all thermal flash symbols.

You can create flash symbols that display in WYSIWYG mode in three ways:

- As a new flash symbol in the symbol mode

Choose *File – New* (`new` command) to create a flash symbol such as a thermal pad for Rastar formats. Allegro PCB Editor saves flash symbols to the symbol library and appends the file name that you specify with an `.fsm` extension.

- From an existing shape symbol `.dra` file in the symbol mode

Choose *Setup – Design Parameters* (`prmed` command), the *Design* tab of the Design Parameter Editor, and *Flash* in the *Drawing Type* section. When you save the change with the `create symbol` command, the `.dra` is saved as a flash symbol.

- From existing flash `.bsm` files

Use the `flash_convert` command when updating pre-14.0 designs for use with later Allegro PCB Editor versions.

The character limit for flash names is 18.

For procedural information, see *File – Create Symbol* (`create_symbol` command) in the *Allegro PCB and Package Physical Layout Command Reference*.

## Choosing a Design Methodology for Negative Planes

The following sections describe the processes required to design databases with negative planes. Beginning with version 14.0, you can choose an old or new methodology for designing with negative planes.

### Methodology Differences

Designs created in Allegro PCB Editor starting with version 14.0 come up automatically in the new methodology; older databases open in the pre-14.0 manner. You must run the conversion program `flash_convert` on all older databases that you open in 14.0 or later versions, regardless of which methodology in which you want to work, to convert `.bsm` flash files to `.fsm` files. (See [Converting Flash Symbols When Migrating Databases](#) for details.) When you convert from one methodology to the other, `.bsm` files referenced locally or in the `PSMPATH` environment variable are converted to `.fsm` files and then imported into the design.

**Note:** See [Treatment of Nonconforming Symbols](#) for instances when you may not want to run `flash_convert`.

**Important:** Once you have migrated a pre-14.0 database to the new methodology, it cannot be converted back to the old style.

You can choose one of the following methodologies in which to work.

### Old Style Methodology

In this mode, flash information is not stored in the design, as is the case in pre-14.0 versions of Allegro PCB Editor. Rather, it is stored in the library referenced by the `PSMPATH` environment variable. This style lets you work with flash files as you did in earlier versions for artwork, display, and plotting.

- For pre-14.0 databases in the old methodology, run `flash_convert` on the design to update flash symbols from `.bsm` to `.fsm` files.



- For new databases created in the old methodology, you need to:
  - ❑ Set the environment variable `OLD_STYLE_FLASH_SYMBOLS`.
  - ❑ Run `flash_convert` on any `.bsm`-formatted flash symbols referenced in the new design.

### ***New Style Methodology***

This is the default style for new databases.

**Note:** See [Treatment of Nonconforming Symbols](#) for instances when you may not want to run `flash_convert` on certain new designs.

- For pre-14.0 databases with the new methodology, run `flash_convert` on the design to update flash symbols from `.bsm` to `.fsm` files.

Then use the `refresh_symbol` command to update the flash symbols.

- For new databases created with the new methodology, run `flash_convert` if you import any `.bsm`-formatted flash symbols into the design.

Then use the `refresh_symbol` command to update the flash symbols.

See [Updating Symbols](#) about refreshing symbols.

## **Converting Flash Symbols When Migrating Databases**

Beginning with version 14.0, flash symbols in databases are referenced as `.fsm` files. The `flash_convert` command, described in the *Allegro PCB and Package Physical Layout Command Reference*, lets you migrate older databases to the new methodology. You can choose to define flash symbols interactively for the database you are currently working in, or for one or more designs in a project hierarchy.

In most cases, `.bsm` symbols in a database are not referenced after running `flash_convert`. This is true regardless of the methodology you are using. The exception is that the `artwork` and `load gerber` commands reference a `.bsm` file if an `.fsm` file cannot be found.

## **Flash Symbols in Padstack Designer**

As long as you use the same name for flash names used in a padstack and flash symbols in a library of shape symbols, you can use the same thermal relief padstacks in either design methodology, because Padstack Designer maintains pad neutrality with respect to `.fsm` files.

In instances where both flash symbol names and shape symbol names are referenced, the methodology in which you are working determines which symbol name takes precedence. If you are working in the old style, the shape figure is referenced; if in the new style, the flash figure is referenced using the PSMPATH environment variable.

Actions that you take in Padstack Designer have no affect on the conversion processes described previously.

## Treatment of Nonconforming Symbols

In cases where the creation of .bsm files has been made using line segments rather than shapes or where voided shapes are used in the board symbol, `flash_convert` is unable to migrate a design's .bsm files to .fsm files. Where this occurs, options are available, based on the methodology you are using.

### Using the Old Methodology

- Use `flash_convert` to migrate all shape-based, non-void .bsm files to .fsm files.

Artwork uses the unconverted .bsm flash when unable to find the .fsm.

### Using the New Methodology for Unconverted Antipad Flashes

1. Create in the local directory of the database dummy .bsm files of any shape geometry for each antipad flash.
2. Run `flash_convert` on the database.

The thermal flashes are converted to dummy .fsm files and loaded into the database in WYSIWYG mode.

3. Delete the dummy .bsm and .fsm files.

The database now contains thermal flashes in WYSIWYG mode. However, artwork does not detect unconverted antipad flashes, so it uses the existing flash.bsm files.

## MDA Format Output Files

For films that include antipads and thermal flashes, MDA format requires two artwork files. For example, when you specify a film named 5v for a layer that contains antipads or thermal flashes, Allegro PCB Editor generates the following files:

- 5v.art

■ 5v\_s.art

MDA format uses paint and scratch commands. The \_s suffix is for the file with the scratch commands.

## Defining a Flash Symbol

To successfully convert an older database to the new methodology or to use new padstacks in new databases, you must provide flash symbol information; that is, you must have defined padstack flashes in .fsm files. For thermal flashes without .bsm files, you must create flash symbols that are used as thermal pads on a negative plane.

For procedural information, see *File – Create Symbol* (create symbol command) in the *Allegro PCB and Package Physical Layout Command Reference*.

## Updating Symbols

To ensure that you are using the latest symbols in a design, Allegro PCB Editor lets you update symbols and symbol padstacks from a library. You can do this interactively in Allegro PCB Editor's layout mode by choosing *Place – Update Symbols* (refresh symbol command), *Tools – Update Symbols* in Allegro PCB Editor's symbol mode, or by running the refresh symbol batch command from an operating system prompt. The menu items and commands are described in the *Allegro PCB and Package Physical Layout Command Reference*.

## Creating a Symbol List File

When you refresh symbols, you can either refresh all the symbols in a design or just the ones in a symbol list file.

List files are ASCII text files that end in the .lst extension and contain the names of symbols that can be updated. Use a text editor to create the symbol list file, and place the file in the current working directory.

The following file format conventions apply:

- Provide only one symbol name on each line.
- Use either upper- or lowercase letters. Allegro PCB Editor always stores symbol names in uppercase but can read mixed case in this file.

- Exclude leading or trailing white space. Allegro PCB Editor removes it if it appears in the file.

### Sample Symbol List File

The following example shows the symbol list file format.

```
a_size_drawing  
100milhole  
dip14_3
```

### Reviewing the Refresh Log File

The `refresh.log` file records refresh symbol processing.

---

## Checking Symbols Automatically

---

### Overview

During the development of physical symbols, such as component footprint symbols, you verify the existence of symbol elements, element layer definition, properties, and other criteria manually. However, manual verification can inadvertently add or omit some of these elements, and you might overlook such errors.

To automate the verification of physical symbols, choose *Tools – Check Symbol* ([check symbol](#) command) described in the *Allegro PCB and Package Physical Layout Command Reference*.

### Configuring the check symbol Command

You can customize the *Tools – Check Symbol* menu in the following ways:

- Modify the values of the global variables. These variables contain information such as the error messages. The global variables and its values are stored in the `rule_check_globals.il` file. This file is also called the Globals file.
- Add rules against which checks can be run. The rules are written in the AXL-SKILL language and then added to the `rule_check_tables.il` file. This file is also called the Rule Table file.

### Globals File

The `rule_check_globals.il` file contains global variables whose values you can change using any text editor. This file is an AXL-SKILL command file that defines one function, `( _PAC_setUserGlobals() )`, for the assignment of global variables.

While you should not change the names of the global variables, you can change the values for these variables, which lets you define error types for specific checks. There are some global variables that have recommended values and should not be changed but are available

in case you want to customize error severity. For information on global variables and their values, see [Global Variables and Values](#).

You can also change messages that appear in this file.

This file must be located in the directory defined by the SKILLPATH environment variable. If stored elsewhere, the symbol check uses the default values.

## Rule Table File

The `rule_check_tables.il` file defines and assigns rule classes, rules that fall into those classes, and what action to take when a rule is marked for execution. The `rule_check_tables.il` file is an AXL-SKILL program file that defines one function, `( _PAC_setUserFormTreeData() )`, for defining the rules tree in the *Physical Symbol Attributes Check* dialog box. You can edit this file using any text editor.

This file must be located in the directory defined by the SKILLPATH environment variable. If stored elsewhere, the symbol check uses the default definitions.

The format of this file is a series of embedded lists. The outermost list is the one that is passed to the `check symbol` command. The next tier of lists is the group or rules class. The first item in this list is the name of the rules class followed by a number of list pairs. This lowest level of lists contains the name of the rule followed by the AXL-SKILL function call.

The syntax is detailed below:

```
list(
  list( ["group_name1"]
    list(
      list(["check_name1"] ["function_call"])
      list(["check_name2"] ["function_call"])
    )
  )
)
```

An example of the Rule Table file is displayed below:

```
Defun( _PAC_setUserFormTreeData
list(
list( "REFDES Checks"
      list( '("REFDES Exist" "check_refdes_exist(file_ptr)')
```

## Allegro PCB Editor User Guide: Defining and Developing Libraries

### Checking Symbols Automatically

---

```
        '("REFDES Text Size" "check_refdes_blocksize(file_ptr)")
        '("REFDES Subclass" "check_refdes_subclass(file_ptr)")
    )
)
list( "COMPONENT VALUE Checks"
    list( '("COMPONENT VALUE Exist" "check_compvalue_exist(file_ptr)")
        '("COMPONENT VALUE Text Size" "check_compval_blocksize(file_ptr)")
        '("COMPONENT VALUE Subclass" "check_compval_subclass(file_ptr)")
    )
)
list( "DEVICE TYPE Checks"
    list( '("DEVICE TYPE Exist" "check_device_exist(file_ptr)")
        '("DEVICE TYPE Text Size" "check_device_blocksize(file_ptr)")
        '("DEVICE TYPE Subclass" "check_device_subclass(file_ptr)")
    )
)
list( "TOLERANCE Checks"
    list( '("TOLERANCE Exist" "check_prttol_exist(file_ptr)")
        '("TOLERANCE Text Size" "check_prttol_blocksize(file_ptr)")
        '("TOLERANCE Subclass" "check_prttol_subclass(file_ptr)")
    )
)
list( "PART NUMBER Checks"
    list( '("PART NUMBER Exist" "check_prtnum_exist(file_ptr)")
        '("PART NUMBER Text Size" "check_prtnum_blocksize(file_ptr)")
        '("PART NUMBER Subclass" "check_prtnum_subclass(file_ptr)")
    )
)
list( "GEOMETRY Checks"
    list( '("SILK SCREEN Geometry" "check_silkscreen_geometry(file_ptr)")
        '("ASSEMBLY Geometry" "check_assembly_geometry(file_ptr)")
    )
)
```

```
)  
list( "Reports"  
    list( ("PIN Location Report" "report_pin_location(file_ptr log_file)")  
        )  
    )  
)
```

## Developing Symbol Check Rules

The `check symbol` command lets you create rules and add them to the command. While writing the rules, you need to follow certain guidelines. These pertain to:

- Choosing the Language for the Source Code
- Deciding on Function Inputs
- Deciding on Function Returns
- Writing to the Marker File
- Writing to the Log File

### Choosing the Language for the Source Code

You must use the AXL-SKILL extension language available in Allegro PCB Editor to develop the source code for symbol rules. You can create as many functions as needed, but there must be one function call that the `check symbol` command calls which requires the passing of at least one variable into the function to record information into the markers file. A secondary variable might be passed to record information in the log file.

### Deciding on Function Inputs

There are two variables that need to be provided to the rule set function call:

- The `file_ptr` variable is used to pass information to the markers file.
- The `log_file` variable is the file pointer to the `<design_name>_symchk.log` file generated when the checks are run.



### Deciding on Function Returns

The rule function call must return a list of two elements:

- The number of errors found
- The number of warnings found during the rule check.

```
return( list( errors warnings))
```

### Writing to the Marker File

The marker file (*<design\_name>\_symchk.mkr*) contains messages and location-specific information for any element found in the drawing that may be an error, warning, or message. You call the `_PAC_report_errors( )` function to write to the marker file.

The complete syntax is as follows:

```
_PAC_report_errors( file_ptr <error_type> <short_message> <long_message>  
                   <object_kind> <object_name> <parent_name> <design_name> )
```

These are the parameters that the function takes:

<i>file_ptr</i>	The pointer to the output port for the markers file.
<i>error_type</i>	An integer value that states what type of message is being displayed. Typically, you use one of the global variables, such as ERROR, WARNING, and so on. For more information, see <a href="#">Global Variables and Values</a> .
<i>short_message</i>	An ASCII string that gives a brief description of the issues being reported. This is a single-string entry.
<i>long_message</i>	An ASCII string that gives a detailed description of the issue being reported. This is a single-string entry.
<i>object_kind</i>	An ASCII string for the name of the element type. The element type is either PIN or TEXT.
<i>object_name</i>	An ASCII string that denotes the X/Y location and the class and subclass description of the item. The string should follow the following format:  <code>x.x:y.y=class/subclass</code>  For example:  <code>1.000:1.250=PACKAGE GEOMETRY/ASSEMBLY_TOP</code>

## Allegro PCB Editor User Guide: Defining and Developing Libraries

### Checking Symbols Automatically

---

<i>parent_name</i>	An ASCII string that describes the element type and X/Y location. This is the format for the string:  item@( x.x y.y)  For example:  PIN@(3.450 4.623)
<i>design_name</i>	An ASCII string that describes the drawing or drawing element

An example of the `_PAC_report` errors is given below.

Assume that the following global variables are defined in the Global file:

```
TEST_CASE_ERROR = ERROR
```

```
TEST_CASE_SHORT = "Test Element missing"
```

```
TEST_CASE_LONG = "Missing Add Test Element at origin.\n Please add."
```

The function call would be:

```
_PAC_report_errors( file_ptr TEST_CASE_ERROR TEST_CASE_SHORT  
TEST_CASE_LONG "TEXT" "0.00:0.00=PACKAGE_GEOMETRY/DISPLAY_TOP"  
"CIRCLE@(0.00 0.00)" "test_symbol.dra")
```

### Writing to the Log File

You can write to the log file (`<design_name>_symchk.log`) using the standard `fprintf` functions available in AXL-SKILL. The pointer variable to the output port for the log file is `log_file`. You must pass this pointer into the function.

For example:

```
Defun( your_function ( file_ptr log_file)  
Prog( ()  
Code  
if( errors < 1 then  
    if(warnings < 1  
        fprintf( log_file "This Check Passed\n")  
    else  
        fprintf( log_file "This check has warnings\n")
```

```
        );end-if  
else  
        fprintf( log_file "This check has errors\n")  
    ); end-if  
return( list( errors warnings))  
))
```

## Installing Custom Rules

Installing the rules that you have created involves two activities:

- Modifying the user defined global variable, if required.
- Including the rules in the check symbol command so that they appear in the `check symbol` dialog box.

## Modifying the User-Defined Global Variables

The Global variable for setting messages and error/checking type values should be created in the `rule_check_globals.il` file. This is a central global variable value repository for all checking functions. Formats can be observed in the file and may be copied or altered as required.



***Do not change the values for the ERROR, WARNING, and INFO definitions.***

***Do not remove or change the name of existing GLOBAL variables defined in this file. You can change values, but do not change the variable names.***

## Including Rules in the check symbol Command

To include a new custom rule, store the SKILL program in a directory to which all those who use the program have at least read access. Then, load the SKILL file upon opening the Symbol Editor. This can be done through the `allegro.ilinit` file. The command or function should then be added in the `rules_check_table.il` file.

## Predefined Rules

The `check symbol` command comes with seven sets of rules against which you can run a check, which appear in the *Physical Symbol Attributes Check* dialog box:

- REFDES Checks
- COMPONENT VALUE Checks
- DEVICE TYPE Checks
- TOLERANCE Checks
- USER PART NUMBER Checks
- GEOMETRY Checks
- Reports

### REFDES Checks

This contains three rules that check for the existence, text block size, and the layer definition for each reference designator in the package or mechanical symbol.

#### REFDES Exist

This rule checks for the existence of the Reference designator in the footprint. The check status is defined by the global variable as an error. For this rule, the following global variables are defined:

```
TEXT_REFDES_ERROR = ERROR
```

```
TEXT_REFDES_SHORT = "No REFDES Text in Symbol"
```

```
TEXT_REFDES_LONG = "No REFDES Text in Symbol \n REFDES Text Has Not  
Been Defined in this Symbol"
```

#### REFDES Text Size

This rule checks for the text block size of the reference designator in the footprint. The Text Size is verified using the global variable `TEXT_REFDES_TEXT_BLOCK` which is set to the default of 3. The check status is defined by the global variable as an error. For this rule, the following global variables are defined:

## Allegro PCB Editor User Guide: Defining and Developing Libraries

### Checking Symbols Automatically

---

```
TEXT_REFDES_TEXT_BLOCK = "3"
```

```
TEXT_REFDES_SIZE_ERROR = ERROR
```

```
TEXT_REFDES_SHORT = "Incorrect Text Size For REFDES Text"
```

```
TEXT_REFDES_LONG = "Text Size for REFDES is Incorrect"
```

### REFDES Subclass

This rule checks for the CLASS/SUBCLASS layer where the REFDES text should be in the footprint symbol. The CLASS/SUBCLASS is verified using the global variable `TEXT_REFDES_SUBCLASS`, which is set to the default value `list(" REF DES/ASSEMBLY_TOP" "REF DES/SILKSCREEN_TOP")`. The check status is defined by the global variable as an error. For this rule, the following global variables are defined:

```
TEXT_REFDES_SUBCLASS = list("REF DES/ASSEMBLY_TOP" "REF DES/SILKSCREEN_TOP")
```

```
TEXT_REFDES_SUBCLASS_ERROR = ERROR
```

```
TEXT_REFDES_SUBCLASS_SHORT = "Incorrect REFDES Text Subclass Assignment."
```

```
TEXT_REFDES_SUBCLASS_LONG = "Subclass Assignment For REFDES Text Not\an Allowed Layer."
```

```
TEXT_REFDES_SUBCLASS_MISSING = "REFDES Text Missing Subclass Definition"
```

### COMPONENT VALUE Checks

This contains three rules that check for the existence, text block size, and the layer definition for the component value in the package or mechanical symbol.

### COMPONENT VALUE Exist

This rule checks for the existence of the component value text in the footprint. The check status is defined by the global variable as a warning. For this rule, the following global variables are defined:

```
TEXT_COMP_VALUE_ERROR = WARNING
```

## Allegro PCB Editor User Guide: Defining and Developing Libraries

### Checking Symbols Automatically

---

```
TEXT_COMP_VALUE_SHORT = "No PART VALUE Text in Symbol"
```

```
TEXT_COMP_VALUE_LONG = "No PART VALUE Text in Symbol \n COMPONENT  
VALUE Text Has Not Been Defined in this Symbol"
```

#### COMPONENT VALUE Text Size

This rule checks for the text block size of the component value in the footprint. The Text Size is verified using the global variable TEXT\_COMP\_VALUE\_TEXT\_BLOCK, which is set to the default of 3. The check status is defined by the global variable as a warning. For this rule, the following global variables are defined:

```
TEXT_COMP_VALUE_TEXT_BLOCK = "3"
```

```
TEXT_ COMP_VALUE _SIZE_ERROR = WARNING
```

```
TEXT_ COMP_VALUE _SHORT = "Incorrect Text Size For COMPONENT VALUE  
Text"
```

```
TEXT_ COMP_VALUE _LONG = "Text Size for COMPONENT VALUE is Incorrect"
```

#### COMPONENT VALUE Subclass

This rule checks for the CLASS/SUBCLASS layer where the component value text should be in the footprint symbol. The CLASS/SUBCLASS is verified using the global variable TEXT\_COMP\_VALUE\_SUBCLASS, which is set to the default value list("COMPONENT VALUE/ASSEMBLY\_TOP"). The check status is defined by the global variable as a warning. For this rule, the following global variables are defined:

```
TEXT_COMP_VALUE_SUBCLASS = list("COMPONENT VALUE/ASSEMBLY_TOP" )
```

```
TEXT_ COMP_VALUE _SIZE_ERROR = WARNING
```

```
TEXT_ COMP_VALUE _SUBCLASS_SHORT = "Incorrect COMPONENT VALUE Text  
Subclass Assignment."
```

```
TEXT_ COMP_VALUE _SUBCLASS_LONG = "Subclass Assignment For component  
Value Text Not\nnon an Allowed Layer."
```

```
TEXT_ COMP_VALUE _SUBCLASS_MISSING = "COMPONENT VALUE Text Missing  
Subclass Definition"
```

## DEVICE TYPE Checks

This contains three rules that check for the existence, text block size, and layer definition for the device type in the package or mechanical symbol.

### DEVICE TYPE Exist

This rule checks for the existence of the device type text in the footprint. The check status is defined by the global variable as a warning. For this rule, the following global variables are defined:

```
TEXT_DEVICE_ERROR = WARNING
```

```
TEXT_DEVICE_SHORT = "No DEVICE Text in Symbol"
```

```
TEXT_DEVICE_LONG = "No DEVICE Text in Symbol \n DEVICE Text Has Not  
Been Defined in this Symbol"
```

### DEVICE TYPE Text Size

This rule checks for the text block size of the device type in the footprint. The text size is verified using the global variable TEXT\_DEVICE\_TEXT\_BLOCK, which is set to the default value of "3". The check status is defined by the global variable as a warning. For this rule, the following global variables are defined:

```
TEXT_DEVICE_TEXT_BLOCK = "3"
```

```
TEXT_DEVICE_SIZE_ERROR = WARNING
```

```
TEXT_DEVICE_SHORT = "Incorrect Text Size For DEVICE Text"
```

```
TEXT_DEVICE_LONG = "Text Size for DEVICE is Incorrect"
```

### DEVICE TYPE Subclass

This rule checks for the CLASS/SUBCLASS layer where the device type text should be in the footprint symbol. The CLASS/SUBCLASS is verified using the global variable TEXT\_DEVICE\_SUBCLASS, which is set to the default value `list("DEVICE TYPE/ASSEMBLY_TOP")`. The check status is defined by the global variable as a warning. For this rule, the following global variables are defined:

```
TEXT_DEVICE_SUBCLASS = list("DEVICE TYPE/ASSEMBLY_TOP" )
```

## Allegro PCB Editor User Guide: Defining and Developing Libraries

### Checking Symbols Automatically

---

```
TEXT_DEVICE_SUBCLASS_ERROR = WARNING
```

```
TEXT_DEVICE_SUBCLASS_SHORT = "Incorrect Subclass Assignment For  
DEVICE Text."
```

```
TEXT_DEVICE_SUBCLASS_LONG = "Subclass Assignment For DEVICE Text  
Not\non an Allowed Layer."
```

```
TEXT_DEVICE_SUBCLASS_MISSING = "DEVICE Text Missing Subclass
```

## TOLERANCE Checks

This contains three rules that check for the existence, text block size, and layer definition for component tolerance in the package or mechanical symbol.

### TOLERANCE Exist

This rule checks for the existence of the tolerance text in the footprint. The check status is defined by the global variable as a warning. For this rule, the following global variables are defined:

```
TEXT_PART_TOL_ERROR = WARNING
```

```
TEXT_PART_TOL_SHORT = "No TOLERANCE Text in Symbol"
```

```
TEXT_PART_TOL_LONG = "No TOLERANCE Text in Symbol \n TOLERANCE Text  
Has Not Been Defined in this Symbol"
```

### TOLERANCE Text Size

This rule checks for the text block size of the tolerance in the footprint. The text size is verified using the global variable TEXT\_PART\_NUMBER\_TEXT\_BLOCK which is set to the default value of 3. The check status is defined by the global variable as a warning. For the rule, the following global variables are defined:

```
TEXT_PART_TOL_TEXT_BLOCK = "3"
```

```
TEXT_PART_TOL_SIZE_ERROR = WARNING
```

```
TEXT_PART_TOL_SHORT = "Incorrect Text Size For TOLERANCE Text"
```

```
TEXT_PART_TOL_LONG = "Text Size for TOLERANCE is Incorrect"
```



## **TOLERANCE Subclass**

This rule checks for the CLASS/SUBCLASS layer where the tolerance text is to be in the footprint symbol. The CLASS/SUBCLASS is verified using the global variable TEXT\_PART\_TOL\_SUBCLASS, which is set to the default value `list("TOLERANCE/ASSEMBLY_TOP")`. The check status is defined by the global variable as a warning. For this rule, the following global variables are defined:

```
TEXT_ PART_ TOL _SUBCLASS = list("TOLERANCE /ASSEMBLY_TOP" )

TEXT_ PART_ TOL _SIZE_ERROR = WARNING

TEXT_ PART_ TOL _SUBCLASS_SHORT = "Incorrect TOLERANCE Text Subclass
Assignment."

TEXT_ PART_ TOL _SUBCLASS_LONG = "Subclass Assignment For TOLERANCE
Text Not\non an Allowed Layer."

TEXT_ PART_ TOL _SUBCLASS_MISSING = "TOLERANCE Text Missing Subclass
Definition"
```

## **USER PART NUMBER Checks**

This rule class contains three rule sets that check for the existence, text block size, and layer definition for the part number in the package or mechanical symbol.

### **USER PART NUMBER Exist**

This rule checks for the existence of the part number text in the footprint. The check status is defined by the global variable as a warning. For this rule, the following global variables are defined:

```
TEXT_PART_NUMBER_ERROR = WARNING

TEXT_ PART_NUMBER _SHORT = "No USER PART NUMBER Text in Symbol"

TEXT_ PART_NUMBER _LONG = "No USER PART NUMBER Text in Symbol \n PART
NUMBER Text Has Not Been Defined in this Symbol"
```

### **USER PART NUMBER Text Size**

This rule checks for the text block size of the component value in the footprint. The text size is verified using the global variable TEXT\_PART\_NUMBER\_TEXT\_BLOCK, which is set to

## Allegro PCB Editor User Guide: Defining and Developing Libraries

### Checking Symbols Automatically

---

the default value of 3. The check status is defined by the global variable as a warning. For this rule, the following global variables are defined:

```
TEXT_ PART_NUMBER _TEXT_BLOCK = "3"
```

```
TEXT_ PART_NUMBER _SIZE_ERROR = WARNING
```

```
TEXT_ PART_NUMBER _SHORT = "Incorrect Text Size For USER PART NUMBER  
Text"
```

```
TEXT_ PART_NUMBER _LONG = "Text Size for USER PART NUMBER is  
Incorrect"
```

### USER PART NUMBER Subclass

This rule checks for the CLASS/SUBCLASS layer that the part number text should be in the footprint symbol. The CLASS/SUBCLASS is verified using the global variable `TEXT_PART_NUMBER_SUBCLASS`, which is set to the default value `list("USER PART NUMBER /ASSEMBLY_TOP")`. The check status is defined by the global variable as a warning. For this rule, the following global variables are defined:

```
TEXT_ PART_NUMBER _SUBCLASS = list("USER PART NUMBER /ASSEMBLY_TOP" )
```

```
TEXT_ PART_NUMBER _SIZE_ERROR = WARNING
```

```
TEXT_ PART_NUMBER _SUBCLASS_SHORT = "Incorrect PART NUMBER Text  
Subclass Assignment."
```

```
TEXT_ PART_NUMBER _SUBCLASS_LONG = "Subclass Assignment For PART  
NUMBER Text Not\nnon an Allowed Layer."
```

```
TEXT_ PART_NUMBER _SUBCLASS_MISSING = "PART NUMBER Text Missing  
Subclass Definition"
```

### GEOMETRY Checks

This rule class contains two rule sets that check for the existence of geometric data in the package or mechanical symbol.

### SILK SCREEN Geometry

This rule checks for the existence of the arcs, lines, and shapes defined on the PACKAGE GEOMETRY class and TOP and bottom silkscreen subclasses in the footprint. The check

status is defined by the global variable as an error. For this rule, the following global variables are defined:

```
SILKSCREEN_GEOMETRY_ERROR = ERROR
```

```
SILKSCREEN_GEOMETRY _SHORT = "No Silk Screen Geometry in Symbol"
```

```
SILKSCREEN_GEOMETRY _LONG = "No Silk Screen Geometry in Symbol \n Silk  
Screen Graphical Data Will Not Exist"
```

### ASSEMBLY Geometry

This rule checks for the existence of arcs, lines and shapes defined on the PACKAGE GEOMETRY class, and the ASSEMBLY\_TOP and ASSEMBLY\_BOTTOM subclasses. The check status is defined by the global variable as an error. For this rule, the following global variables are defined:

```
ASSEMBLY_GEOMETRY_ERROR = ERROR
```

```
ASSEMBLY _GEOMETRY _SHORT = "No Assembly Geometry in Symbol"
```

```
ASSEMBLY _GEOMETRY _LONG = "No Assembly Geometry in Symbol \n Assembly  
Graphical Data Will Not Exist"
```

### Reports

This allows report data to be generated as output. This contains one rule that reports the pin number, the XY location and the padstack name for each pin in a symbol drawing. This rule is hard-coded because check status INFO and cannot be modified.

## Global Variables and Values

Variable Name	Type	Default Value
ERROR	Integer (Do not change)	40
WARNING	Integer (Do not change)	30
INFO	Integer (Do not change)	20
ASSEMBLY_GEOMETRY_SHORT	String (short message)	"No Assembly Geometry in Symbol."
ASSEMBLY_GEOMETRY_LONG	String (long message)	"No Assembly Geometry in Symbol \n Assembly Graphical Data Will Not Exist"
ASSEMBLY_GEOMETRY_ERROR	Integer	ERROR
SILKSCREEN_GEOMETRY_SHORT	String (short message)	"No Silk Screen Geometry in Symbol."
SILKSCREEN_GEOMETRY_LONG	String (long message)	"No Silk Screen Geometry in Symbol \n Silk Screen Graphical Data Will Not Exist"
SILKSCREEN_GEOMETRY_ERROR	Integer	ERROR
TEXT_REFDES_SHORT	String (short message)	"No REFDES Text in Symbol."
TEXT_REFDES_LONG	String (long message)	"No REFDES Text in Symbol \n REFDES Text Has Not Been Defined in This Symbol."
TEXT_REFDES_SIZE_SHORT	String (short message)	"Incorrect Text Size For REFDES Text."
TEXT_REFDES_SIZE_LONG	String (long message)	"Text Size For REFDES is Incorrect."
TEXT_REFDES_SUBCLASS_SHORT	String (short message)	"Incorrect REFDES Text Subclass Assignment."
TEXT_REFDES_SUBCLASS_LONG	String (long message)	"Subclass Assignment For REFDES Text Not \n on an Allowed Layer."

## Allegro PCB Editor User Guide: Defining and Developing Libraries

### Checking Symbols Automatically

TEXT_REFDES_SUBCLASS_MISSIN G	String	"REFDES Text Missing Subclass Definition"
TEXT_REFDES_ERROR	Integer	ERROR
TEXT_REFDES_BLOCK_ERROR	Integer	ERROR
TEXT_REFDES_SUBCLASS_ERROR	Integer	ERROR
TEXT_COMP_VALUE_SHORT	String (short message)	"No COMPONENT VALUE Text in Symbol."
TEXT_COMP_VALUE_LONG	String (long message)	"No COMPONENT VALUE Text in Symbol \n COMPONENT VALUE Text Has Not Been Defined in This Symbol."
TEXT_COMP_VALUE_SIZE_SHORT	String (short message)	"Incorrect Text Size For COMPONENT VALUE Text."
TEXT_COMP_VALUE_SIZE_LONG	String (long message)	"Text Size For COMPONENT VALUE is Incorrect."
TEXT_COMP_VALUE_SUBCLASS_S HORT	String (short message)	"Incorrect Subclass Assignment For COMPONENT VALUE Text."
TEXT_COMP_VALUE_SUBCLASS_L ONG	String (long message)	"Subclass Assignment For COMPONENT VALUE Text Not \n on an Allowed Layer."
TEXT_COMP_VALUE_SUBCLASS_MI SSING	string	"COMPONENT VALUE Text Missing Subclass Definition"
TEXT_COMP_VALUE_ERROR	Integer	WARNING
TEXT_COMP_VALUE_BLOCK_ERRO R	Integer	WARNING
TEXT_COMP_VALUE_SUBCLASS_E RROR	Integer	WARNING
TEXT_DEVICE_SHORT	String (short message)	"No DEVICE Text in Symbol."
TEXT_DEVICE_LONG	String (long message)	"No DEVICE Text in Symbol \n DEVICE Text Has Not Been Defined in This Symbol."
TEXT_DEVICE_SIZE_SHORT	String (short message)	"Incorrect Text Size For DEVICE Text."

## Allegro PCB Editor User Guide: Defining and Developing Libraries

### Checking Symbols Automatically

TEXT_DEVICE_SIZE_LONG	string	"Text Size For DEVICE is Incorrect."
TEXT_DEVICE_SUBCLASS_SHORT	String (short message)	"Incorrect Subclass Assignment For DEVICE Text."
TEXT_DEVICE_SUBCLASS_LONG	String (long message)	"Subclass Assignment For DEVICE Text Not \n on an Allowed Layer."
TEXT_DEVICE_SUBCLASS_MISSING	string	"DEVICE Text Missing Subclass Definition"
TEXT_DEVICE_ERROR	Integer	WARNING
TEXT_DEVICE_BLOCK_ERROR	Integer	WARNING
TEXT_DEVICE_SUBCLASS_ERROR	Integer	WARNING
TEXT_PART_NUMBER_SHORT	String (long message)	"No USER PART NUMBER Text in Symbol."
TEXT_PART_NUMBER_LONG	String (long message)	"No USER PART NUMBER Text in Symbol \n PART NUMBER Text Has Not Been Defined in This Symbol."
TEXT_PART_NUMBER_SIZE_SHORT	String (long message)	"Incorrect Text Size For PART NUMBER Text."
TEXT_PART_NUMBER_SIZE_LONG	String (long message)	"Text Size For USER PART NUMBER is Incorrect."
TEXT_PART_NUMBER_SUBCLASS_SHORT	String (short message)	"Incorrect Subclass Assignment For USER PART NUMBER Text."
TEXT_PART_NUMBER_SUBCLASS_LONG	String (long message)	"Subclass Assignment For USER PART NUMBER Text Not \n on an Allowed Layer."
TEXT_PART_NUMBER_SUBCLASS_MISSING	string	"USER PART NUMBER Text Missing Subclass Definition"
TEXT_PART_NUMBER_ERROR	Integer	ERROR
TEXT_PART_NUMBER_BLOCK_ERROR	Integer	ERROR
TEXT_PART_NUMBER_SUBCLASS_ERROR	Integer	ERROR
TEXT_PART_TOL_SHORT	String (short message)	"No TOLERANCE Text in Symbol."

## Allegro PCB Editor User Guide: Defining and Developing Libraries

### Checking Symbols Automatically

TEXT_PART_TOL_LONG	String (long message)	"No TOLERANCE Text in Symbol \n PART TOLERANCE Text Has Not Been Defined in This Symbol."
TEXT_PART_TOL_SIZE_SHORT	String (short message)	"Incorrect Text Size For TOLERANCE Text."
TEXT_PART_TOL_SIZE_LONG	String (long message)	"Text Size For TOLERANCE is Incorrect."
TEXT_PART_TOL_SUBLCASS_SHORT	String (short message)	"Incorrect Subclass Assignment For TOLERANCE Text."
TEXT_PART_TOL_SUBCLASS_LONG	String (long message)	"Subclass Assignment For TOLERANCE Text Not \n on an Allowed Layer."
TEXT_PART_TOL_SUBCLASS_MISSING	string	"TOLERANCE Text Missing Subclass Definition"
TEXT_PART_TOL_ERROR	Integer	WARNING
TEXT_PART_TOL_BLOCK_ERROR	Integer	WARNING
TEXT_PART_TOL_SUBCLASS_ERROR	Integer	WARNING
TEXT_REFDES_SUBCLASS	List (string class/ subclass)	list( "REF DES/ASSEMBLY_TOP" "REF DES/SILKSCREEN_TOP")
TEXT_COMP_VALUE_SUBCLASS	List (string class/ subclass)	list( "COMPONENT VALUE/ ASSEMBLY_TOP")
TEXT_DEVICE_SUBCLASS	List (string class/ subclass)	list( "DEVICE TYPE/ ASSEMBLY_TOP")
TEXT_PART_TOL_SUBCLASS	List (string class/ subclass)	list( "TOLERANCE/ ASSEMBLY_TOP")
TEXT_PART_NUMBER_SUBCLASS	List (string class/ subclass)	list( "USER PART NUMBER/ ASSEMBLY_TOP")
TEXT_REFDES_TEXT_BLOCK	String (text block size)	"3"

## Allegro PCB Editor User Guide: Defining and Developing Libraries

### Checking Symbols Automatically

---

TEXT_COMP_VALUE_TEXT_BLOCK	String (text block size)	"3"
TEXT_DEVICE_TEXT_BLOCK	String (text block size)	"3"
TEXT_PART_NUMBER_TEXT_BLOCK	String (text block size)	"3"
TEXT_PART_TOL_TEXT_BLOCK	String (text block size)	"3"



---

## Preparing Device Files

---

Device files are ASCII text files that provide logic information for unique components. During design database creation, links device files to related package symbols to obtain a complete description of these unique components.

Device files provide Allegro PCB Editor with the following information:

- Package configuration (for example, DIP and SIP)
- Placement class of the package (for example, IC, discrete)
- Number of pins in the component package
- Electronic description of the component pins (for example, logical use and pin swap information)
- Number of functions in the physical package
- How functions are mapped to package slots
- Pin use (how pins are used)—for example, input pins, output pins, and bidirectional pins
- How the logical function pins correspond to the physical pins of the package
- Which pins are tied to power and ground
- Definition properties, such as alternate symbols that can be used instead of the primary package symbol during placement

You need device files only when you are passing the electrical configuration to Allegro PCB Editor with a third-party netlist, and you are not using Concept for schematic entry. (For details, see the *Transferring Logic Design Data* user guide in your documentation set. Allegro PCB Editor uses a netlist to determine which electrical components the package symbols represent and how signals are connected. The netlist lists the logic functions (NAND gates, resistors, capacitors, connect pins, and so on) or electrical components of the design and their interconnections.

The netlist contains various sections that require the names of the device files for a design:

- The \$PACKAGES section identifies the components to be added to the design.

For each user-created package symbol in the \$PACKAGES section, provide the name of the device file corresponding to the package symbol and the reference designators for the device type.

**Note:** You can also assign reference designators by choosing *Logic – Assign Refdes* (assign\_refdes command) and *Tools – Assign Refdes* (for Allegro PCB Performance Option L only), described in the *Allegro PCB and Package Physical Layout Command Reference*.

- The \$FUNCTIONS section identifies the devices with functions.

You must provide the device file names in the \$FUNCTIONS section of the netlist.

The netlist also carries property information for components, functions, or nets. For details about netlists, see the *Transferring Logic Design Data* user guide in your documentation set.

Because Allegro PCB Editor requires device file information when creating the design database, create device files before choosing *File – Import – Logic* (netin command) to create the design database. This command is described in the *Allegro PCB and Package Physical Layout Command Reference*.

Create device files using a text editor on your system. You can obtain much of the information for device files from manufacturers' product specification data books.

## Device File Records

A device file consists of separate lines (records) that provide the logical data for the device. Each line contains a keyword followed by one or more data fields. A keyword is an attribute that describes the device, and fields are units of information that further define the keyword. The following shows the format of a device file.

## Device File Format

Optional but recommended	_____	(comment line)
	_____	PACKAGE <matching_package_symbol_name>
		CLASS <type>
Required	_____	PINCOUNT <number_of_pins>
		PINORDER <function_type> <list_of_pin_names>
		PINUSE <function_type> <list_of_pin_use_codes>
Optional but recommended	_____	PINSWAP <function_type> <list_of_pin_names>
		FUNCTION <slot_name> <function_type>
		<list_of_pin_numbers>
		POWER <net_name> ; <list_of_pin_numbers>
		GROUND <net_name> ; <list_of_pin_numbers>
		NC ; <list_of_pin_numbers>
Optional but recommended	_____	PACKAGEPROP <property_type> <property_value>
		END

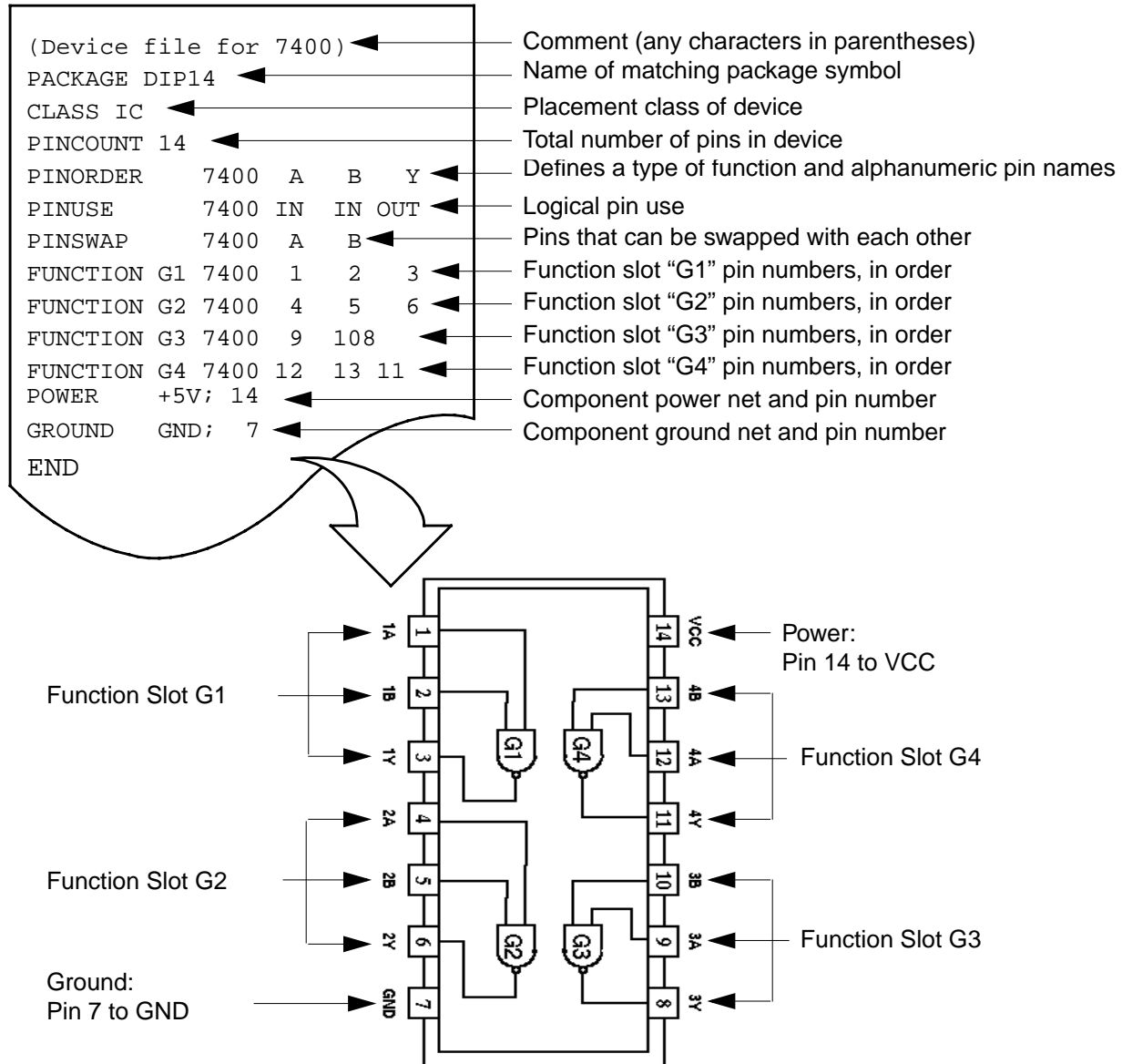
The records not labelled as required or recommended are optional.

Figure 5-1 shows how each line in a sample device file is used in Allegro PCB Editor to construct a physical package. This example shows how a 7400 maps the functions to slots and the pin names to pin numbers.

# Allegro PCB Editor User Guide: Defining and Developing Libraries

## Preparing Device Files

**Figure 5-1 Device File for 7400 Component**



## Syntax and Field Descriptions

This section describes each of the device file records and their syntax.

### PACKAGE

Optional, but highly recommended. The PACKAGE record describes the physical package for the device. Allegro PCB Editor uses this information during interactive and automatic placement.

```
PACKAGE <name>
```

*name*                      The package symbol name to which the device corresponds. If there is a mismatch, automatic placement fails.

### CLASS

Optional, but highly recommended. The CLASS record lets you place the device by class type.

```
CLASS <type>
```

*type*                      The type of class:

- IC: Integrated circuit
- IO: Input/output—connectors
- DISCRETE

If the device file does not contain a pin order section, Allegro PCB Editor uses the class to determine how many functions are contained in the device. If you specify class *IO*, Allegro PCB Editor sees each pin as a function. If you specify class *IC*, Allegro PCB Editor considers all pins on the device, except for power and ground pins, to be in one function.

Test prep uses the class to determine which holes are defined as test point sites. Class is also used during placement.

## PINCOUNT

Required. The PINCOUNT record tells Allegro PCB Editor the total number of pins (connect points) on the physical package. This number is used in interactive and automatic placement.

PINCOUNT *<number\_of\_pins>*

*number\_of\_pins*            The total number of pins on the device. The pincount is the total number of all pins, including mounting holes. For example, a 20-pin connector with 2 mounting holes has a pincount of 22.

## PINORDER

Optional. The PINORDER record contains information about a unique function type in a device. You must also supply PINUSE, PINSWAP, and FUNCTION records to fully define the function.

PINORDER *<function\_type>* *<list\_of\_pin\_names>*

*function\_type*            The name of the function.

*list\_of\_pin\_names*        A list of pins associated with the function. Separate each pin name with a space or tab. Pin names can be alphanumeric. Each pin name corresponds to a pin of the function.

If you have a device with multiple functions like 7431, the device file must contain multiple PINORDER statements that identify each function. See [Specifying Multiple Functions in a Device File](#) for details on defining multiple functions.

### PINUSE

Optional. The PINUSE record defines the logical use of each pin within a gate. You must specify a pin use code for each pin listed in the PINORDER record.

```
PINUSE <function_type> <list_of_pin_use_codes>
```

*function\_type*

The name of the gate.

*list\_of\_pin\_use\_codes*

A list of the types of pins used in the gate:

- IN: Input
- OUT: Output
- BI: Bidirectional
- TRI: Tristate—sending, receiving, or held at some state
- OCA: Open Emitter
- OCL: Open Collector
- POWER
- GROUND
- NC: Not connected internally
- UNSPEC

Separate each pin type with a space or tab.

Pinuse codes are required when using more advanced features in the Allegro PCB Editor toolset. Pinuse codes apply for Allegro PCB SI L, XL, or GXL and other applications (such as scheduling) that merely need information as to whether the pin is an output or input, or to determine a default simulation model. Allegro PCB Editor uses the logical information during scheduling and terminator assignment. Test prep also uses PINUSE information.

Scheduling considers OUT, OCA, OCL, NC, and TRI as drivers; IN, UNSPEC, POWER, GROUND as loads. The pin type BI can be either a load or a driver, depending on other PINUSE codes on the net. For example, if no other driver exists on a given ECL net, the BI pin is considered the driver. If a driver exists on the net, then the BI pin is located between the driver and the load.

The OCL and OCA pinuse codes can be used for devices that tie directly to a certain leg of the output transistor and therefore require an external resistor. They can also be used for

## Allegro PCB Editor User Guide: Defining and Developing Libraries

### Preparing Device Files

---

devices tied together to represent Wired logic configurations (Wired-AND or Wired-OR depending on positive or negative logic). The following table describes their function:

PINUSE	PINTYPE	Description	Default SI Model	SI Model Type
OCL	OC	Open Collector	Open Drain	Open PullUp
OCA	OE	Open Emitter	Open Source	Open PullDown

### PINSWAP

Optional. The PINSWAP record tells the interactive and automatic swapping routines which pins inside a gate are legal for swapping.

```
PINSWAP <function_type> <list_of_pin_names>
```

*function\_type*                      The name of the gate. Must match the gate defined in the pinorder record.

*list\_of\_pin\_names*                A list of pins inside the gate that can be swapped.

Allegro PCB Editor uses the PINSWAP record during interactive and automatic swapping to determine which pins inside a gate are available for pin swapping.

If a gate has more than one group of swappable pins, enter a pinswap record for each group of swappable pins to the device file.

### FUNCTION

Optional. The FUNCTION record identifies the slots available in the package for the function type described in the PINORDER record.

```
FUNCTION <slot_name> <function_type> <list_of_pin_numbers>
```

*slot\_name*                              The name of the gate. Must match the gate defined in the PINORDER record.

*function\_type*                        The name of the physical slot.

*list\_of\_pin\_numbers*                A list of alphanumeric pin numbers in the device. The pin numbers must match the pin numbers on the package symbol that corresponds to this device. Separate each pin number with a space.



## Allegro PCB Editor User Guide: Defining and Developing Libraries

### Preparing Device Files

---

One symbol on the schematic corresponds to one function in the device file. If you do not enter a PINORDER record, Allegro PCB Editor assumes that for each non-power and non-ground pin there is a schematic symbol for CLASS IO devices.

For CLASS IC and DISCRETE devices, Allegro PCB Editor assumes there is one symbol showing all pins in the device, except for power and ground pins. For example, Figure 5-1 shows a 7400 device file. The corresponding schematic symbol for the 7400 device file would contain four separate gates, each with three pins. However, if the symbol were a 14-pin CLASS IO connector, the corresponding schematic would contain 12 one-pin symbols.

### POWER

Optional. The POWER record ties the pins of a device to a particular power signal. Allegro PCB Editor uses this power information when you use the `netin` command to create a database. The net name given is the default net for the defined power pins. This net is assigned to these pins if there is no net specified for the pins in the netlist.

```
POWER <net_name> ; <list_of_pin_numbers>
```

*net\_name*                      The default net for the power pins listed. The net is assigned to these pins if you do not specify a net for pins in the netlist.

*list\_of\_pin\_numbers*      A list of alphanumeric pin numbers in the device. Separate each pin number with a space or tab.

### GROUND

Optional. The GROUND record ties the pins of a device to a particular ground signal. Allegro PCB Editor uses this ground information when you use the `netin` command to create a database. The net name given is the default net for the defined ground pins. This net is assigned to these pins if there is no net specified for the pins in the netlist.

```
GROUND <net_name> ; <list_of_pin_numbers>
```

*net\_name*                      The default net for the ground pins listed. The net is assigned to these pins if you do not specify a net for pins in the netlist.

*list\_of\_pin\_numbers*      A list of alphanumeric pin numbers in the device. Separate each pin number with a space or tab.

### NC

Optional. The NC (no connect) record defines pins on a device that will never be connected.

NC ; <list\_of\_pin\_numbers>

*list\_of\_pin\_numbers* A list of alphanumeric pin numbers in the device. Separate each pin number with a space or tab.

## **PACKAGEPROP**

Optional. The PACKAGEPROP record(s) identifies definition properties that apply to the device. For further information on definition properties that you can assign to a device, see [Specifying Definition Properties in a Device File](#).

## **END**

Optional, but highly recommended. This ends the device file description. If you do not enter an END statement, Allegro PCB Editor issues a warning during the `netin` process.

# **Guidelines for Creating a Device File**

You can create a device file by choosing *File – Create Device* ([create device](#) command) in the Symbol Editor, or you can observe the following guidelines when creating device files:

- Follow DOS naming conventions and use the file extension `.txt`.
- Enter text in upper- or lowercase letters.
- Add any comments as the first line in the file, enclosed in parentheses.  
Allegro PCB Editor ignores comments. Include the file name as a comment.
- Separate fields in a record with spaces, unless the syntax for a record specifies a different separator.
- Enter each record on a separate line.
- Enclose a text string that contains non-alphanumeric characters in single quotes.
- End the device file with an END statement to avoid receiving a warning.

## **Creating a Device File**

1. Change to the directory where you are placing the device file.
2. Use a text editor, such as vi or Notepad, to write the device file.

3. To include comment information at the beginning of the file, enter a beginning parenthesis followed by the text and a closing parenthesis.

For example:

```
(device file for 7400)
```

4. Enter the device file records as described in [Device File Records](#).

For details about multiple functions, see [Specifying Multiple Functions in a Device File](#).  
If you want to include definition properties for the device, see [Specifying Definition Properties in a Device File](#).

5. it is recommended that you end the file with the following keyword:

```
END
```

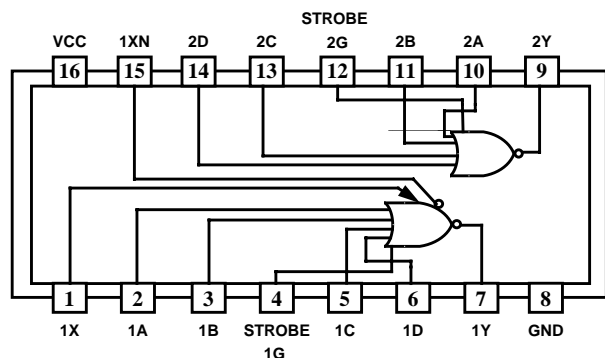
6. Save the file.

## Specifying Multiple Functions in a Device File

If a device has different functions, you must define additional PINORDER, PINUSE, PINSWAP, and FUNCTION records in the device file for each different function, so that Allegro PCB Editor can determine which function slot to assign.

For example, in Figure 5-2, all the functions are of the same type, 7400 NAND2 gates. Allegro PCB Editor assigns function slots.

**Figure 5-2 7423 Component**



If you specify only 7423 in the PINORDER and FUNCTION lines of the device file for the package shown, Allegro PCB Editor cannot determine which 7423 function is referenced—the four-input NOR gate or the expanded NORX gate.

## Allegro PCB Editor User Guide: Defining and Developing Libraries

### Preparing Device Files

You must define two different PINORDER, PINUSE, PINSWAP, and FUNCTION records, as Figure 5-3 shows.

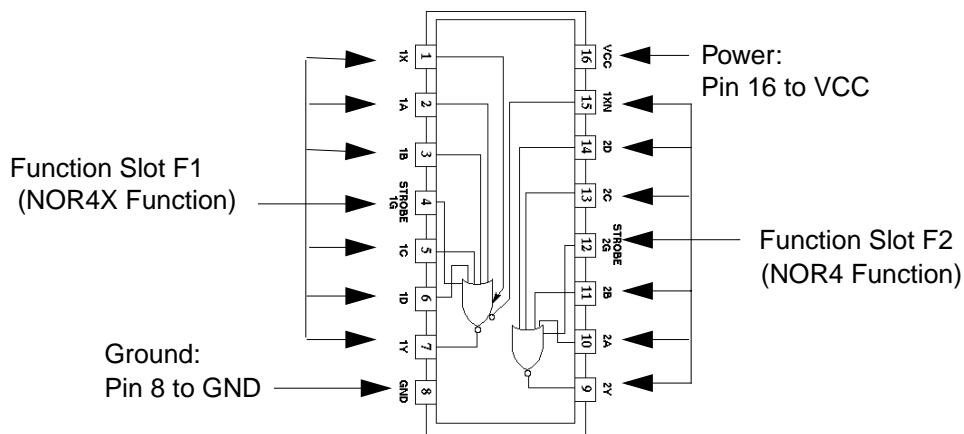
**Figure 5-3 Two Functions in a Device File**

```
(Device file for 7423)
PACKAGE DIP16
CLASS IC
PINCOUNT 16

PINORDER  NOR4X  A   B   C   D   G   X  XN  Y
PINUSE     NOR4X  IN  IN  IN  IN  IN  IN  IN  OUT
PINSWAP    NOR4X  A   B   C   D               1  15  7
FUNCTION F1 NOR4X  2   3  5   6   4               1  15  7

PINORDER  NOR4    A   B   C   D   G   Y
PINUSE     NOR4    IN  IN  IN  IN  IN  OUT
PINSWAP    NOR4    A   B   C   D
FUNCTION F2 NOR4   10  11  13  14  12   9

POWER VCC; 16
GROUND GND; 8
END
```



## Specifying Definition Properties in a Device File

You can assign properties to a device by creating a PACKAGEPROP property record for each property in the following format:

```
PACKAGEPROP <property_name> <property_value>
```

See the *Allegro Platform Properties Reference* for more information

**<property\_name>** Specifies the property you are assigning to the device. These are the property names, described in the rest of this section:

- From the VOLT\_TEMP\_MODEL property group, you can use MAX\_POWER DISS.
- DEVICE\_LABEL
- TOL
- VALUE
- PART\_NUMBER
- INSERTION\_CODE
- TERMINATOR\_PACK
- HEIGHT
- ALT\_SYMBOLS

**<property\_value>** Specifies the value of the property you are defining. See the property descriptions for information about values.

## Checking a Device File

Before you create a design database, use the `dev_check` command to make sure the device files correspond to the correct package symbols. The *Allegro PCB and Package Physical Layout Command Reference* describes the command.

## Reviewing the dev\_check.log File

The names of the devices and their matching package names are generated in the `dev_check.log` file. If you specify only one device file in the `dev_check` command, only the information for that file appears in the log.

The log file identifies which devices have errors by printing

## Allegro PCB Editor User Guide: Defining and Developing Libraries

### Preparing Device Files

---

\*\*\* ERROR: <message>

beneath each device file that contains an error, and provides the total number of errors at the end of the `dev_check.log` file.

### Error Messages

The following is a list of common errors that you might find in the log file:

An error occurred while finding the symbol

This can indicate a problem with the symbol. For example, there could be a database error.

The symbol is not found

Check to see if a .psm file was created. Also check the environment path and naming conventions for the symbol and the device file.

The symbol has an extra pin that the device file does not

The pin number of the extra pin is provided so that you can check the pin numbers.

The device has an extra pin that the symbol does not

The pin number of the extra pin is provided so that you can check the pin numbers.

The symbol does not have a reference designator

The symbol is missing a reference designator.

### Sample dev\_check.log File

```
(-----)
( DEVICE FILE CHECKER )
(
( Drawing : dev_check.brd )
( Date/Time : Tue Sep 28 15:54:24 1993 )
(-----)
Checking device Z8581_1, with symbol DIP18_3.
Checking device Z8001_52_1, with symbol PLCC52.
Checking device XTAL_2, with symbol CRYSTAL.
Checking device WR_ENABLE_1, with symbol SOIC20.
Checking device TERMINATOR_33, with symbol SIP8.
Checking device SIPRES_4_7K, with symbol SIP10.
Checking device RESISTOR_4_7K, with symbol SMDRES.
```

## Allegro PCB Editor User Guide: Defining and Developing Libraries

### Preparing Device Files

---

```
Checking device IOBUF_1, with symbol SOIC20.
Checking device INTERRUPT_1, with symbol SOIC20.
Checking device HM6116_2_1, with symbol SOIC24.
Checking device HM4864_2_1, with symbol SIP30.
Checking device ECON_1, with symbol ECON62_100.
Checking device DIPRES_33, with symbol SOIC16.
Checking device CLEAR_1, with symbol SOIC20.
Checking device CAPACITOR_470PF, with symbol SMDCAP.
Checking device CAPACITOR_0_01UF, with symbol SMDCAP.
Checking device 74LS51_1, with symbol SOIC14.
Checking device 74LS393_1, with symbol SOIC14.
Checking device 74LS373_1, with symbol DIP20_3.
Checking device 74LS373_1, with symbol SOIC20.
Checking device 74LS373_1, with symbol SOIC20.
Checking device 74LS373_1, with symbol SOIC20_PE.
***ERROR: Symbol not found.
Checking device 74LS32_1, with symbol SOIC14.
Checking device 74LS245_1, with symbol SOIC20.
Checking device 74LS157_1, with symbol SOIC16.
Checking device 74LS138_1, with symbol SOIC16.
Checking device 74LS08_1, with symbol SOIC14.
Checking device 74LS04_1, with symbol SOIC14.
Checking device 74LS04_1, with symbol DIP14_3.
Checking device 74LS04_1, with symbol SOIC14_PE.
Checking device 74F74_1, with symbol DIP14_3.
Checking device 74F02_1, with symbol DIP14_3.
Checking device 74F00_1, with symbol DIP14_3.
Checking device 2716_1_1, with symbol SOIC24.
1 Error(s) occurred.
Tue Sep 28 15:54:19 2005   Page 1
Allegro NETLIST IN Log File
=====
Netlist File Name: `/usr1/QA/pcb/devices/dev_check.net' Layout File Name:~
`/usr1/QA/pcb/devices/dev_check.brd'
=====
$PACKAGES
!2716_1_1 ; U1
!74f00_1 ; U2
!74f02_1 ; U3
!74f74_1 ; U4
!74ls04_1 ; U5
```

## Allegro PCB Editor User Guide: Defining and Developing Libraries

### Preparing Device Files

---

```
!74ls08_1 ; U6
!74ls138_1 ; U7
!74ls157_1 ; U8
!74ls245_1 ; U9
!74ls32_1 ; U10
!74ls373_1 ; U11
!74ls393_1 ; U12
!74ls51_1 ; U13
!Altos ; U14
```

^

ERROR: Cannot find device file for 'ALTOS'.

```
-----
!capacitor_0_01uf ; U15
!capacitor_470pf ; U16
!clear_1 ; U17
!devices ; U18
```

^

ERROR: Cannot find device file for 'DEVICES'.

```
-----
!dipres_33 ; U19
!econ_1 ; U20
!hm4864_2_1 ; U21
!hm6116_2_1 ; U22
!interrupt_1 ; U23
!iobuf_1 ; U24
!resistor_4_7k ; U25
!sipres_4_7k ; U26
!terminator_33 ; U27
!wr_enable_1 ; U28
!xtal_2 ; U29
!z8001_52_1 ; U30
!z8581_1 ; U31
$END
```

```
=====
End of NETLIST IN Syntax/Logic Check
=====
```

Total Netlist Warnings = 0.

Total Netlist Errors = 2.

Tue Sep 28 15:54:19 1993 Page 1

Parsing device file: '/usr1/QA/pcb/devices/2716\_1\_1.txt'.

```
=====
```



## Allegro PCB Editor User Guide: Defining and Developing Libraries

### Preparing Device Files

---

```
(DEVICE FILE: 2716_1_1 - used for device: '2716-1-1')

PACKAGE SOIC24
CLASS IC
PINCOUNT 24
PINORDER '2716-1-1' '-CS' '-OE' 'A<0>' 'A<10>' 'A<1>' 'A<2>' 'A<3>' 'A<4>'
'A<~5>' 'A<6>' 'A<7>', 'A<8>' 'A<9>' 'Q<0>' 'Q<1>' 'Q<2>' 'Q<3>' 'Q<4>'
'Q<5>' 'Q<6>' 'Q<7>' VPP
PINUSE '2716-1-1' IN IN IN IN IN IN IN IN IN IN IN IN IN IN IN TRI TRI TRI TRI
TRI T~RI TRI TRI IN
FUNCTION G1 '2716-1-1' 18 20 8 19 7 6 5 4 3 2 1 23 22 9 10 11 13 14 15 16
17 21
GROUND GND ; 12
POWER VCC ; 24
PACKAGEPROP MAX_POWER DISS '.5'
END
```



---

## Using Technology and Parameter Files

---

Both technology and parameter files are essential components in the process of leveraging re-usable design information during the database creation stage of the PCB design flow. Technology (tech files) are used to enter cross-section, drawing and constraint settings into the database while parameter files are used to enter settings for global and application-based functions.

### Working with Tech Files

Technology files, also called tech files, contain the following types of neutral design data:

- Drawing parameters (includes units and design extents)
- Layout cross section
- DRC modes
- Spacing, physical, electrical, and design constraints
- User property definitions

Tech files are eXtensible Markup Language (XML)-based files. The tech file syntax is described in the Document Type Definition (DTD) file located at:

```
<cdsroot>/share/pcb/xml-formats/techfile.dtd
```

If you have families of designs that share similar technologies, you should create design-specific tech files pertaining to these families. You can create these tech files by exporting the rules from an existing design or using one of the physical editors to set up the required design rules in an empty design and then export the data. The tech file name should describe the rules contained within the file.

You can have tech files for specified design information. For example, you can have one set of tech files containing only stack-up information; another set, constraints; and a final set, corporate user property definitions.

When using tech files, if you find that a constraint value does not contain an explicit unit, then it is assumed that the value is in the design units specified in the header of the tech file. If the tech file is read into a design with different units, the values are converted to the current Allegro database design units.

Cadence recommends that you place tech files in a central location using the CDS\_SITE strategy (`<CDS_SITE>/pcb/tech`). By locating technology files in a centralized directory, you promote the sharing of design rules across similar designs. Refer to the *Getting Started User Guide* in your product documentation for additional information on CDS\_SITE.

Tech files are located using the TECHPATH environment variable. You can manage this variable in the User Preferences Editor (`enved` command).

Tech files use the `.tcf` extension. Your layout tool also supports pre-Release 16.0 tech files using the `.tech` extension. It automatically uprevs the pre-Release 16.0 tech files to the Release 16.0 `.tcf` files. Tech files created in the current release are forward-compatible with future releases (an uprev process may be required). Tech files are not backward-compatible. For example, you cannot import a technology file that you created in or upreved to Release 15.0 into a Release 14.0 design.

## Accessing Tech Files

Using any Allegro design database, you can import, export, or compare tech files.

### Exporting Tech Files

You can create a tech file from an existing design. Use one of the following methods to export the file:

- *File – Export* menu command in Constraint Manager
- *File – Export* menu command (`techfile out`) in one of the physical design editors
- `techfile` batch command

Currently, the Constraint Manager export command offers more options than the other methods.

### Importing Tech Files

You can import a tech file into an existing design. Use one of the following methods to import the file:

- *File – Import* menu command in Constraint Manager
- *File – Import* menu command (techfile in) in one of the physical design editors
- The new board wizard (layout wizard)
- techfile batch command

During import, if an error exists in the tech file, the tool continues reading the file, and writing warning and error messages, but does not create an updated design. After import, check the `techfile.log` for any warnings or errors.

In *Overwrite* mode, importing a technology file overwrites any values that already exist in the design. If a constraint does not exist in the design, it is added.

Currently, the Constraint Manager import command offers more options than the other methods.

## Comparing Tech Files to Designs

Comparing a tech file to a design can help determine if the values in a design conform to the intended values residing in the tech file, before you send the design to manufacturing.

The `techfile.log` records the values of the file and the design for side-by-side comparison. Only the constraints specifically contained in the tech file are checked against their counterparts in the design. The `techfile.log` also contains any warnings or errors encountered while reading the tech file.

Use one of the following methods to compare a tech file to a design:

- *Tools – Technology File Compare* menu command in one of the physical design editors (techfile compare)
- techfile batch command
- *File – Import* menu command in Constraint Manager

## Upreving Tech Files

If you created tech files in previous versions of Allegro, you can uprev them to the latest XML version (`.tcf`). The tool automatically uprevs the tech file when you import it but you may want (for performance reasons) to update all your tech files to the latest version by using the techfile batch command.

DRC analysis modes are not layer-specific in the new format. The tool updates the database as follows:

- If any layer is set to ALWAYS, the tool sets all layers to ALWAYS.
- If any layer is set to BATCH, the tool sets all layers to BATCH.

Uprev restrictions include the following:

- Comments in the old version of the tech file (.tech) do not appear in the new file.
- Pre-Release 16.0 tech files do not formally support segmented data. For example, in Release 15.7 you could manually edit a tech file so that it contained only the user-defined property section. When you uprev this tech file, it will be updated to a full tech file and you will need to manually edit the result to restore it to its original intent.

**Note:** Tech files are not backward-compatible. You cannot import a technology file for a newer release into an older release.

### Locking Constraint Sets

The tech file supports op flags that determine whether all values in a physical, spacing, or electrical constraint set in a design are locked from editing. Importing a tech file is the only way to lock, and later unlock, these items.

The op flag syntax that locks or unlocks constraint sets is defined in the DTD file located at:

```
<cdsroot>/share/pcb/xml-formats/techfile.dtd
```

When a constraint set is locked, its values cannot be changed from within Allegro PCB Editor. You cannot override a locked constraint value by setting the corresponding property on particular design elements.

A locked constraint set can only be changed by importing one of these types of tech file:

- A tech file that changes the constraint values (but may still keep the constraint set locked).
- A tech file that clears the op flag for the constraint set. The absence of the read only op flag indicates the CSet is editable.

### Examples of Locked and Unlocked Items in Tech Files

Read only or locked:

```
</attribute>
```

```
<objectFlag>fObjectReadOnly</objectFlag>
```

Editable or unlocked:

```
</attribute>
```

```
<objectFlag>fObjectNOTReadOnly</objectFlag>
```

## Technology Constraints File

The .tcf file supports all the information in the .tech file. No design specific information such as nets, xnets or buses are saved to the tech file.

Section	Description
Drawing	Consists of design units, design extents, and origin. The drawing extents and origin are used only for new boards.
Cross-section	Defines the design stackup.
User-defined constraint definitions	Contains any property dictionary entries.
opFlags	Specifies the DRC analysis mode setting for all constraints.
Constraints	Contains all objects, for example, CSets, Net Classes, and Regions, and their constraints.

## Working with Parameter Files

Database parameter files contain customized parameter records, which are those that have a single instance in the database and include the following types of customized settings:

- Design settings (such as global values and grid settings)
- Artwork
- Text size settings
- Application or command parameters (includes auto rename, auto assignment, auto silkscreen, global dynamic fill, autovoid, export logic, drafting, gloss line fattening, gloss dielectric generation, Options window tab settings, test prep, automatic placement, auto swap, thieving, backdrill, interactive flow planner, and Signoise analysis)

Global values are those such as dynamic fill; grid settings; artwork format; and Xhatch style, line width, spacing, and angle, for example.

### Parameter File Syntax

Parameter files are eXtensible Markup Language (XML)-based files. The database parameter file syntax is described in the Document Type Definition (DTD) file located at:

```
<cdsroot>/share/pcb/xml-formats/parameter.dtd
```

Database parameter files use the `.prm` extension and are located using the `PARAMPATH` environment variable in the User Preferences Editor, available by choosing *Setup – User Preferences* (enved command).

## Accessing Parameter Files

Using any Allegro design database, you can import or export parameter files.

### Exporting Parameter Files

You can create a parameter file from an existing design. Use one of the following methods to export the file:

- *File – Export – Parameters* (param\_out command)
- techfile batch command



After export, view error messages and other process information in the current or last generated `param_write.log` report.

## Importing Parameter Files

When you initially begin a design, import the `.prm` file from a centrally located corporate library, or your local working directory. The environment variable `PARAMPATH` determines the path of library-based files.

Use one of the following methods to import the file:

- *File – Import – Parameters* (`param in` command)
- The new board wizard (`layout wizard`)
- `techfile` batch command

If a parameter record of the same name already exists in the design database, the `.prm` file overwrites the existing record when you import. When no parameter name exists, a new record is created. After import, view error messages and other process information in the current or last generated `param_read.log` report.



---

## Generating Allegro PCB Editor Libraries

---

Allegro PCB Editor lets you define new libraries based on libraries from existing Allegro PCB Editor designs. This is useful for:

- Creating a library for a design originating from a different CAE or physical layout system, where the original library is not provided
- Creating a library that is compatible with the padstacks and symbols already in a design, when the design is from an earlier library revision
- Recovering libraries that may have been lost by obtaining the library data from a design that contains the correct library data
- Creating a clipboard library that contains elements from designs or symbol drawings

You can create libraries from existing Allegro PCB Editor designs by extracting:

- Device files
- Padstacks
- Symbols

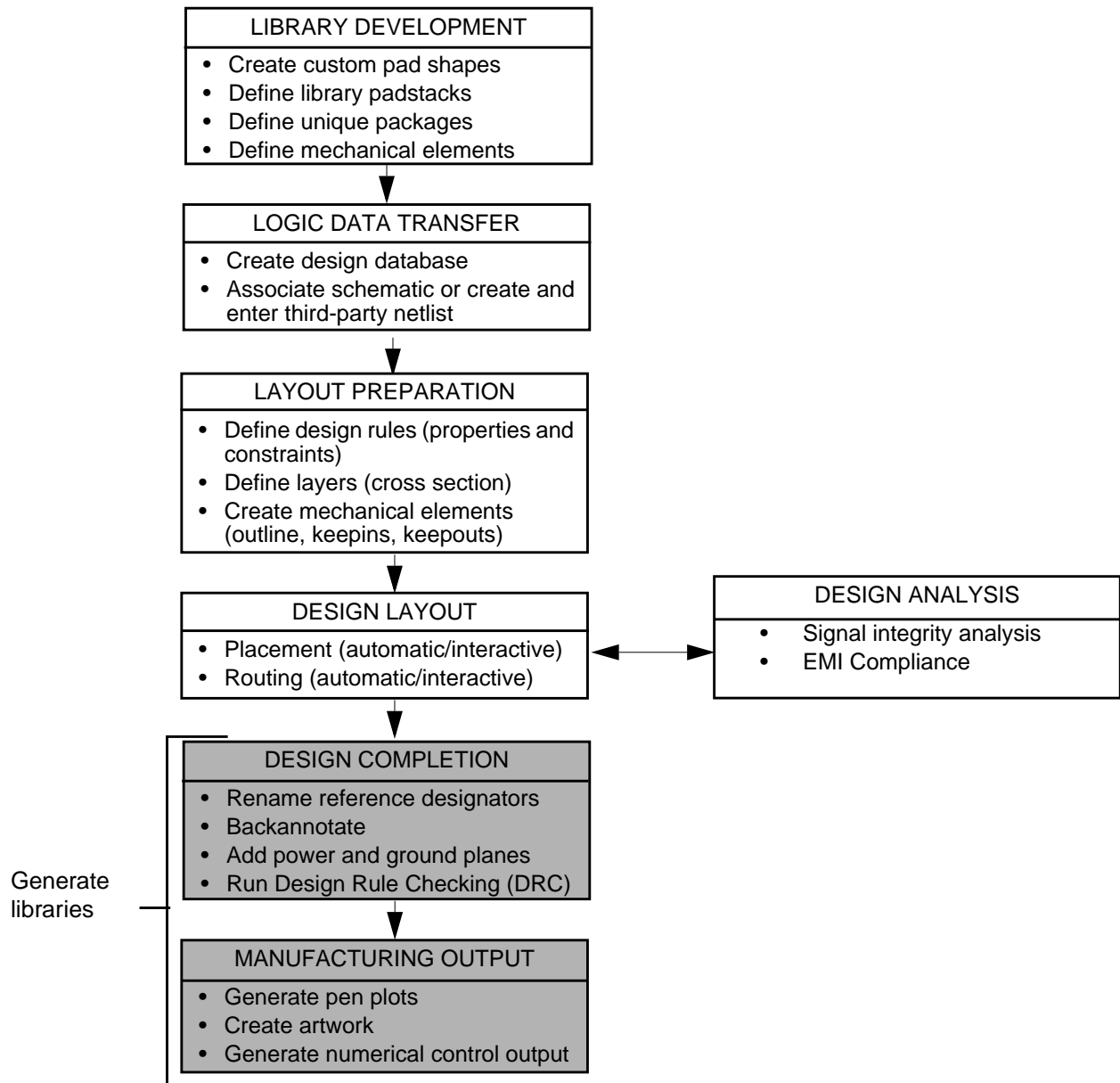
Creating new libraries based on existing designs occurs late in a design flow or following its completion, as shown in Figure [7-1](#).

# Allegro PCB Editor User Guide: Defining and Developing Libraries

## Generating Allegro PCB Editor Libraries

---

**Figure 7-1 Generating new libraries in a design flow**



## Creating Libraries from Existing Designs

To obtain device files, padstack definitions, and symbol definitions from an existing design, choose *File – Export – Libraries* (dlib command) (dump libraries), described in the *Allegro PCB and Package Physical Layout Command Reference*.

## Creating Device and Symbol Files with Batch Commands

You can obtain device files and symbol definitions from an existing Allegro PCB Editor design by running these batch commands:

- The create\_devices batch command creates device files.
- The create\_sym batch command creates symbol definitions.

Both commands are described in the *Allegro PCB and Package Physical Layout Command Reference*.

## Creating a Clipboard Library

You can create a library (directory) of clipboard files for use in designs and symbol drawings. These are elements you can place in such a library:

- Connect lines
- Connect points
- Connect line segments (including connect arc segments)
- Drafting symbols
- Filled rectangles
- Groups
- Lines
- Line segments (including arc segments)
- Package, mechanical, and format symbols
- Pins
- Rectangles
- Shapes

- Text
- Vias

These are the elements you cannot place in clipboard files:

- Components
- DRC violations
- Figures
- Functions
- Nets
- Ratsnest lines

You can copy and paste these elements between designs or symbol drawings by choosing *File – Import – Sub\_Drawing* (clppaste command) and *File – Export – Sub\_Drawing* (clpcopy command), described in the *Allegro PCB and Package Physical Layout Command Reference*.

## Setting the CLIPPATH environment

The clippath environment variable in the Allegro PCB Editor global or local environment file identifies the directory in which clipboard elements are stored. The default setting for the clippath environment variable is the current directory from which Allegro PCB Editor is run, as indicated by a period:

```
set clippath = .
```

**Note:** You can check the current setting for the clippath variable by entering `set` at the command line. A list of the defined environment variables appears. See the *Getting Started with Physical Design* user guide in your documentation set for details on the various environment variable settings.

To change the directory in which the clipboard elements are stored, do one of the following:

- Temporarily change the clippath using the set command, described in the *Allegro PCB and Package Physical Layout Command Reference*.
- Change the clippath setting in the local `env` environment file. For details, see the *Getting Started with Physical Design* user guide in your documentation set.

---

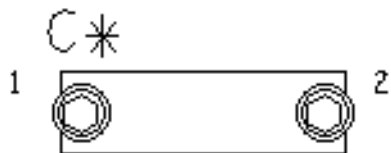
## Package Symbol Library

---

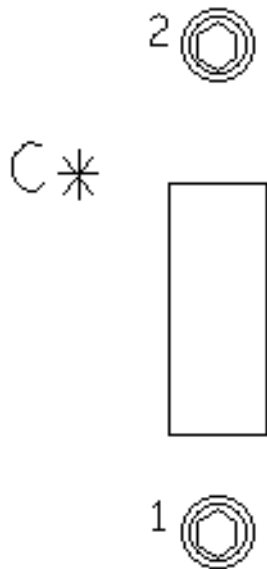
This chapter illustrates some of the package symbols available in the library.

## Capacitors

### cap300




### cap400





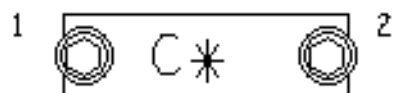
## cap600

2 

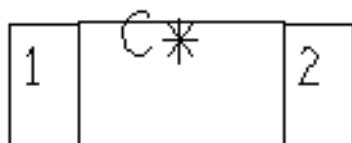


1 

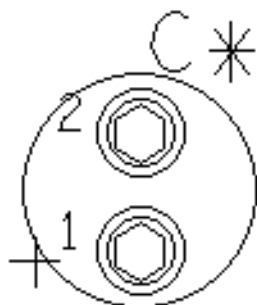
## dipcap



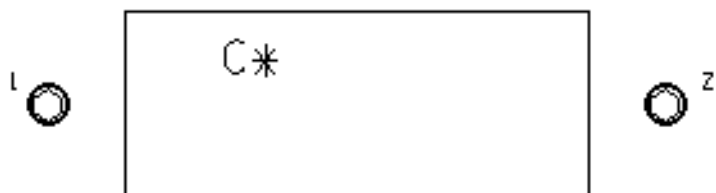
## smdcap



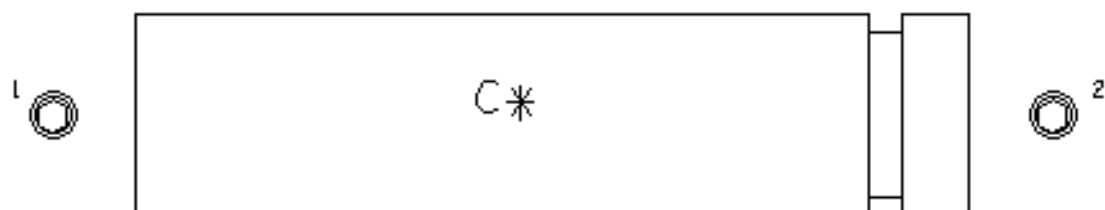
### cap196



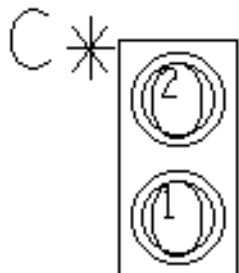
### cap1000



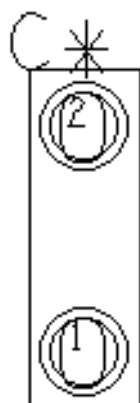
### cap1500



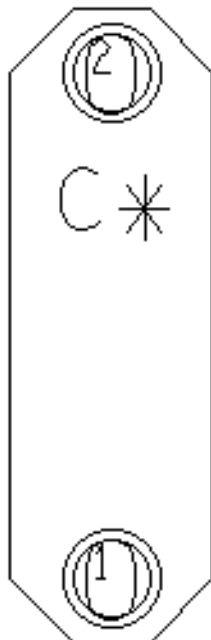
**capck05**



**capck06**



**capck60**



**capck62**



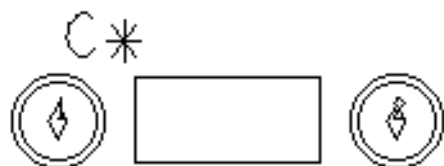
## case17-02



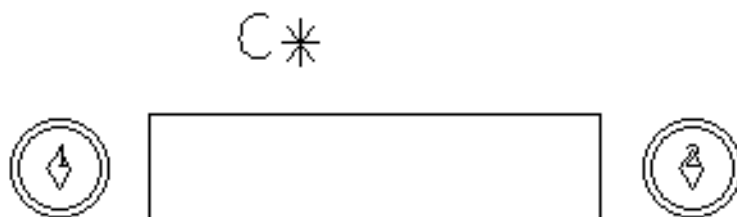
U\*



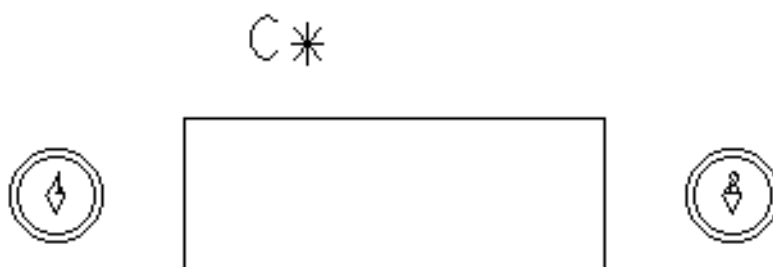
## ck12-10pf



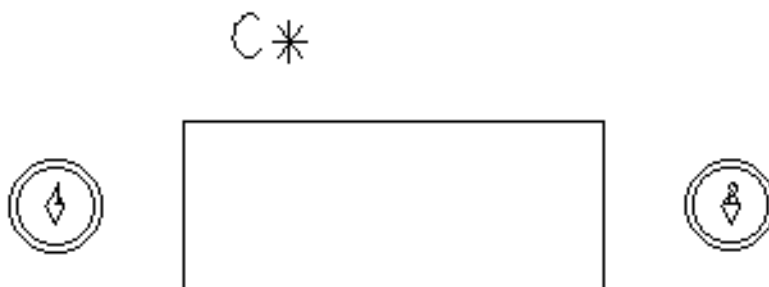
**ck13-10pf**



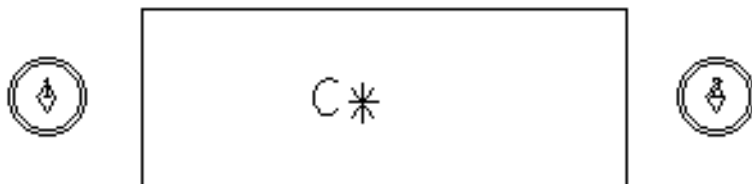
**ck14-10pf**



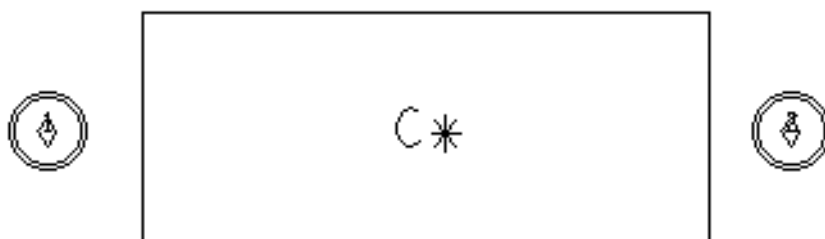
**ck15-10pf**



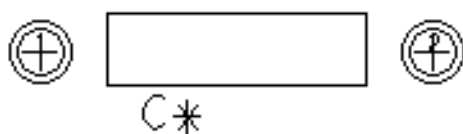
### ck16-10pf



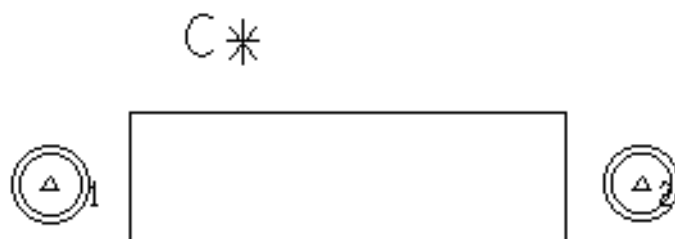
### ck17-10pf



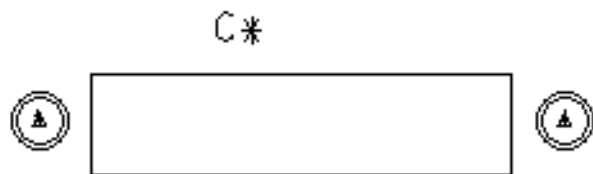
### cy10



### cy15

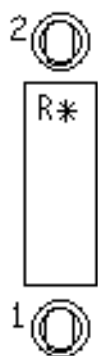


**cy20**

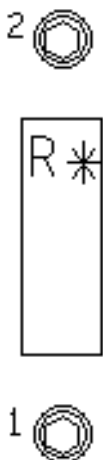


## Resistors

**res400**

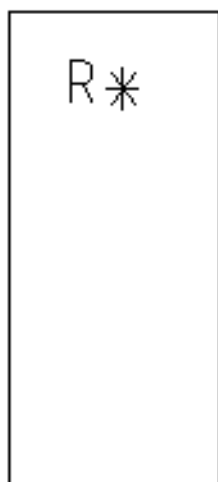


**res500**





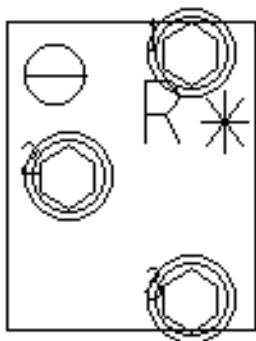
### res1000



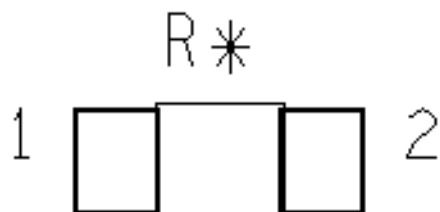
### res800



## resadj

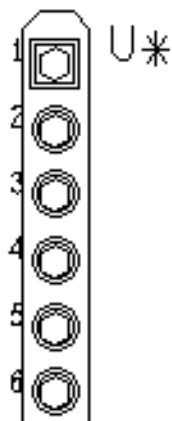


## smdres

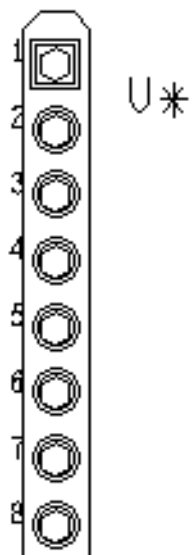


## SIPs

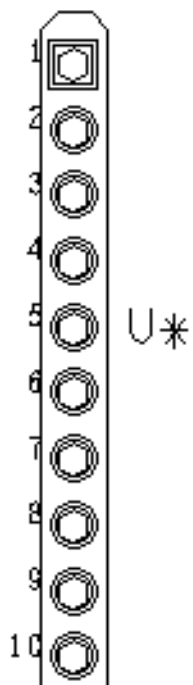
### sip6



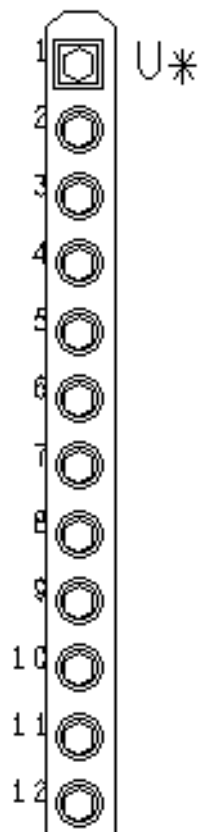
## sip8



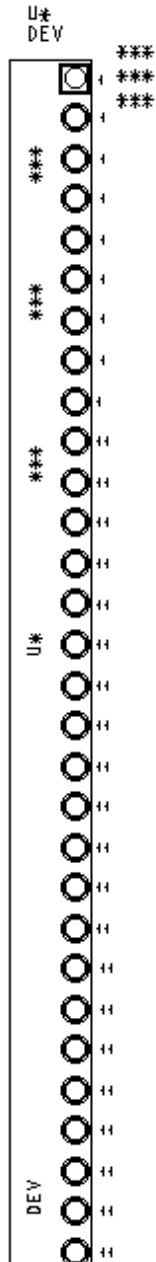
## sip10



## sip12

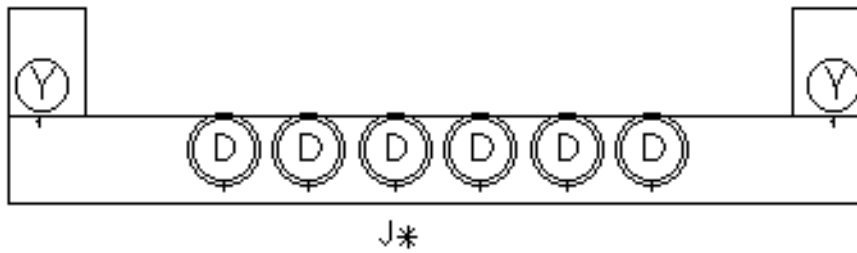


## sip30

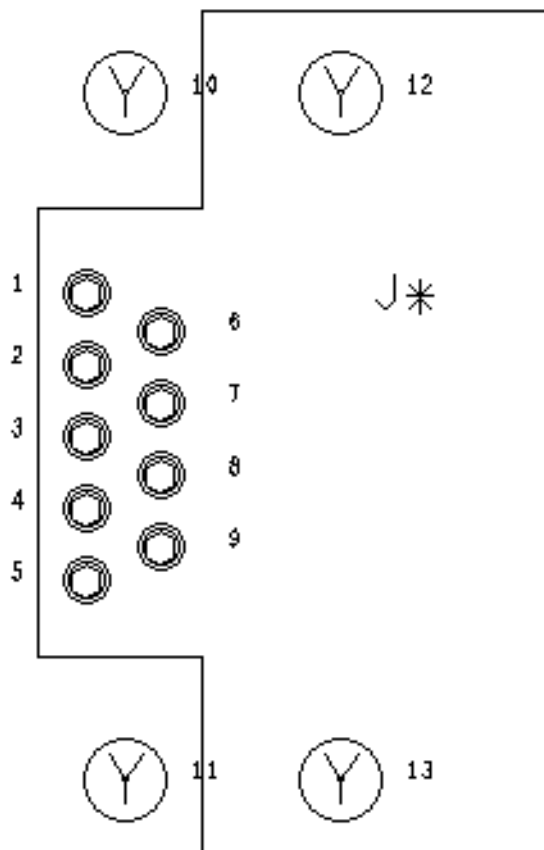


## Connectors

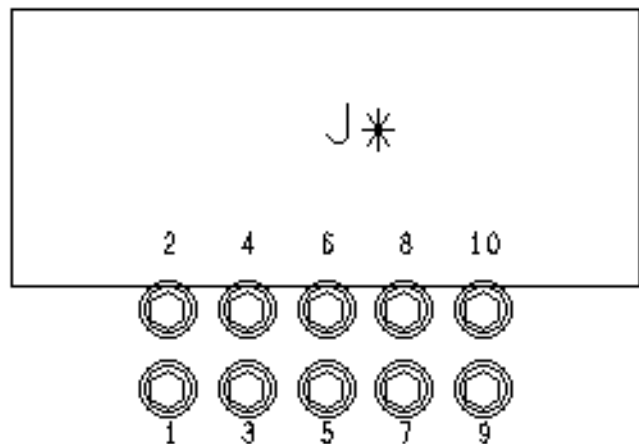
### conn6



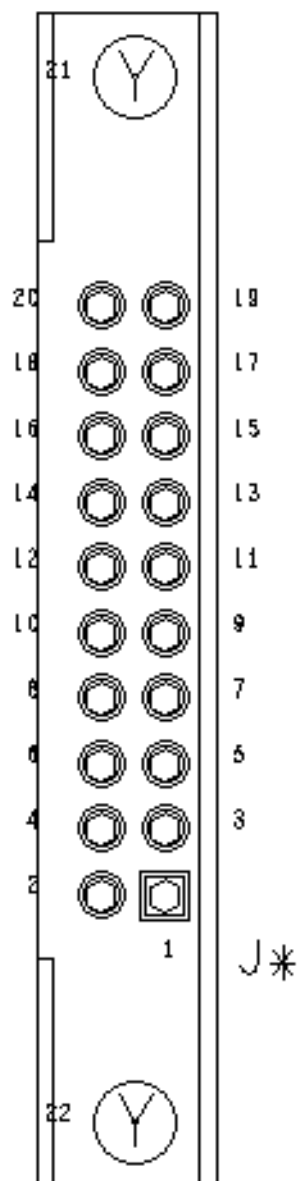
### conn9



## conn10

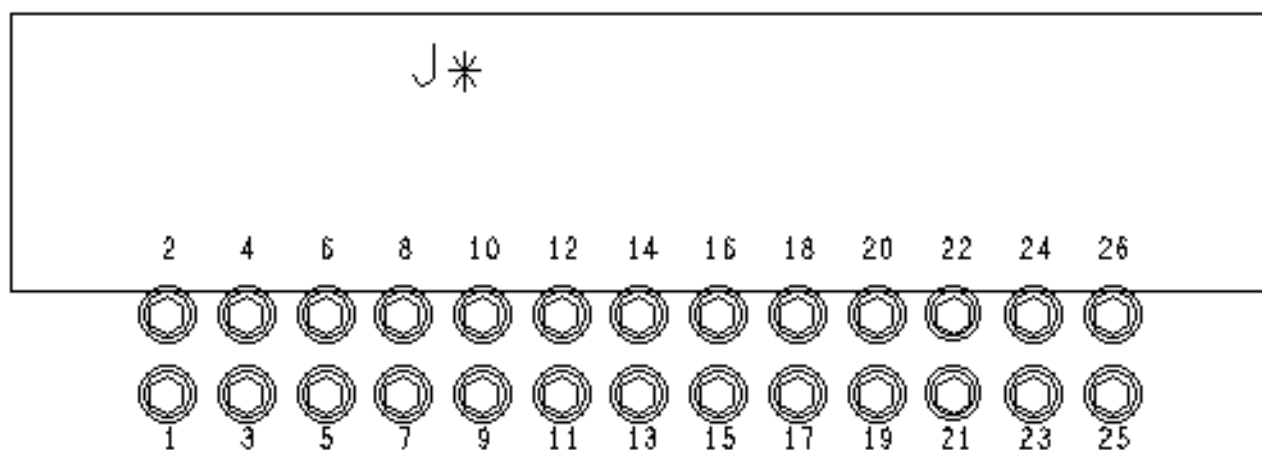


## conn20

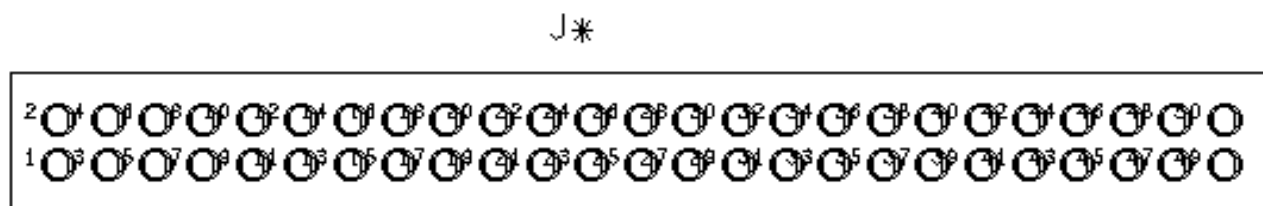




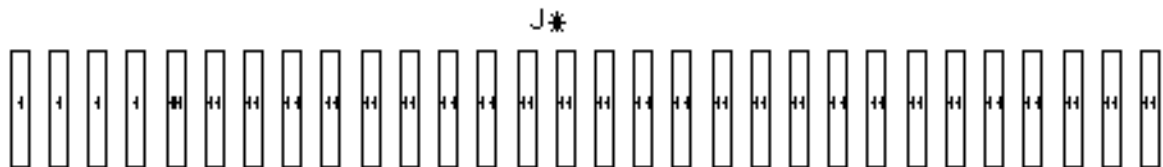
## conn26



## conn50



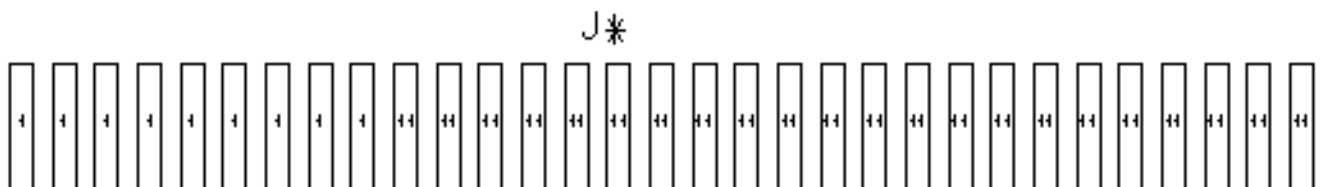
## multiconn30



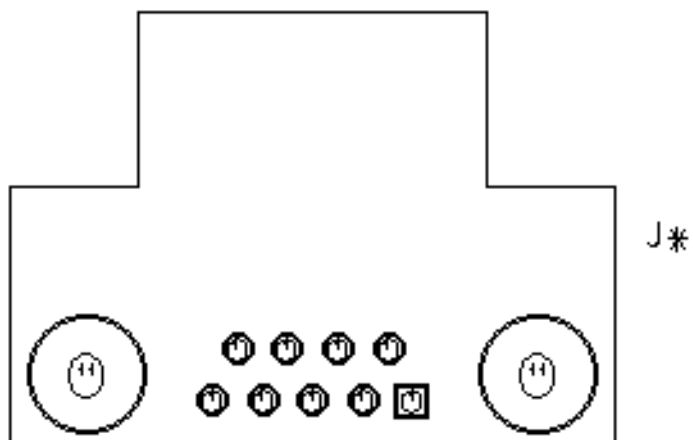
## multicon43



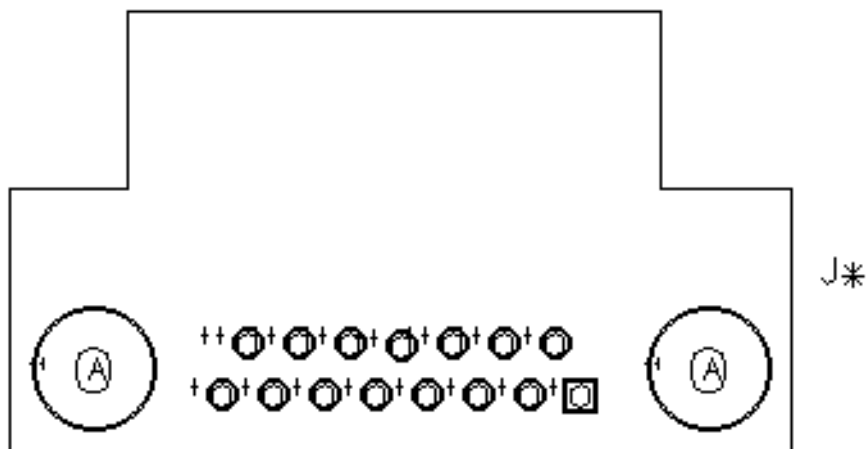
## ibmconn



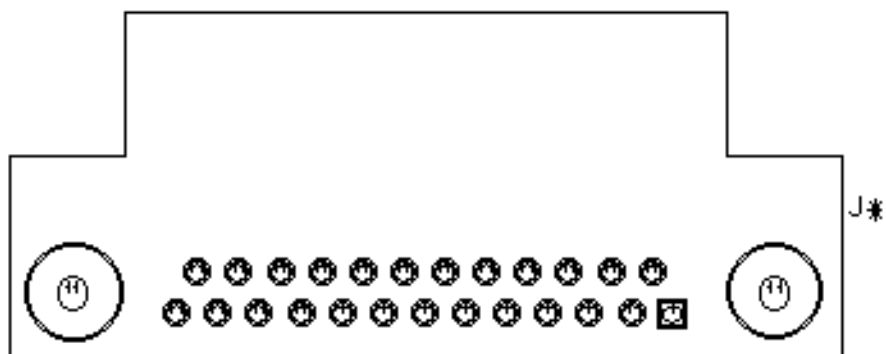
## db9



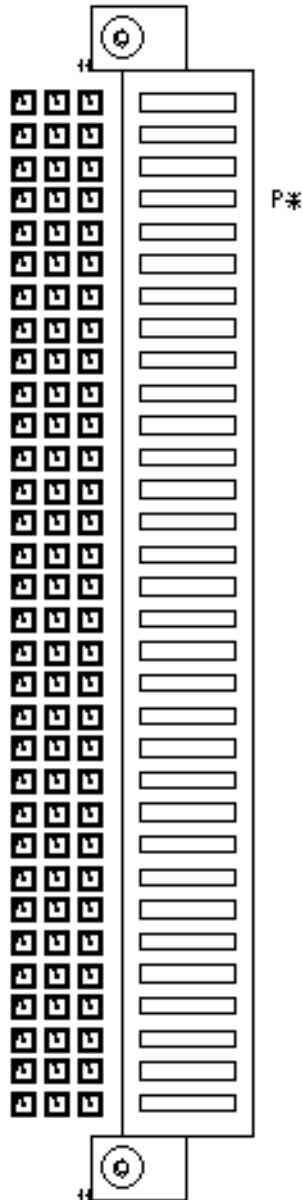
## db15



## db25

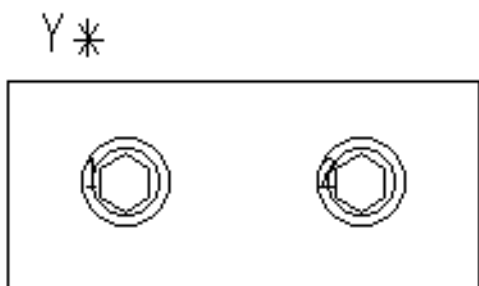


## eurocon

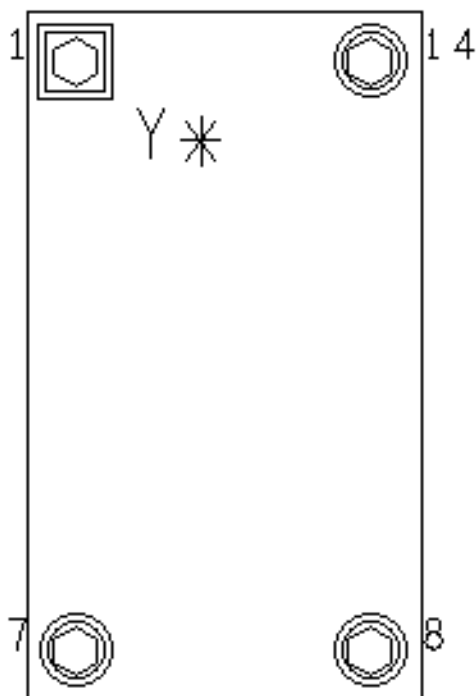


## Crystals

### crys11mhz

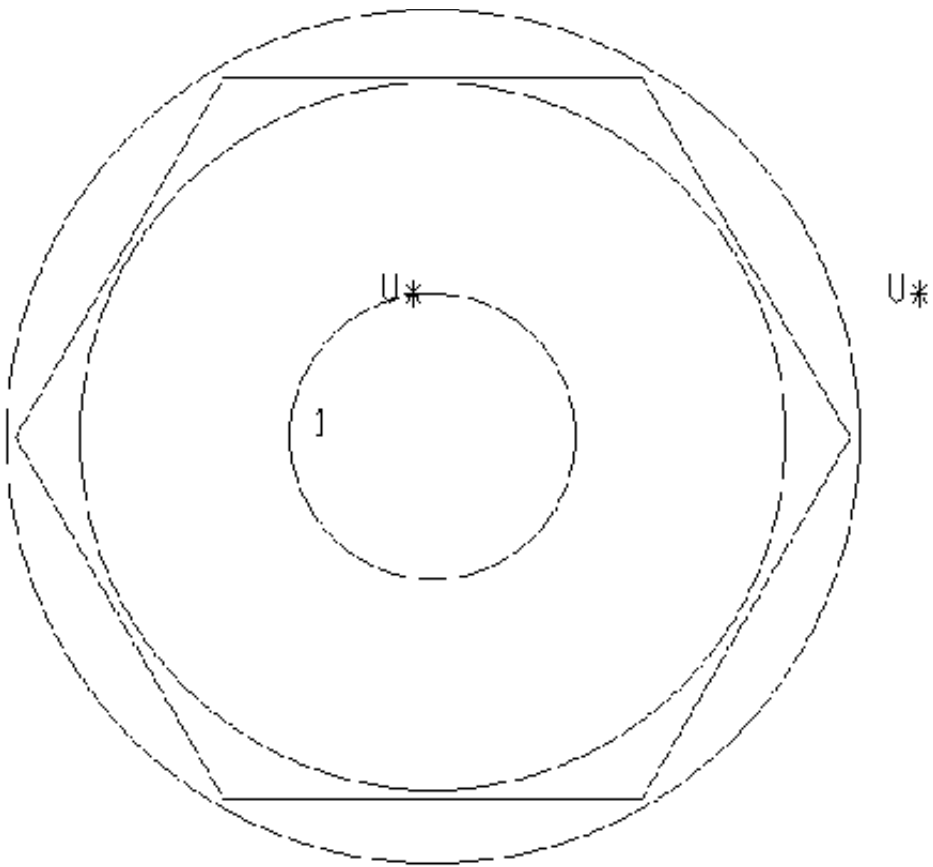


### crys14

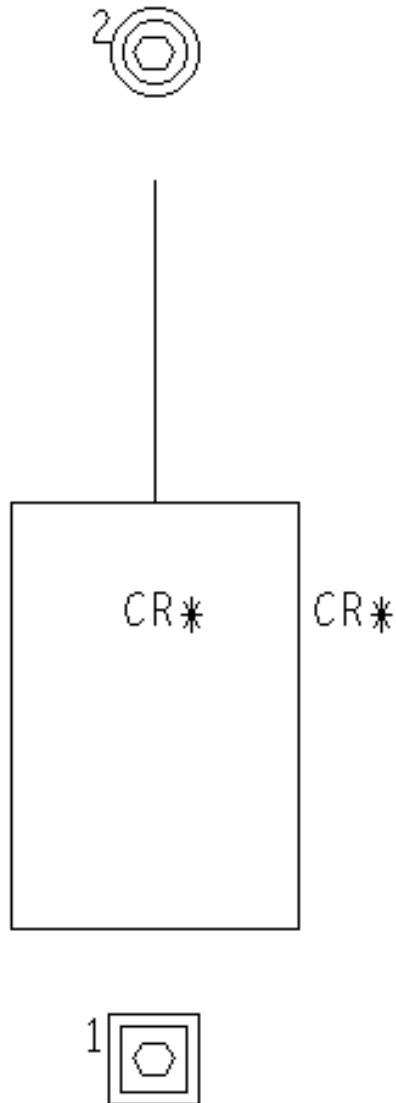


## Diodes

do5

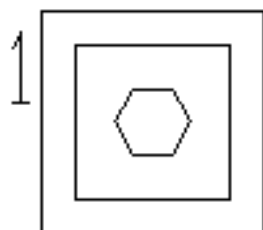
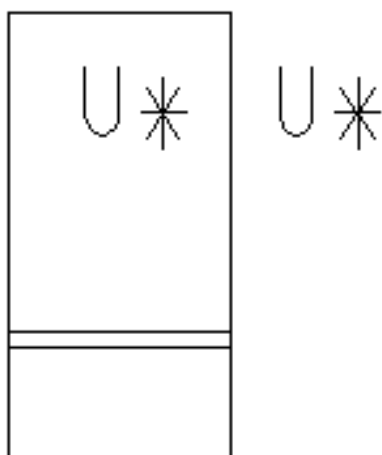


**do13**

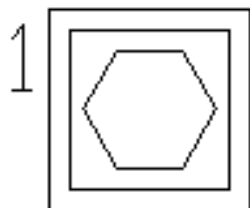
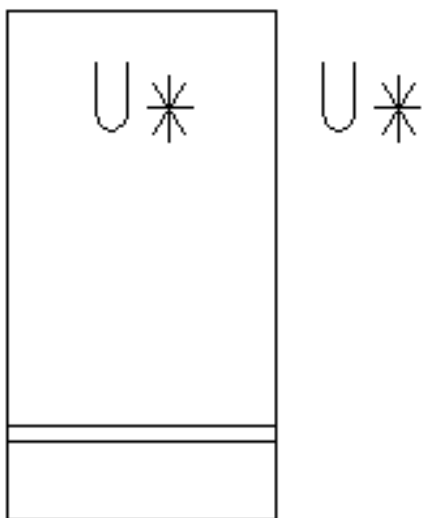




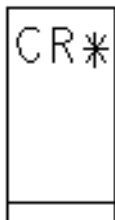
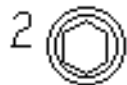
## do35



**do41**



## dio400

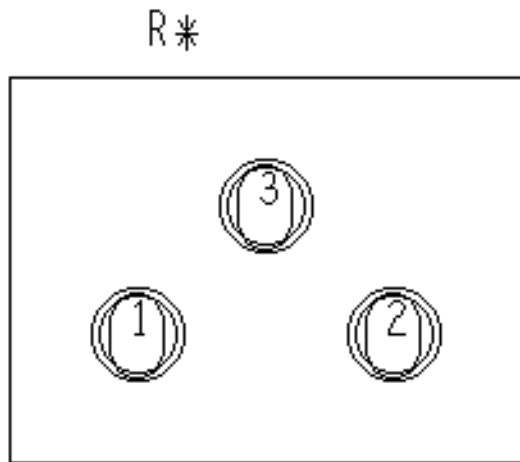


## dio500



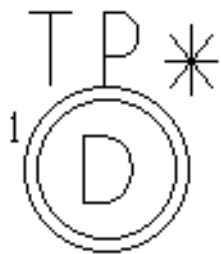
## Potentiometer

pot



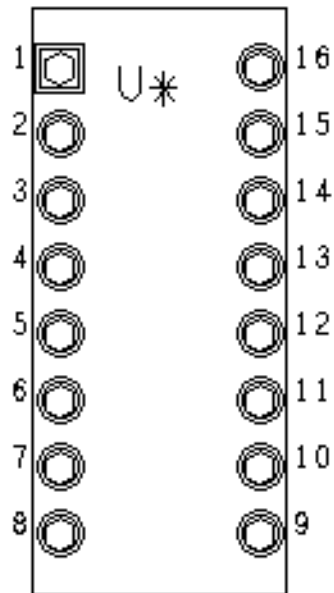
## Test Point

tp

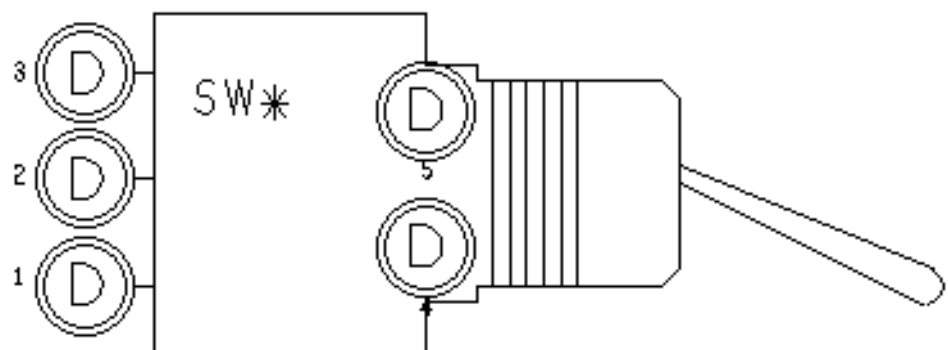


## Switches

### dipswitch

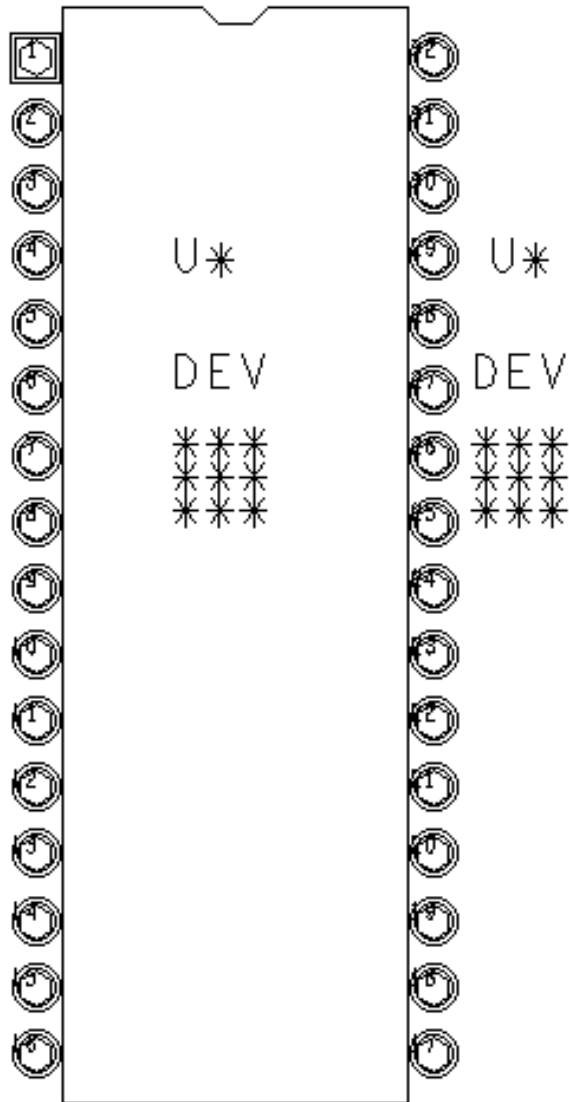


### switch

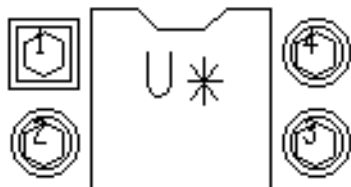


## DIPs

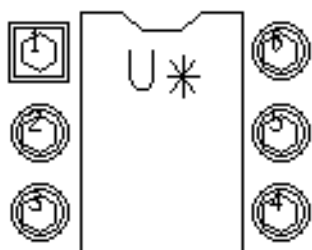
### dip32\_6



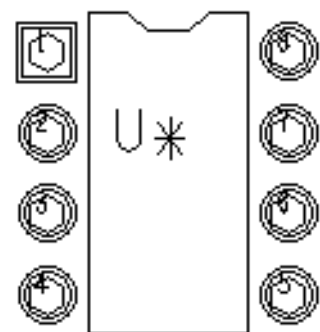
### **dip4\_3**



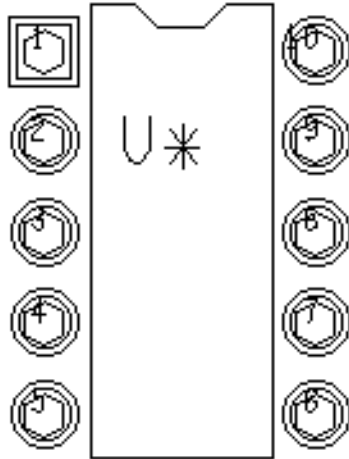
### **dip6\_3**



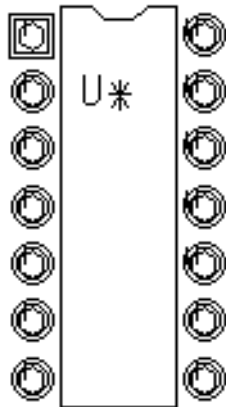
### **dip8\_3**



### dip10\_3

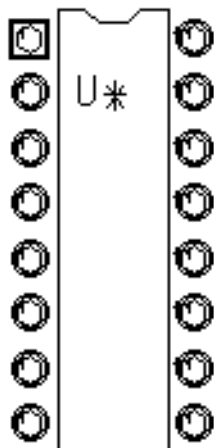


### dip14\_3

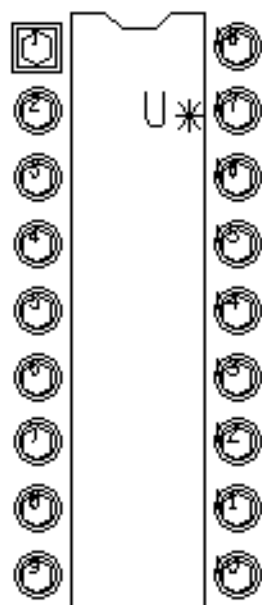




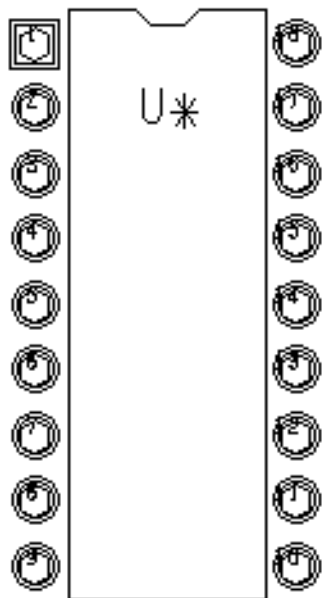
### dip16\_3



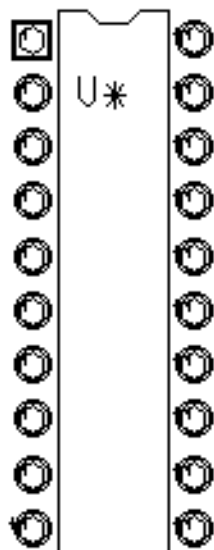
### dip18\_3



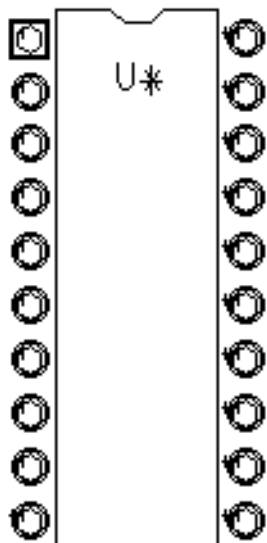
### dip18\_4



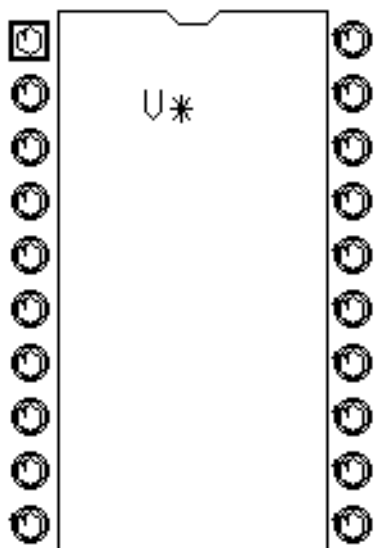
### dip20\_3



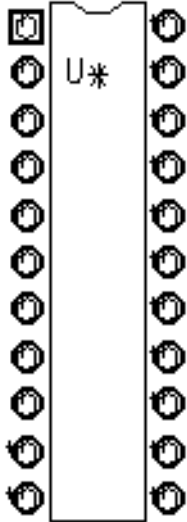
### dip20\_4



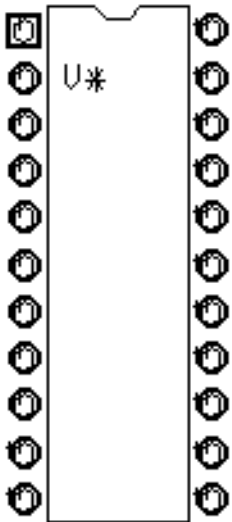
### dip20\_6



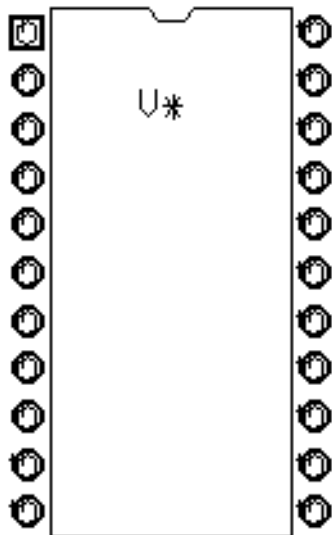
### dip22\_3



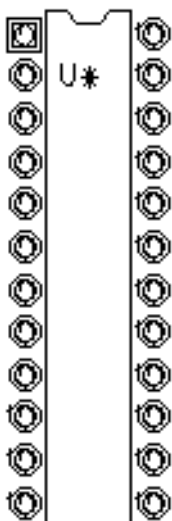
### dip22\_4



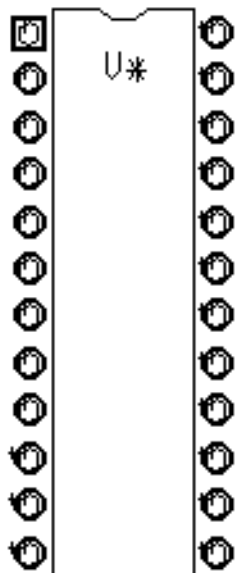
### dip22\_6



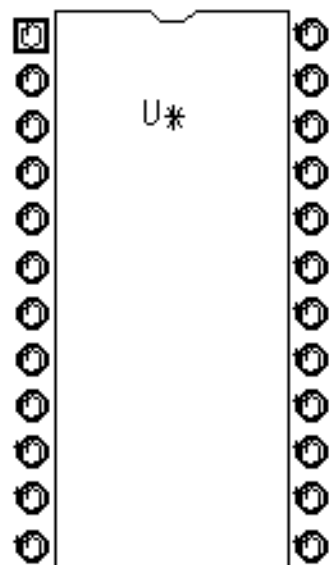
### dip24\_3



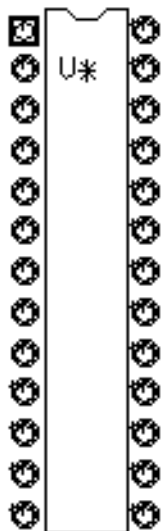
### dip24\_4



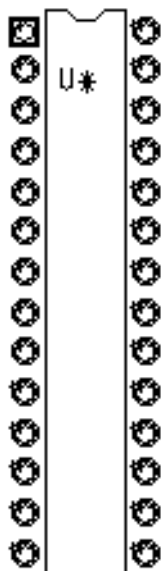
### dip24\_6



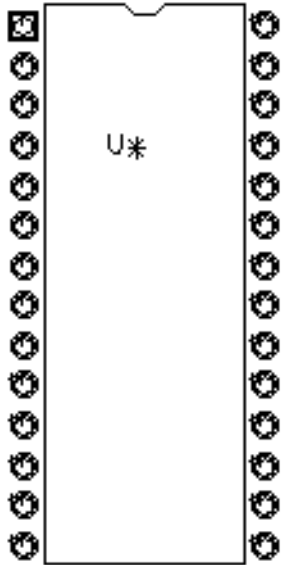
### dip26\_3



### dip28\_3

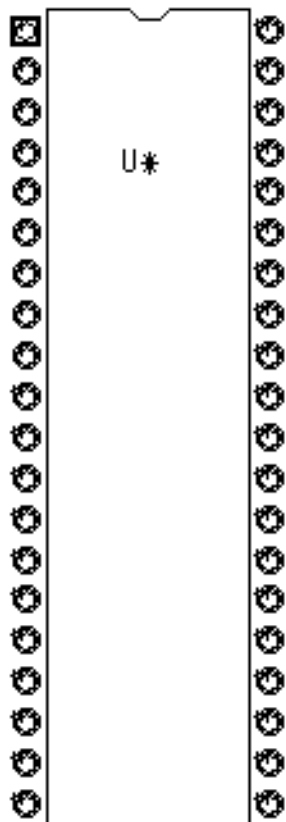


## dip28\_6

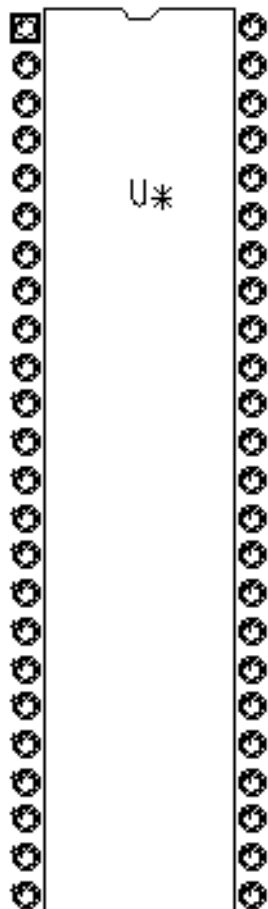




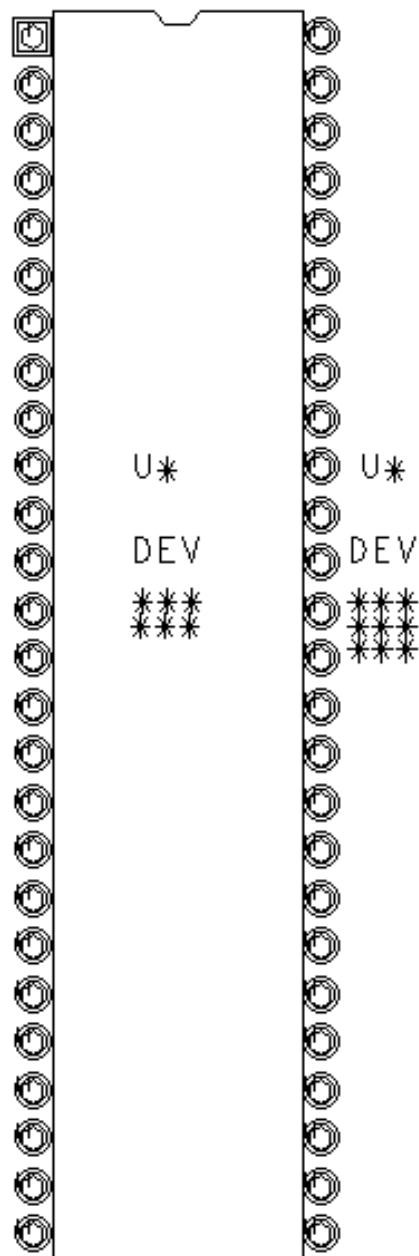
## dip40\_6



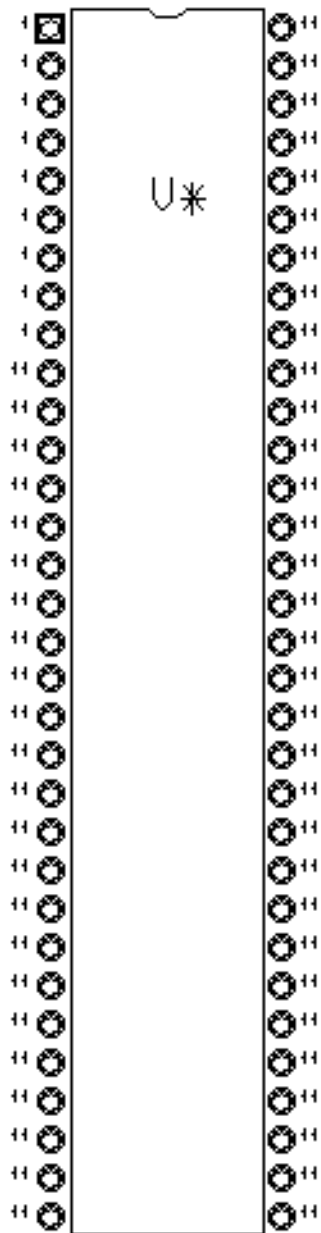
## dip48\_6



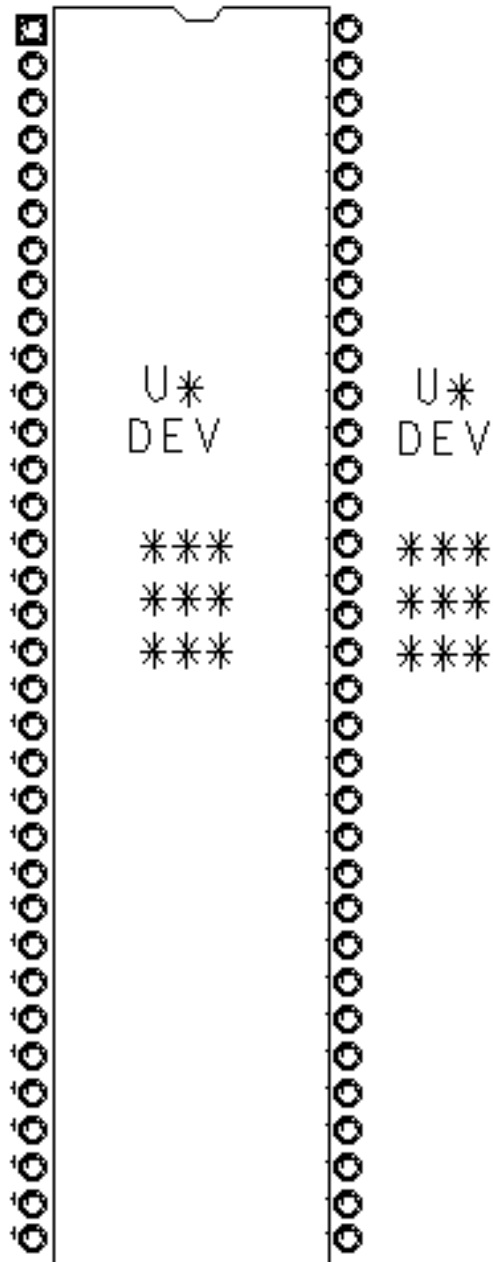
## dip52\_6



## dip64\_6

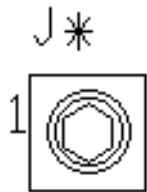


## dip68\_6

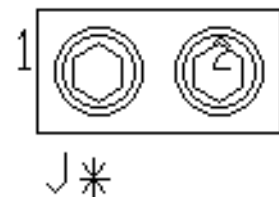
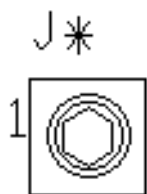


## Jumpers

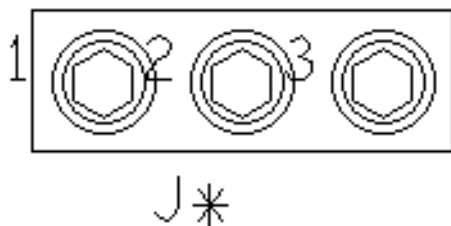
### jumper1



### jumper2

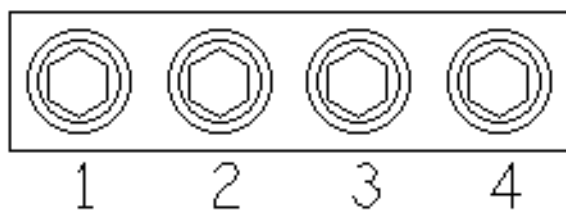


### jumper3



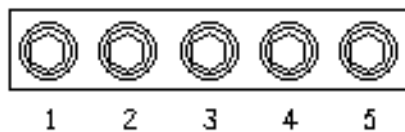
### jumper4

J \*



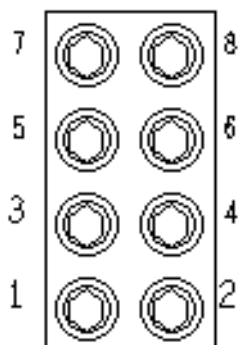
### jumper5

J \*

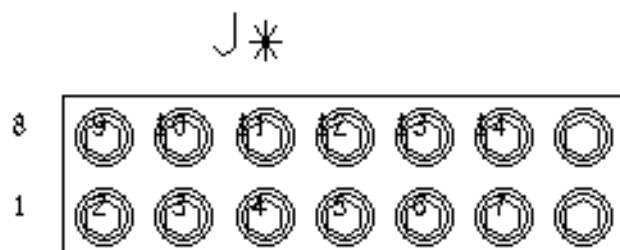


### jumper8

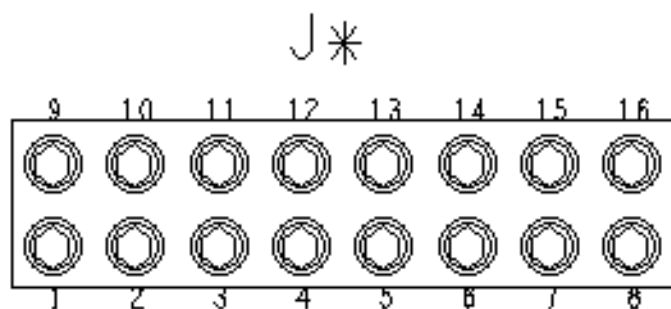
J \*



## jumper14



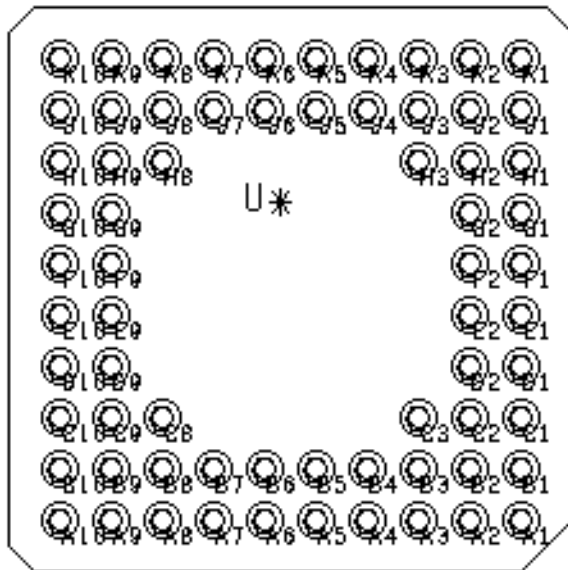
## jumper16



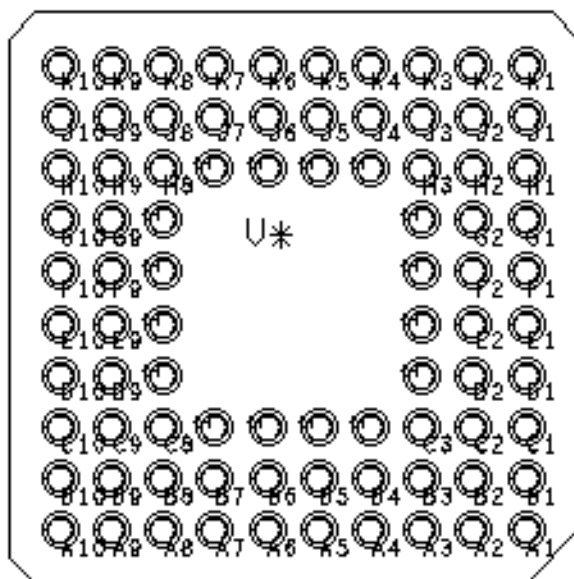


## Pin Grid Arrays

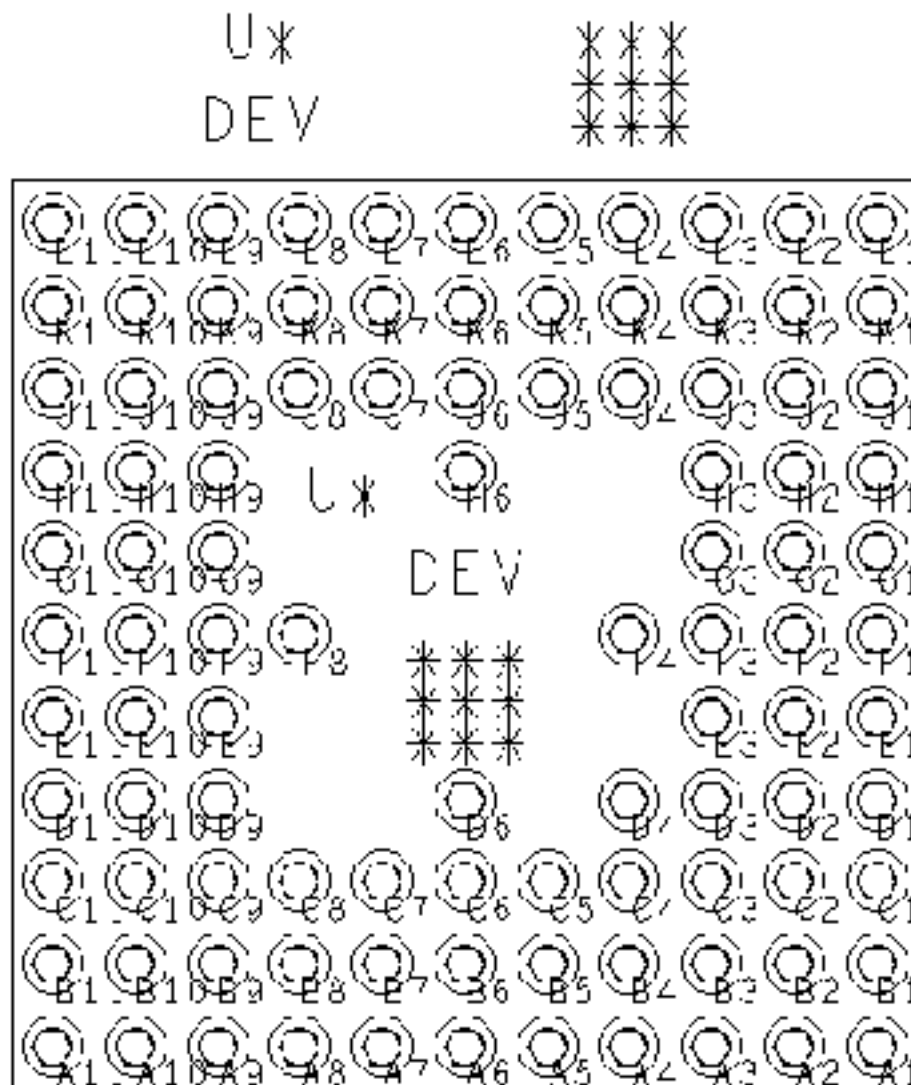
### pga68



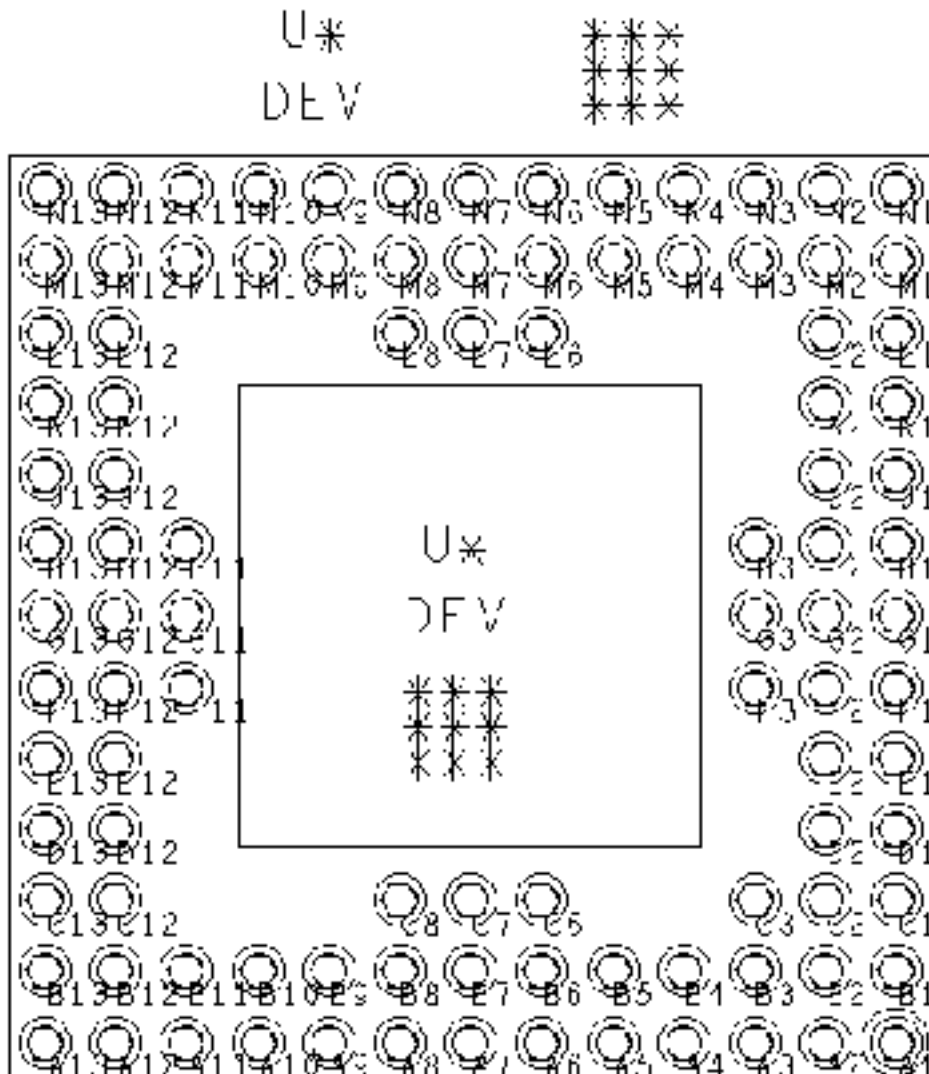
### pga84



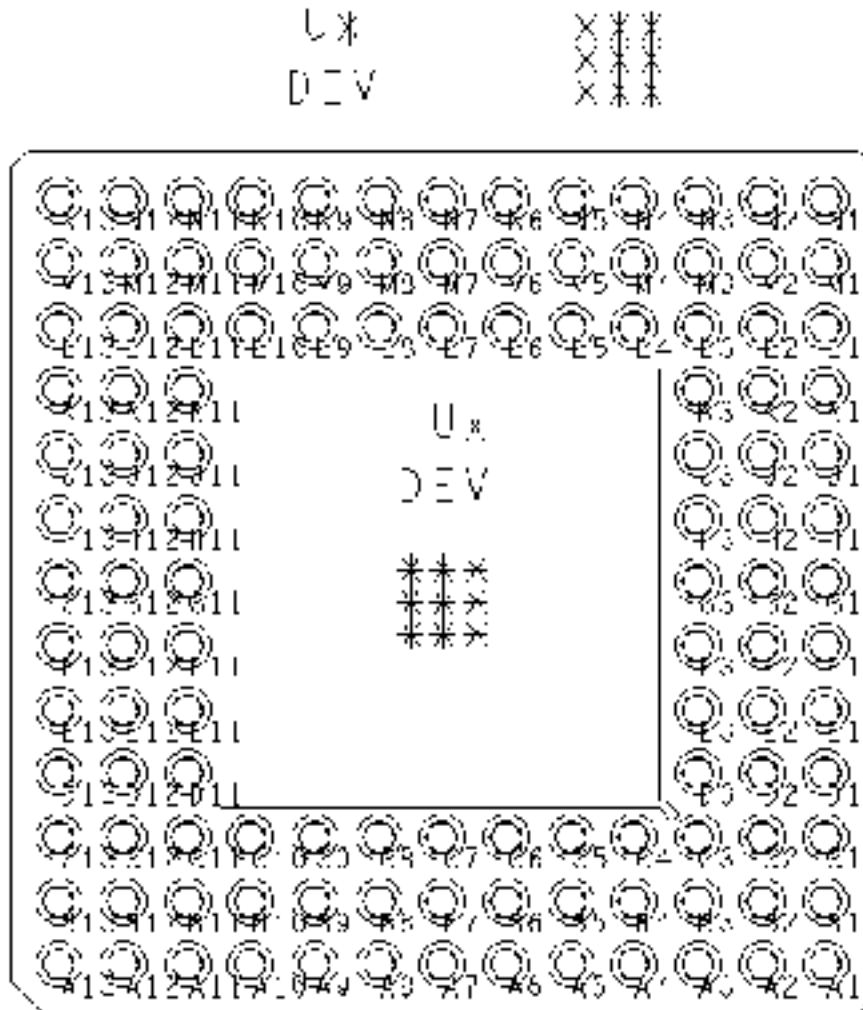
## pga100



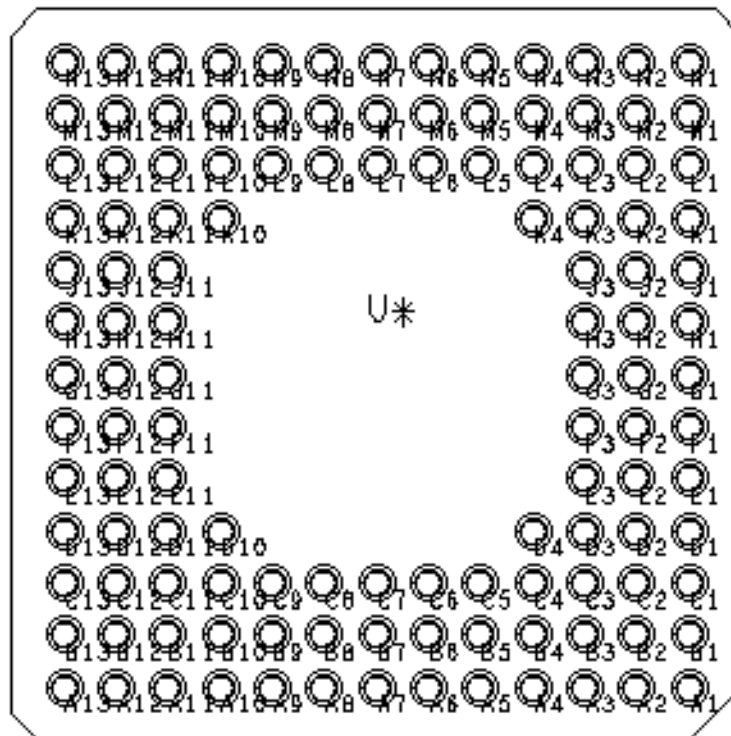
## pga101



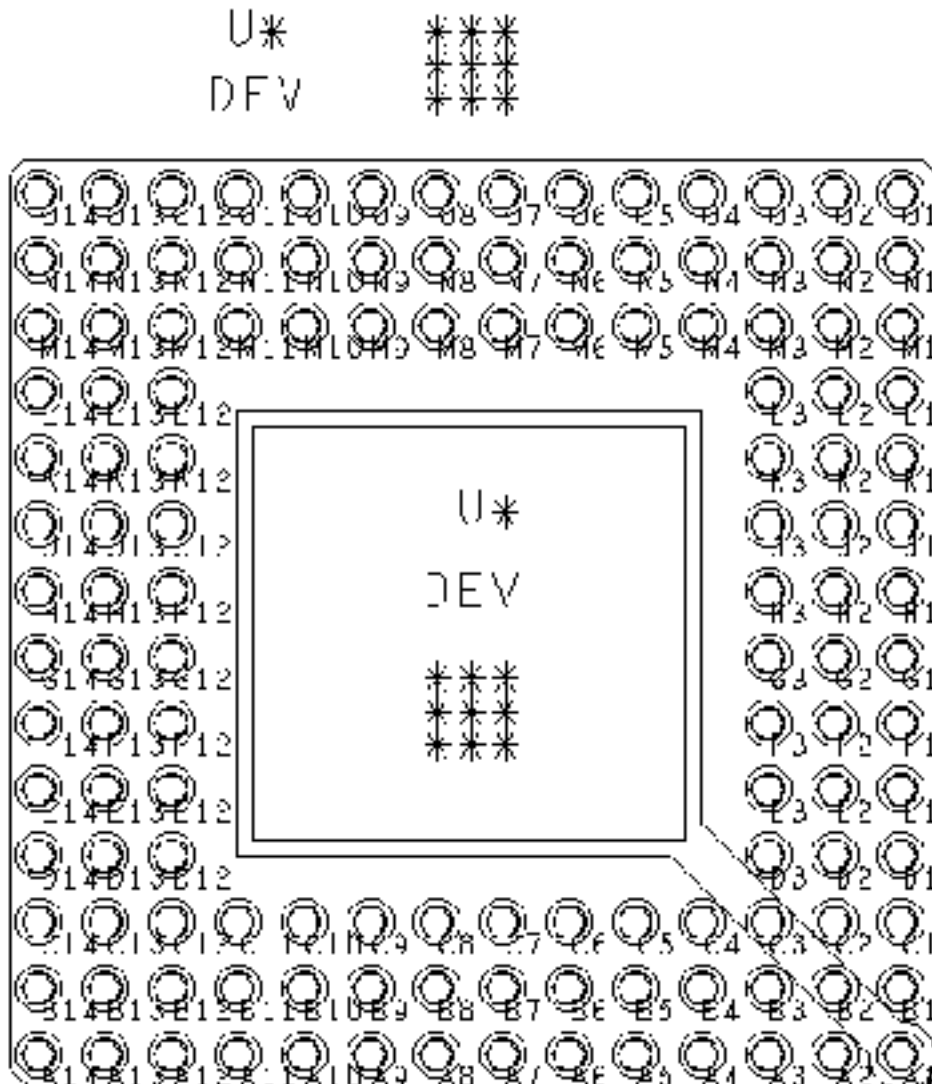
## pga120



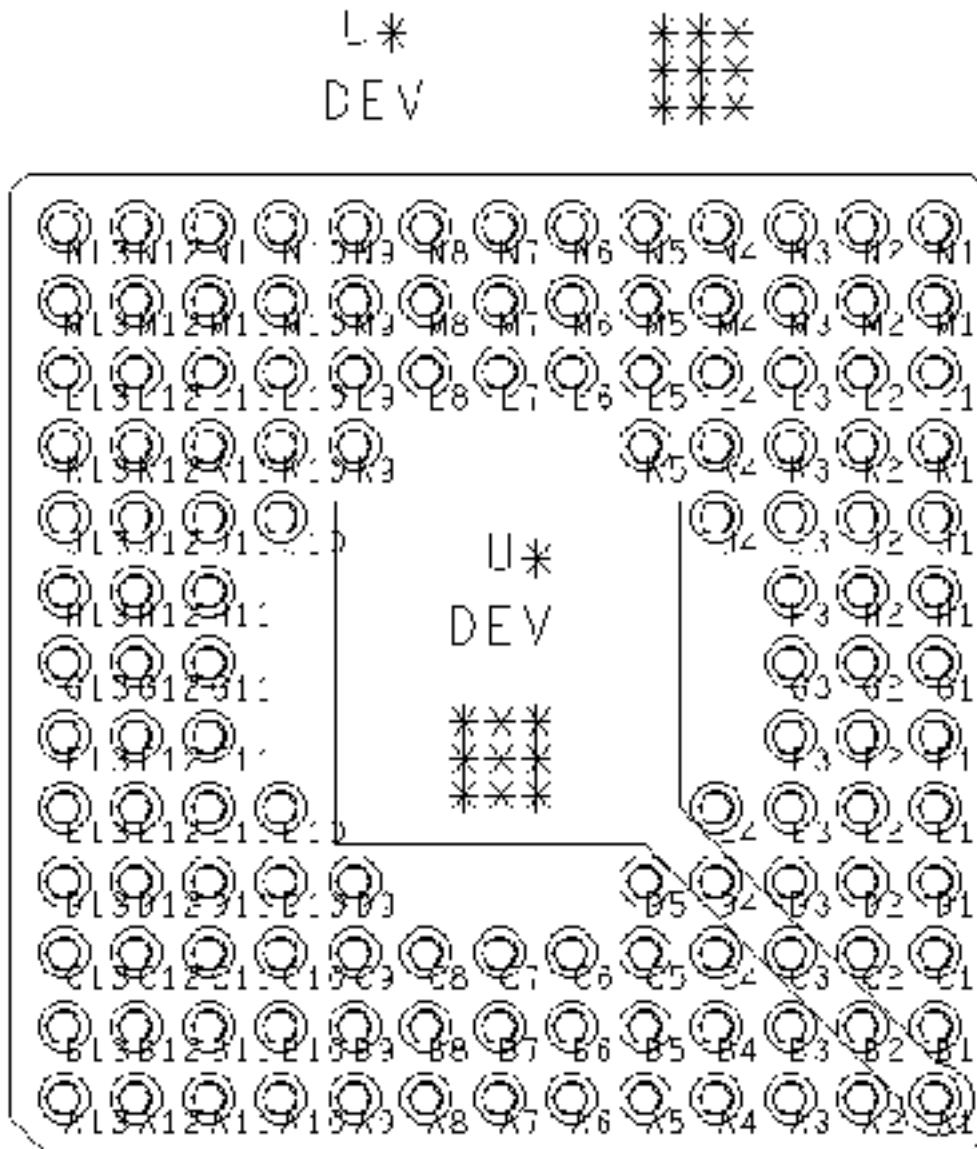
## pga124



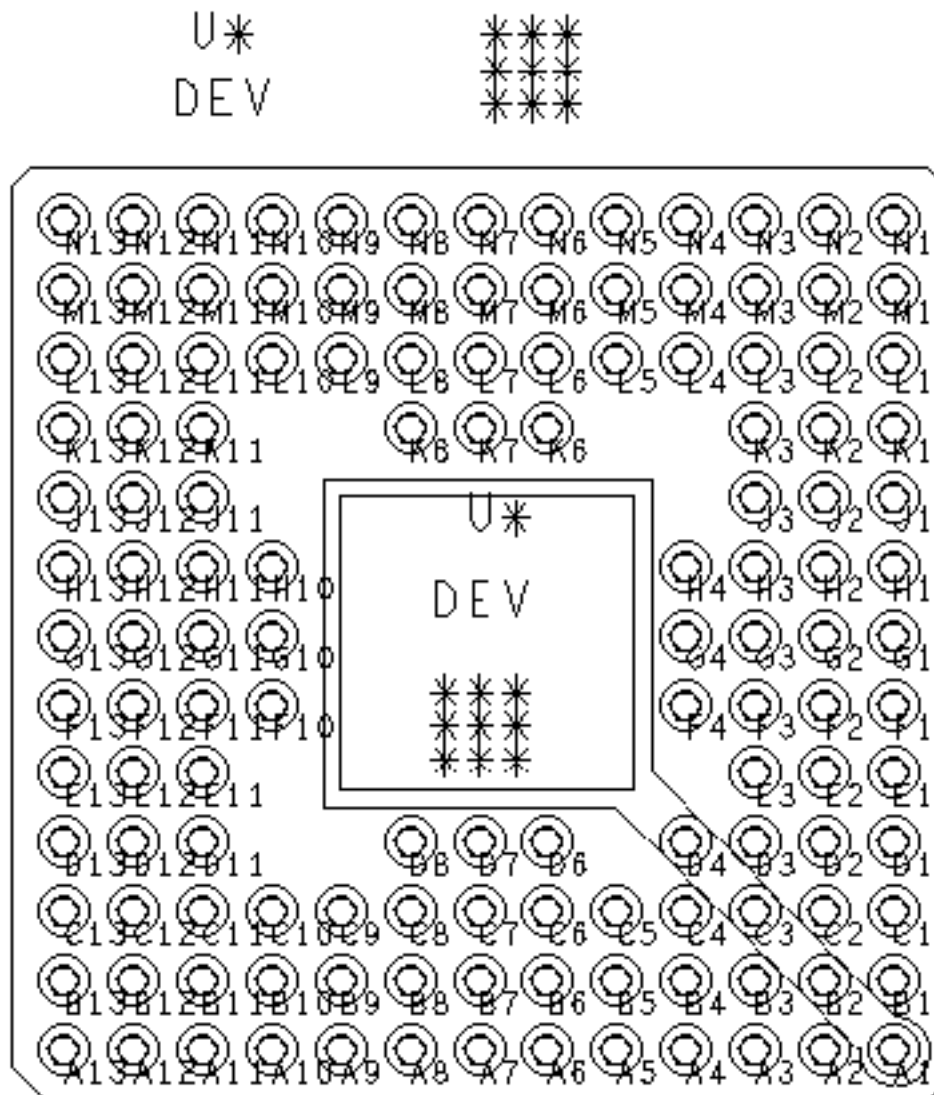
## pga132



## pga132\_ci

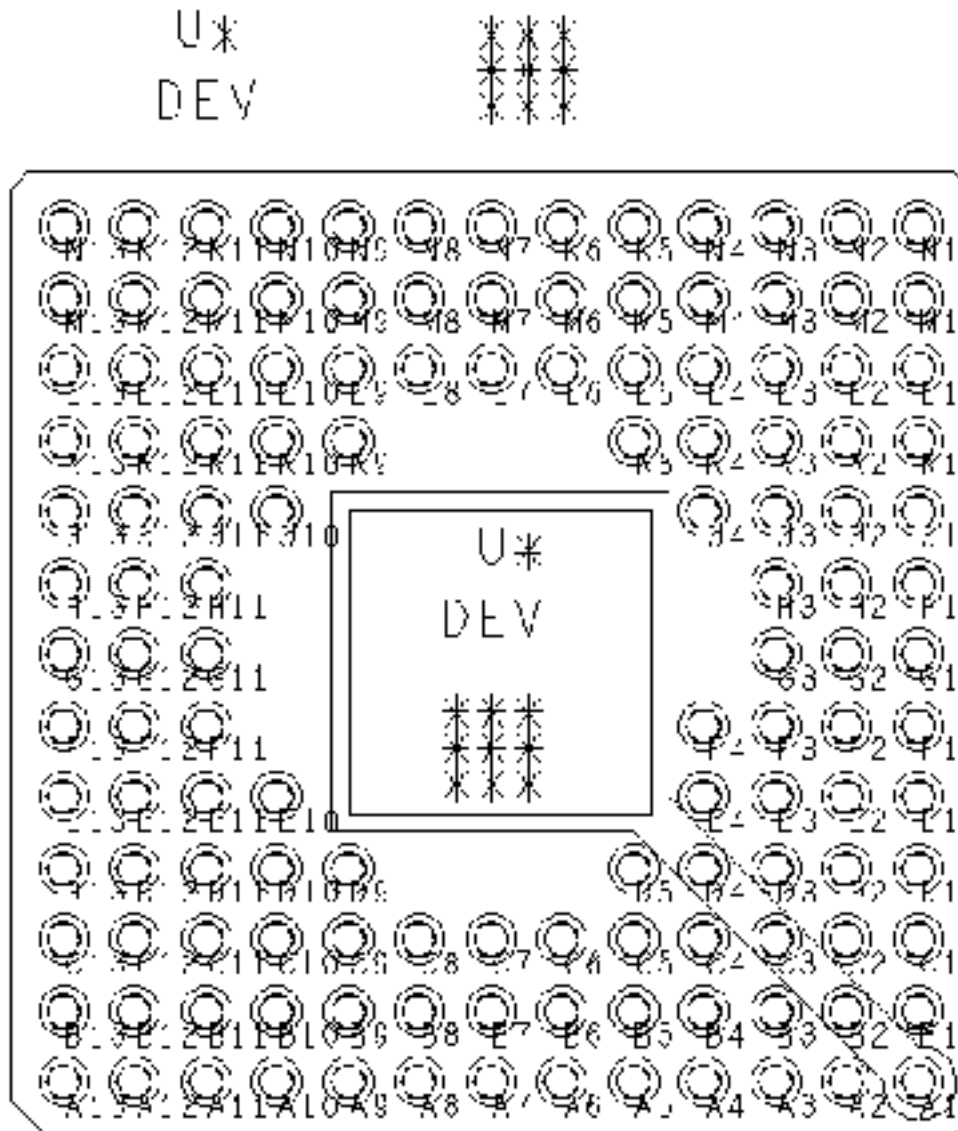


## pga132-x

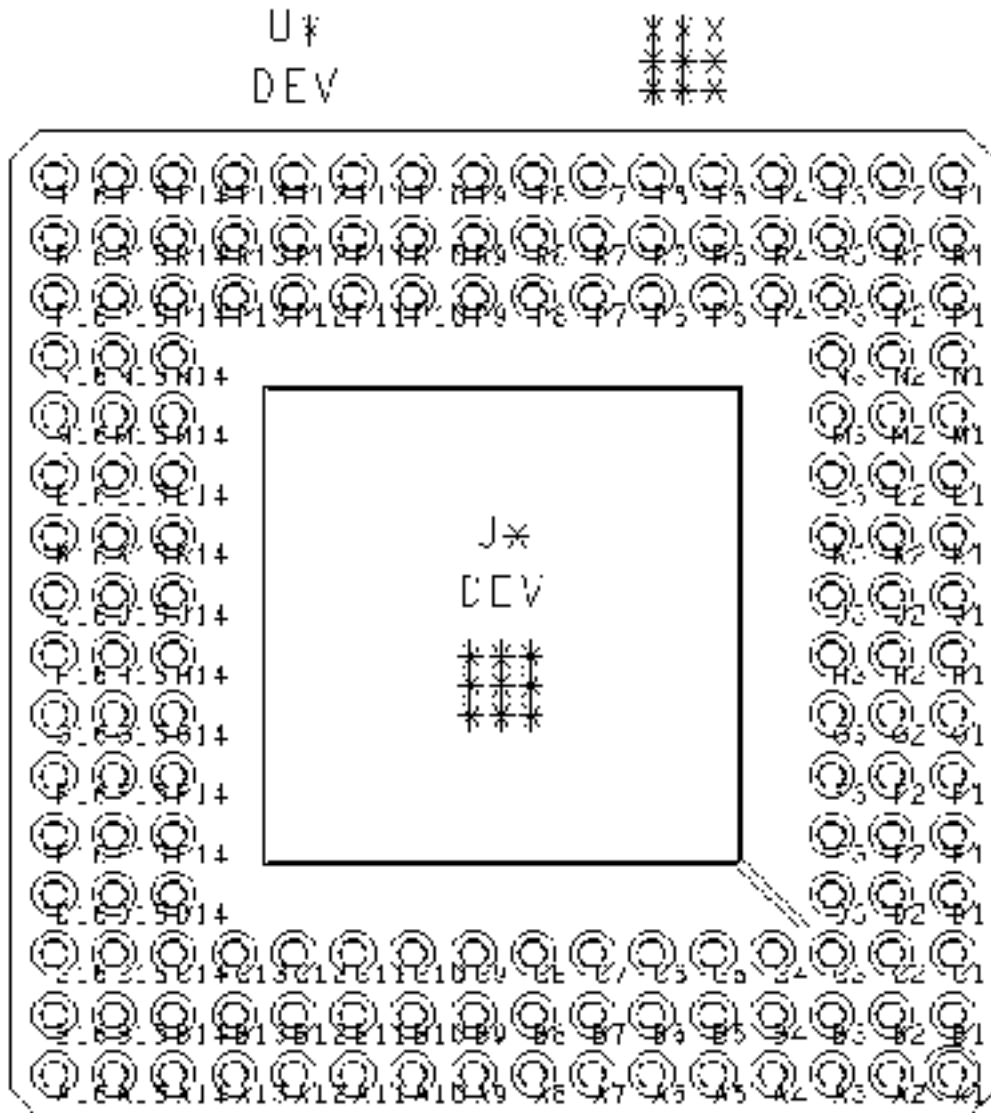




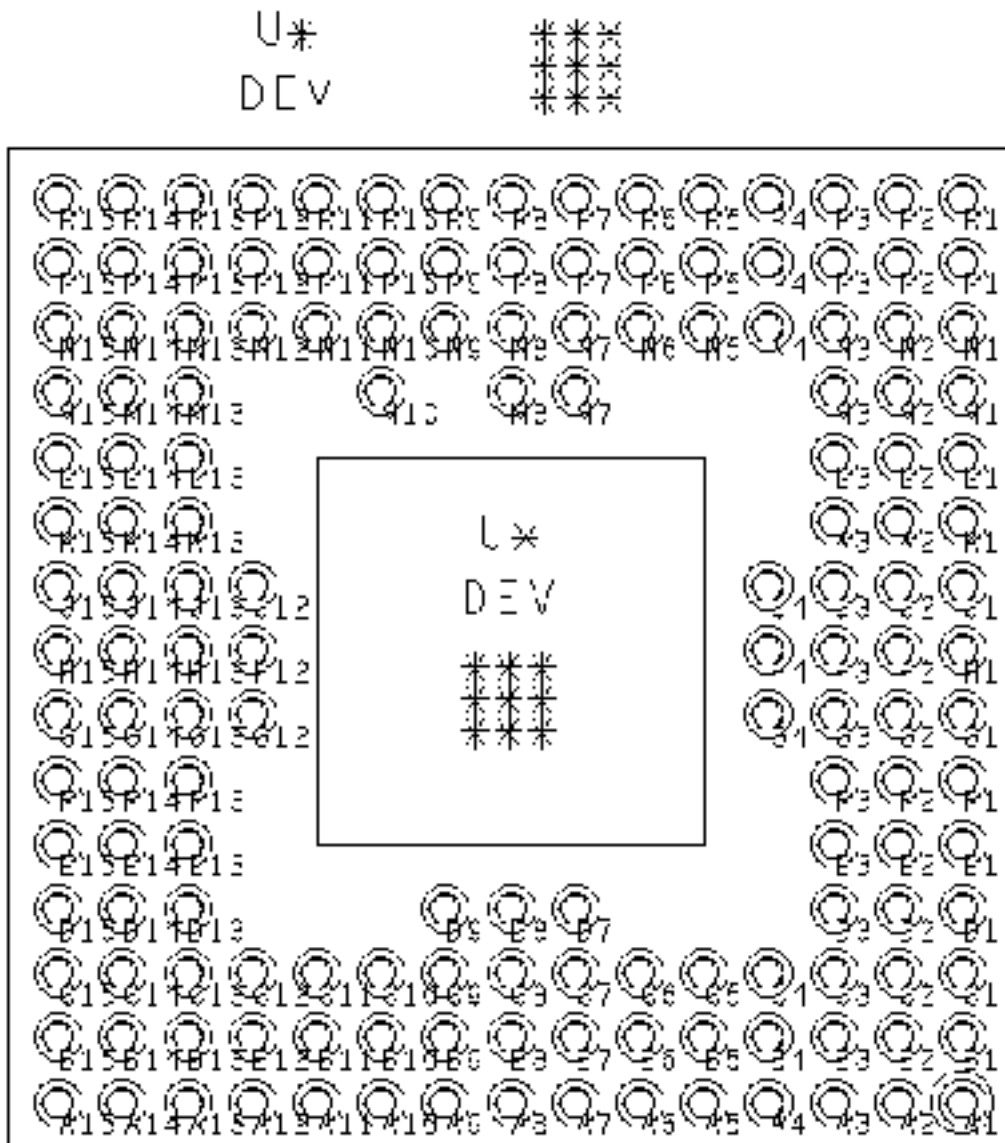
## pga133



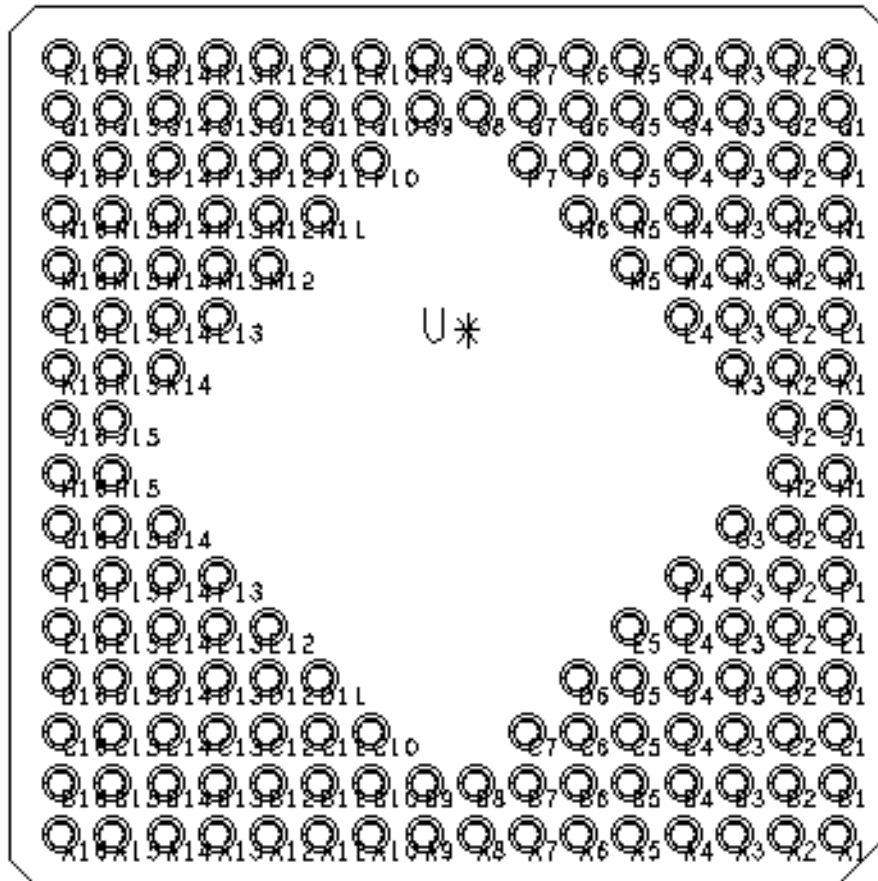
## pga156



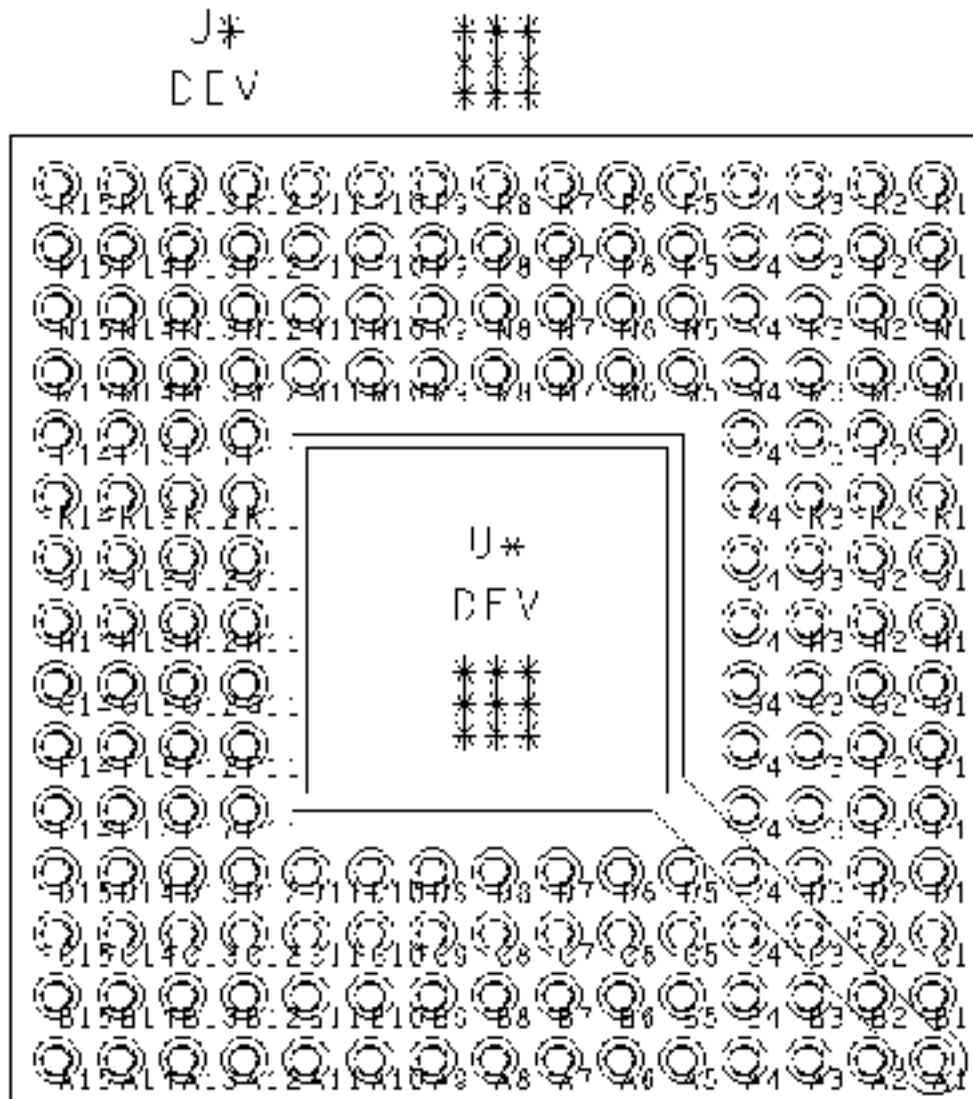
## pga156\_x



## pga172

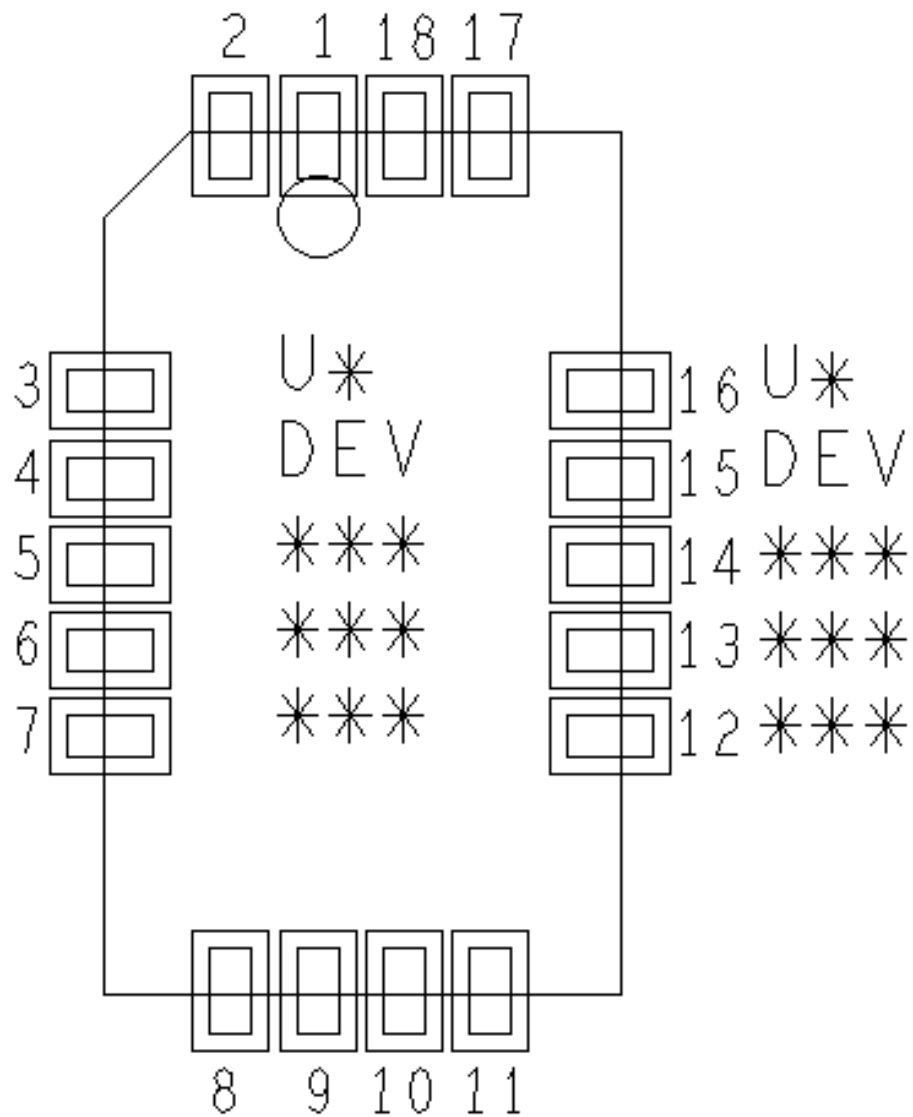


## pga176

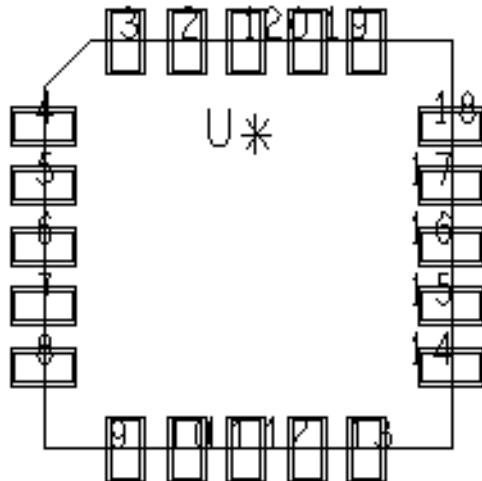


## PLCCs

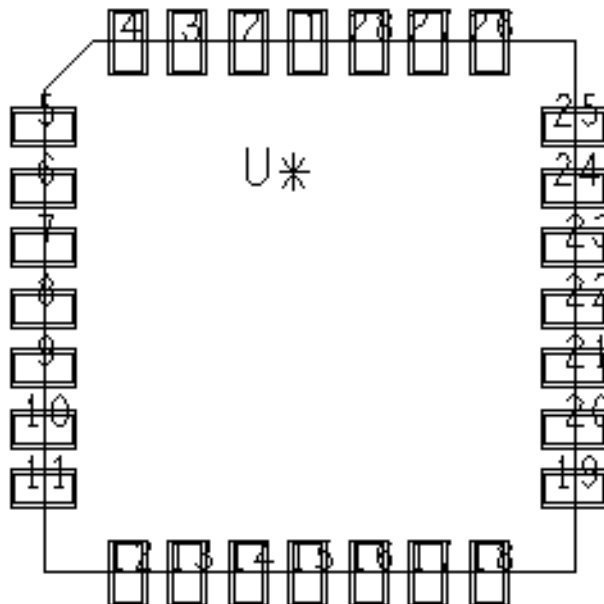
### picc18



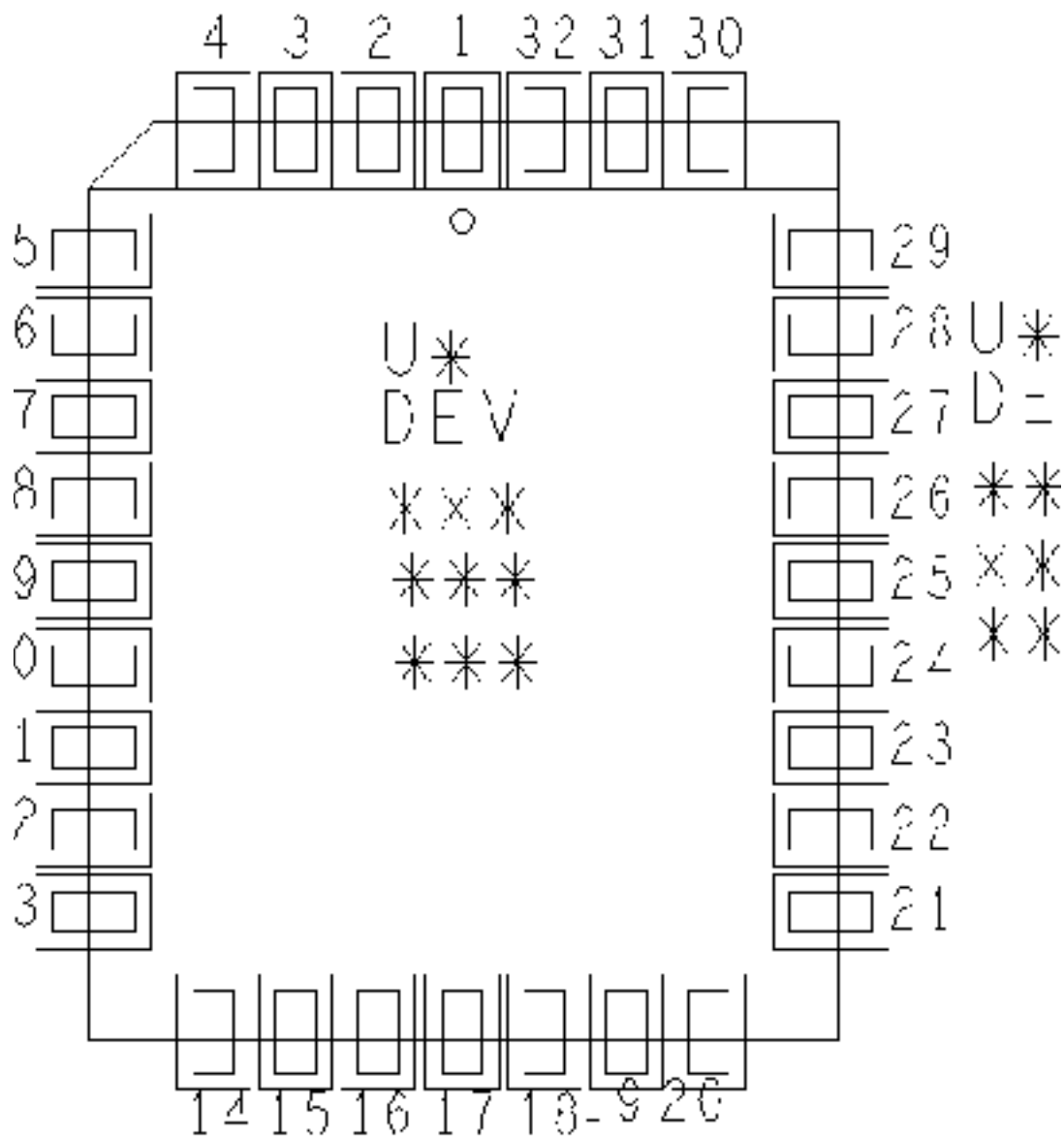
## plcc20



## plcc28

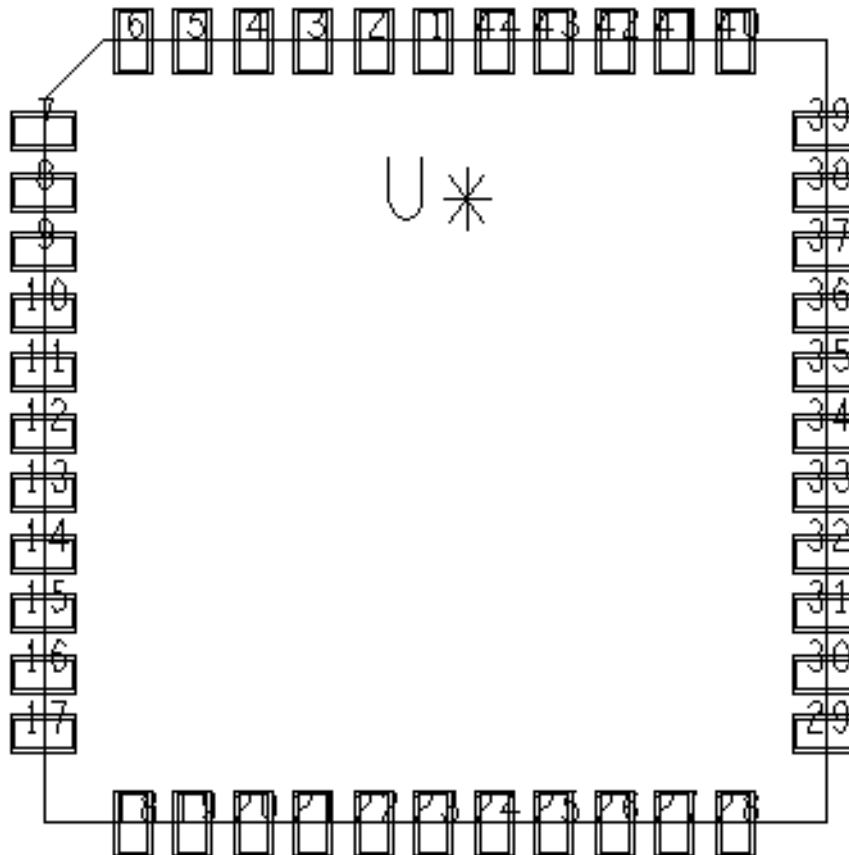


## picc32

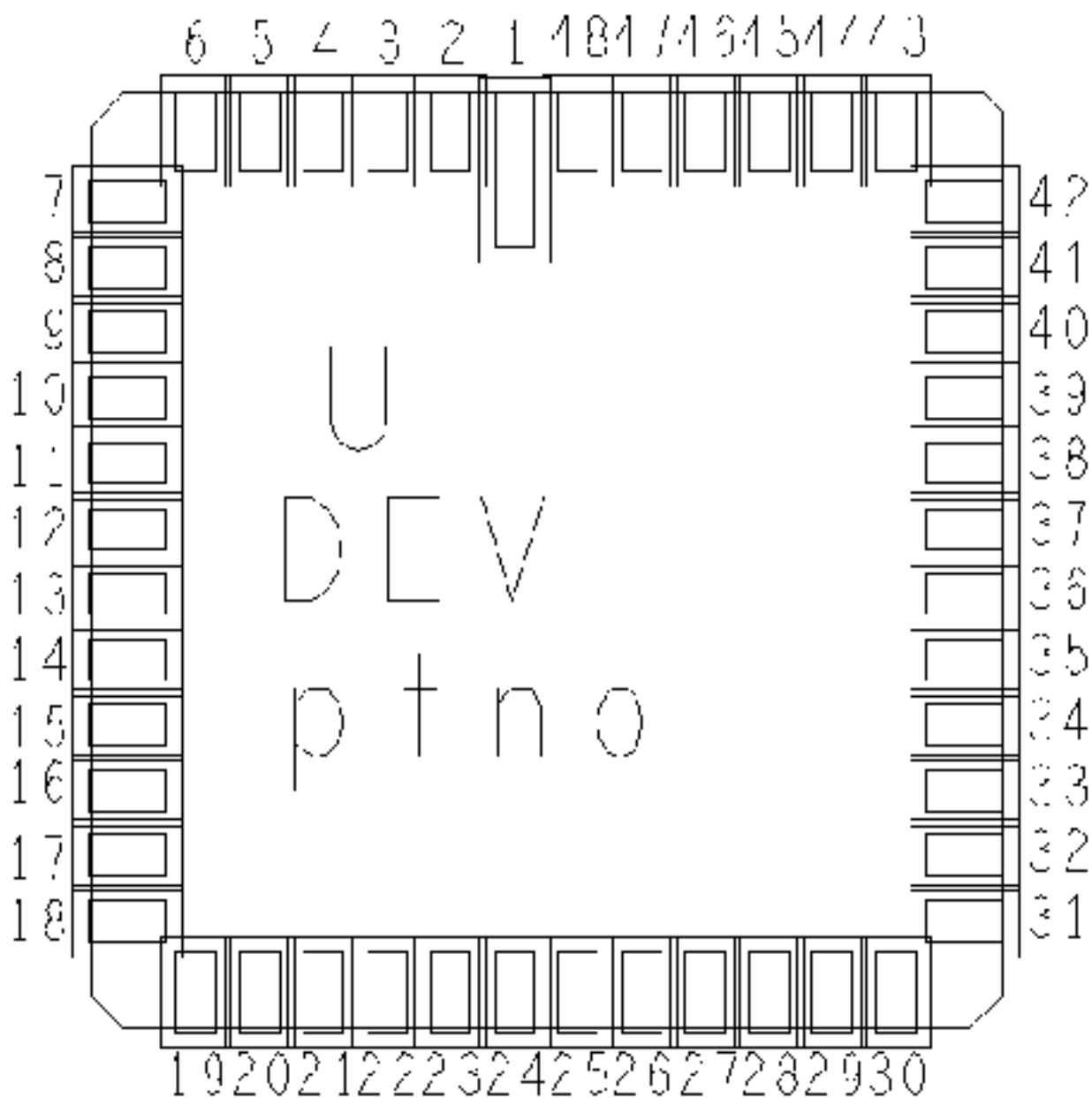




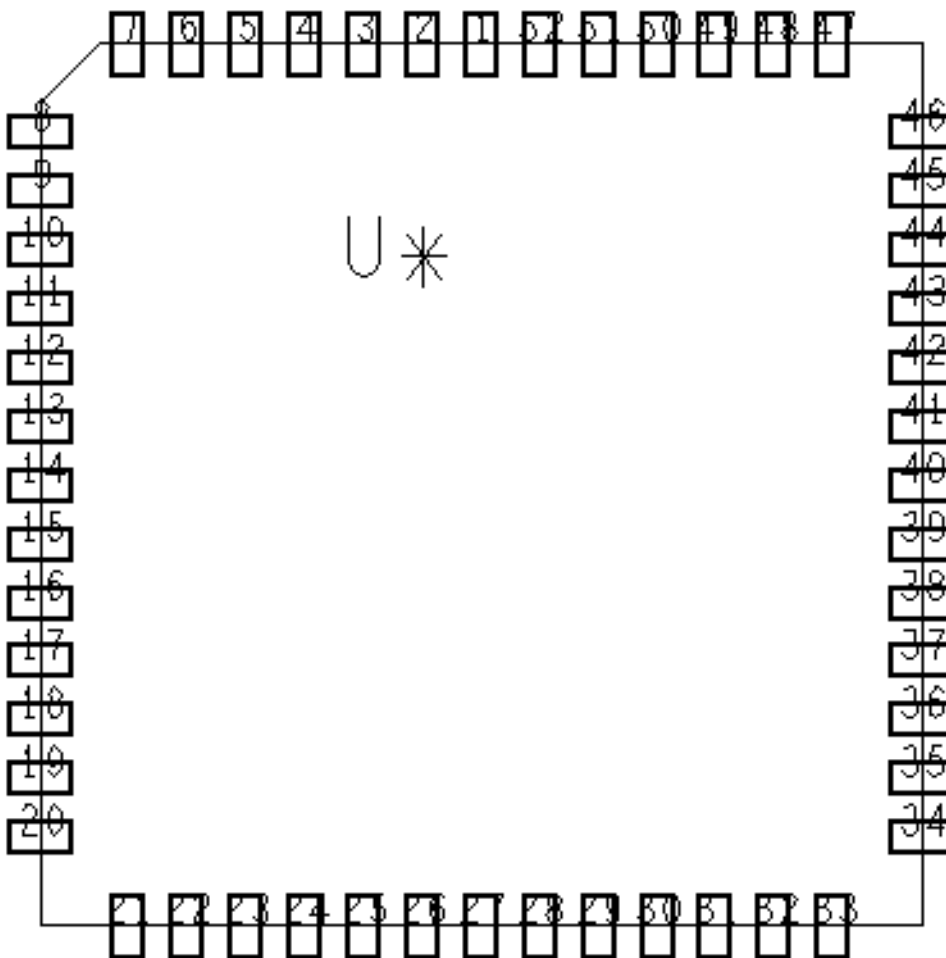
## plcc44



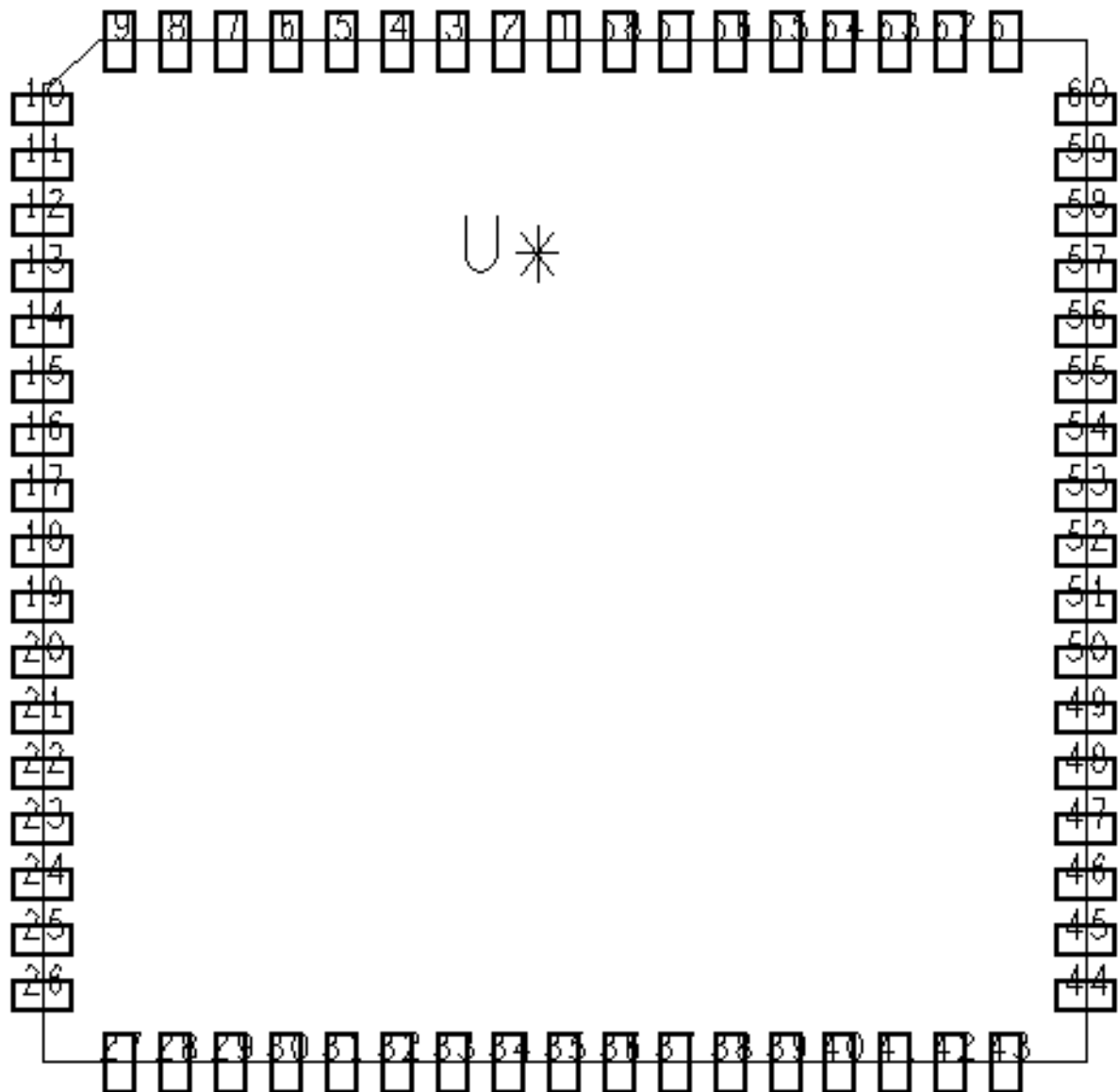
## plcc48



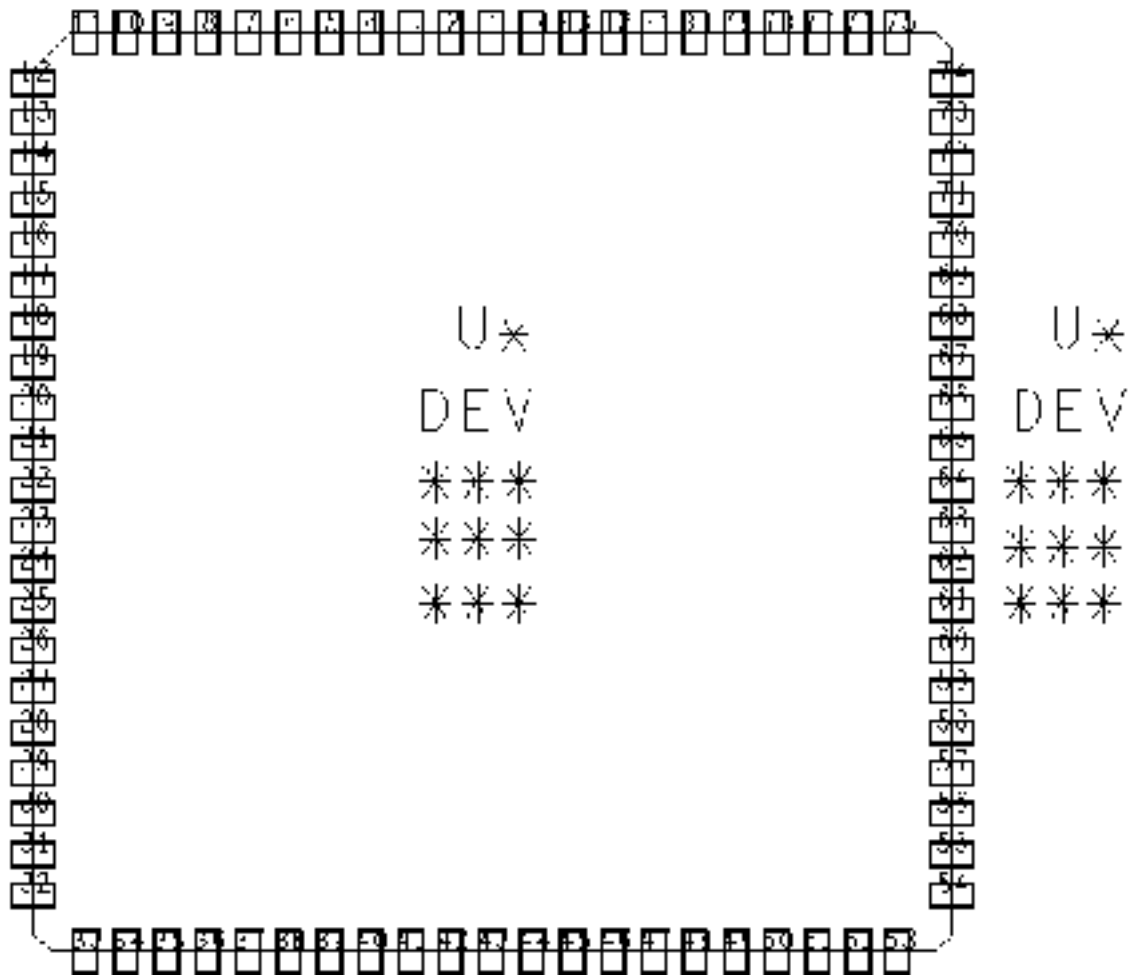
## picc52



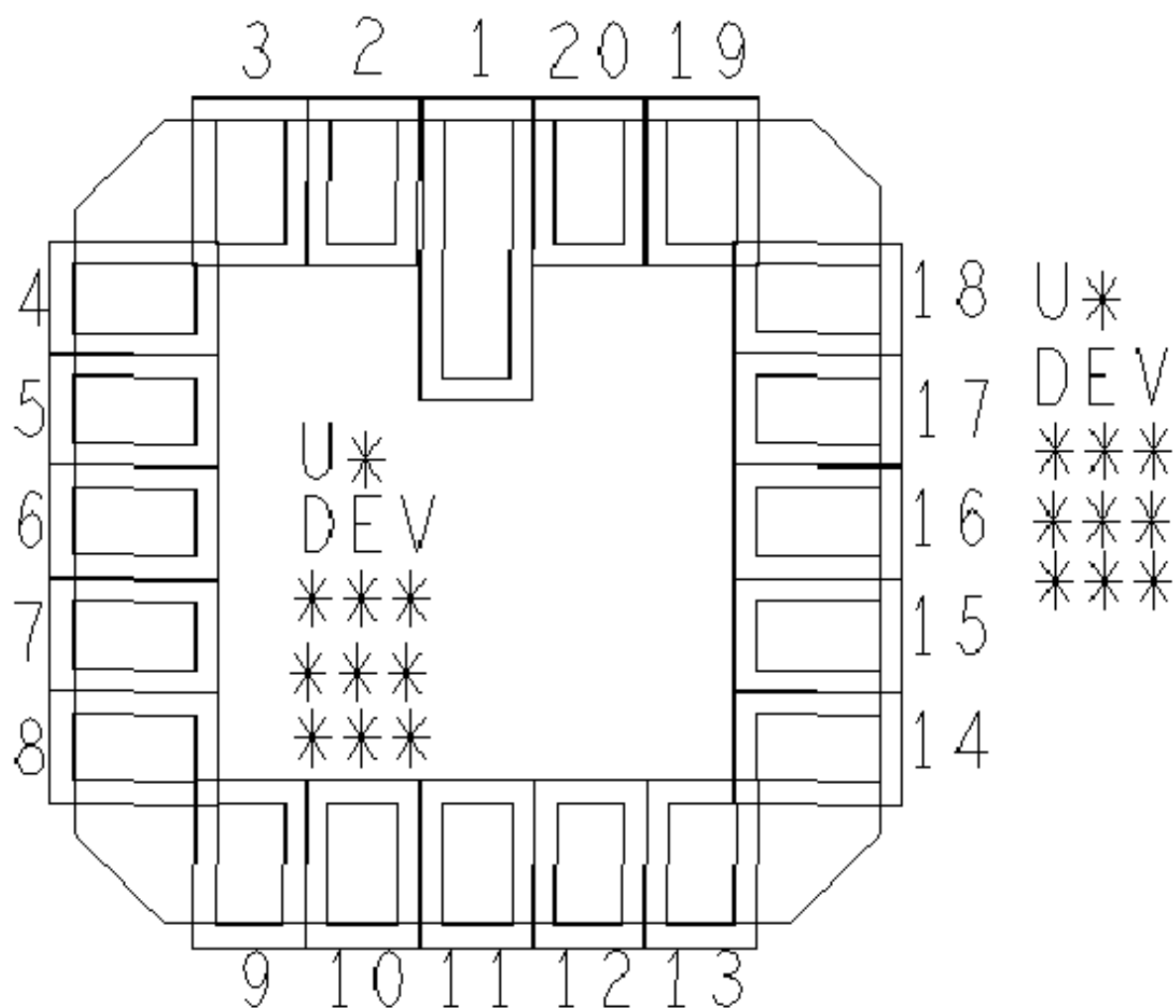
## picc68



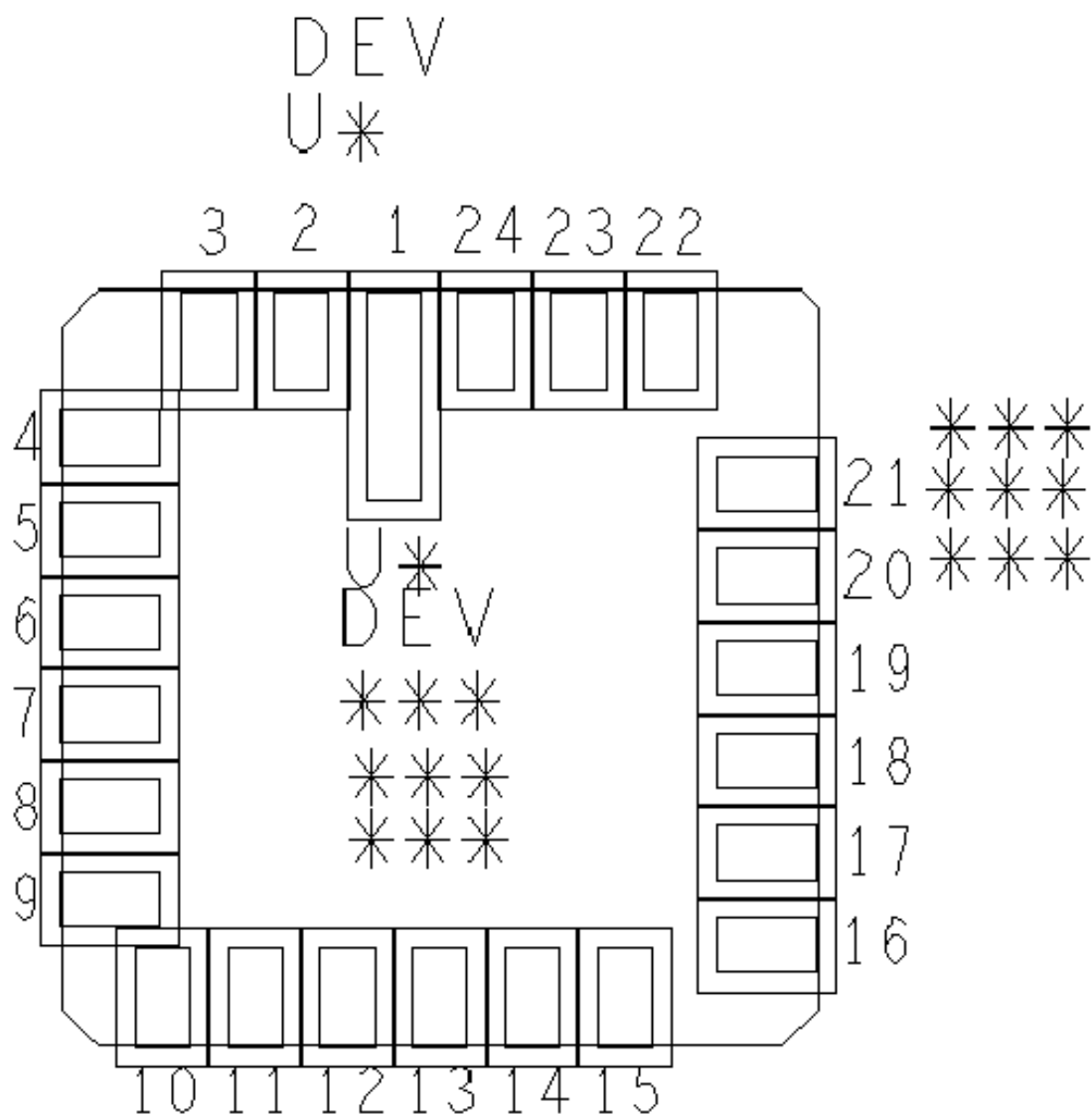
## picc84



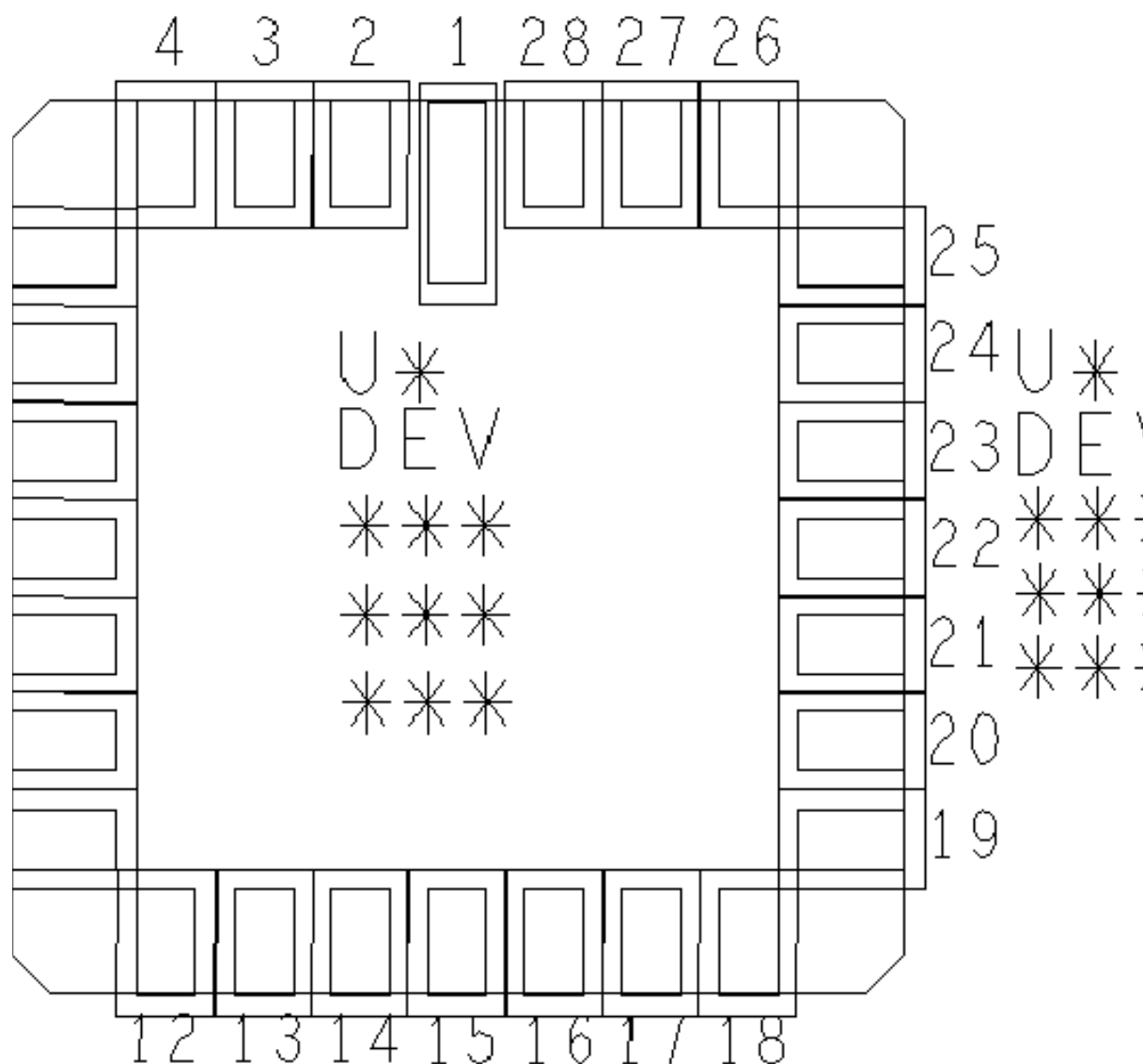
## icc20



## icc24

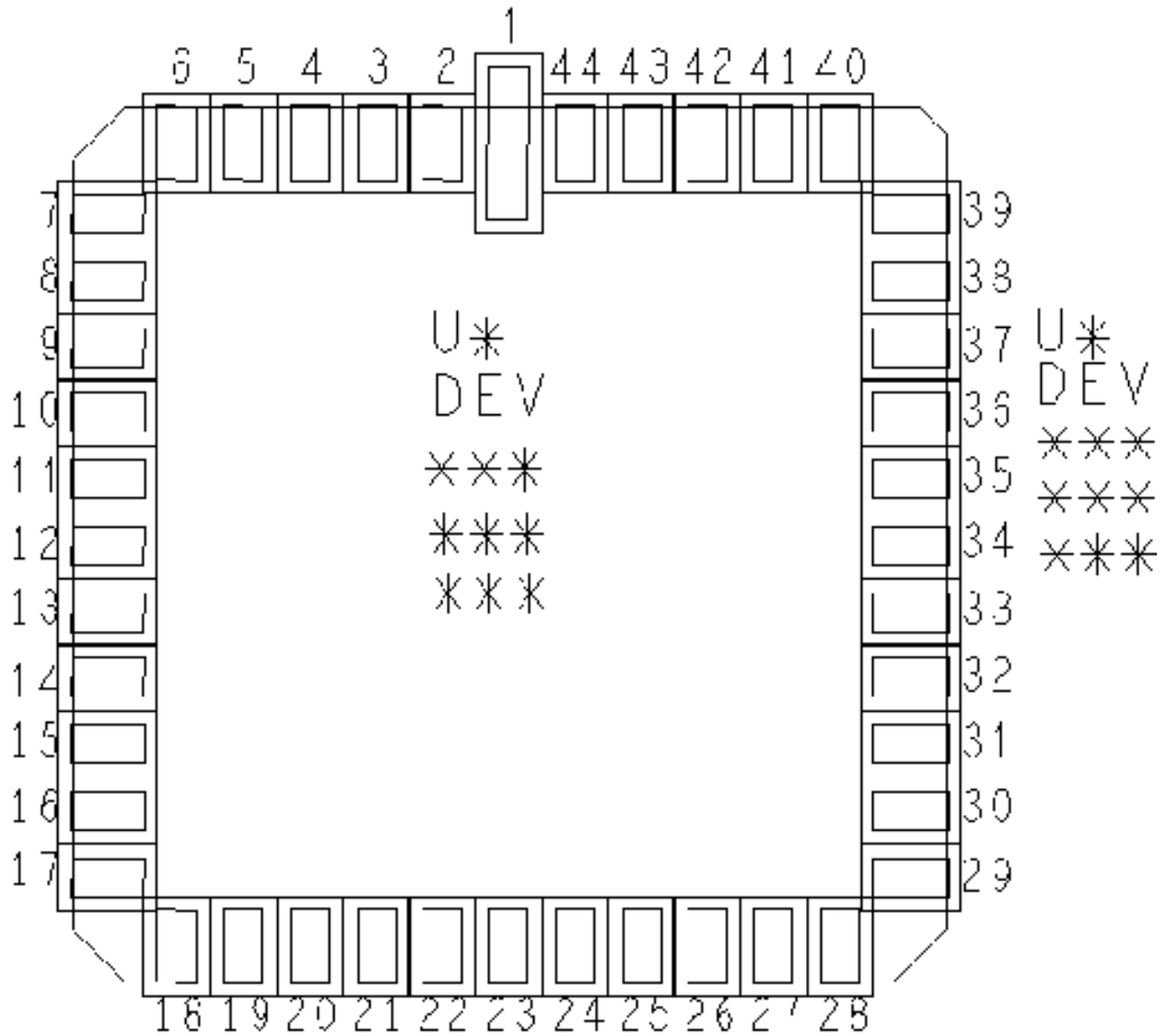


## icc28

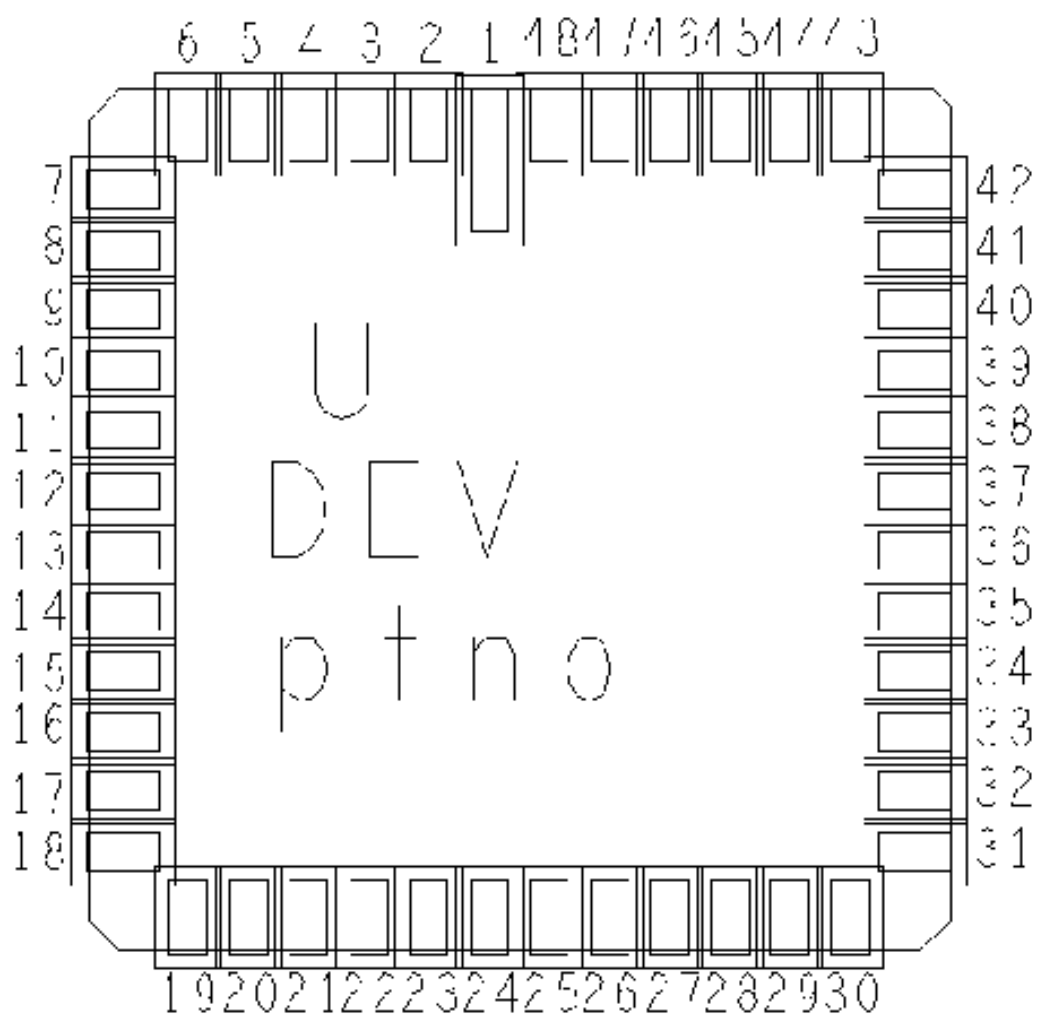




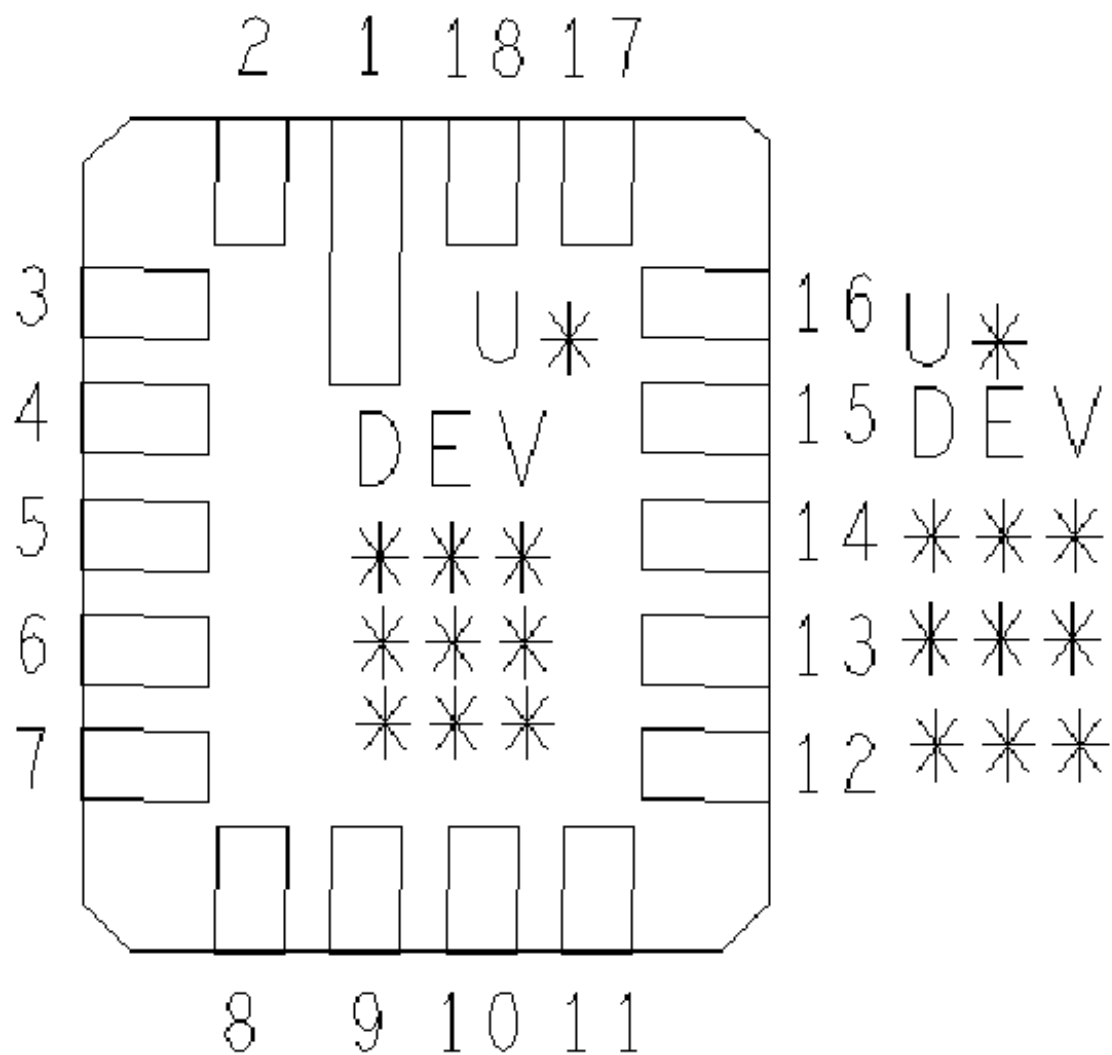
## icc24



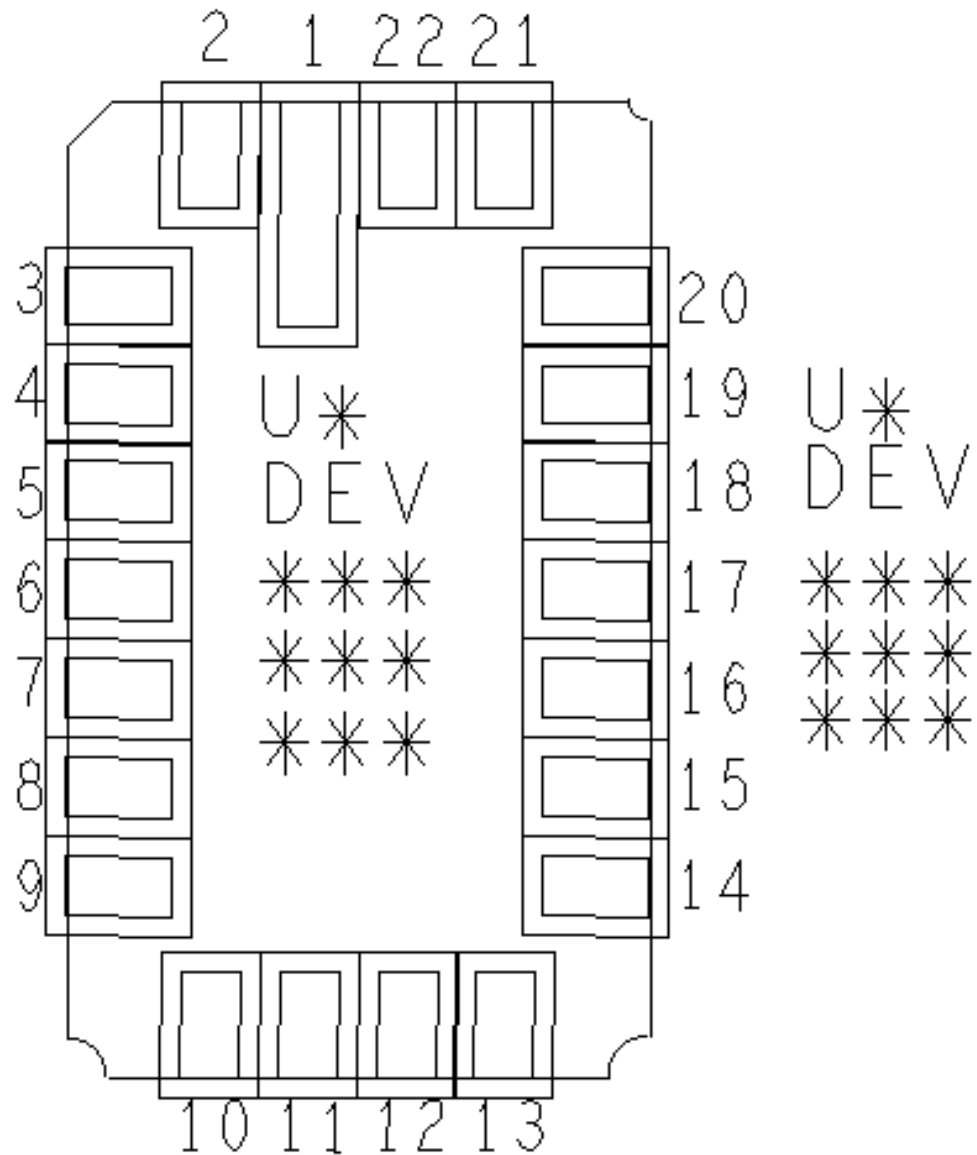
## icc48



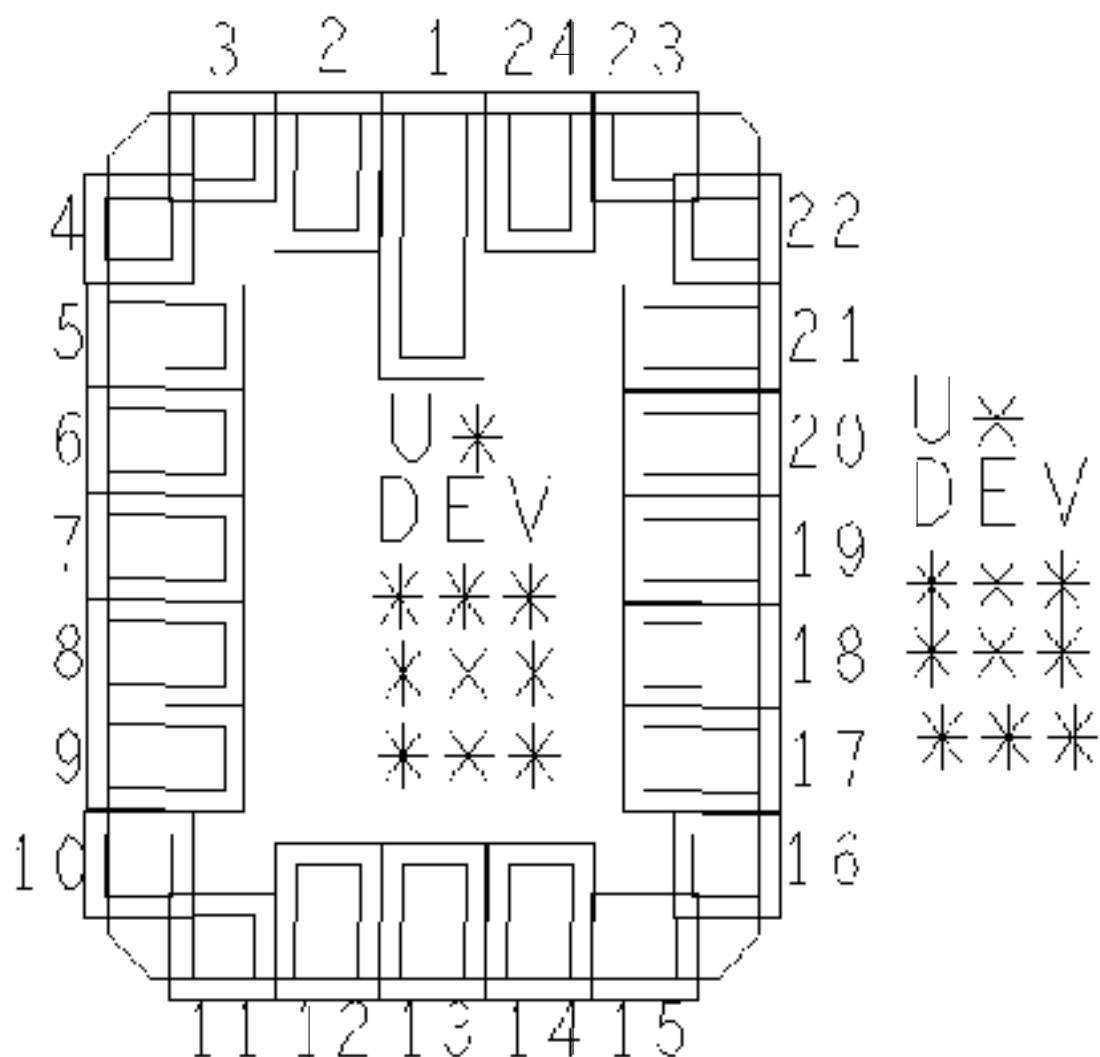
## iccs18



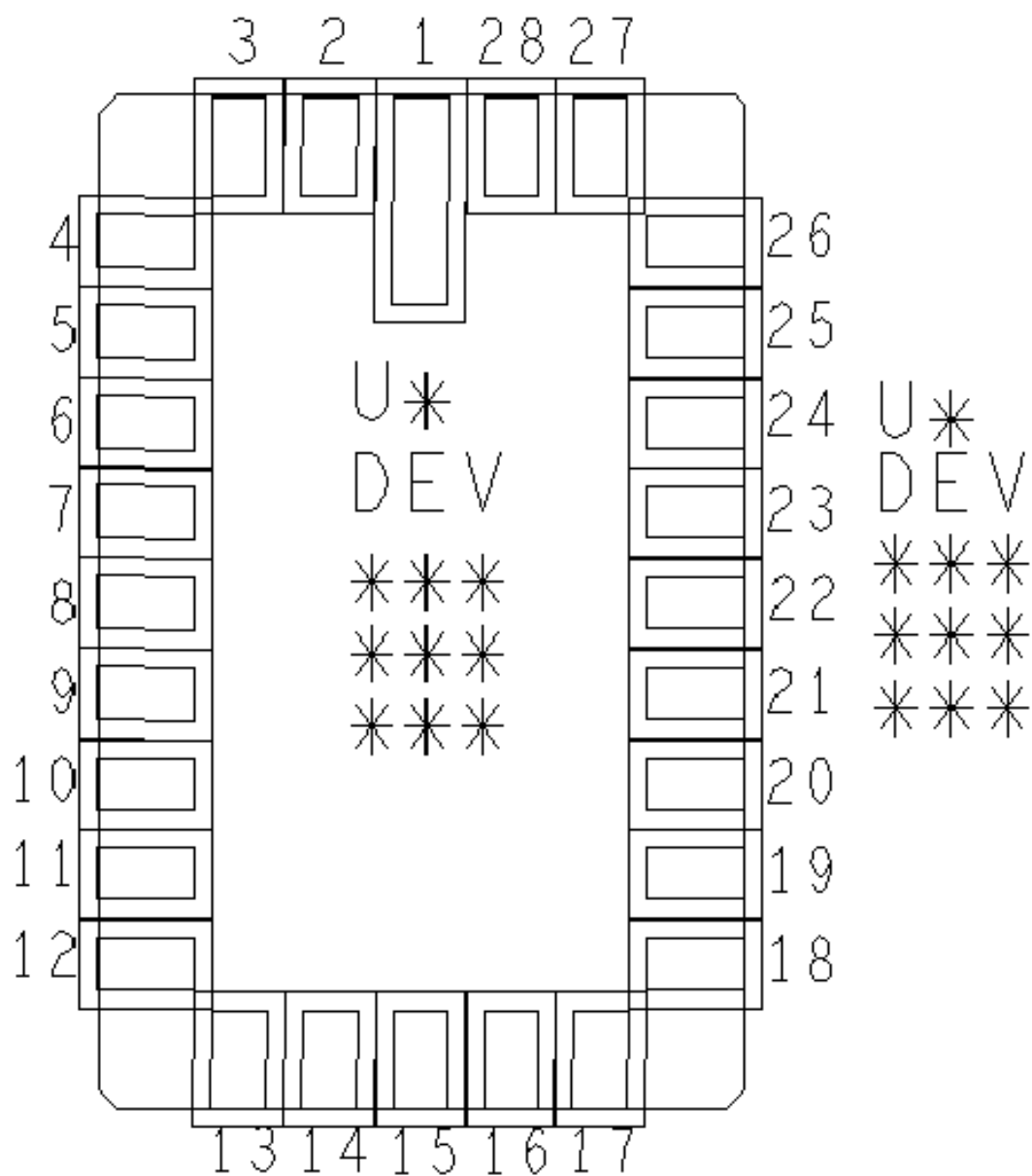
## iccs22



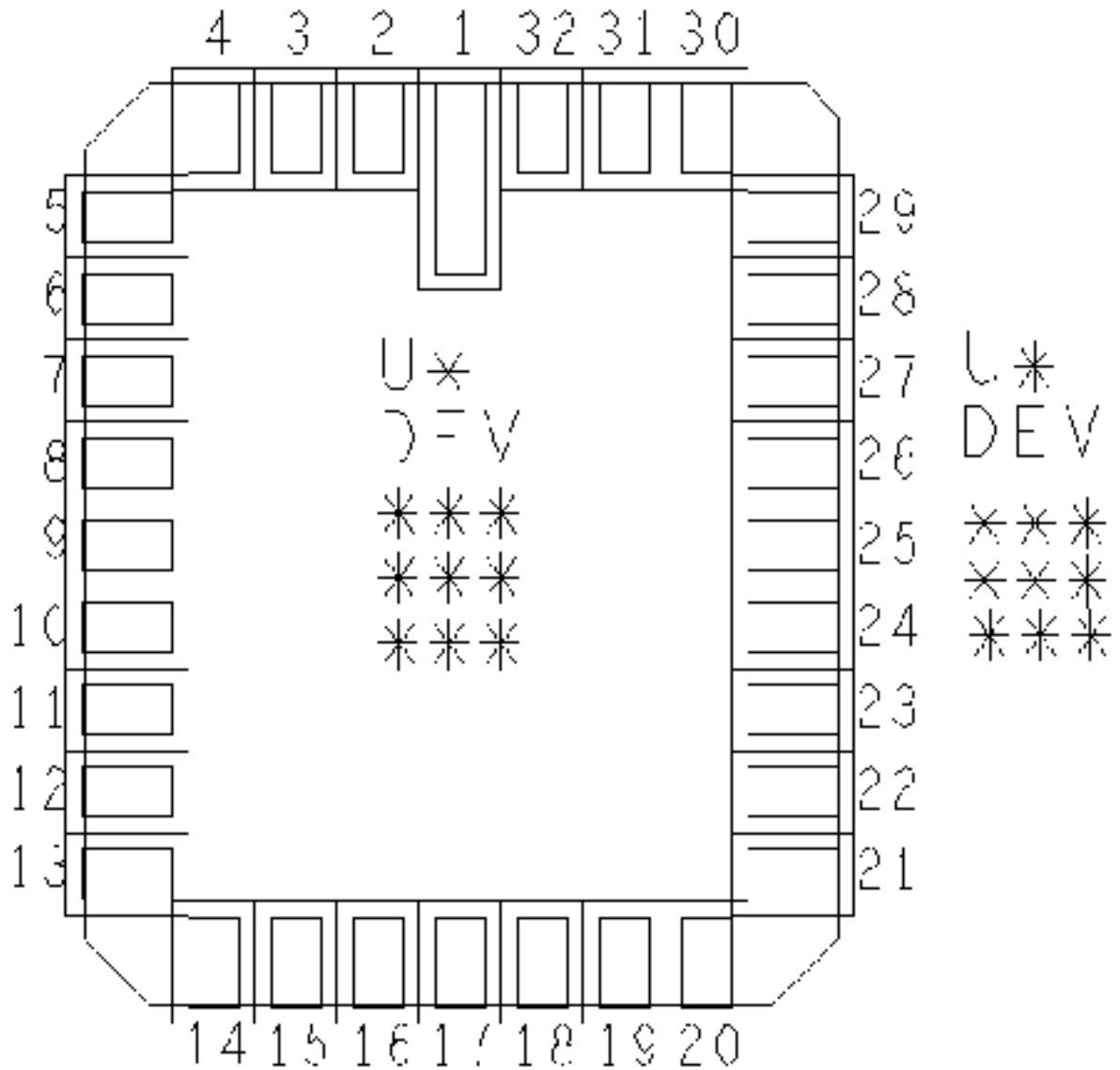
## iccs24



## iccs28

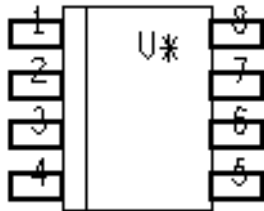


## iccs32

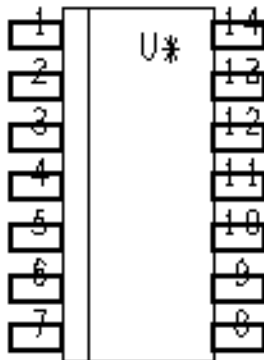


## SOICs

### soic8

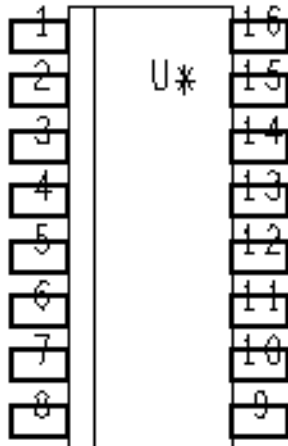


### soic14

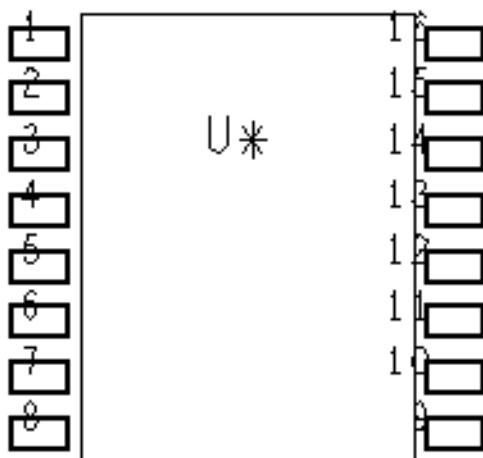




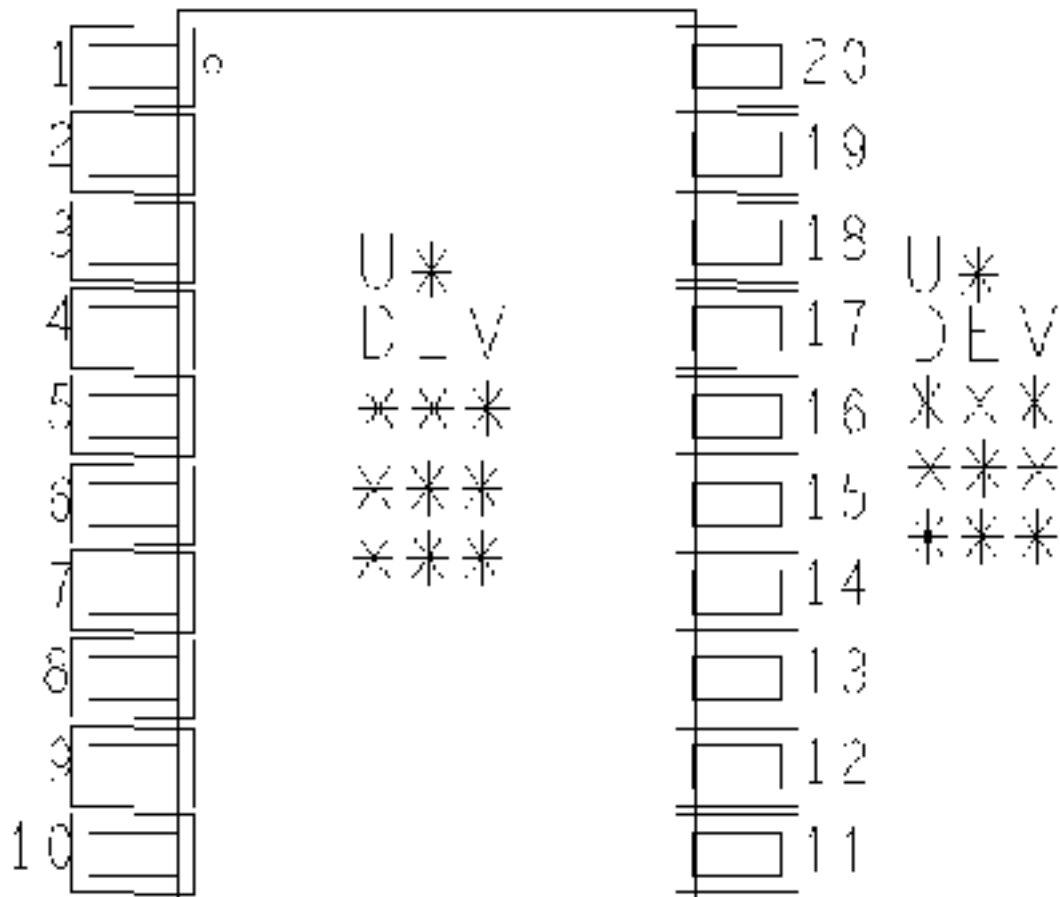
### soic16



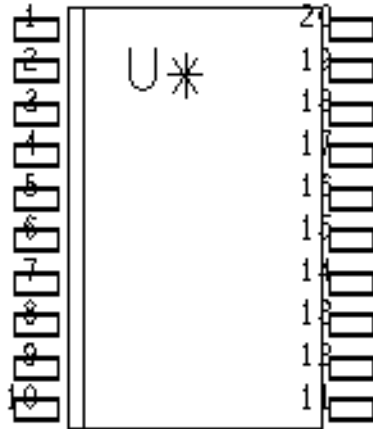
### soic16w



## soic20

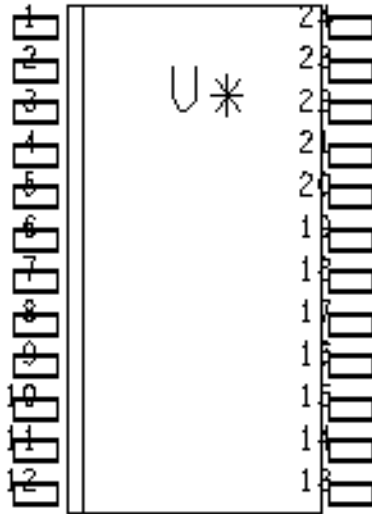


## soic20w

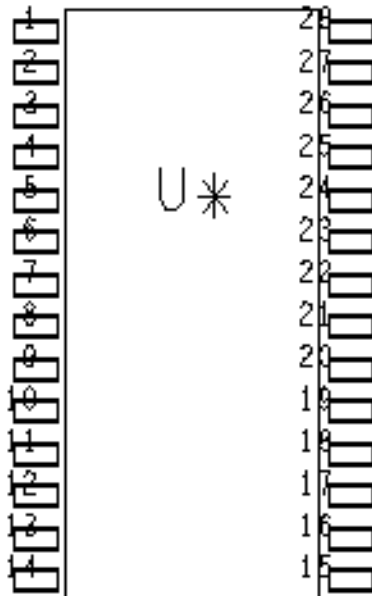




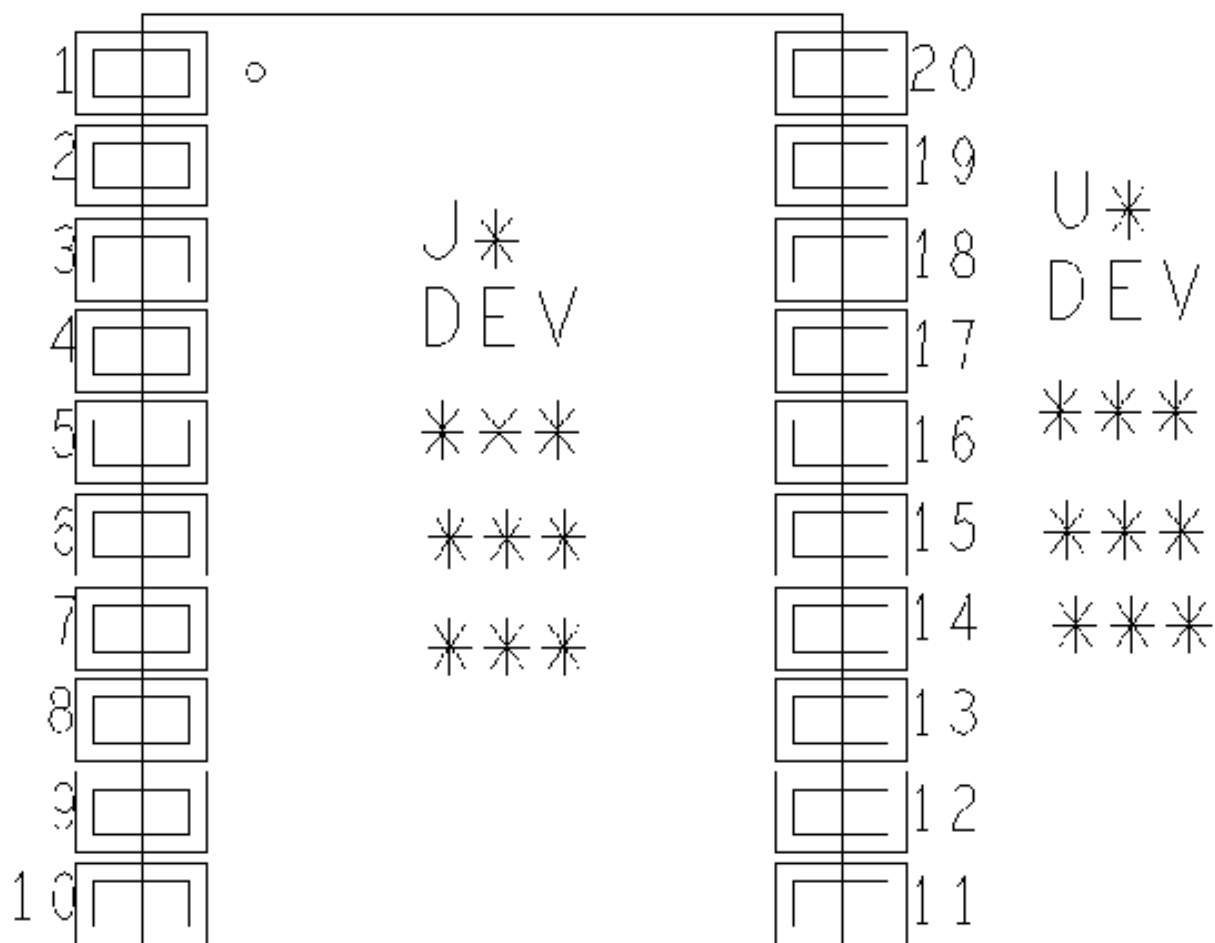
### soic24w



### soic28w

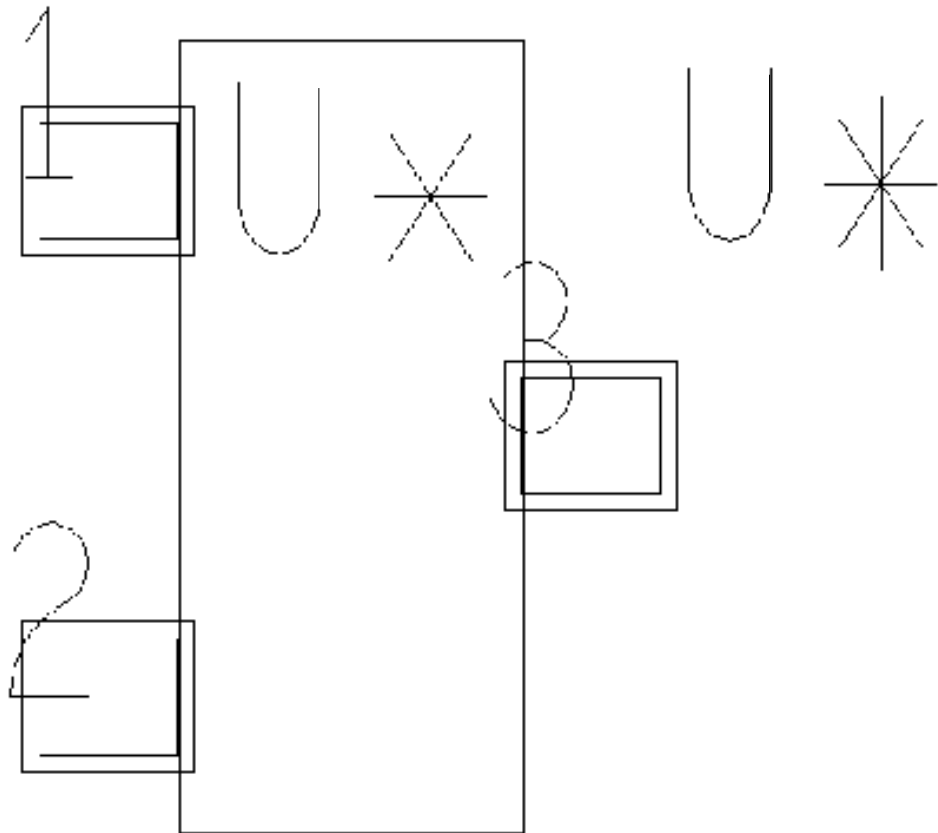


## sol120

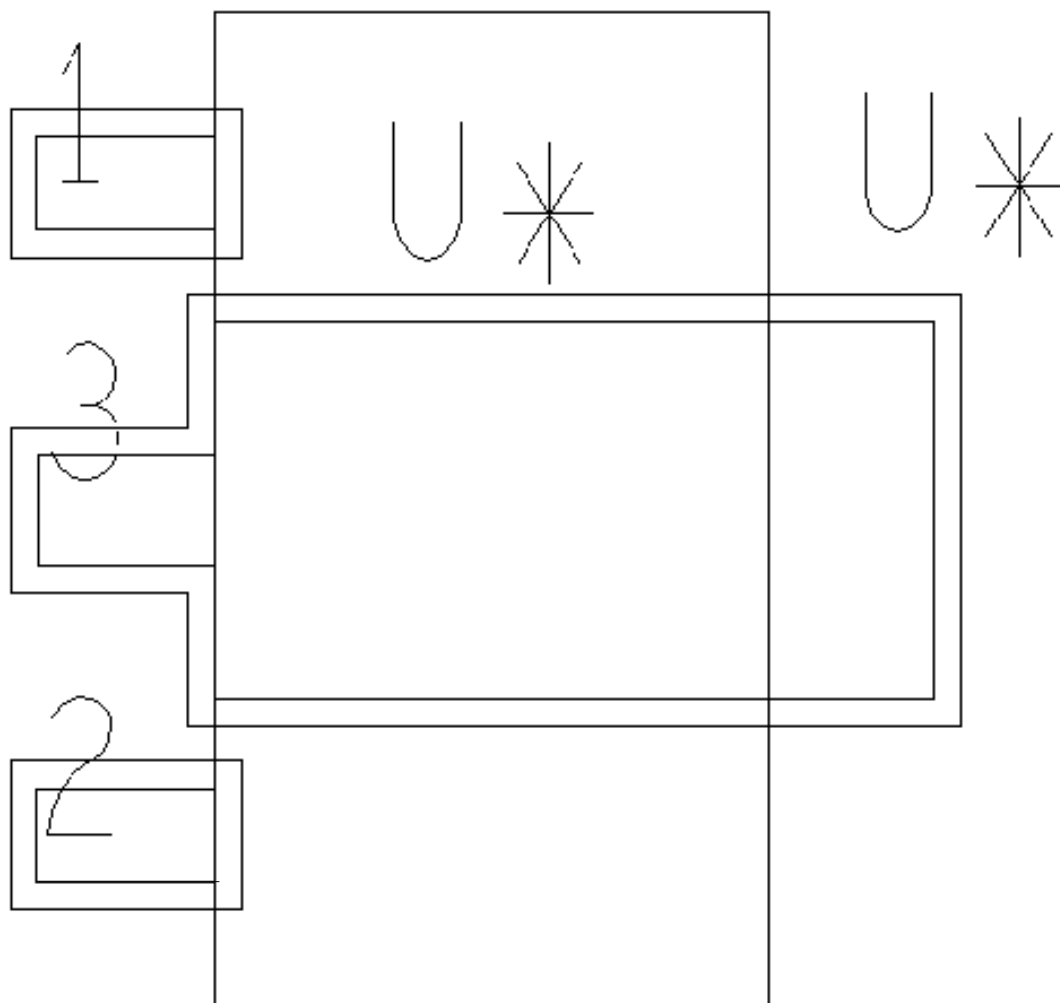


## Transistors

### sot23

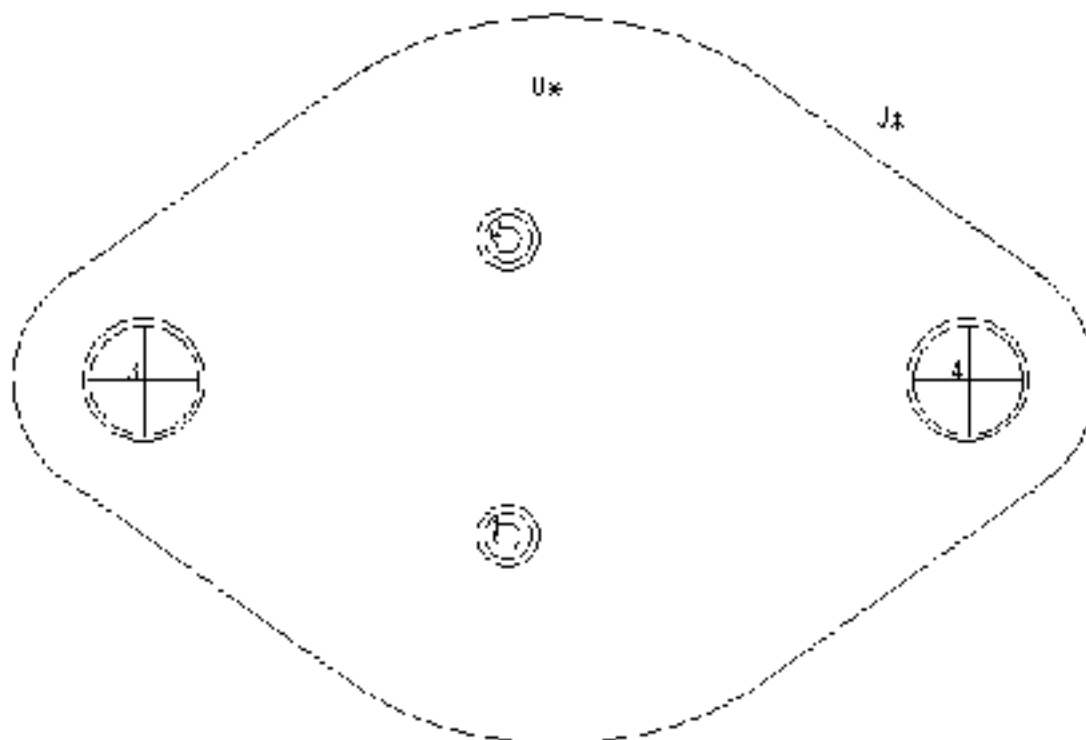


**sot89**

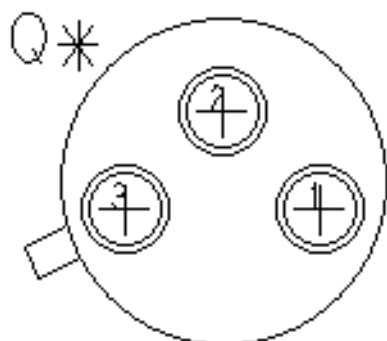




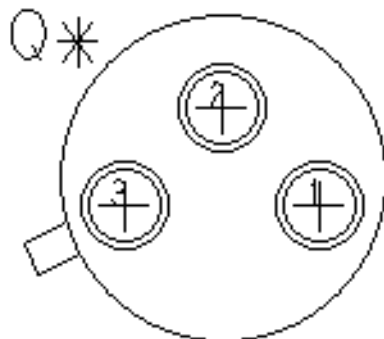
to3



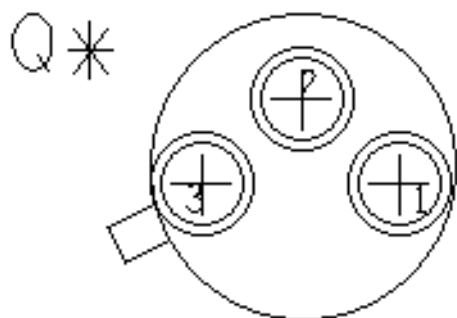
to5



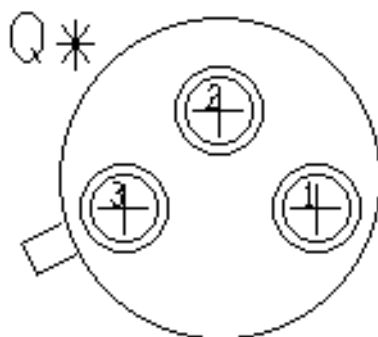
**to12**



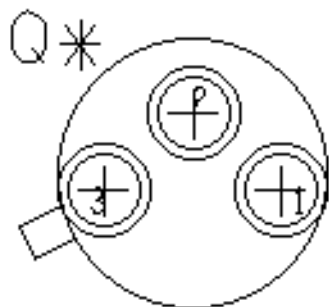
**to18**



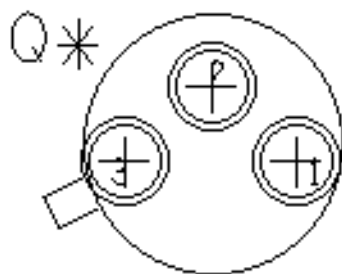
**to39**



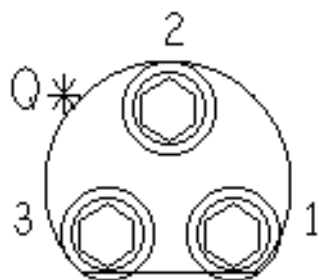
**to46**



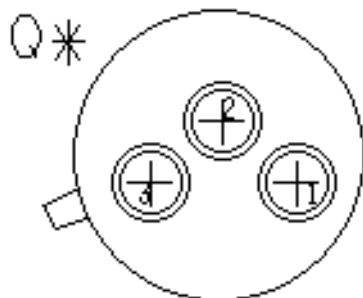
**to52**



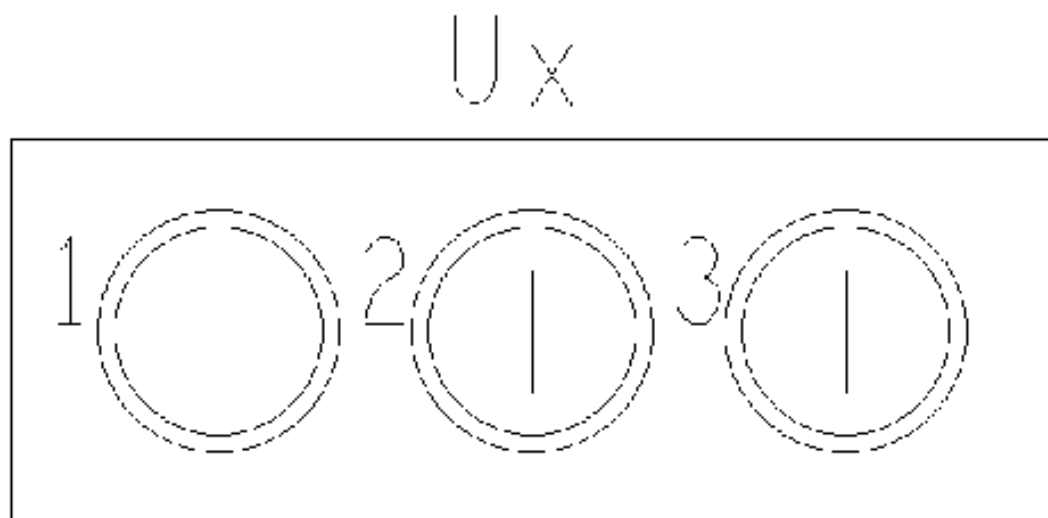
**to92**

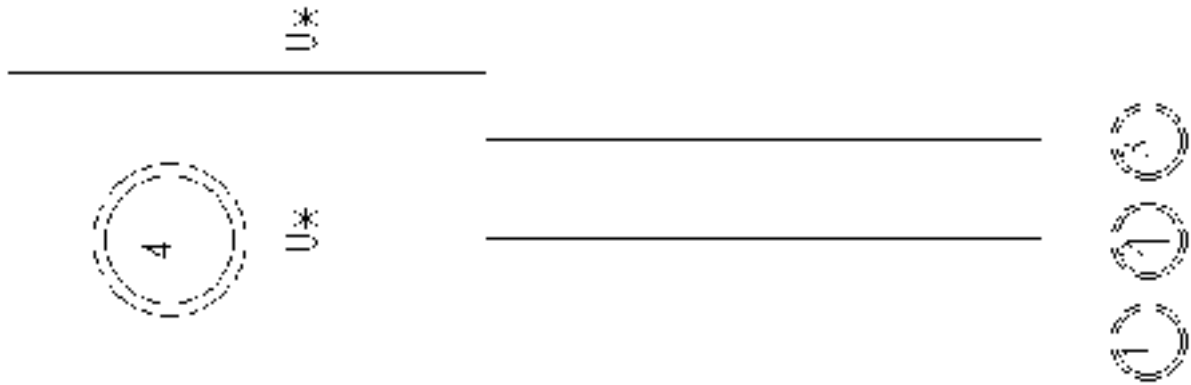


**to107**

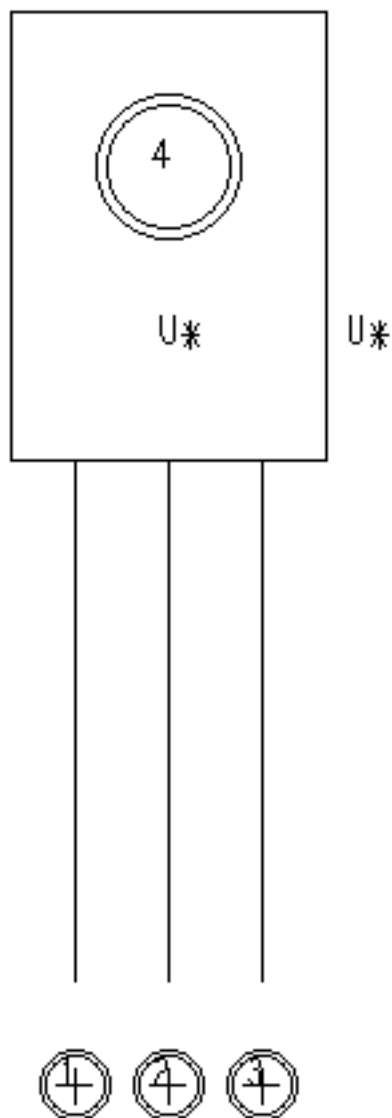


**to126**

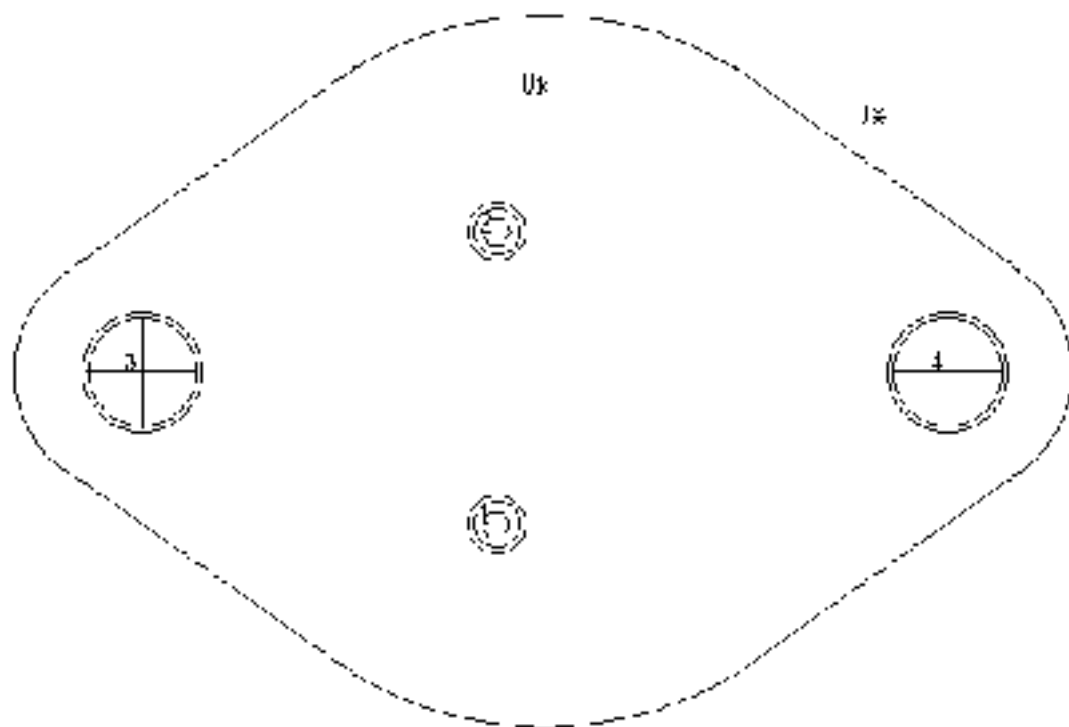




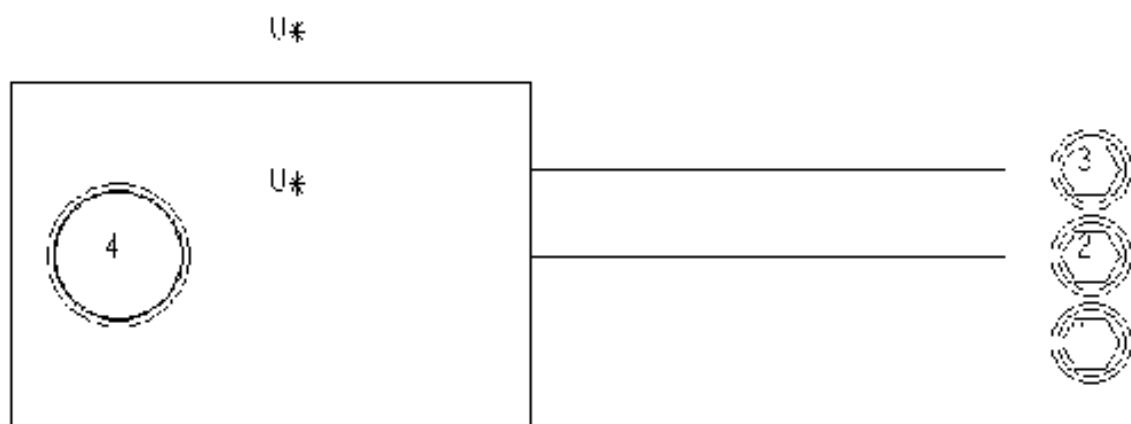
to126v



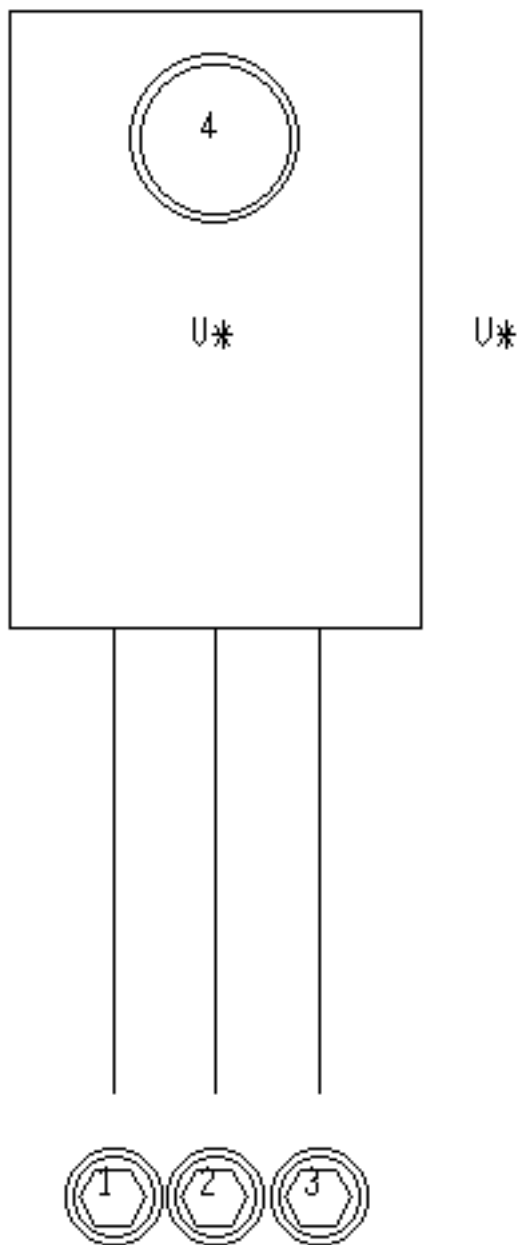
## to204aa



## to220abh

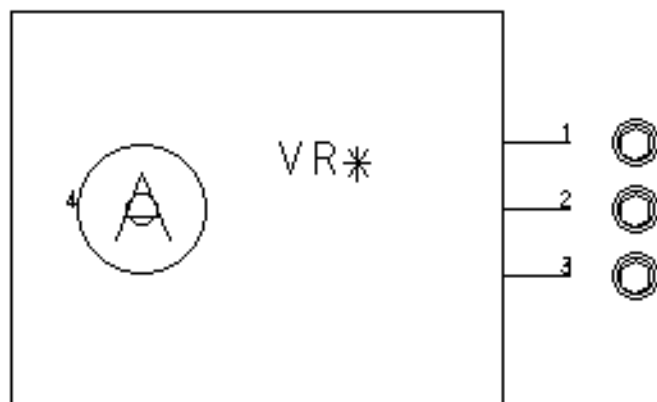


**to220abv**

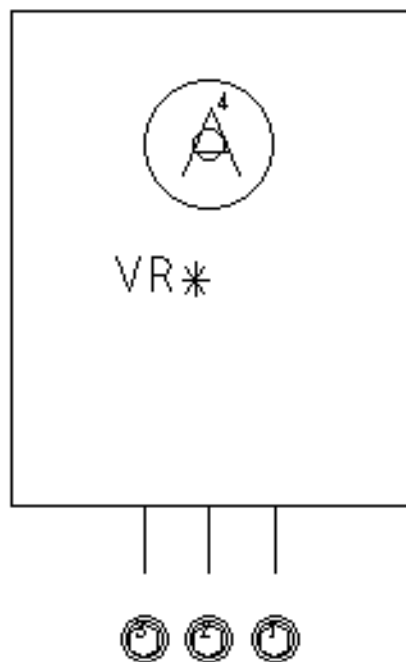




## to220h

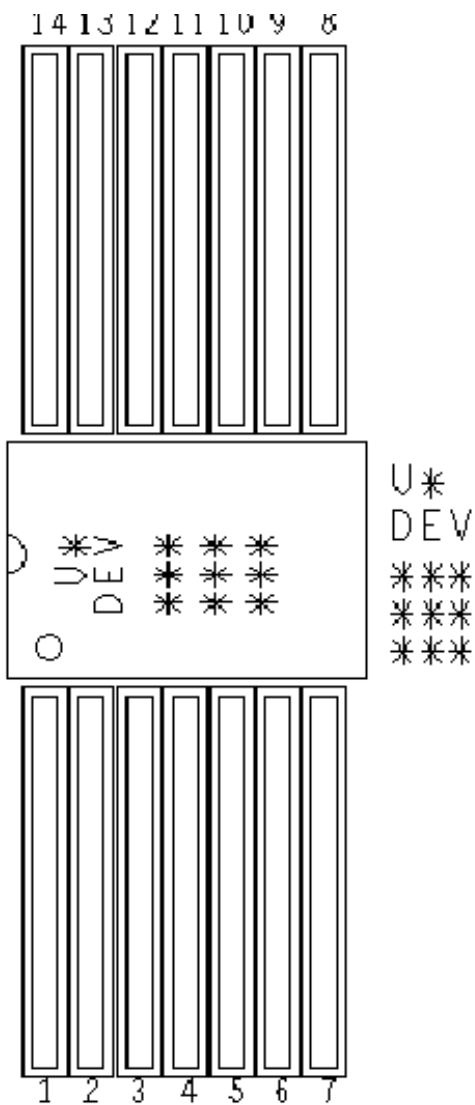


## to220v

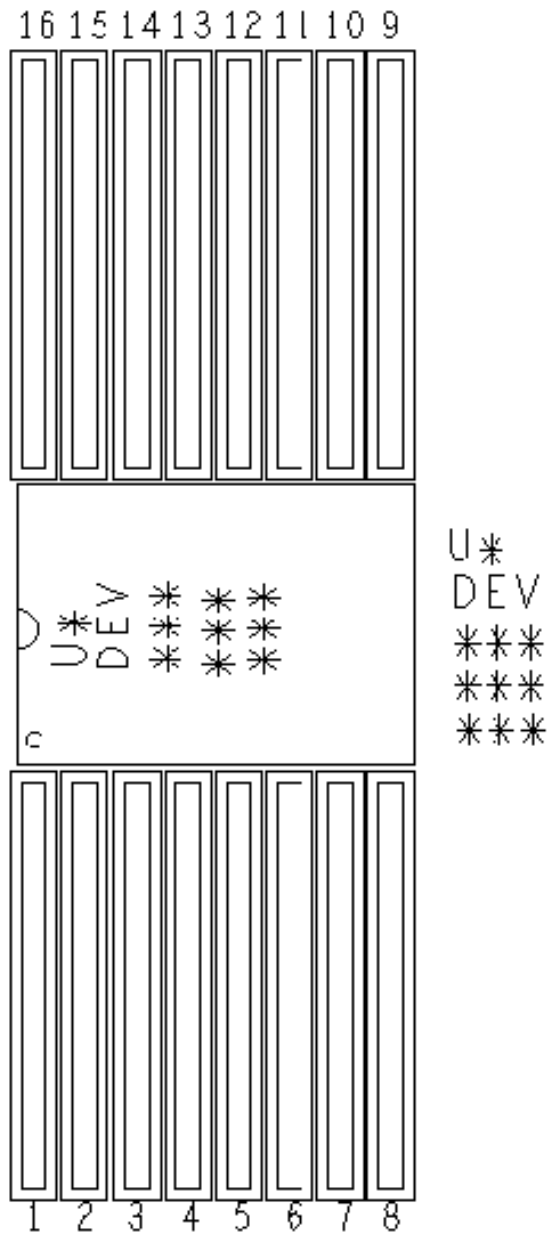


## Flat Packs

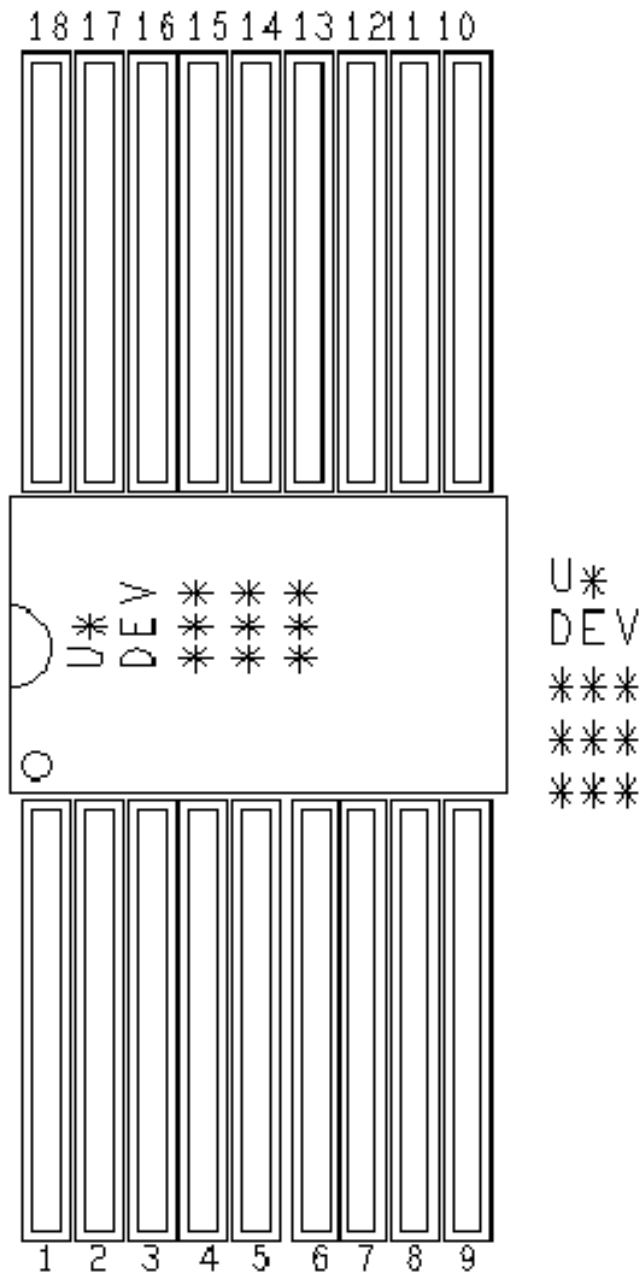
### flat14



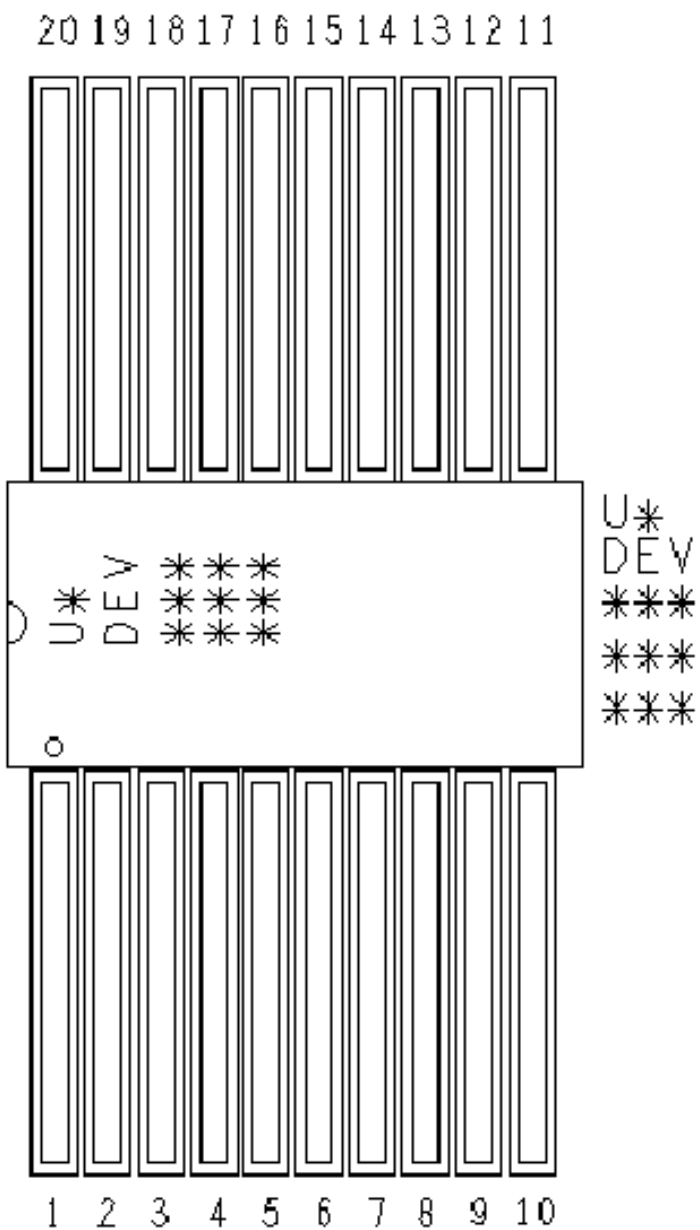
## flat16



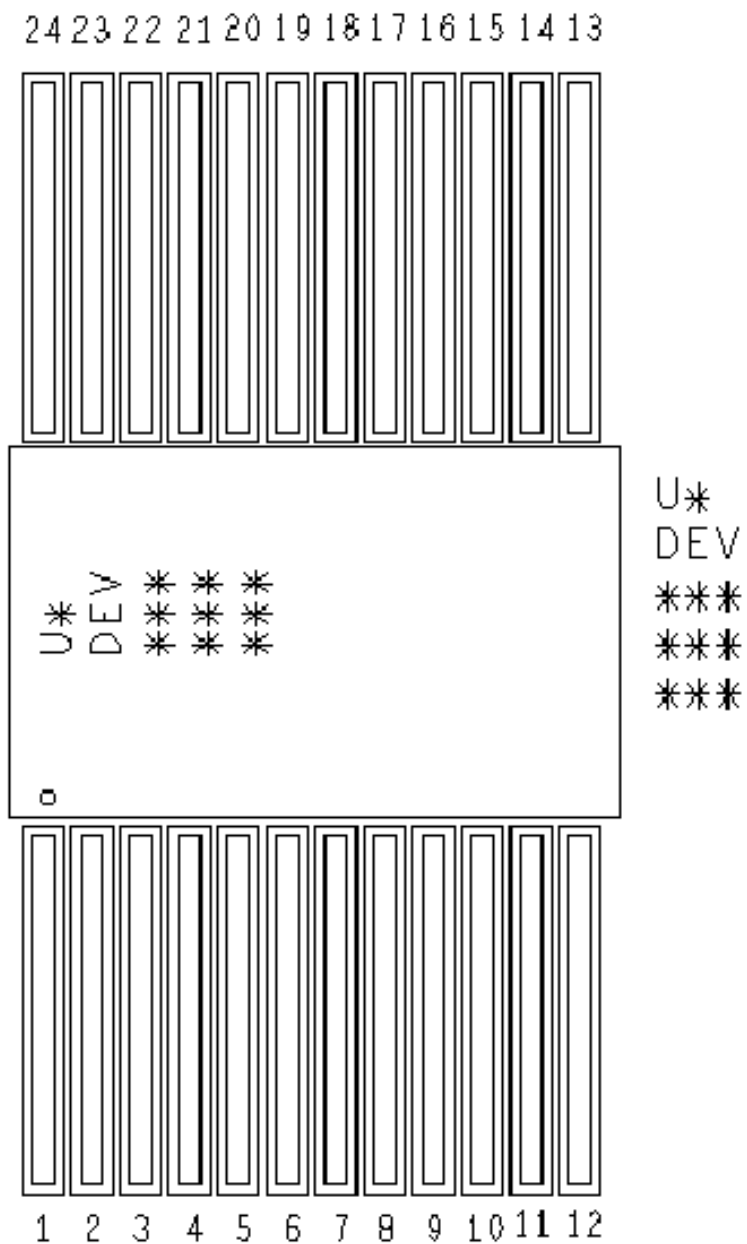
## flat18



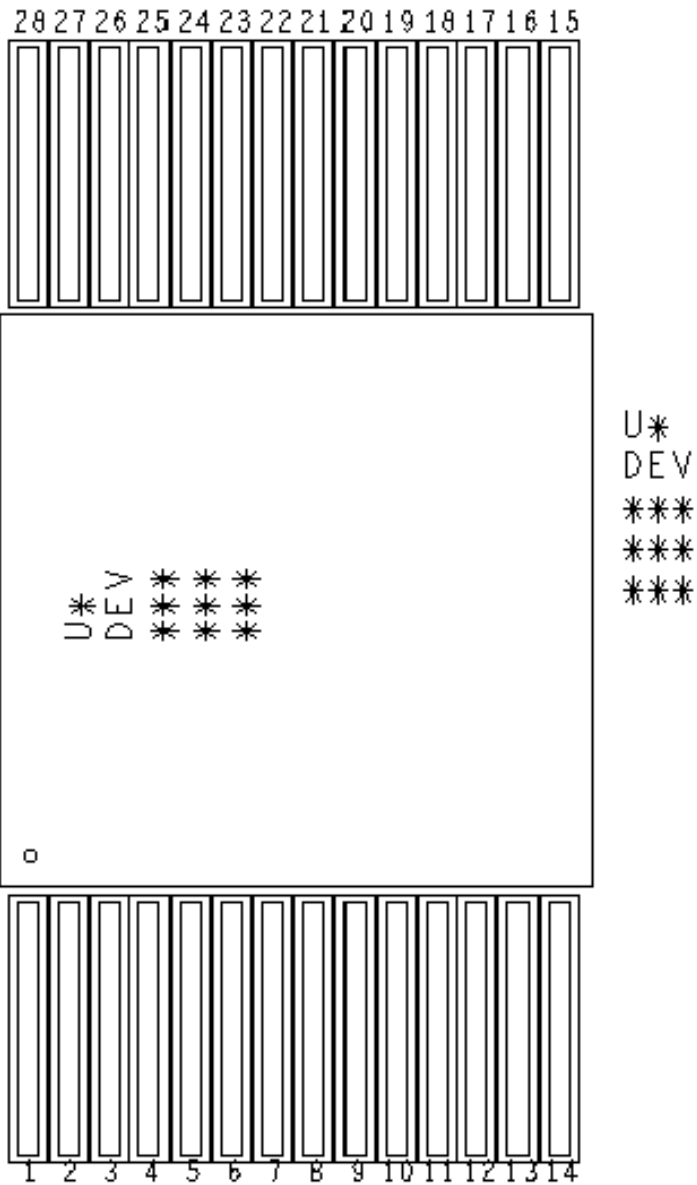
## flat20



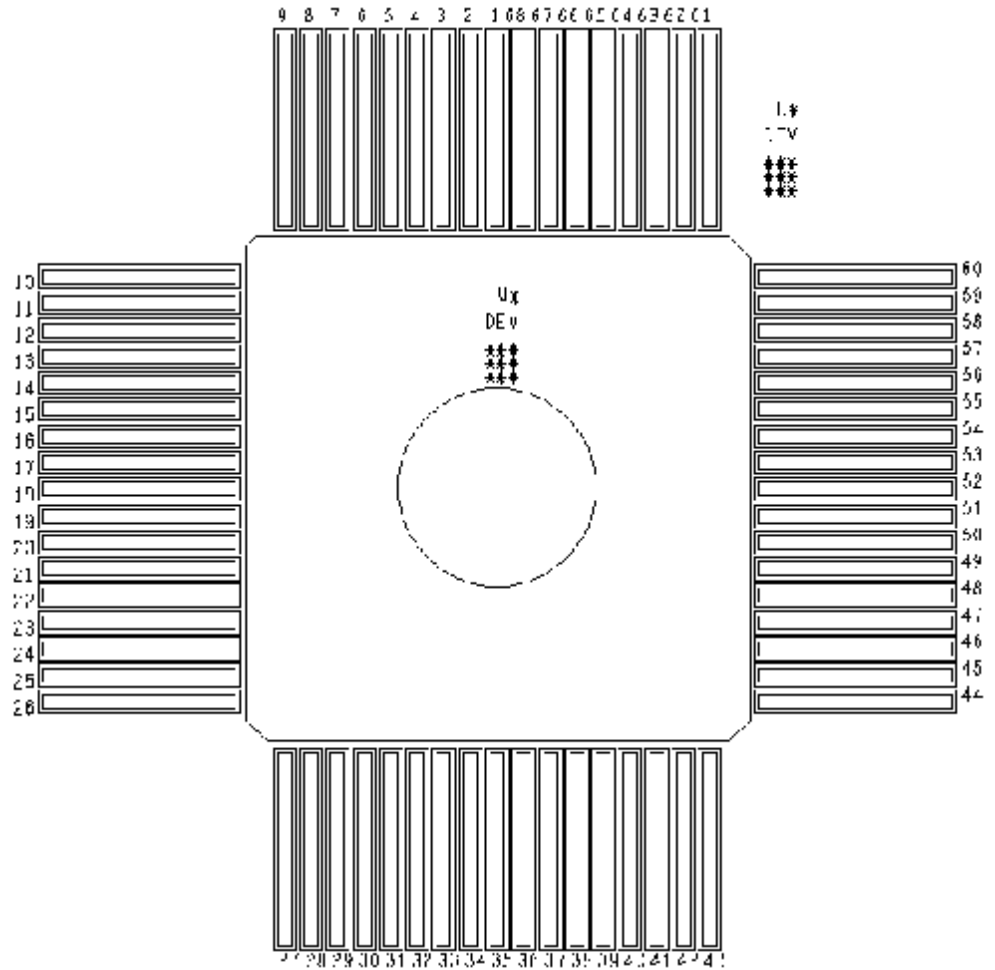
## flat24



## flat28

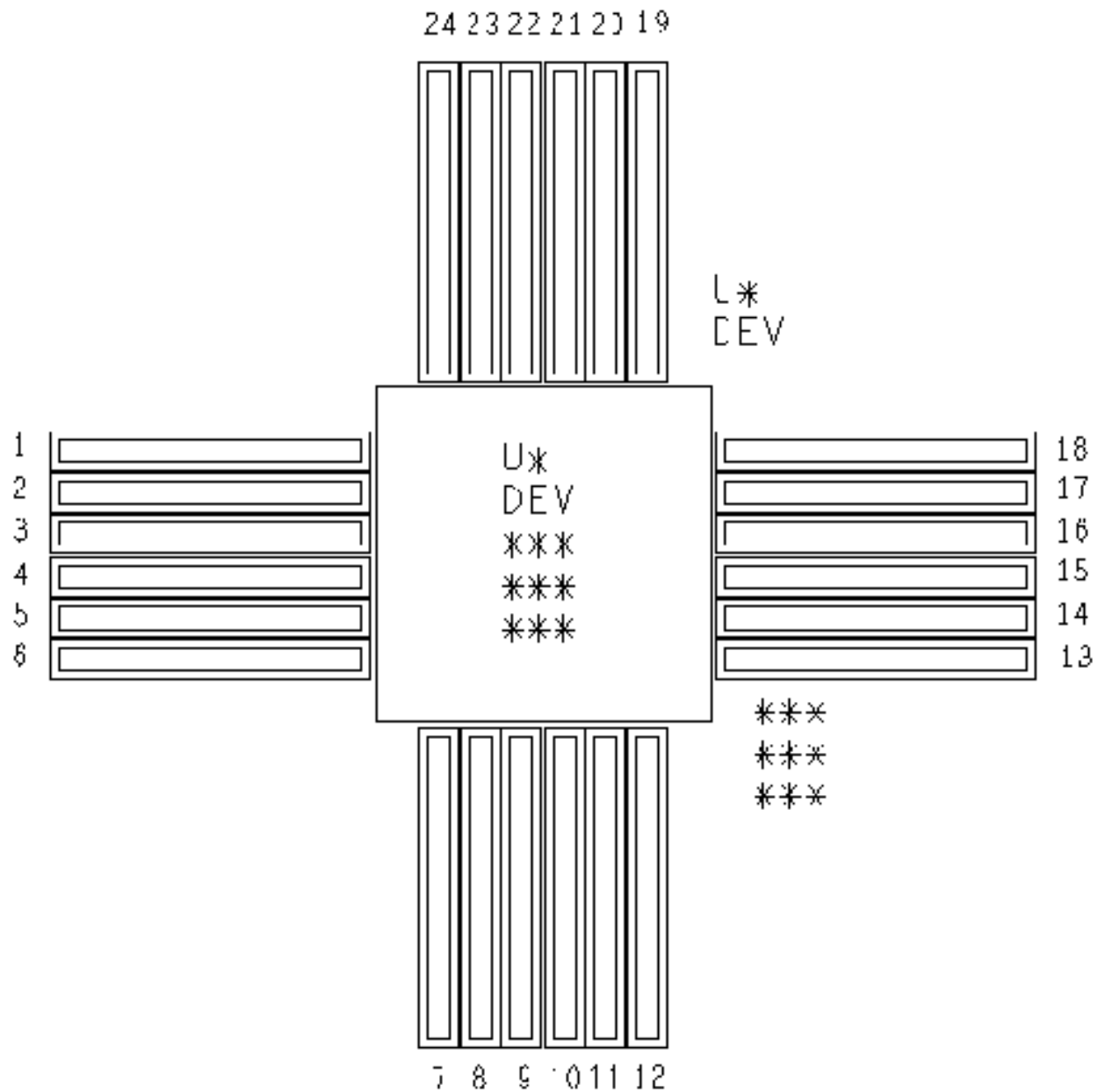


### cpfp68



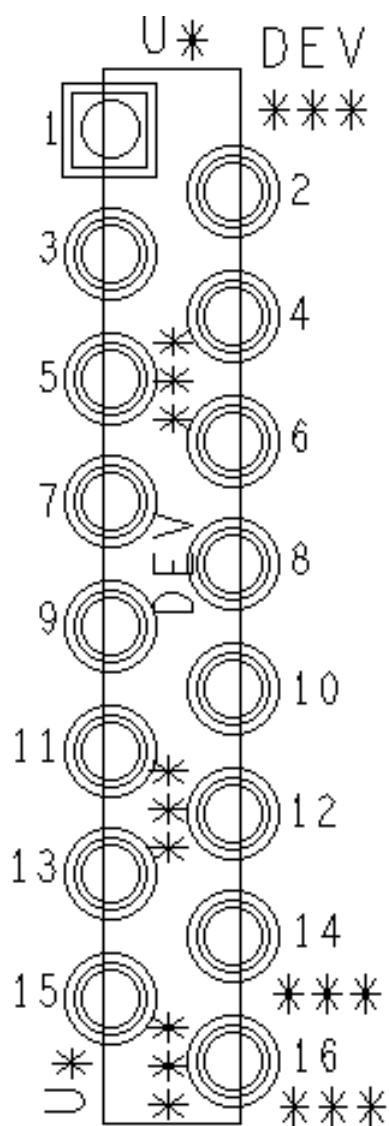


## quadflat24

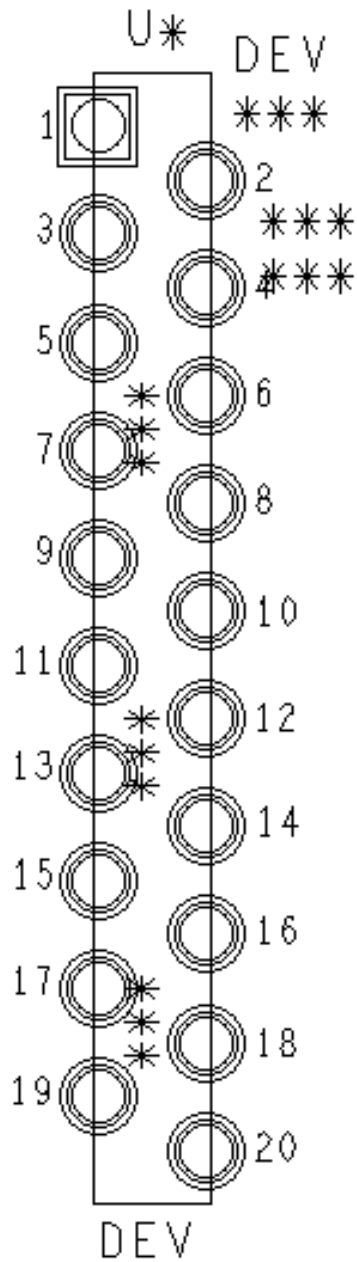


## ZIPs

### zip16



## zip20





---

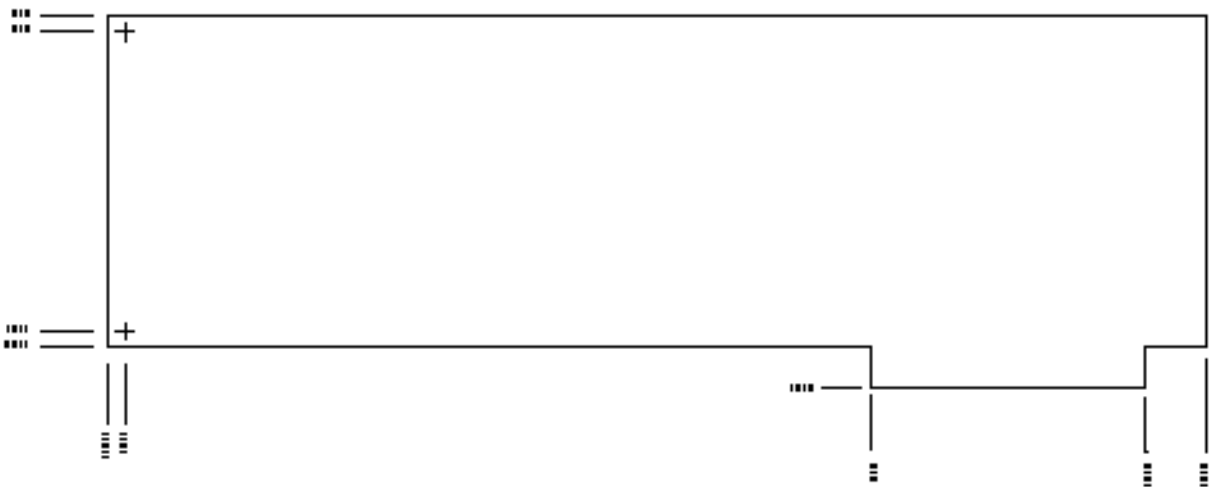
## Mechanical Symbol Library

---

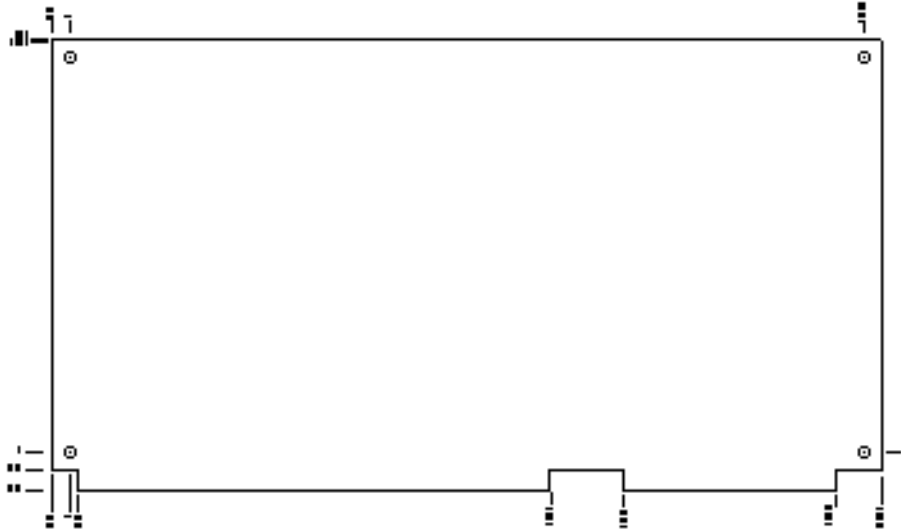
This chapter illustrates the mechanical symbols provided in the library.

### Card Outlines

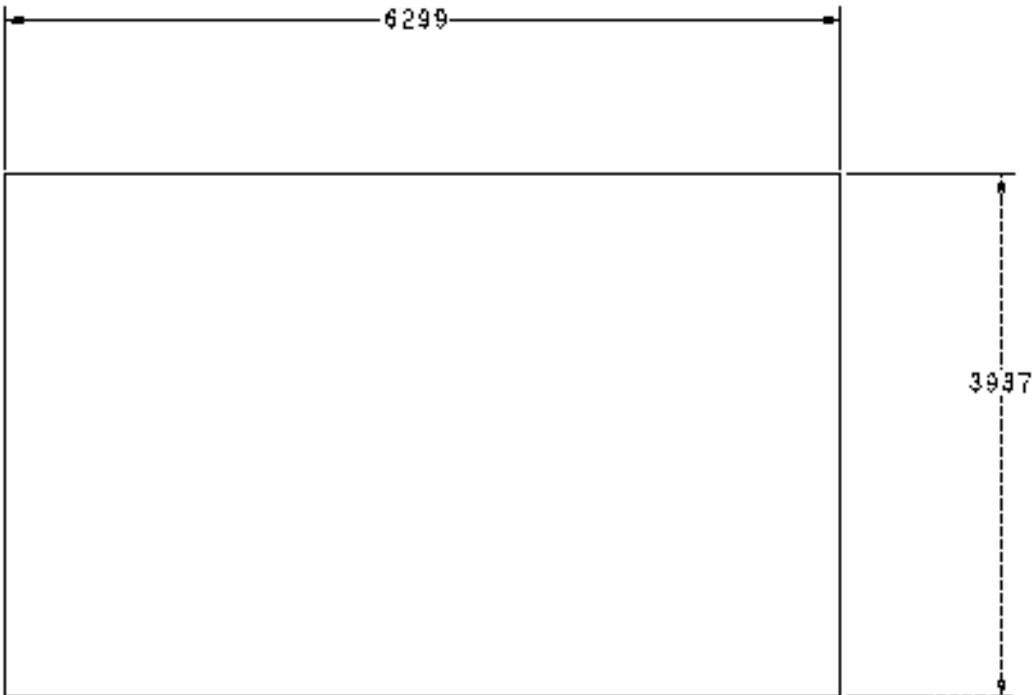
ibm



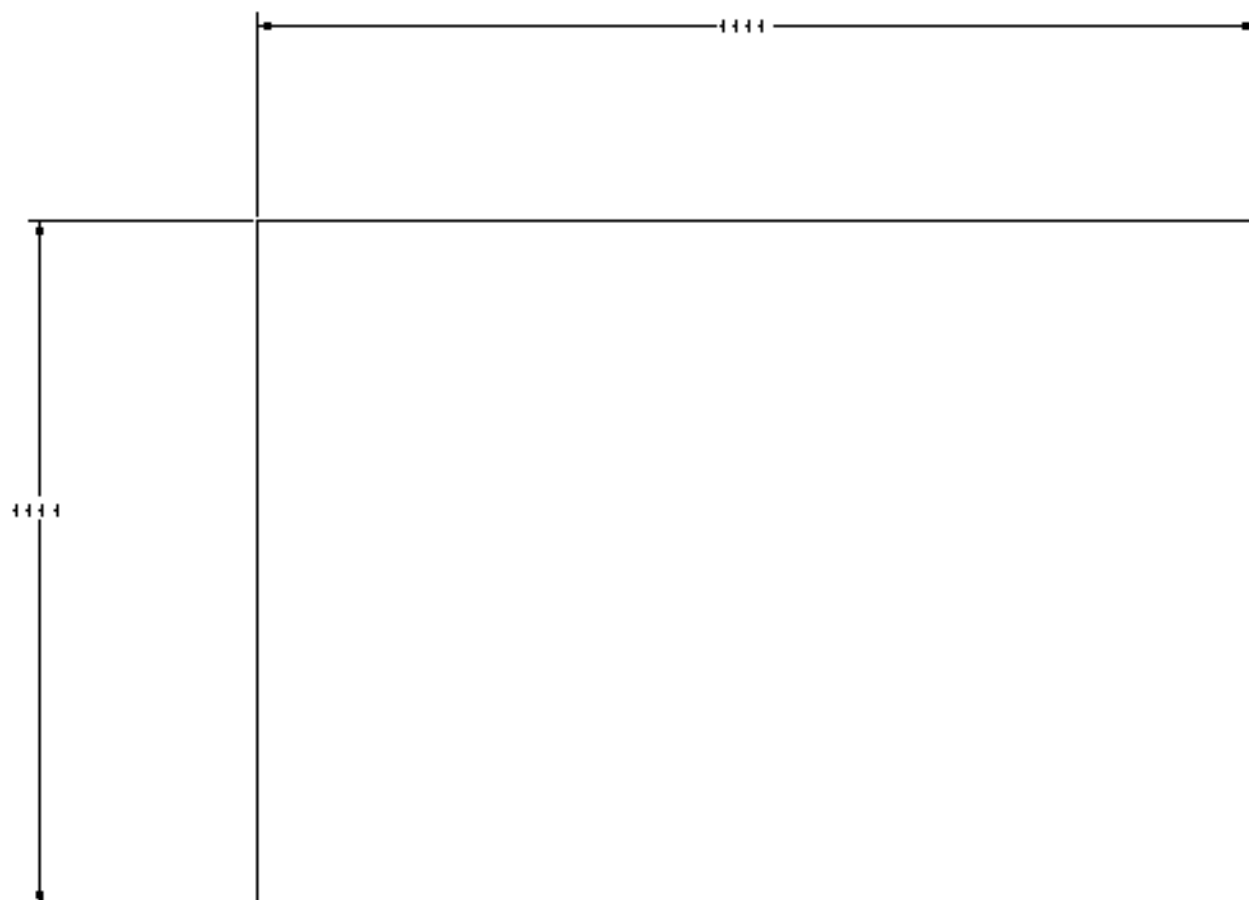
## multibus



## euro

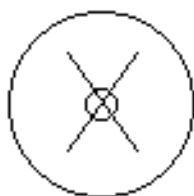


**eurod**



## **Mounting Holes**

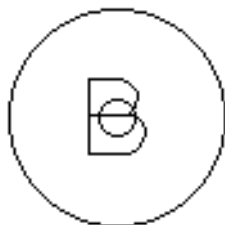
**mtq125**



**mtq156**



**mtq250**





---

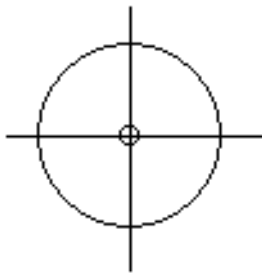
## Format Symbol Library

---

This chapter illustrates the format symbols provided in the Allegro PCB Editor library.

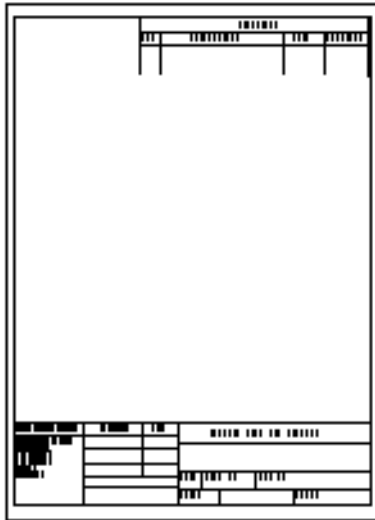
### Target

target

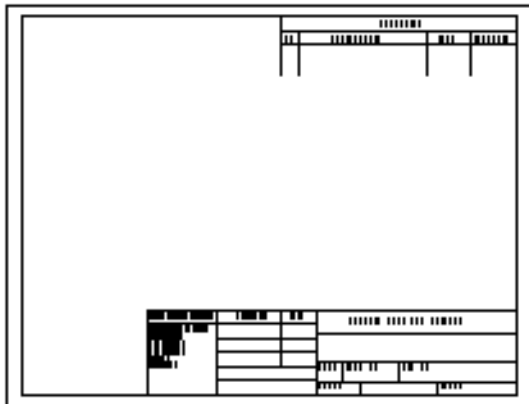


## Drawing Formats

### asizev



### asizeh

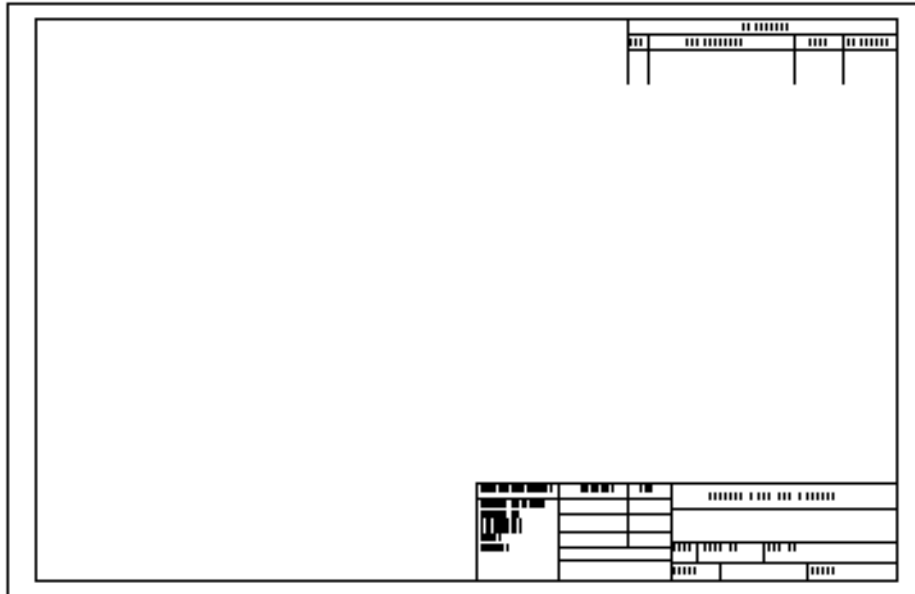


# Allegro PCB Editor User Guide: Defining and Developing Libraries

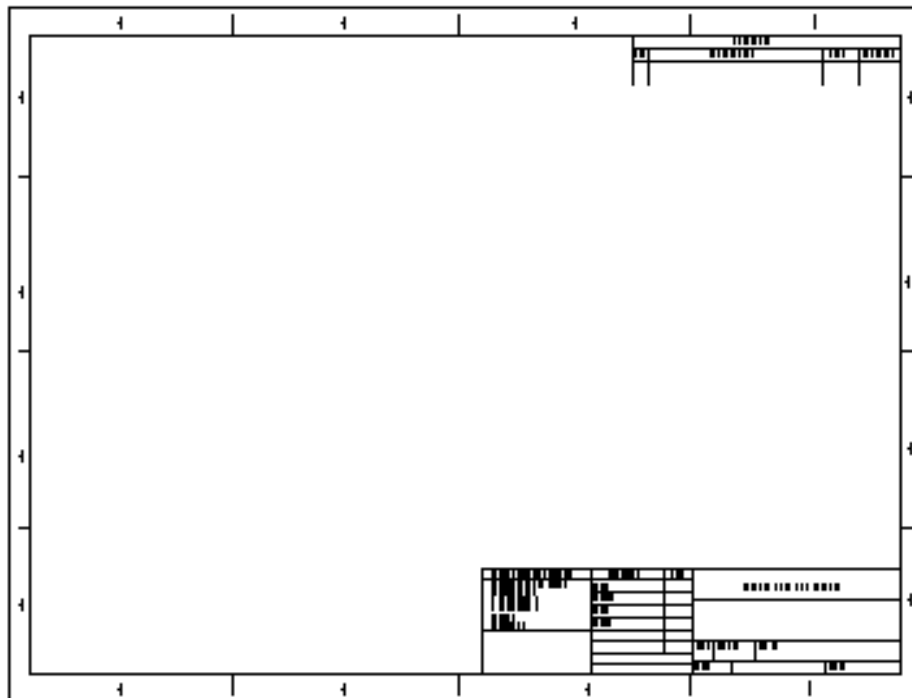
## Format Symbol Library

---

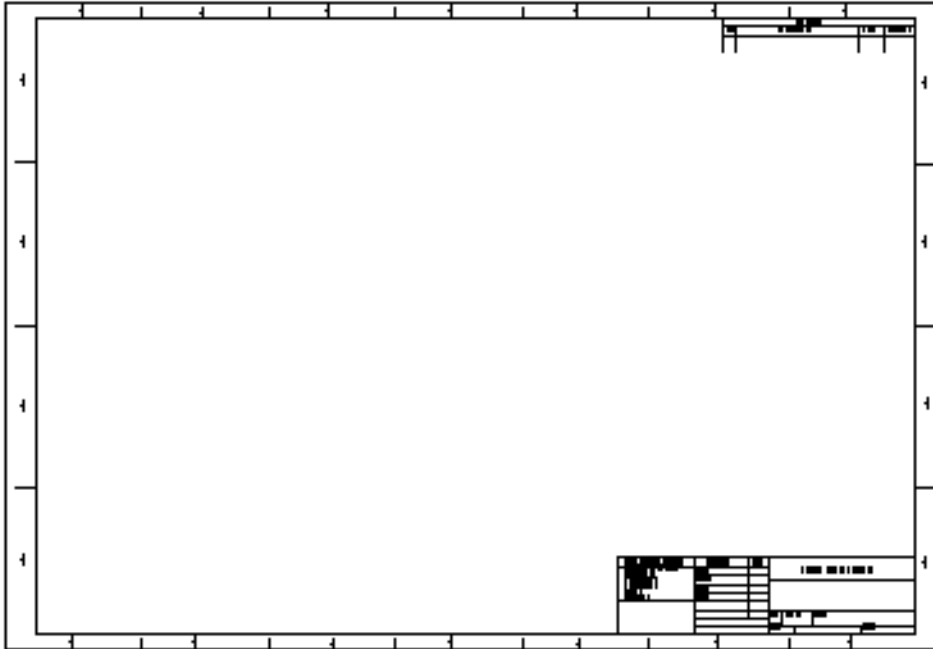
### bsize



### csize



## **dsize**



## **esize**

