

EMControl User Guide

Product Version 15.2
June 2004

© 1999-2004 Cadence Design Systems, Inc. All rights reserved.
Printed in the United States of America.

Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134, USA

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

All other trademarks are the property of their respective holders.

Restricted Print Permission: This publication is protected by copyright and any unauthorized use of this publication may violate copyright, trademark, and other laws. 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. This statement grants you permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used solely for personal, informational, and noncommercial purposes;
2. The publication may not be modified in any way;
3. Any copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement; and
4. Cadence reserves the right to revoke this authorization at any time, and any such use shall be discontinued immediately upon written notice from Cadence.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. The information contained herein is the proprietary and confidential information of Cadence or its licensors, and is supplied subject to, and may be used only by Cadence's customer in accordance with, a written agreement between Cadence and its customer. 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>	11
<u>About This Guide</u>	11
<u>How to Use This Guide</u>	11
<u>Brief Outline of Different Chapters</u>	12
<u>Typographic and Syntax Conventions</u>	13
<u>1</u>	
<u>Introduction to EMControl</u>	15
<u>EMControl Overview</u>	15
<u>EMControl Users</u>	17
<u>EMControl Tasks</u>	18
<u>Checking for EMC Rule Violations</u>	19
<u>Tasks to be Performed</u>	19
<u>2</u>	
<u>Installing and Accessing EMControl</u>	21
<u>Installing EMControl</u>	21
<u>EMControl System Directory Structure</u>	21
<u>EMControl Run Directory Structure</u>	23
<u>Default EMControl Rule Set</u>	24
<u>Advanced EMControl Rule Set</u>	25
<u>License for EMControl</u>	25
<u>EMControl Product License</u>	25
<u>Default Rule Set License</u>	25
<u>Advanced Rule Set License</u>	25
<u>Accessing EMControl</u>	26
<u>3</u>	
<u>Setting Up the EMControl System Environment</u>	27
<u>Overview</u>	27

EMControl User Guide

<u>Setting Up the System Directory</u>	29
<u>The EMControl Mapping File</u>	30
<u>Setting Up Mapping Files</u>	31
<u>Setting Up Rule Files</u>	33
<u>Setting Up Rule Parameter Files</u>	34
<u>Setting Up SKILL Files</u>	36
<u>Setting Up System Preference</u>	38

4

<u>Setting Up the EMControl Design Environment</u>	41
<u>Overview</u>	41
<u>Setting Up Power/Ground Plane</u>	43
<u>Setting Up Power/Ground Net Matching List</u>	45
<u>Automatically Attaching Properties to Design Objects</u>	46
<u>Manually Attaching Properties to Selected Design Objects</u>	48
<u>Attaching Component Properties</u>	49
<u>Attaching Net Properties</u>	51
<u>Attaching Room Properties</u>	53
<u>Saving the EMC Properties Assigned to a Design</u>	56
<u>Loading EMC Properties Previously Saved</u>	57

5

<u>Performing EMControl Rule Checking</u>	61
<u>Overview</u>	61
<u>Define the Scope of the Checking</u>	62
<u>Select Rules to be Checked</u>	64
<u>Displaying Rule Files Selected</u>	64
<u>Displaying the Rules in a File</u>	64
<u>Selecting Rules to Use for Design Checking</u>	65
<u>Getting Help on a Rule</u>	65
<u>Customize EMControl Variables</u>	66
<u>Customizing EMControl Rule Variables</u>	66
<u>Customizing EMControl Global Variables</u>	67
<u>Audit EMControl Rules</u>	68
<u>Execute EMControl Rules</u>	69

EMControl User Guide

<u>View Results</u>	70
<u>View the Log File for Automatically Property Tagging</u>	70
<u>View Audit Reports</u>	74
<u>View Checking Reports</u>	78
<u>Reset Design After Rule Checking</u>	78

6

<u>Resolving EMC Rule Violations in Your Design</u>	79
<u>Overview of Rule-Checking</u>	79
<u>Viewing the Results of Rule-Checking</u>	79
<u>Viewing a Violation</u>	80
<u>Highlighting Objects</u>	81
<u>Filtering Violations</u>	81
<u>Cross-Probing Multiple Violations</u>	84
<u>Loading a Markers File</u>	85
<u>Hiding a Violation Message</u>	85
<u>Saving a Markers File</u>	86
<u>Reading an Execute Report</u>	86
<u>Header Information</u>	86
<u>Rule-Specific Information</u>	87
<u>Summary Information</u>	88

7

<u>Rule Development</u>	89
<u>Overview</u>	89
<u>Develop New Predicates</u>	91
<u>Add a New Predicate</u>	93
<u>Delete a Predicate</u>	93
<u>Edit an Existing Predicate</u>	94
<u>Develop New Parameters</u>	95
<u>Add a Global Parameter</u>	96
<u>Delete a Global Parameter</u>	96
<u>Edit a Global Parameter</u>	96
<u>Develop New Rules</u>	97
<u>Create a Rule File</u>	98

EMControl User Guide

<u>Create Rule Parameters</u>	98
<u>Compile a Rule File</u>	99
<u>Create a Rule Help File</u>	99
<u>Create an Audit Rule File</u>	101
<u>Running new rules</u>	102
<u>Adding a Directory</u>	103
<u>Adding a Parameter</u>	104

A

<u>ARL Training Guide</u>	105
<u>Introduction</u>	105
<u>Language Highlights</u>	105
<u>EMControl Objects</u>	105
<u>EMControl Predicates</u>	106
<u>Getting a Feel for the Language</u>	107
<u>Basic Language Constructs</u>	107
<u>Variables and Base Objects</u>	108
<u>Base Objects and Implied Looping</u>	109
<u>Predicate Calls</u>	109
<u>Variable Typing</u>	110
<u>ARL Operators</u>	110
<u>Exercises</u>	111
<u>List Manipulation</u>	112
<u>What are Lists</u>	113
<u>List Manipulation Routines</u>	113
<u>Foreach Construct</u>	116
<u>Saving Intermediate Results Within a Foreach construct</u>	116
<u>If Construct</u>	117
<u>Exercise</u>	118
<u>Dissection of an existing rule</u>	119
<u>Rule critical net via count</u>	119
<u>Rule conn in low freq regions</u>	120
<u>Laboratory Exercises</u>	122
<u>Basic Rules</u>	122
<u>Modifying existing EMControl rule (conn in low freq regions)</u>	122

EMControl User Guide

<u>Custom EMC rule writing</u>	123
 B	
<u>EMControl Rules</u>	125
<u>Overview</u>	125
<u>Placement Rules</u>	126
<u>central clock</u>	126
<u>conn in low freq regions</u>	127
<u>gnd screw between clock and conn</u>	128
<u>Bypass Rules</u>	129
<u>bypass cap type</u>	129
<u>bypass drvr rcvr bidir</u>	130
<u>bypass fast sw trans</u>	131
<u>critical IC 3caps C 2C 4C</u>	132
<u>critical IC loop area</u>	134
<u>decouple emc regions</u>	137
<u>fence off emc regions</u>	138
<u>DC Routing Rules</u>	139
<u>bypass pwr trace</u>	139
<u>filters to clean ground</u>	140
<u>max pwr gnd resistance</u>	141
<u>nets over clean gnd</u>	142
<u>Signal Routing Rules</u>	143
<u>critical net man ratio</u>	143
<u>critical net via count</u>	144
<u>filtered IO signals</u>	145
<u>max critical net xtalk</u>	145
<u>Signal Quality Rules</u>	146
<u>clock spectral content</u>	147
<u>critical net ringing</u>	149
<u>critical net termination</u>	150
<u>single diff mode EMI</u>	151
<u>sum diff mode EMI</u>	153
<u>Default Rules</u>	156
<u>bypass cap per plane square</u>	156

EMControl User Guide

<u>bypass critical IC</u>	157
<u>comp not conn dist</u>	158
<u>comp to conn dist</u>	159
<u>gnd under clock</u>	160
<u>pwr gnd plane separation</u>	161
<u>pwr gnd trace width</u>	162
<u>critical net card edge dist</u>	162
<u>no critical net thru IO comps</u>	163
<u>critical net hole dist</u>	164
<u>serial resistor position</u>	165
<u>Advanced Rules</u>	167
<u>bypass plane split</u>	167
<u>critical net return path</u>	168
<u>critical net via pin ratio</u>	169
<u>return path near signal via</u>	170
<u>shield net</u>	171
<u>vias per plane square</u>	173
<u>plane overlap</u>	175
<u>ic power sequence</u>	176
<u>bulkcap decap trace</u>	176
<u>ceracap decap trace</u>	178
<u>gap between split plane</u>	179
<u>matched via number diffpair</u>	180
<u>symmetry of via position of diffpair</u>	180
<u>termination of diffpair</u>	181

C

<u>EMControl Predicates</u>	183
<u>Overview</u>	183
<u>Physical Environment Objects and Predicates</u>	183
<u>General-Purpose Predicates</u>	184
<u>Design Predicates</u>	189
<u>Component Predicates</u>	191
<u>Net Predicates</u>	196
<u>Pin Predicates</u>	203

EMControl User Guide

<u>Via Predicates</u>	209
<u>Shape Predicates</u>	210
<u>Polygon Predicates</u>	213

EMControl User Guide

Preface

This preface discusses the following:

- [About This Guide](#)
- [How to Use This Guide](#)
- [Brief Outline of Different Chapters](#)
- [Typographic and Syntax Conventions](#)

About This Guide

This user guide shows you how to use the EMControl tool to check the electromagnetic compliance of systems. This ensures that electronic systems operate in their environment without affecting any other system.

The EMControl tool checks for electromagnetic compliance through a set of rules. This guide describes these rules, explains all the necessary concepts and procedures required for using them, and also explains the Advanced Rule Language (ARL) in which these rules are written. An understanding of ARL enables the user to write his own rules as well as edit the existing rules to suit his requirements.

How to Use This Guide

The user guide is organized in a way that it begins with a brief introduction of the EMControl tool followed by one chapter each on the various high level tasks that can be performed by the tool.

The purpose behind this guide is to:

- provide conceptual understanding of the tool
- explain the various rules and predicates which are used in EMControl
- enable the user to use the rules according to his design requirements
- enable the user to write new rules

If you are a new user and do not have any prior working experience with the EMControl tool, then start your learning process from the first chapter and continue exploring the different tools in sequence as covered in the user guide. If you are using the user guide as reference, then you may directly reference any chapter corresponding to a particular topic. Refer to the [Brief Outline of Different Chapters](#) section below for details.

Brief Outline of Different Chapters

This guide is organized into five chapters:

1. [Chapter 1, "Introduction to EMControl"](#)

This chapter gives a brief introduction to the EMControl tool. It discusses the use model for EMControl, the various tasks that can be performed by EMControl and the tasks performed by different users of the tool.

2. [Chapter 2, "Installing and Accessing EMControl"](#)

This chapter describes the installation and accessing of EMControl tools. The license issues about the default and advanced rule set are also discussed here.

3. [Chapter 3, "Setting Up the EMControl System Environment"](#)

This chapter describes the EMControl system environment, and also talks about how user can change the default settings to his own environment.

4. [Chapter 4, "Setting Up the EMControl Design Environment"](#)

This chapter describes the tasks for specifying various properties on the PCB design board. These properties deal with such issues as power/ground plane designation, power/ground net matching, etc. These properties must be specified prior to perform EMControl rule checking.

5. [Chapter 5, "Performing EMControl Rule Checking"](#)

This chapter discusses the various stages at which the various rules of the EMControl rule can be used. Then, it also talks about the tasks to be performed for checking the design for electromagnetic compliance.

6. [Chapter 6, "Resolving EMC Rule Violations in Your Design"](#)

This chapter discusses various steps after the rules have been executed on the design. It talks of how the results are displayed and where to find information about the execution results.

7. Chapter 7, “Rule Development”

This chapter describes the steps for adding new predicates, manipulating parameters, writing new rules, and getting rules working.

Typographic and Syntax Conventions

This list describes the syntax conventions used for tools used in the Allegro Design Entry HDL Rules Checker User Guide.

<code>literal (LITERAL)</code>	Nonitalic or (UPPERCASE) words indicate key words that you must enter literally. These keywords represent command (function, routine) or option names.
<code>argument</code>	Words in italics indicate user-defined arguments for which you must substitute a value.
<code> </code>	Vertical bars (OR-bars) separate possible choices for a single argument. They take precedence over any other character. For example, <code>command argument argument</code>
<code>[]</code>	Brackets denote optional arguments. When used with OR-bars, they enclose a list of choices. You can choose one argument from the list.
<code>{ }</code>	Braces are used with OR-bars and enclose a list of choices. You must choose one argument from the list.
<code>...</code>	Three dots (...) indicate that you can repeat the previous argument. If they are used with brackets, you can specify zero or more arguments. If they are used without brackets, you must specify at least one argument, but you can specify more. <code>argument...: specify at least one argument, but more are possible</code> <code>[argument]...: you can specify zero or more arguments</code>
<code>,...</code>	A comma and three dots together indicate that if you specify more than one argument, you must separate those arguments by commas.
<code>Courier font</code>	Indicates command line examples.

EMControl User Guide

Preface

Introduction to EMControl

EMControl Overview

Systems can adversely impact each other due to electromagnetic interference (EMI), or due to unwanted coupling of energy between conductors, components, and systems.

Electromagnetic compatibility (EMC) is the ability of electronic systems to function as expected within their intended environment without adversely affecting other systems.

An effective way to help meet EMC requirements is to use EMC design guidelines or EMC rules to screen the design for potential problems. You can use the EMControl tool to detect problem areas in your design early in the design cycle and take immediate steps to resolve those problems.

EMControl provides this capability of detecting problems by enabling you to repeatedly check your design against selected sets of EMC rules chosen by a user who has expertise in EMC.

The EMControl product includes several default sets of EMC rules. You can also write your own rules to verify specific design, environment, and regulatory requirements. Running EMControl early in the design cycle often helps to detect potential EMC problems before they can significantly impact product development.

The typical use model for EMControl to check electromagnetic compatibility (EMC) in high-speed printed circuit board (PCB) design is:

- Initialize
- Property Setup
- Rule Select
- Customize Parameters
- Audit
- Execute
- Results/Reports

Initialize

- Specifies EMC run directory.
- Specifies the parameter files and rule files
- Specifies the mapping files

Property Setup

- Sets the critical components, nets, and regions. The setup can be done automatically or interactively.
- Identifies EMC components by attaching the EMC_COMP_TYPE property.

Rule Select

- Selects the rules to be run.
- Specifies the scope of the design to be checked.

Customize Parameters

- Customizes the selected rule parameters, if required.
- Customizes the global parameters, if required.

Audit

- Performs a check to verify whether or not the required properties are setup.

Execute

- Executes the selected rules on the specified design.

Results/Reports

- Views the execute and audit reports.
- Cross-probes the EMC violations using the markers utility.

EMControl Users

EMC verification tasks can be performed by three types of EMControl users:

- EMC expert
- Design engineer
- Layout designer

The EMC expert determines the EMC requirements that must be met by the completed design and then maps these requirements to EMC rules. A combination of past experience and knowledge of the requirements of the design influence the EMC expert in determining:

- which EMC rules can be used as is
- which Cadence-supplied EMC rules must be modified
- whether or not any new EMC rules must be implemented in order to ensure that the design meets EMC requirements.

After the EMC expert has identified the EMC rule sets to be used to verify the design, the design engineer can set up the EMControl tool. The design engineer then implements the required EMC rule sets and associated variables; classifies the components, nets, and regions in the design according to their various levels of criticality; and attaches the PCB Editor and EMControl properties required by the selected EMC rules.

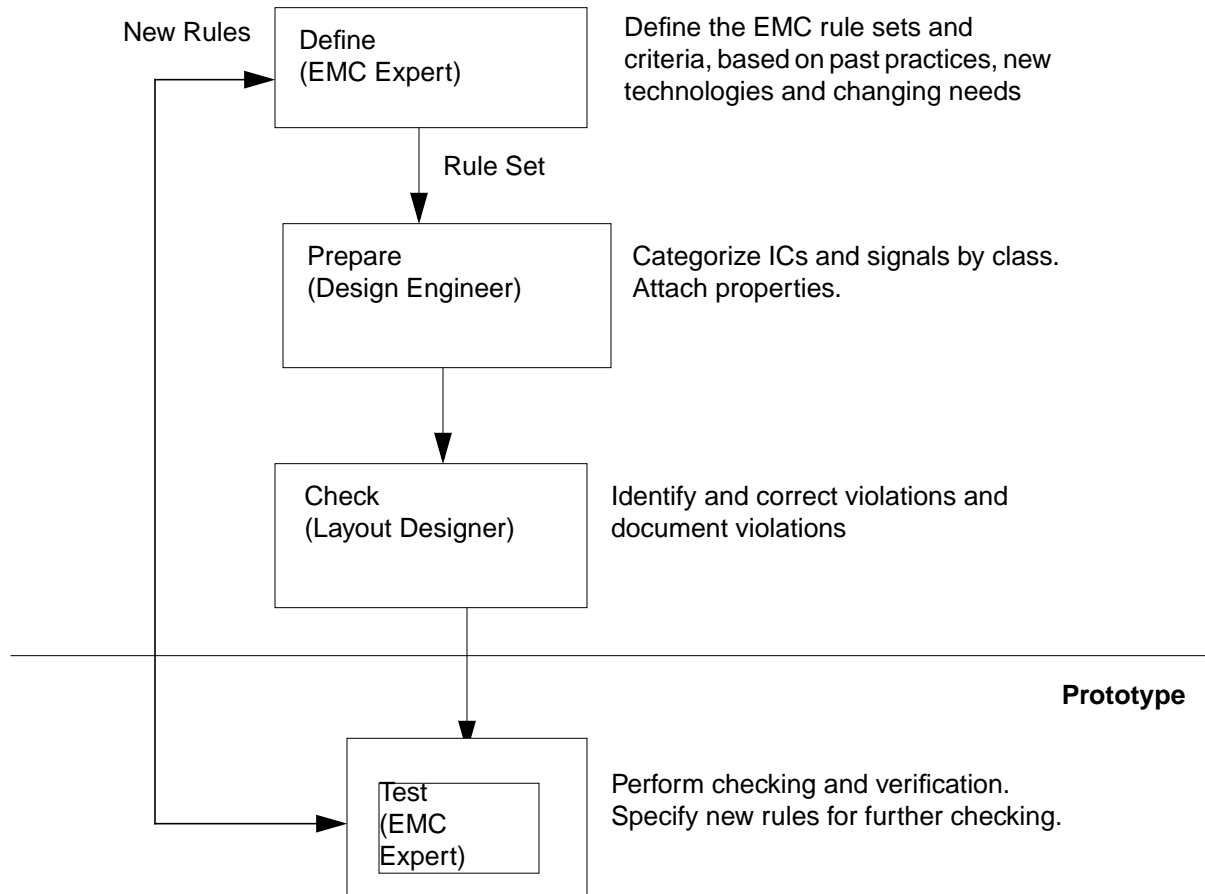
Once the rule set has been identified and property assignments are made, the layout designer can set up and run EMC rule checks. The layout designer identifies and corrects any EMC rule violations reported during EMC rule checking.

After a prototype is prepared using this method, the EMC expert can verify the design and identify new sets of EMC rules for further EMC rule checking.

The use of EMControl for design development is an iterative process. Throughout design development, the EMC expert, the design engineer, and the layout designer confer to ensure that all constraints are being met. They also define trade-offs between constraints and verify whether or not the completed design will meet all EMC requirements.

Figure [1-1](#) illustrates a typical workflow that you might use when designing for EMC.

Figure 1-1 EMControl Checking Flow



EMControl Tasks

EMControl enables the design engineer and the layout designer to perform the following tasks:

- Select specific EMC rule sets to check against the design.
- View a help file for each rule to determine:
 - ☐ Whether or not the rule should be used for an EMC rule-checking run.
 - ☐ Whether or not the rule should be customized.
 - ☐ The properties and the property values required by the rule.
 - ☐ The variables and their values required by the rule.

- Select the scope of the design that requires checking.
- Use Audit commands to verify that the properties required by the selected rules have been properly defined.
- Run the EMC rule checker to search for EMC violations.
- View the results of the check in a list of rule violations.
- Highlight an individual violation within the design.
- View the feedback from the Markers for correcting the EMC violations.

An EMC expert can use EMControl to customize the default rule-checking in the following ways:

- Customizing default EMC rules locally by editing variable values through the `emc_custom.par` file.
- Customizing default EMC rules for an entire site of users by editing variable values in the `emc_param.par` site parameter file.
- Writing and compiling new EMC rules, in the Cadence Advanced Rule Language (ARL). These rules can be applied in EMC rule checking.

Checking for EMC Rule Violations

The EMC expert, design engineer, and the layout designer have specific tasks to perform for verifying whether or not a design is EMC compliant.

EMControl can be used during:

- placement (pre-route) stage
- routing stage
- post-route stage

Tasks to be Performed

EMC expert

- ☐ Use established EMC standards and past experience to determine the EMC requirements for the design.

EMControl User Guide

Introduction to EMControl

- ☐ Specify or develop project-specific, customized EMC rules and modify system-provided default EMC rules, if necessary.
- ☐ Select EMC rule sets for the different stages of design verification.
- ☐ Review the results of EMC rule-checking runs and specify new EMC rule sets for further testing.

Design engineer

- ☐ Set up the EMControl product, if this has not already been done.
- ☐ Customize EMC rule variables (parameters) where required. The EMC expert can provide key inputs here.
- ☐ Assign relevant EMControl and PCB Editor properties to components and set the initial property values.
- ☐ Set up the EMC rule-checking environment.
- ☐ Specify the set of EMC rules to use during each rule-checking run.
- ☐ Specify the portion of the design to check for each rule set.
- ☐ Audit the EMControl tool setup and design preparation.
- ☐ Confer with the EMC expert to resolve any setup issues.

Layout designer

- ☐ Run EMC rule checking.
- ☐ View EMC rule violation messages and correct problems in the design.
- ☐ Confer with the design engineer and the EMC expert as required while resolving violations.
- ☐ Run rule checking again, if required.

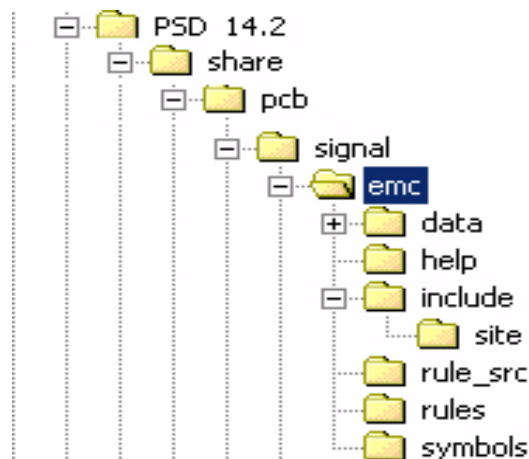
Installing and Accessing EMControl

Installing EMControl

EMControl is not offered as a standalone product. Rather, it is a tool that is part of Allegro PCB SI and Allegro Package SI. Customers need only install one of these products in order to access EMControl.

EMControl System Directory Structure

Figure 2-1 EMControl System Directory Structure



The EMControl installation directory is located on your system at `<your_install_dir>/share/pcb/signal/emc`, where `your_install_dir` specifies the path of the directory where the design software is installed.

The EMControl installation directory contains the following subdirectories:

- `rule_src`: This directory contains seven source rule files (source rule files) and the corresponding rule-verification files:
 - `emc_placement.arl` and `emc_placement.arl.verify`

EMControl User Guide

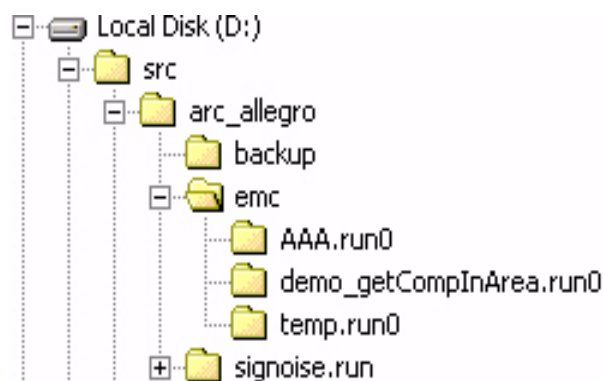
Installing and Accessing EMControl

- ❑ `emc_bypass.arl` and `emc_bypass_verify.arl.verify`
- ❑ `emc_pwr_gnd_dist.arl` and `emc_pwr_gnd_dist.arl.verify`
- ❑ `emc_dc_route.arl` and `emc_dc_route.arl.verify`
- ❑ `emc_sig_route.arl` and `emc_sig_route.arl.verify`
- ❑ `emc_sig_qual.arl` and `emc_sig_qual.arl.verify`
- ❑ `emc_default_rules.arl` and `emc_default_rules.arl.verify`
- **rules:** This directory contains the compiled Cadence mapping file (`emc_allegro.env`). This `rules` directory also contains the seven compiled rule files, and the corresponding compiled rule-verification files:
 - ❑ `emc_placement.rle` and `emc_placement.rle.verify`
 - ❑ `emc_bypass.rle` and `emc_bypass.rle.verify`
 - ❑ `emc_pwr_gnd_dist.rle` and `emc_pwr_gnd_dis.rle.verify`
 - ❑ `emc_dc_route.rle` and `emc_dc_route.rle.verify`
 - ❑ `emc_sig_route.rle` and `emc_sig_route.rle.verify`
 - ❑ `emc_sig_qual.rle` and `emc_sig_qual.rle.verify`
 - ❑ `emc_default_rules.rle` and `emc_default_rules.rle.verify`
- **include:** This directory contains the following files and directories:
 - ❑ `emc_param.par`: This file defines the Cadence supplied user-definable parameters and is used to assign them default values.
 - ❑ **site directory:** This directory contains the site parameter file, `emc_param.par`. You can use the `emc_param.par` file to customize the values of different parameters for your site. However, you must have write permissions to modify this file.
- **help:** This directory contains the help (`.hlp`) files for the rules. You can display the help file contents in the EMC Rule Selection form rule browser. For more information about displaying help file contents, refer to [“Chapter 5, “Performing EMControl Rule Checking.”](#)
- **data:** This directory contains miscellaneous files.
- **symbols:** This directory contains miscellaneous files.

EMControl Run Directory Structure

By default, the EMControl run directory is located on your system at `<your_working_dir>/emc`, where the *your_working_dir* is your current working directory. In most cases, it is the directory holding your design board file.

Figure 2-2 EMControl Run Directory Structure



The EMControl working directory contains the following files and subdirectories.

- `design.conf`: This file contains the system variables for the current instance of EMControl, if it exists. The variables defined here will override the values defined in `system.conf`. Each time you exit EMControl, all of your system configuration information will be saved to this file.
- `emc_custom.par`: This is a local rule parameter file created in the working directory. Initially, it is empty. When you modify the values of any parameters, the changes are reflected in this file. The parameter values defined here will override the parameter values defined in the site parameter file and the system default parameter file.
- `<design>.run[0-999]`: EMControl uses this directory to read and store the temporary files and the result files that it generates for rule checking of a specific design. The `<design>` variable is the design board name. These files include:
 - `emcrc.log`: This file contains a summary of the EMControl session, including the name of the design, the rule names selected, the start and stop time, and the total number of violations detected.
 - `emcrc.setup`: This file contains the rule file information settings used during the latest rule checking.
 - `emcrc_execute.msg`: This file contains the short and advisor messages that describe the violations encountered in an EMC rule-checking session.

EMControl User Guide

Installing and Accessing EMControl

- ❑ `emcrc.mkr`: This file contains the information required by the markers utility for each rule violation.
- ❑ `emcrc.verify`: This file contains the rule file information settings used during the latest run to verify (audit) the property setup for the selected rules.
- ❑ `emcrc_verify.msg`: This file contains the short and advisor messages that describe the violations encountered during property verification by the Audit function.
- ❑ `emcrc.res`: This file contains the number of violations recorded against each rule in the latest run.
- ❑ `emcrc_propedit.log`: This file contains the messages displayed in the most recent run of automatically property tagging.

Default EMControl Rule Set

A free default rule set, provided by Cadence, is installed with EMControl. You can find these default EMControl rules in the following directories:

- Rule Source Code:
`<your_install_dir>/share/pcb/signal/emc/rule_src`
- Compiled Rule:
`<your_install_dir>/share/pcb/signal/emc/rules`
- Rule Help File:
`<your_install_dir>/share/pcb/signal/emc/help`
- Parameter File:
`<your_install_dir>/share/pcb/signal/emc/include`

Advanced EMControl Rule Set

An advanced EMControl rule set is also supplied as a Methodology Service. You will need to contact Cadence to get the advanced package and license for this service if they are not already available. After installation, you can find the advanced EMControl rule set in the following directories:

- Rule Source Code:

Note: Cadence does not provide source code for the advanced rules.

- Compiled Rule:

`<your_install_dir>/share/pcb/signal/emc/rules`

- Rule Help File:

`<your_install_dir>/share/pcb/signal/emc/help`

- Parameter File:

`<your_install_dir>/share/pcb/signal/emc/include`

License for EMControl

EMControl Product License

EMControl requires the PX3100 (Allegro PCB SI) or the PX4000 (Allegro Package SI) license. If neither license exists, EMControl is not accessible.

Default Rule Set License

The default EMControl rules are free to run. No additional license is needed to run them.

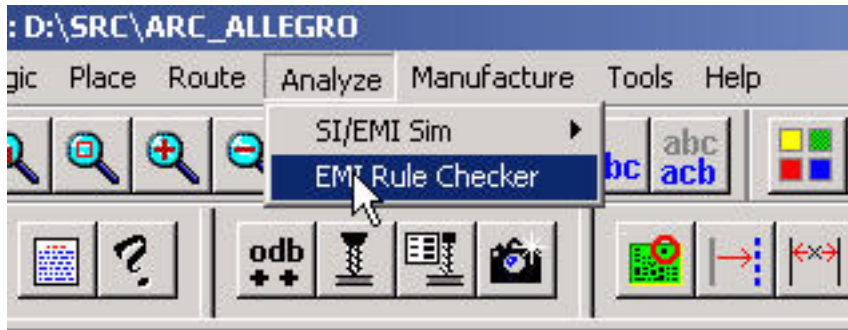
Advanced Rule Set License

The advanced EMControl Rules are distributed as a Methodology Service, and a specific license is needed to run these rules. Please contact Cadence for license support.

Accessing EMControl

You can access EMControl using either Allegro PCB SI and Allegro Package SI.

Figure 2-3 Access EMControl



To access EMControl from the menu,

- Select *Analyze – EMI Rule Checker*.

- or -

To access EMControl from the command line,

- Type `emcontrol`.

Setting Up the EMControl System Environment

Overview

When you run EMControl in the current working directory for the first time, you will be prompted to fill in your system configuration information first. You can also access this dialog from EMControl Main Window as shown in the following figure.

Figure 3-1 Access EMControl System Configuration Dialog

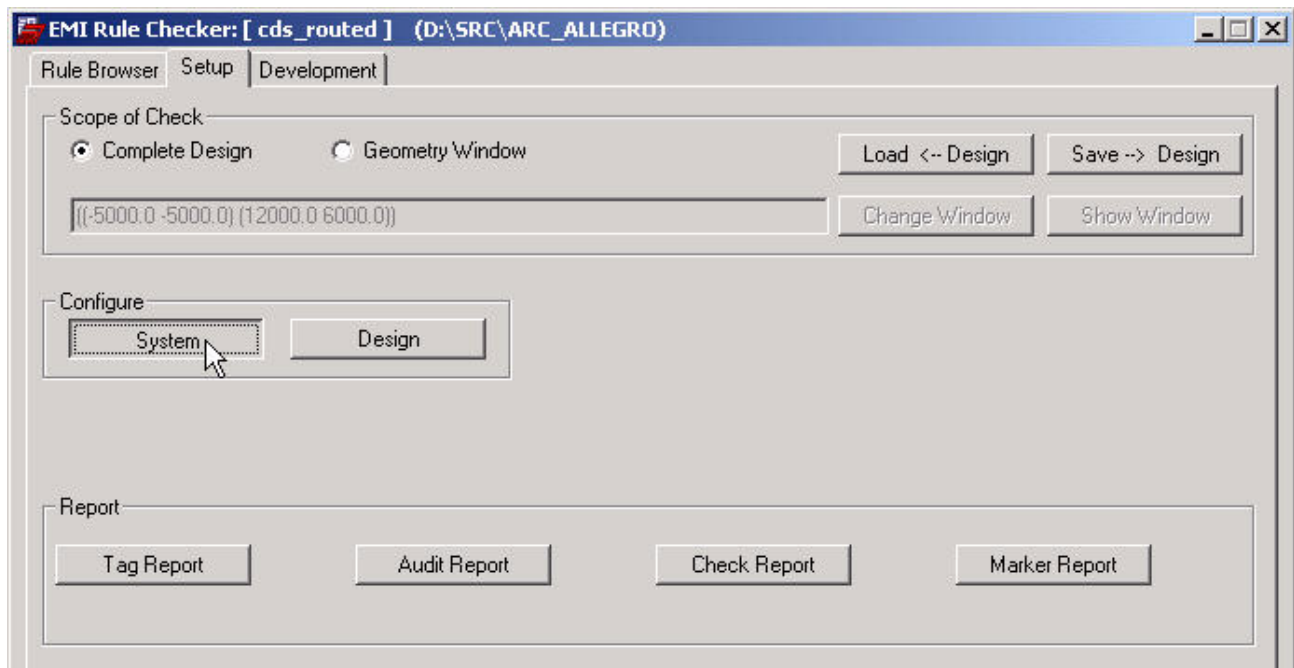
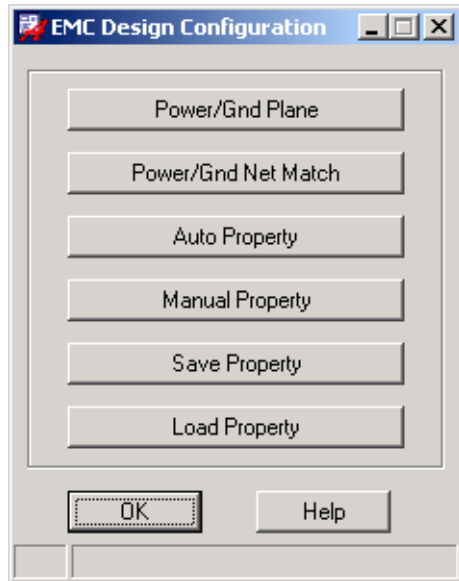


Figure 3-2 illustrates the dialog box used to set up the EMControl system environment. Click on a button to go to the corresponding task.

Figure 3-2 EMControl System Configuration Dialog



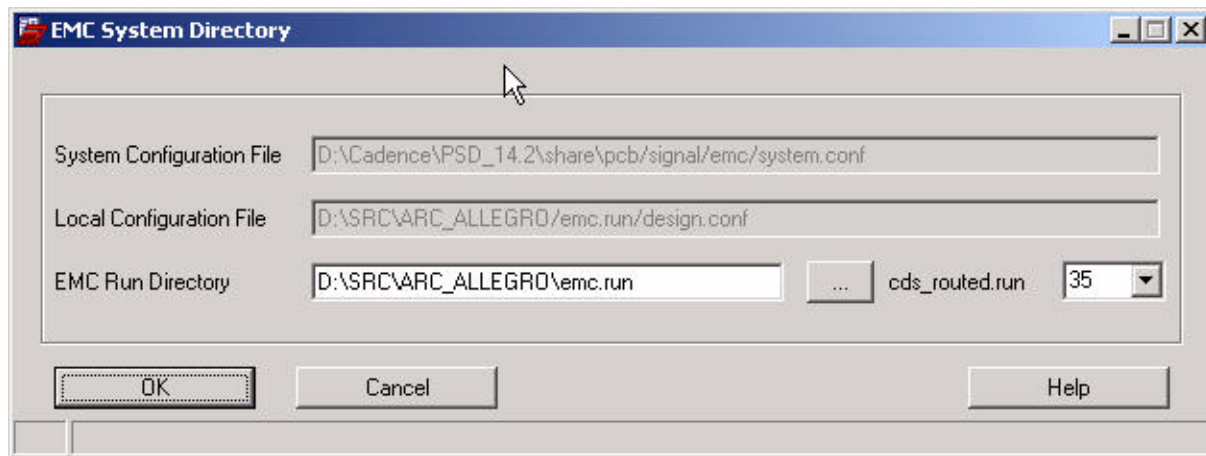
The following sections describe EMControl tasks related to system configuration and provide general information needed to use EMControl. You can modify everything or only the particular items you are interested in.

- [Setting Up the System Directory](#)
- [Setting Up Mapping Files](#)
- [Setting Up Rule Files](#)
- [Setting Up Rule Parameter Files](#)
- [Setting Up SKILL Files](#)
- [Setting Up System Preference](#)

Setting Up the System Directory

Before you can perform EMControl checking on a design, you may want to know where the system and design configuration files are stored and modify the default EMControl run directory. Figure 3-3 illustrates the dialog used to set up the EMControl system directory.

Figure 3-3 EMC System Directory Dialog Box



System Configuration File

This field is for information use only. It indicates where the system configuration file is. You can modify this ASCII file to set system-level configuration information.

Local Configuration File

This field is also for information use only. It indicates where the local configuration file is. You can modify the related entry in the system configuration file to re-direct the local configuration file to another place, if necessary.

EMC Run Directory

This field indicates where the EMControl working directory is. You can change the working directory and, if the directory does not exist, EMControl will create it by default. To change the location, specify the full path in the field or use the browse button to select the directory at another location. The temporary file and the result files generated by EMControl will be stored here.

Note: When EMControl is used for the first time in the current working directory, the default run directory is created and assigned the name `emc.run`.

cds_routed.run

This field indicates where the EMControl design working directory is. This field is a drop-down list where you can select one of the existing design working directories or input a new number [0–999] to create a new directory. The temporary files and the result files related to your specific PCB design will be stored here.

Note: The EMControl Design-Specific Working Directory name is stored in a design-level property called EMC_RUN_DIR.

Note: When EMControl is used to check a design for the first time, the default path is assigned the name *<design>.run0*. Notice that the name has suffix 0. Subsequently, whenever you run EMControl, the suffix number in the directory is increased by 1.

To save your modifications and close the dialog,

- Click *OK*

- or -

To discard your modifications and close the dialog,

- Click *Cancel*

The EMControl Mapping File

EMControl uses a default mapping file to define the various aspects of the rule-checking environment. The *.env* file name extension is required. By default, EMControl searches for a mapping file in the software installation hierarchy.

Mapping File Functions

The default mapping file, *emc_allegro.env*:

- Identifies the objects in the layout physical environment that are used by the EMC rules.
- Identifies the EMControl predicates and the objects that they use.

EMC rules are implemented using predicates to perform their functions. For reference information on the EMControl predicates, see [Appendix C, “EMControl Predicates.”](#)
- Maps EMControl predicates to the SKILL/C functions that perform their tasks.

- Defines message severity levels for EMControl rule violations.

Message Severity Levels

The severity levels assigned to rule violations are defined in the mapping file. User can use any of the following severity levels to display a violation message in the markers dialog box:

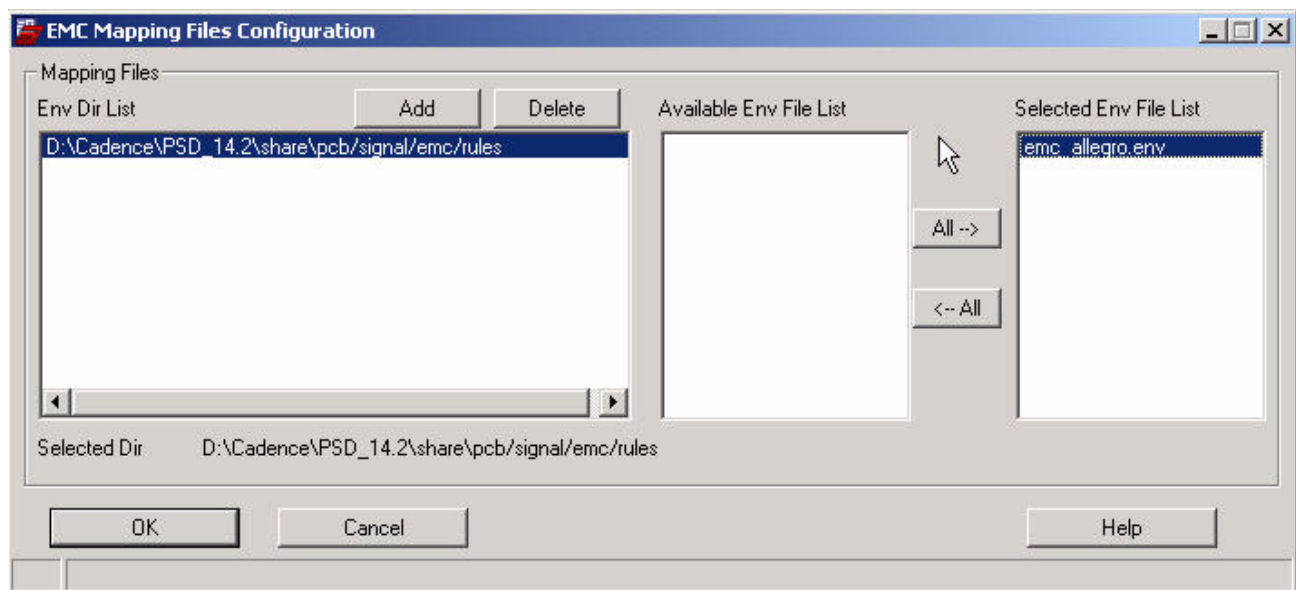
- FATAL (terminates rule execution)
- ERROR
- WARNING
- OVERSIGHT (a probable warning)
- INFO

Note: Severity levels are listed in the descending order of importance.

Setting Up Mapping Files

In addition to the default mapping file, you can also write your own predicates and create new .env files for specifying mappings to SKILL/C functions. You can specify multiple mapping files using the EMC Mapping Files Configuration dialog box.

Figure 3-4 EMC Mapping Files Configuration Dialog Box



EMControl User Guide

Setting Up the EMControl System Environment

Using the EMC Mapping Files Configuration dialog box, you can:

- *Add* a new directory to indicate where your new `env` file is located.
- *Delete* the existing `env` directory.
- Add an `env` file by double-clicking the file in the *Available Env File List*.
The file you selected will appear in the *Selected Env File List*.
- Add all `env` files by clicking the *All-->* button.
All files in *Available Env File List* will appear in the *Selected Env File List*.
- Delete an `env` file by double-clicking the file in the *Selected Env File List*.
The file you selected will appear in the *Available Env File List*.
- Delete all `env` files by clicking the *<-- All* button.
All files in *Selected Env File List* will appear in the *Available Env File List*.

To save the mapping file configuration information and close the dialog box,

- Click *OK*

- or -

To discard your modifications and close the dialog box,

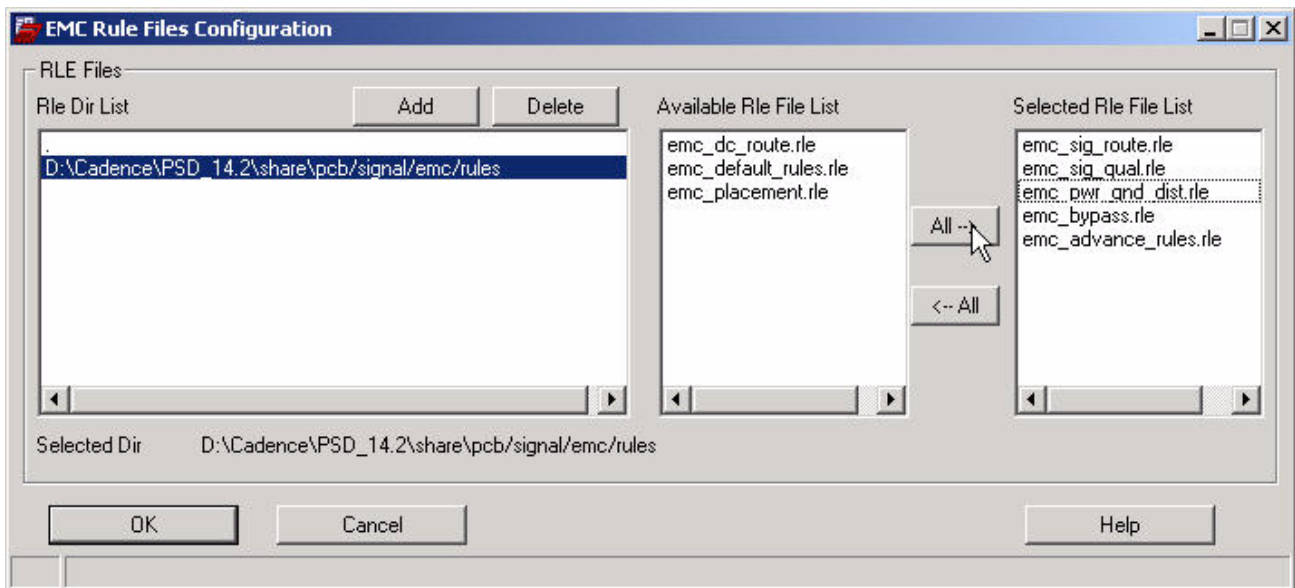
- Click *Cancel*

Note: The system default `env` directory and the `emc_allegro.env` file can not be deleted.

Setting Up Rule Files

EMControl Rule Files have to be setup before checking the design. After you specify the rule files and their paths, the rules contained in the rule files will display in the EMC Rule Files Configuration dialog box.

Figure 3-5 EMC Rule Files Configuration Dialog Box



Using this dialog box, you can:

- *Add* a new directory to indicate where your new rule file is located.
- *Delete* the existing rule file directory.
- Add a rule file by double-clicking the file in *Available Rle File List*.
The file selected will appear in the *Selected Rle File List*.
- Add all rule files by clicking the *All-->* button.
All files in *Available Rle File List* will appear in the *Selected Rle File List*.
- Delete a rule file by double-clicking the file in *Selected Rle File List*.
The file selected will appear in the *Available Rle File List*.
- Delete all rule files by clicking the *<--All* button.
All files in *Selected Rle File List* will appear in the *Available Rle File List*.

To save the rule file configuration information and close the dialog box,

- Click *OK*

- or -

To discard your modifications and close the dialog box,

- Click *Cancel*

Setting Up Rule Parameter Files

There are three types of Parameter Files used in EMControl:

■ Default Parameter File

Cadence provides a system default parameter file, `emc_param.par`, which contains the global variables and rule variables for the default rule set. You can create new default parameter files for your new rules or add the variables in your new rules to this file. In either case, you should always include the variables defined in new rules in default parameter files.

■ Site Parameter File

You can set the site-default value in this file for variables defined in default parameter files. The site parameter file, `emc_param.par`, is always located in the following directory: `<your_install_dir>/share/pcb/signal/emc/include/site`. EMControl experts can modify this file to customize variables for site use.

Note: You cannot re-direct the location of the site parameter file.

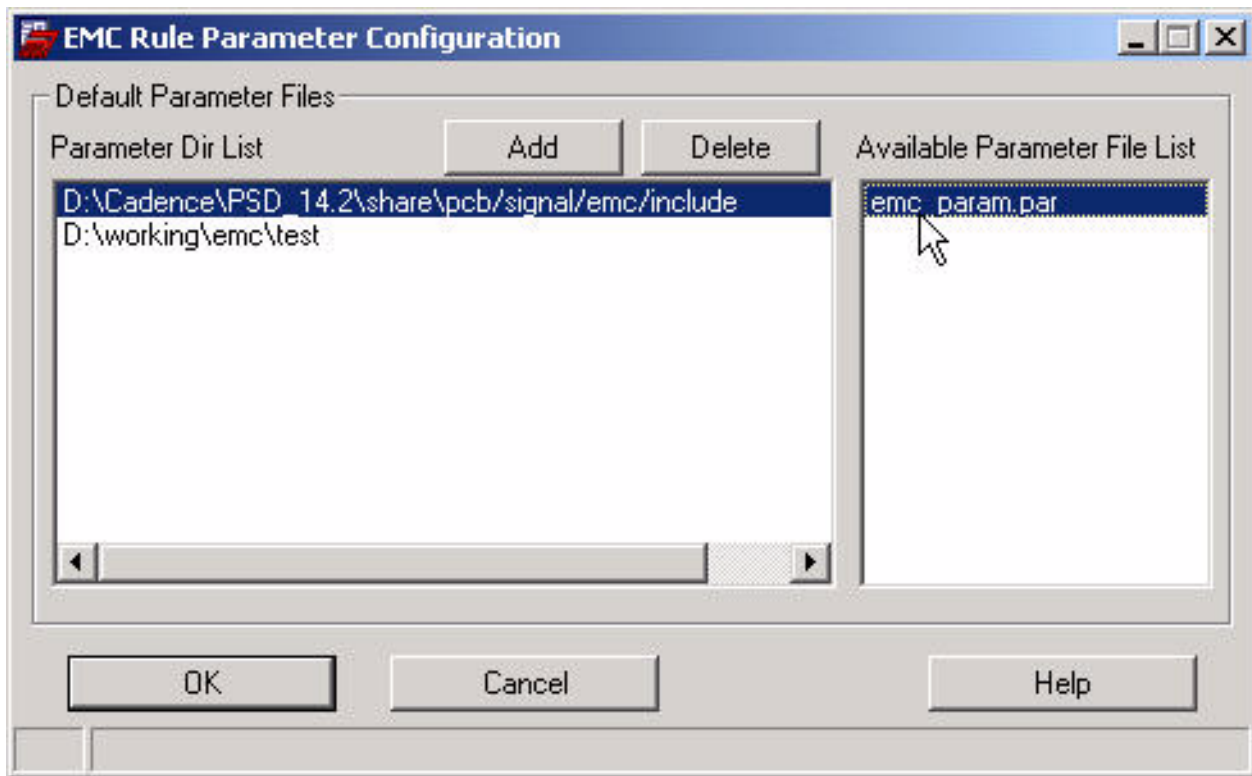
■ Custom Parameter File

You can also set the variable values for design checking. All local customizations for design checks are stored in the `emc_custom.par` file which is always located in the EMControl Working Directory.

Note: You cannot re-direct the location of the custom parameter file.

EMControl default parameter files need to be setup before you can check the design. The following figure illustrates the dialog box used to configure the parameter files.

Figure 3-6 EMC Rule Parameter Configuration Dialog Box



Using this dialog box, you can:

- *Add* a new path where the default parameter file is located.
- *Delete* an existing default parameter file path.
- View the *Available Parameter File List* to see the default parameter files in the selected path.

To save the rule parameter configuration information and close the dialog box,

- Click *OK*

To discard your modifications and close the dialog box,

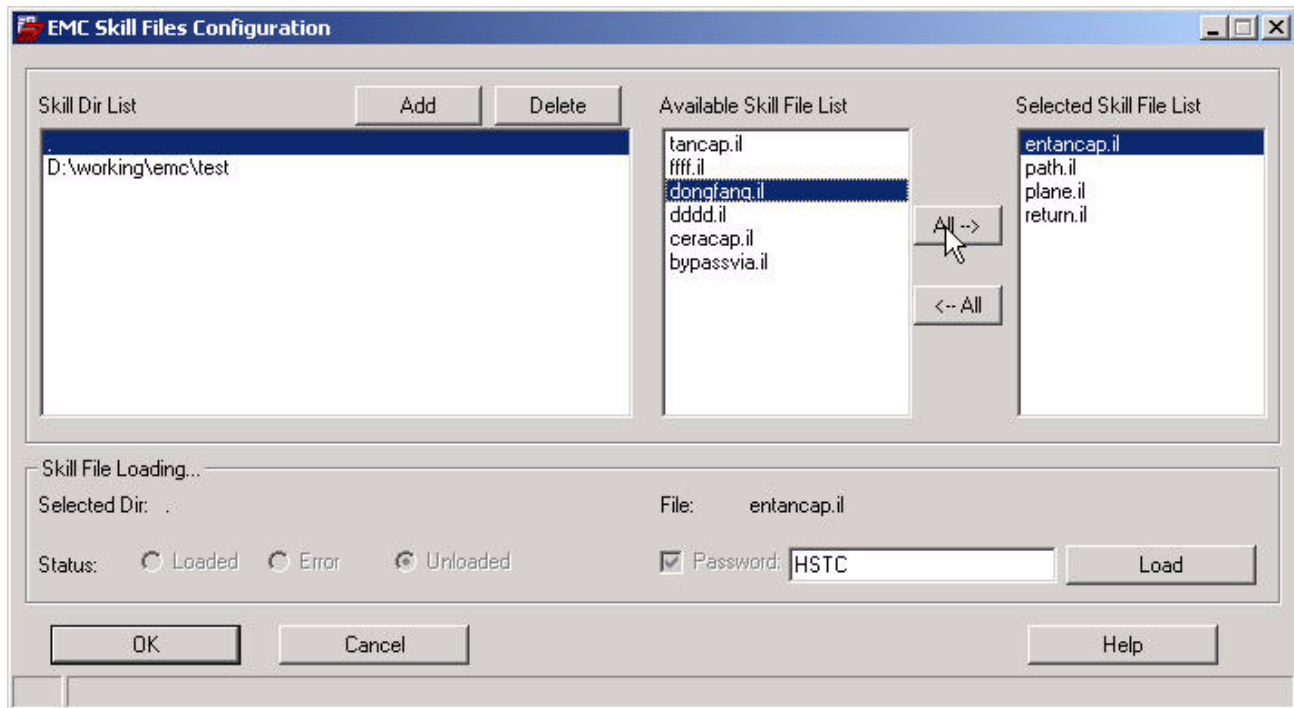
- Click *Cancel*

Setting Up SKILL Files

When you create your own predicates or `env` files, the SKILL/C functions need to be loaded before rule checking. EMControl enables you to specify what SKILL files are loaded.

The following figure illustrates the dialog box used to set up SKILL Files:

Figure 3-7 EMC SKILL Files Configuration



In this dialog box, you can:

- *Add* a new path where the SKILL files are located.
- *Delete* an existing SKILL file path .
- Add a SKILL file by double-clicking the file in *Available Skill File List*.
The file selected will appear in the *Selected Skill File List*.
- Add all SKILL files by clicking the *All-->* button.
All files in *Available Skill File List* will appear in the *Selected Skill File List*.
- Delete a SKILL file by double-clicking the file in *Selected Skill File List*.
The file selected will appear in the *Available Skill File List*.

EMControl User Guide

Setting Up the EMControl System Environment

- Delete all SKILL files by clicking the <--*All* button.
All files in *Selected Skill File List* will appear in the *Available Skill File List*.
- View the status of selected SKILL files in the *Status* field. There are three types:
 - ☐ *Loaded*
 - ☐ *Error*
 - ☐ *Unloaded*
- Change or input a password using the *Password* field, if necessary.
- *Load* the selected SKILL file.

The *Status* indication will be updated according to the results.

To save the SKILL file configuration information and close the dialog box,

- Click *OK*

- or -

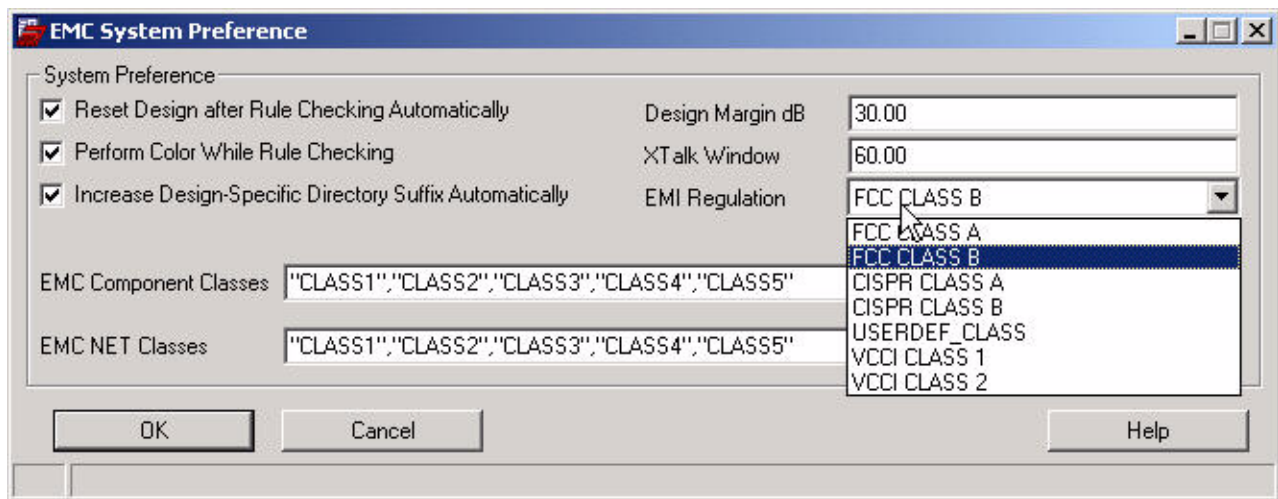
To discard your modifications and close the dialog box,

- Click *Cancel*

Setting Up System Preference

EMControl provides an easy method to set global variables and preferences through the EMC System Preference dialog box.

Figure 3-8 EMC System System Preference



The current options in this dialog box are:

Reset Design after Rule Checking Automatically

Some EMC rules will color and highlight the objects in the design while checking. When this option is set, EMC will reset all of them when the checking is completed.

Perform Color While Rule Checking

Some EMC rules need to color nets and other components in a design to indicate violations. When this option is not set, EMC will not color the violations while checking.

Increase Design Specific Directory Suffix Automatically

When this option is set, the suffix in the EMControl design-specific directory name is increased automatically every time EMControl is launched. It is used to prevent previous rule checking results from being overwritten as EMControl performs new rule checks.

Design Margin dB

Defines the maximum acceptable level of emissions in dB (decibels).

EMControl User Guide

Setting Up the EMControl System Environment

XTalk Window Uses the maximum distance to search for neighbor nets when calculating crosstalk.

EMI Regulation Defines the EMI regulation used to check the design.

EMC Component Classes

There are up to five classes for components in a design. Using this field, you can specify the corresponding variable value for each component class.

EMC Net Classes

There are up to five classes for nets in a design. Using this field, you can specify the corresponding variable value for each net class.

To save the system preference information and close the dialog box,

➤ Click *OK*

- or -

To discard your modifications and close the dialog box,

➤ Click *Cancel*

EMControl User Guide

Setting Up the EMControl System Environment

Setting Up the EMControl Design Environment

Overview

To determine which design objects to check, the EMC rules examine certain property values associated with design objects. For example, the EMC bypass_critical_IC rule searches for ICs with the `EMC_CRITICAL_IC` property assigned. For each critical IC it finds, EMControl searches for a minimum number of bypass capacitors attached to the component. EMControl identifies the bypass capacitors by verifying whether or not the `EMC_COMP_TYPE` property is assigned the `BYPASS_CAP` value. EMControl also evaluates each bypass capacitor by its assigned `TOL` and `VALUE` property values and reports any problems.

EMC regions are areas within a design where components and nets with similar EMC characteristics are clustered together. EMC regions are defined using the Allegro PCB SI room mechanism.

Schematic implementation is the recommended time for the design engineer to assign EMC-specific classes and other required properties to components and nets. Property requirements are determined by the specific rules included in an EMC rule-checking run.

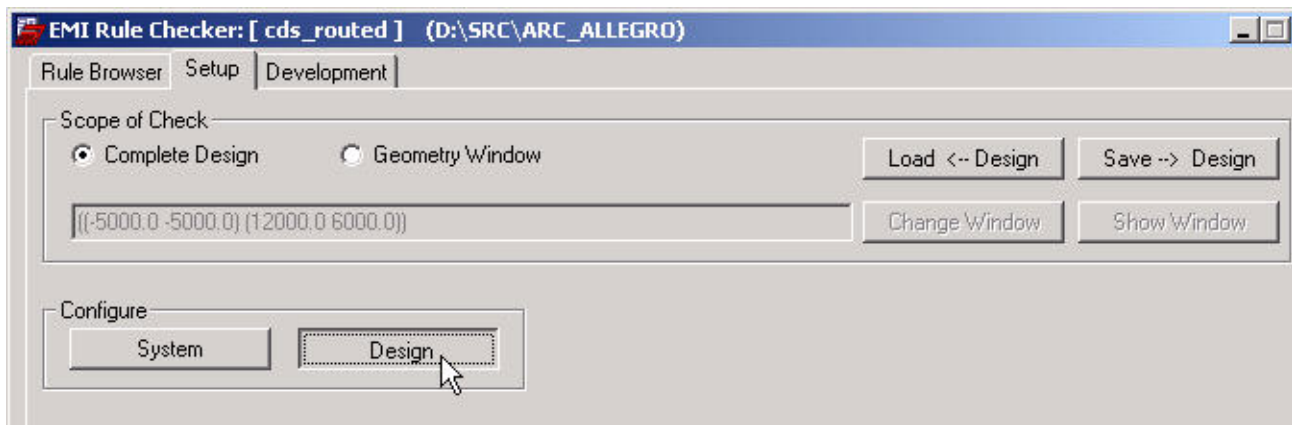
EMControl supplies up to five critical frequency classifications to describe the EMC characteristics of the ICs and nets in a design. User can either accept these default classes or redefine the boundary values to create up to five critical frequency classes that better describe the EMC characteristics of the components and nets comprising your design.

When you run EMControl to perform rule checking on a specific design for the first time, you will be prompted to fill in the design configuration first. You can also access this dialog from EMControl Main Window as shown in the figure [4-1](#).

EMControl User Guide

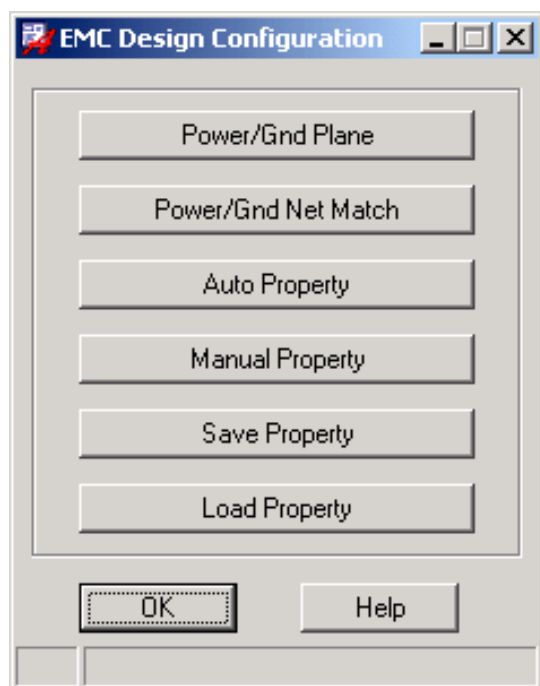
Setting Up the EMControl Design Environment

Figure 4-1 Accessing EMC's Design Configuration Dialog Box



The following figure illustrates the dialog box used to setup the EMControl design environment. Click on a button to go to the corresponding task.

Figure 4-2 EMC Design Configuration Dialog Box



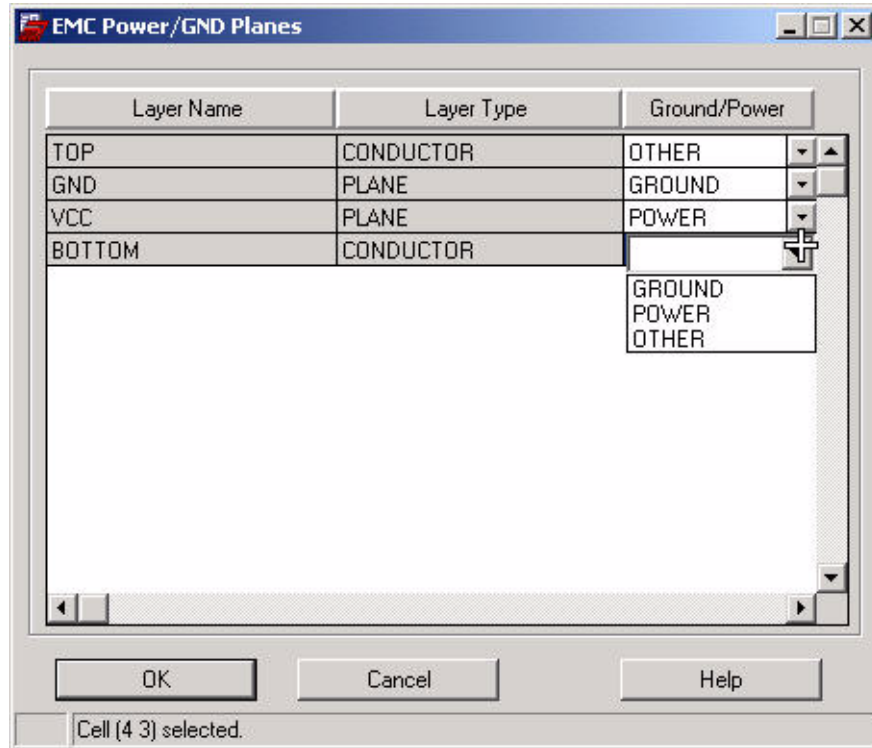
The following sections describe the EMControl tasks related to design configuration and provide general information you need to use EMControl. You can modify everything or only the particular information you are interested in.

- [Setting Up Power/Ground Plane](#)
- [Setting Up Power/Ground Net Matching List](#)
- [Automatically Attaching Properties to Design Objects](#)
- [Manually Attaching Properties to Selected Design Objects](#)
- [Saving the EMC Properties Assigned to a Design](#)
- [Loading EMC Properties Previously Saved](#)

Setting Up Power/Ground Plane

EMControl enables you to specify the power planes for the design. The power plane setting is very important to PI analysis. You can specify whether the plane is *POWER*, *GROUND*, or *OTHER*.

Figure 4-3 EMC Power/Ground Planes Dialog Box



When the EMC Power/Ground Planes dialog box is displayed, EMControl will list all layers in design together with their layer types. EMControl also provides the estimation on whether the layer is power plane, ground plane, or other for user reference. You can modify it if it is not correct. The modification will be stored in the design for future rule checking.

To modify the power/ground plane types,

1. Select the *Grond/Power* value from the drop-down list in the associated layer.
2. Click *OK* to save the power/ground plane configuration and close the dialog box.

- or -

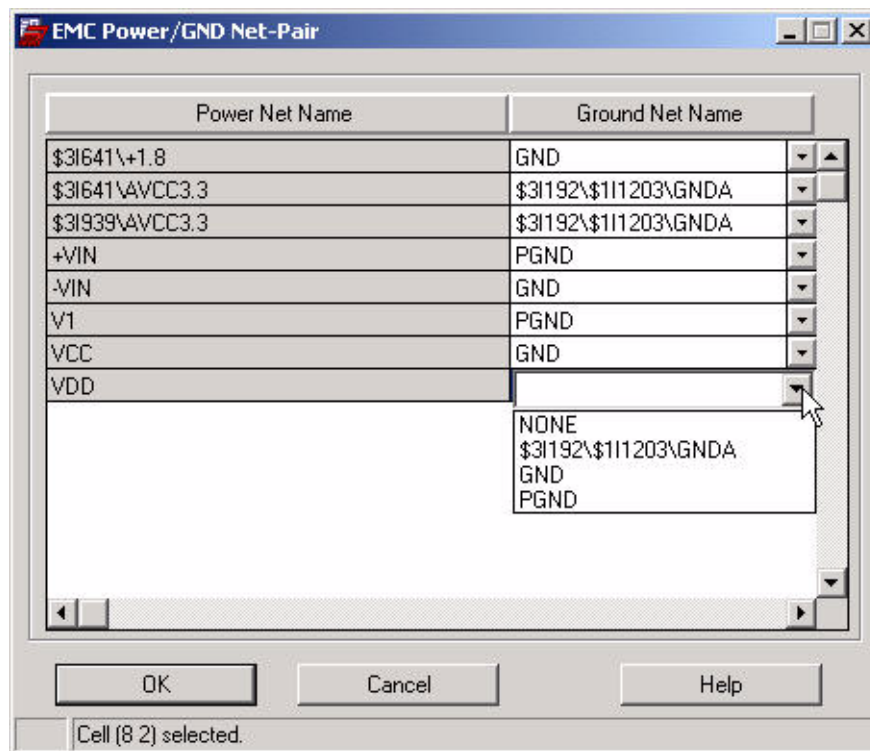
To discard your modifications and close the dialog box,

- Click *Cancel*.

Setting Up Power/Ground Net Matching List

EMControl enables you to specify the power/ground net pair for a design in. The power/ground net match is very important to PI analysis. You can specify which ground net a power net matches within the following dialog box.

Figure 4-4 EMC Power/GND Net-Pair Dialog Box



When this dialog box is displayed, EMControl will list all power nets in a design, and the corresponding ground net matched from the current design setting. You can modify it if it is not correct. All ground nets are listed in the drop-down list. All of the power/ground net match settings will be stored in the design for future rule checking.

To modify the power/ground net match,

1. Select the corresponding *Ground Net Name* value from the drop-down list for the associated *Power Net Name*.
2. Click *OK* to save the power/ground net matching configuration and close the dialog box.

- or -

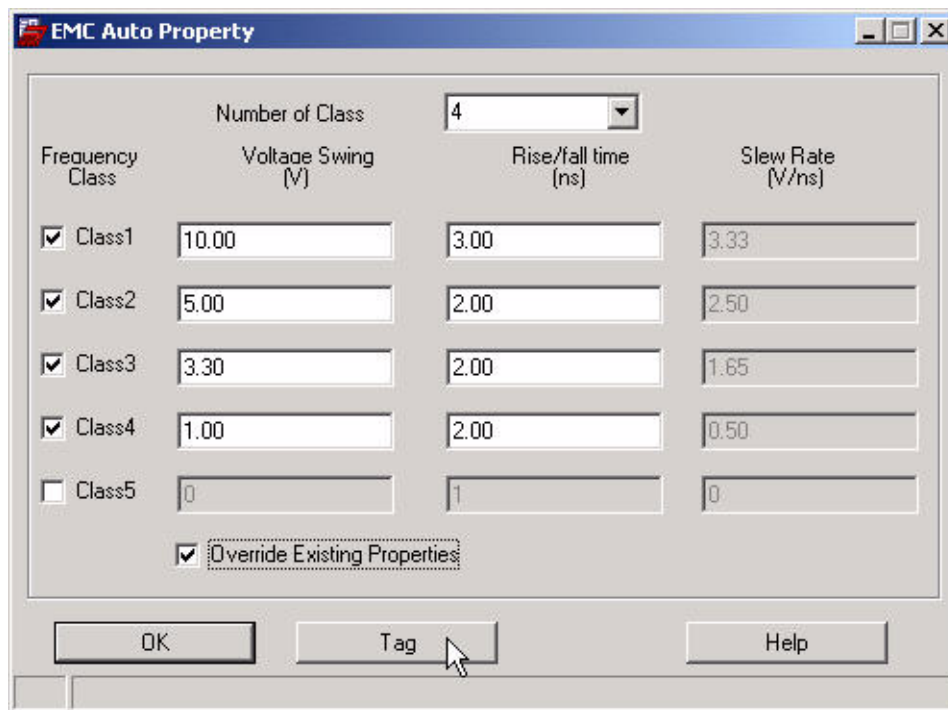
To discard your modifications and close the dialog box,

- Click *Cancel*

Automatically Attaching Properties to Design Objects

You can attach critical frequency classes (EMC_CRITICAL_IC, EMC_CRITICAL_NET, EMC_CRITICAL_REGION) to design objects automatically using the dialog box:

Figure 4-5 Automatically Attaching Properties to Design Objects



In this dialog box, you can attach the properties in the following steps:

1. Select the critical *Frequency Class* toggle, if needed.
2. Select the *Number of Class* drop-down list to specify the number of classes.

You can have up to five consecutive critical frequency classifications for components, nets, and regions. For example, when you deselect CLASS4, CLASS5 is also deselected. You are left with CLASS1, CLASS2, and CLASS3.

EMControl User Guide

Setting Up the EMControl Design Environment

3. Specify the *Voltage Swing* and *Rise/Fall Time* values. The values you enter here are saved as properties on the design. The *Slew Rate* will be calculated automatically.
4. Click the toggle button to select or deselect the *Override Existing Properties* option.

When the *Override Existing Properties* check box is selected, any existing frequency class boundary values are replaced. When the *Override Existing Properties* check box is not selected, new frequency classes are added, but the existing frequency class boundary values are not changed and only a warning message is reported.

5. Click the *Tag* button.

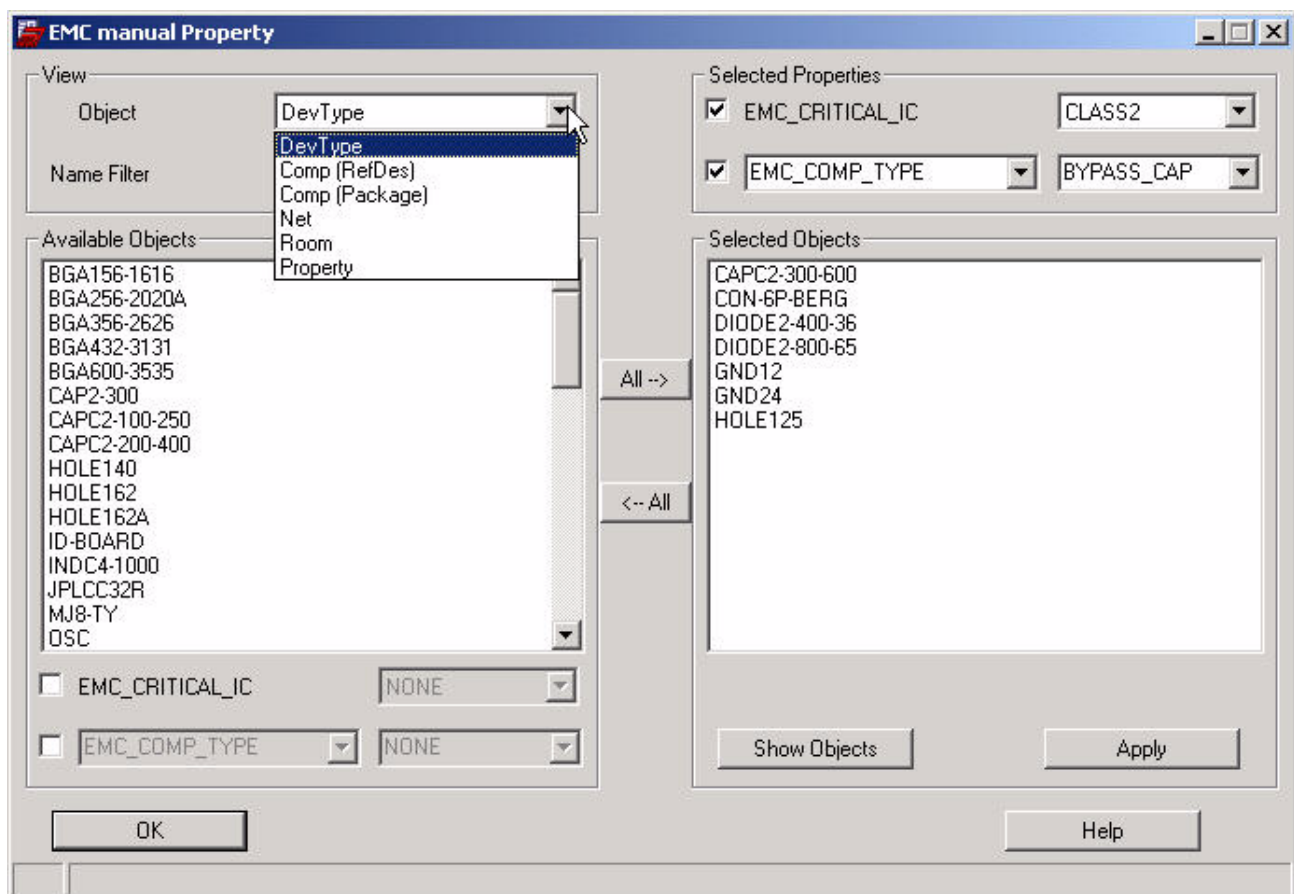
When property tagging is completed, the Auto Tag Report appears. See [View the Log File for Automatically Property Tagging](#) on page 70 for detail.

Note: When the `EMC_CRITICAL_IC`, `EMC_CRITICAL_NET`, and `EMC_CRITICAL_REGION` properties are attached, the EMC AutoPropertyTag log file (`emcrc_propedit.log`) is created.

Manually Attaching Properties to Selected Design Objects

You can attach a specific property to selected design objects manually in the dialog box illustrated below. You can select a design object for property tagging through its reference designator, device type, package type, net name, or property.

Figure 4-6 EMC Manual Property Dialog Box



You can attach the following properties to the selected design objects:

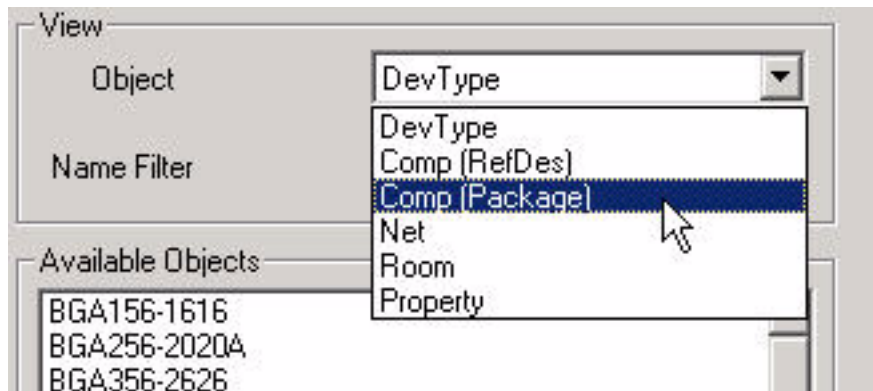
- Attaching Component Properties
- Attaching Net Properties
- Attaching Room Properties

Attaching Component Properties

To attach component properties to design component objects,

1. Specify the *Object* type for search options.

Figure 4-7 Select the Design Object Type



There are four ways to search components,

- ☐ Specify the *Object* field as *DevType*.

The components' device types are loaded in the *Available Objects* list.

- ☐ Specify the *Object* field as *Comp (RefDes)*.

The components' reference designators are loaded in the *Available Objects* list.

- ☐ Specify the *Object* field as *Comp (Package)*.

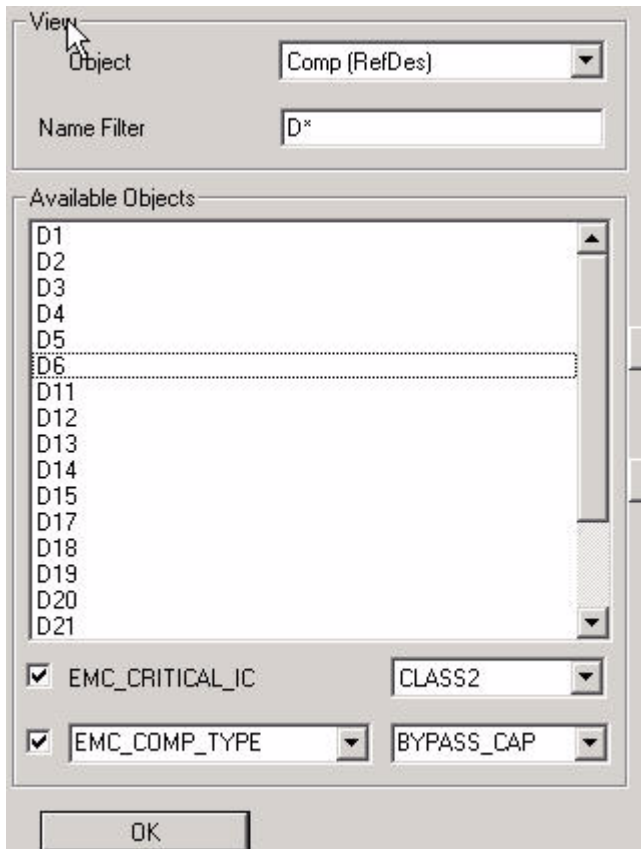
The components' package types are loaded in the *Available Objects* list.

- ☐ Specify the *Object* field as *Property*

All properties with values are loaded in the *Available Objects* list.

2. Filter the available objects by specifying *Name Filter*. The corresponding device types, package types, properties, and component RefDeses that meet the filter requirements are loaded in the *Available Objects* list.
3. Filter the available objects further by selecting the `EMC_CRITICAL_IC`, `EMC_COMP_TYPE` check boxes and by setting the class and type for other component properties respectively.

Figure 4-8 Filter the Components in Available Object List



4. Click a component in the list.

This transfers the component to the *Selected Objects* list.

Note: You can use the *All-->* and *<--All* buttons to transfer all components between the *Available Objects* list and the *Selected Objects* list.

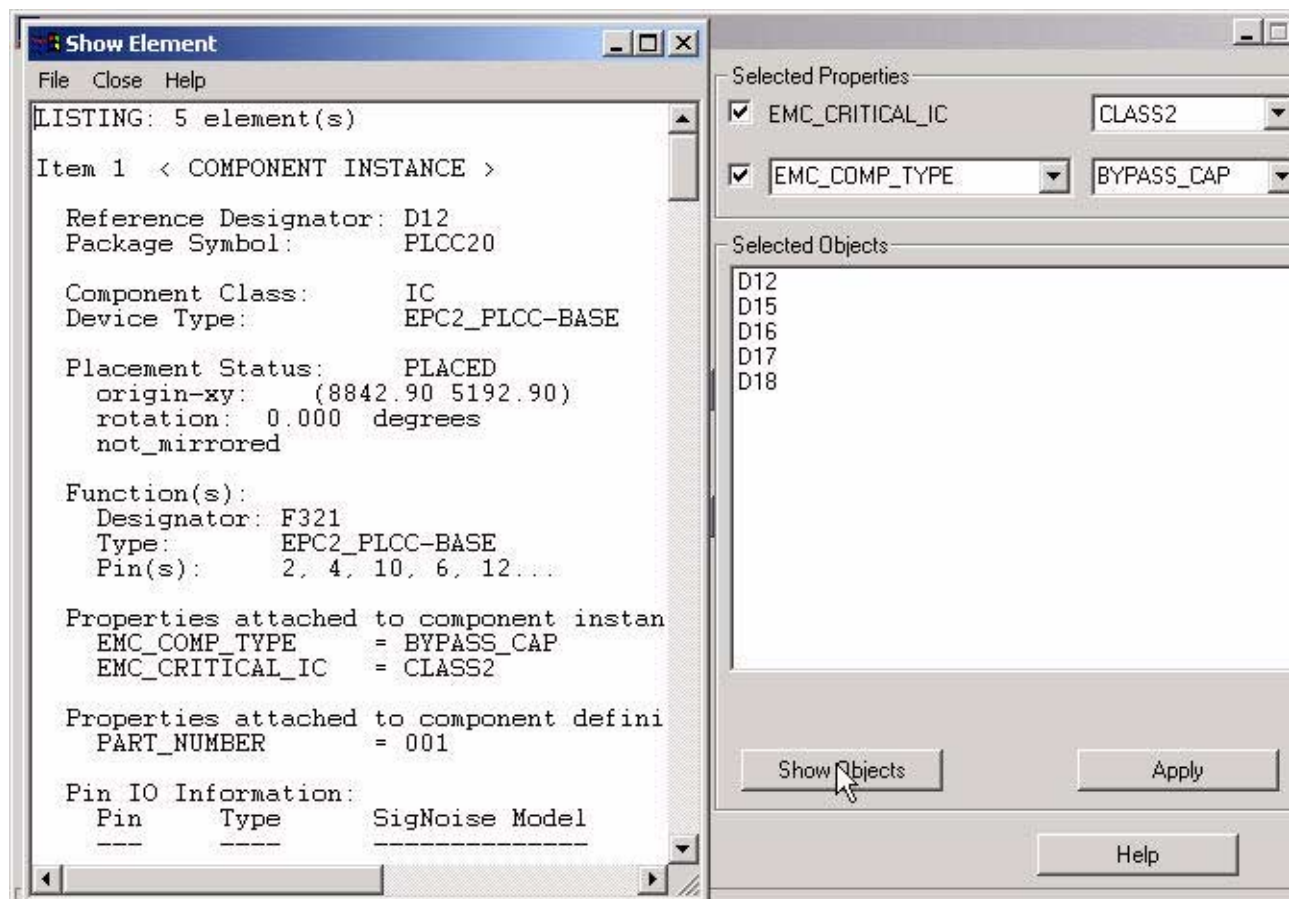
5. Specify the property value in the *Selected Properties* list.
6. Click *Apply*.

All of the components in the *Selected Objects* list are now tagged with the property value displayed in the *Selected Properties* list.

7. Click *Show Objects*.

This will show the object information in the *Selected Objects* list.

Figure 4-9 Objects Properties Attached.



Tip

If you assign a property the value `NONE`, EMControl will delete the property.

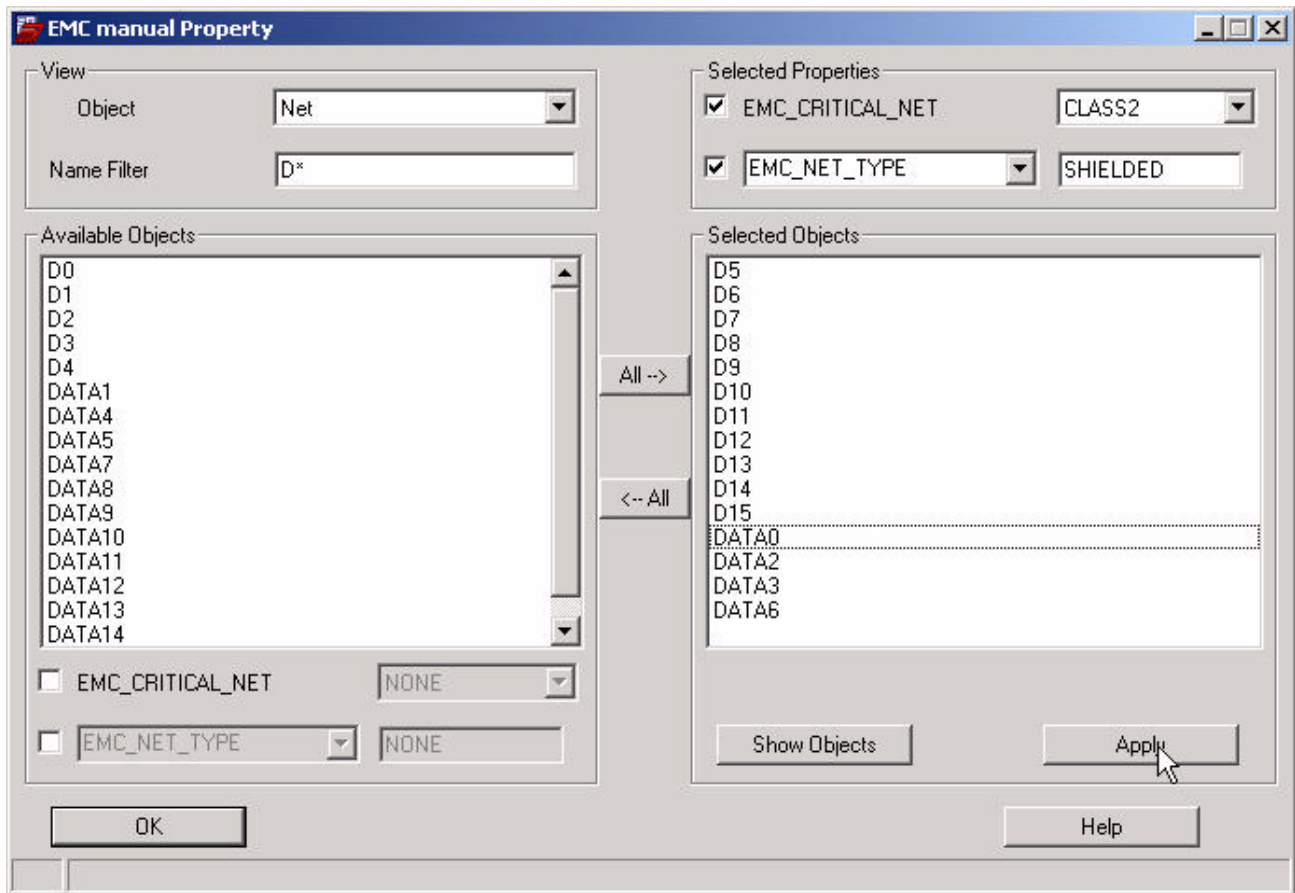
Attaching Net Properties

To attach net properties to design net objects,

1. Specify the *Object* field as *Net*.

The selected nets are loaded in the *Available Objects* list.

Figure 4-10 Attach Net Properties



2. Filter the *Available Objects* by specifying *Name Filter*.
3. Filter the *Available Objects* further by checking the `EMC_CRITICAL_NET` check box and setting the class.
4. Click the net in the list.

This transfers the net from the *Available Objects* list to the *Selected Objects* list.

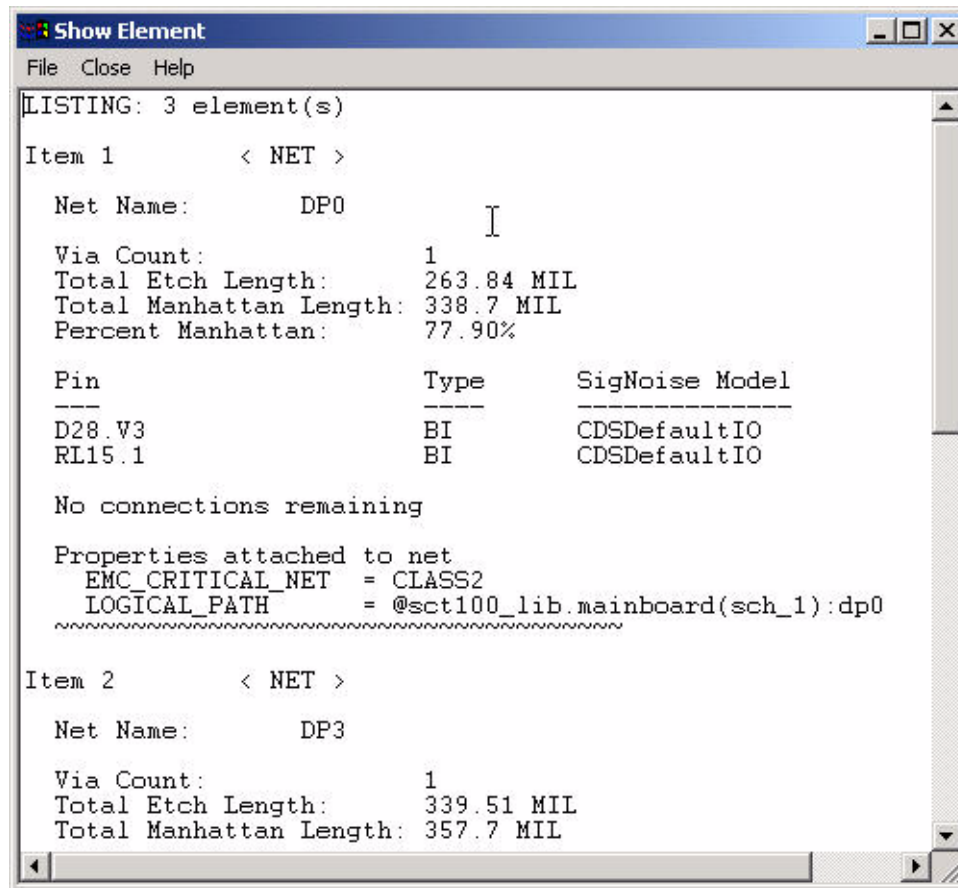
Note: You can use the *All-->* and *<--All* buttons to transfer all nets between the *Available Objects* list and the *Selected Objects* list.

5. Specify the property value in the *Selected Properties* list.
6. Click *Apply*.

All the nets in the *Selected Objects* list are tagged with the property value given in the *Selected Properties* list.

7. Click *Show Objects*. This will show the object information in the *Selected Objects* list.

Figure 4-11 Net Properties Attached



Tip

If you give a property the value `NONE`, EMControl will delete the property.

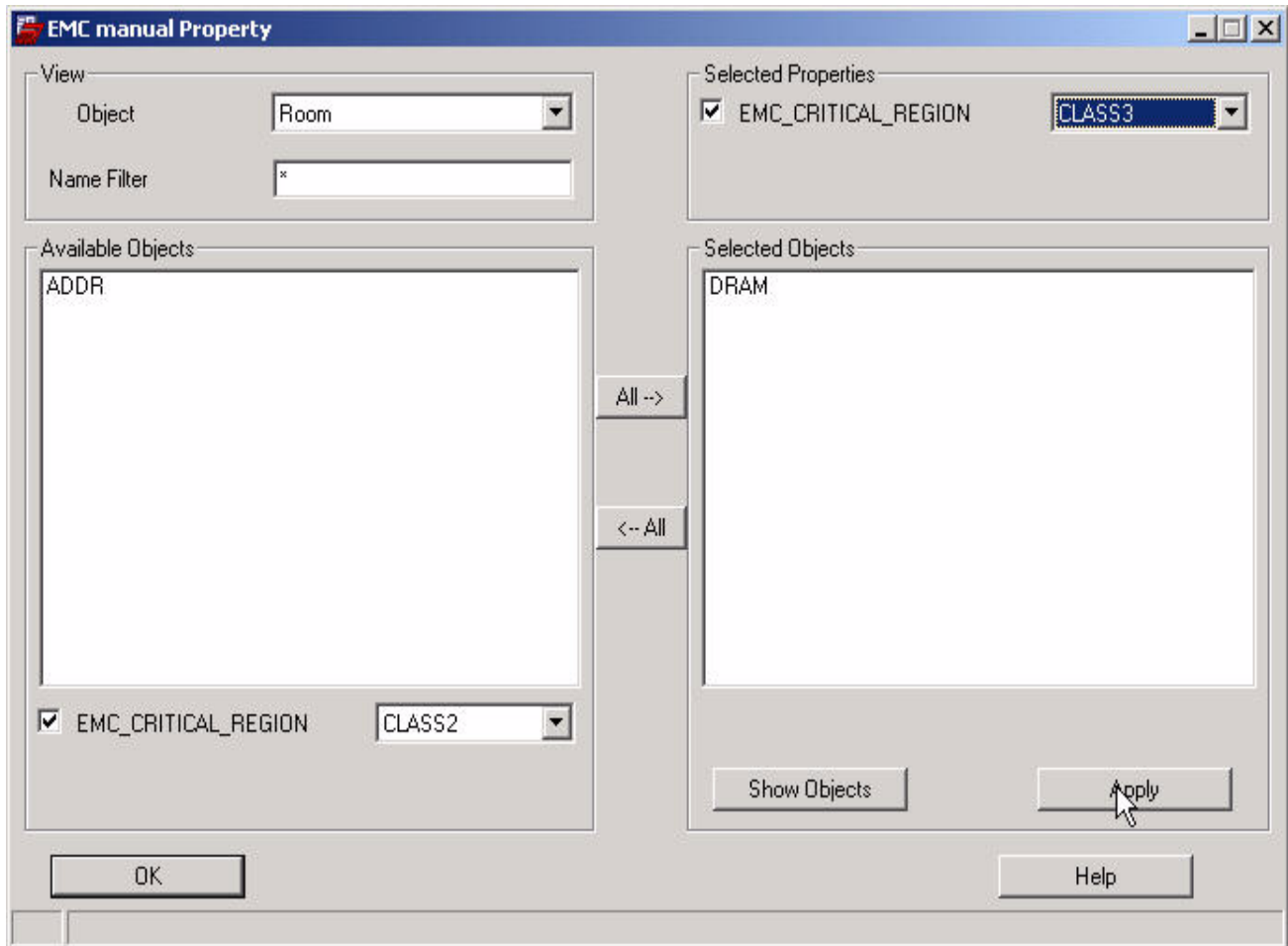
Attaching Room Properties

To attach room properties on design room objects,

1. Specify the *Object* field as *Room*.

The selected rooms are loaded in the *Available Objects* list.

Figure 4-12 Attaching Room Properties



2. Filter the *Available Objects* list by specifying *Name Filter*.
3. Filter the *Available Objects* list further by checking the `EMC_CRITICAL_REGION` check box and setting the class.
4. Click on the room in the list.

This transfers the room from the *Available Objects* list to the *Selected Objects* list.

Note: You can use the *All-->* and *<--All* buttons to transfer all rooms between the *Available Objects* list and the *Selected Objects* list.

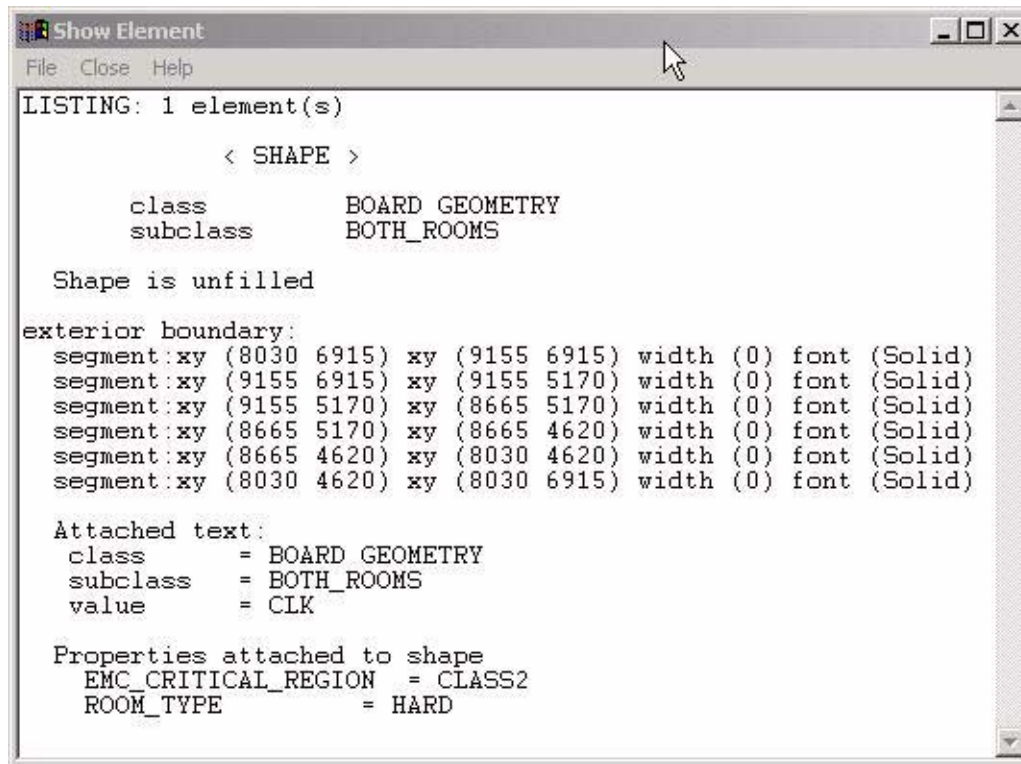
5. Specify the property value in the *Selected Properties* list.
6. Click *Apply*.

All of the rooms in the *Selected Objects* list are tagged with the property value given in the *Selected Properties* list.

7. Click *Show Objects*.

This will show the object information in the *Selected Objects* list.

Figure 4-13 Room Properties Attached



Tip

If you give a property the value `NONE`, EMControl deletes the property.

Saving the EMC Properties Assigned to a Design

You can save the EMC properties assigned to your design during rule checking. Once saved, these properties can be loaded back into the design and reused in a subsequent work session.

To save the EMC properties in the design

1. Click the *Save Property* button in the EMC Design Configuration dialog box (see [Figure 4-2](#) on page 42).

The Save EMC Properties dialog box appears as shown in [Figure 4-14](#) on page 56.

2. In the *File* text box, enter a pathname to the file (`emc_properties.txt`) where the properties are written to.

- or -

Click the Browse (...) button to navigate to the file.

3. Click *OK*.

A Message dialog appears verifying that EMC properties in the design were stored in the specified file. See [Figure 4-15](#) on page 56.

Figure 4-14 Save EMC Properties Dialog Box

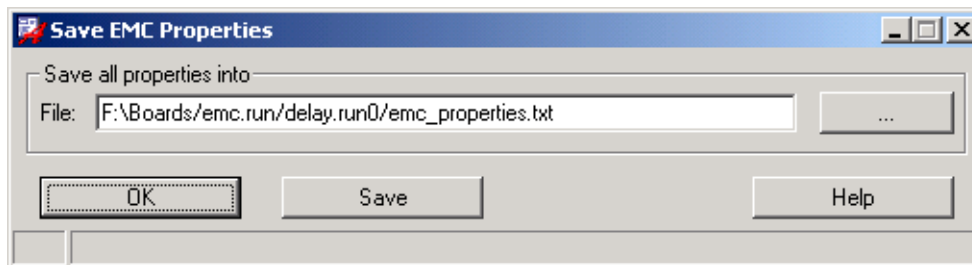
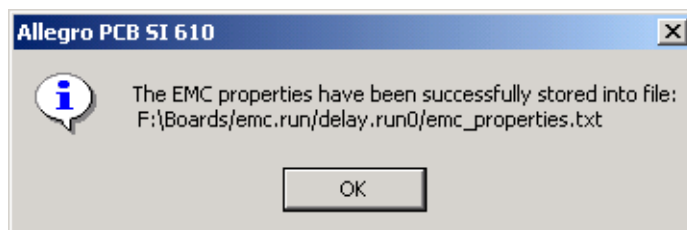


Figure 4-15 Saved EMC Properties Verification Message



Loading EMC Properties Previously Saved

If you had previously saved the EMC properties assigned to a design, you can load the same properties back into the design for further rule checking.

To load EMC properties back into a design

1. Click the *Load Property* button in the EMC Design Configuration dialog box (see [Figure 4-2](#) on page 42).

The Load EMC Properties dialog box appears as shown in [Figure 4-16](#) on page 58.

2. In the *File* text box, enter a pathname to the file (`emc_properties.txt`) where the properties are stored.

- or -

Click the Browse (...) button to navigate to the file.

3. Click the *Load* button.

The properties in the specified file are loaded into the dialog box Property grid for selection.

4. Display each tab in the dialog box and select the properties you wish to load back into your design.

You select a property for loading by clicking its checkbox located in the left column (an **x** appears in the box). You can easily select all properties for loading by clicking the *Select All* button.



Tip

To list only specific properties, you can filter and sort the entries in the Property grid.

- ☐ To filter, enter a unique filter string in the *Name*, *Property*, or *Value* text boxes followed by an asterisk (*) character, then click anywhere in the grid to refresh the list. For example `EMC*` in the *Property* text box lists only properties whose name leads with EMC as shown in [Figure 4-16](#) on page 58.
- ☐ To sort, click-right on a column header and selecting the desired Sort option from the menu as shown in [Figure 4-16](#) on page 58.

5. Click *Attach*.

The selected EMC properties are loaded back into the design.

EMControl User Guide

Setting Up the EMControl Design Environment

Figure 4-16 Load EMC Properties Dialog Box

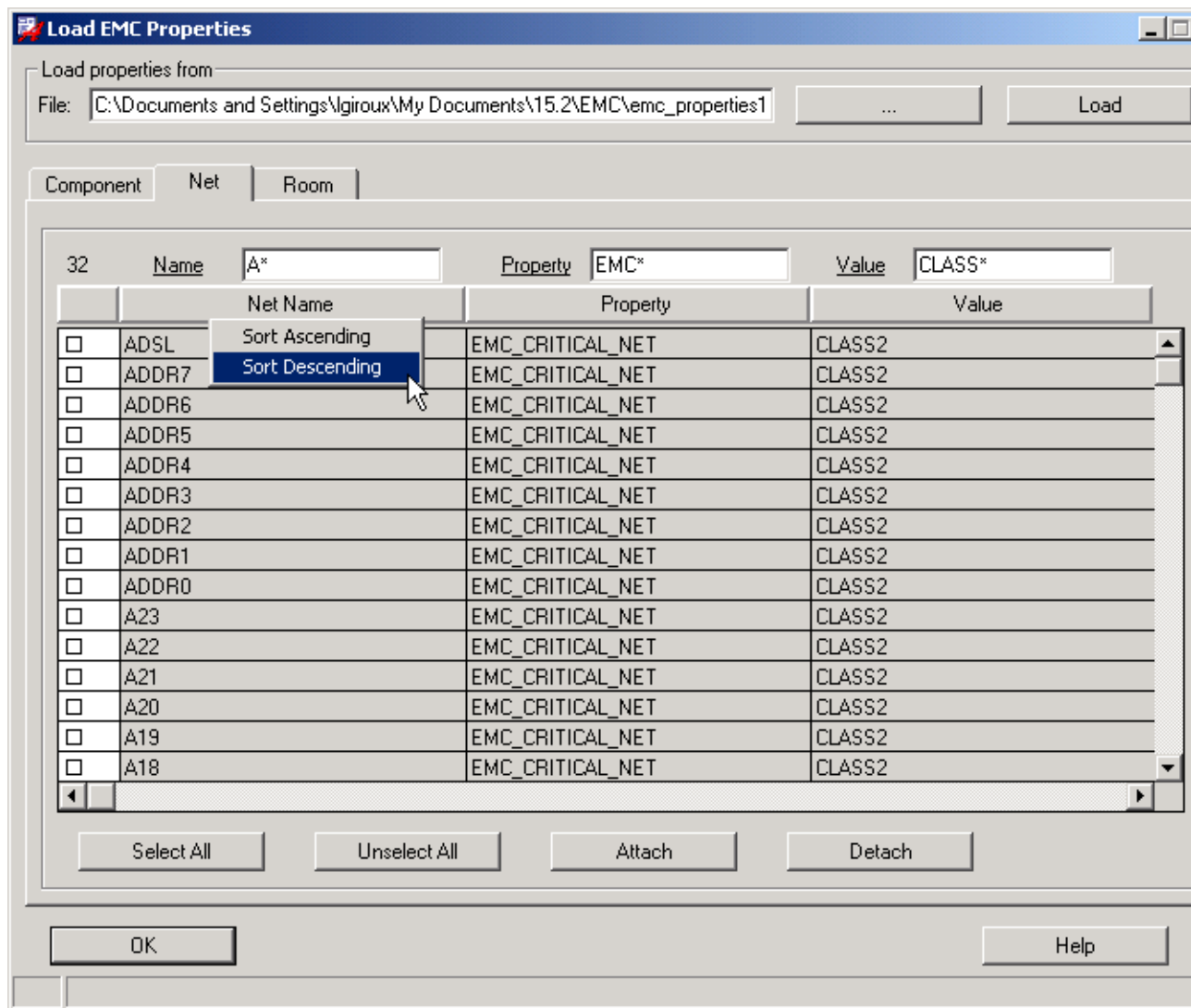


Table 4-1 Load EMC Properties Options

<i>File</i>	Specifies the path where your property file is located. You can use the <i>browse(...)</i> button to select the path or input a new path.
<i>Load</i>	Populates the Spreadsheet window with the properties in the specified file.
<i>#</i>	Total number of properties listed in the Spreadsheet window.

EMControl User Guide

Setting Up the EMControl Design Environment

<i>Name</i>	Specifies a unique character or string to filter by net name. The default is * which lists all nets (no filtering). For example, A* lists only nets with names leading with A.
<i>Property</i>	Specifies a unique character or string to filter by property name. The default is * which lists all property names (no filtering). For example, EMC* lists only properties with a name leading with EMC.
<i>Value</i>	Specifies a unique character or string to filter by property value. The default is * which lists all property values (no filtering). For example, CLASS* lists only properties with a value leading with CLASS.
<i>Select All</i>	Selects all properties for loading.
<i>Unselect All</i>	Unselects all properties for loading.
<i>Attach</i>	Attaches the selected (checked) properties to the named objects in the design.
<i>Detach</i>	Detaches (removes) the selected (checked) properties from the named objects in the design.
<i>OK</i>	Closes the dialog box.

EMControl User Guide

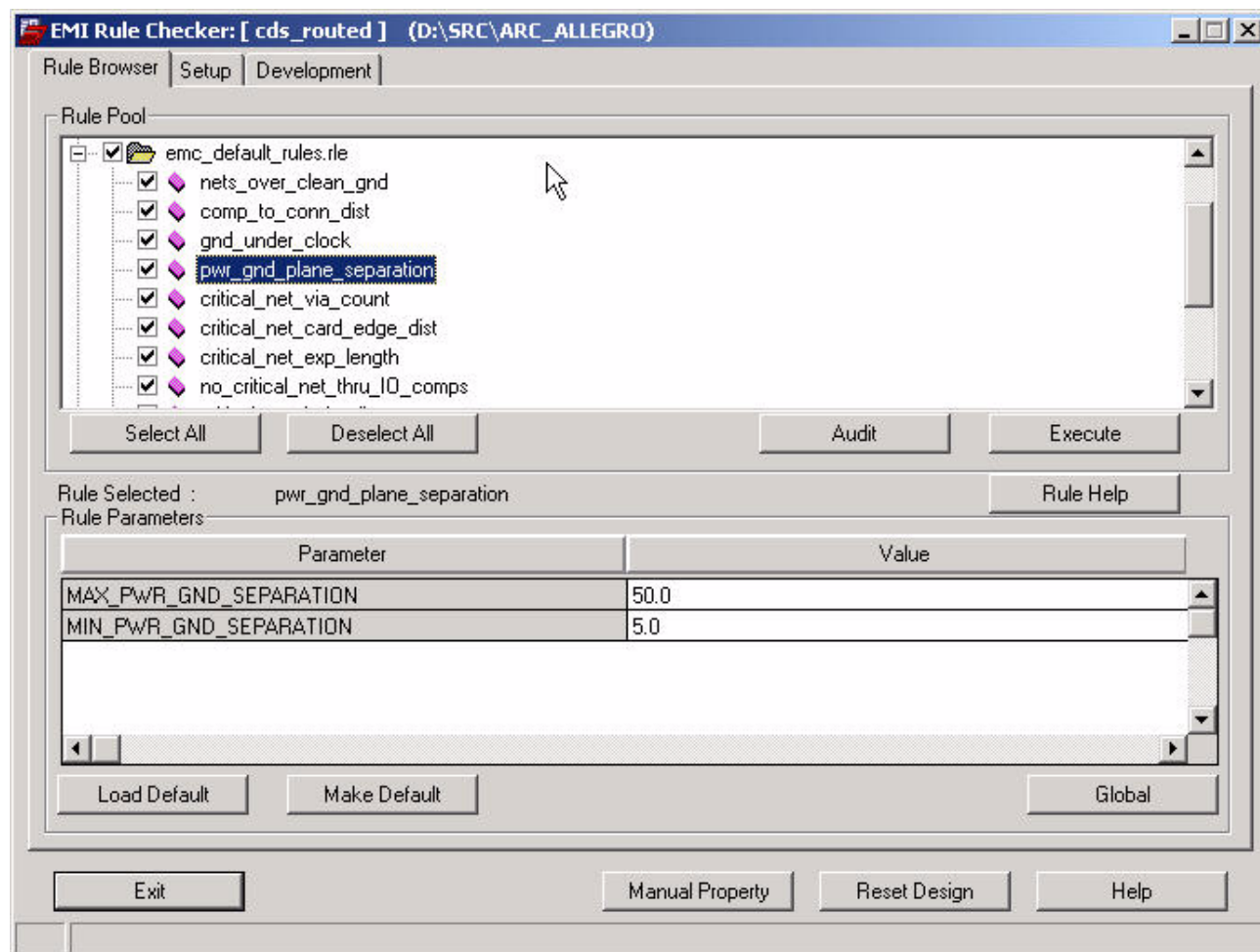
Setting Up the EMControl Design Environment

Performing EMControl Rule Checking

Overview

The EMControl Rule Checker allows you to check the EMI/EMC on your design. The Rule Checker is illustrated below.

Figure 5-1 EMC Rule Checker Dialog Box



The EMC Rule Checker dialog box enables you to:

- Define the Scope of the Checking
- Select Rules to be Checked
- Customize EMControl Variables
- Audit EMControl Rules
- Execute EMControl Rules
- View Results
- View Checking Reports
- Reset Design After Rule Checking

Click the *Exit* button to save the configuration information and exit the EMControl.

Define the Scope of the Checking

If you run EMControl for an entire design, you may abuse checking rules by applying them to entire areas of the design where they do not apply. This increases rule checking time. Therefore, it is recommended that you check the design by first defining windows, then select only those rules that are relevant to that portion of the design.

You can perform EMControl design checking for:

- The complete design
- A selected window

The following figures [5-2](#) and [5-3](#) show the two different user interfaces for the above two check types.

To check the entire design,

- Select *Complete Design* in the *Scope of Check* section.

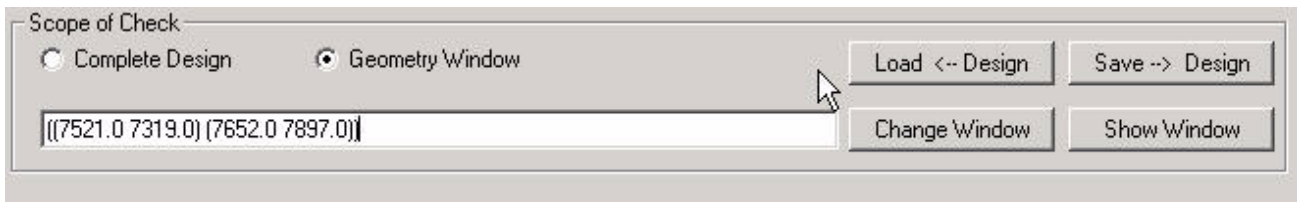
Figure 5-2 Rule Checking for the Complete Design



To check a portion of the design,

1. Select *Geometry Window* in the *Scope of Check* section.

Figure 5-3 Rule Checking for Specified Rectangular Window



2. Click on *Change Window*.

Specify the extent of the design area to check (the bounding box).

3. Click in the layout window to select a corner of the bounding box.
4. Click again to select the opposite corner (to define the diagonal extent) of the bounding box.
5. The coordinates that you selected are displayed in the text field to the right of the *Geometry Window* button.

Important

If you do not specify a bounding box for the rule check, EMControl uses the coordinates of the previous check run, if any. If you have not defined coordinates and none of them are available from the previous run, EMControl checks the entire design.

To highlight the scope you just selected,

- Click *Show Window*.

- or -

To dehighlight the scope you just selected,

- Click *Hide Window*.

To save the *Scope of Check* into the design for future use.

- Click *Save->Design*.

Any previous scope saved will be overwritten.

- or -

To load the *Scope of Check* information stored in a design,

- Click *Load->Design*.

Select Rules to be Checked

The dialog box illustrated in figure [5-1](#) is the user interface used to select rules to be checked and specify rule parameters, if any.

Displaying Rule Files Selected

When you open the *EMC Rule Browser* dialog box for the first time, only the names of the compiled rule files are displayed. The files are in the unexpanded condition (that is, no groups or rules within that file are visible). EMControl searches for compiled rule files you select. You can add and delete rule files by following the instructions in [Setting Up Rule Files](#) on page 33. When EMControl checks the design for EMC problems, it uses only the rules that you have selected in the *Rule Browser*. When the list of rules extends beyond the browser window in either the vertical or the horizontal direction, the scroll bar lets you view the remaining rules.

Displaying the Rules in a File

You can choose either to display all the rules in a rule file (expand the rule), or to turn off (collapse) such a display.

To display all the rules in a file when only the file name is currently visible,

- Click on the + sign of the tree view.

The tree view expands to show all the rules in the rule file. All the rules are displayed either selected (that is, highlighted) or deselected, according to the setting of the file name when you expanded it.

To turn off the display of all the rules in a file,

- Click on the - sign of the tree view.

The tree view contracts to remove from the display all the rules in the rule file. The file name remains either selected or deselected, according to the setting of the file name when you unexpanded it.

Selecting Rules to Use for Design Checking

The following conventions apply to selecting and deselecting rule files:

- Selecting the name of a rule file automatically selects all the rules in the corresponding file for checking, whether the individual rules are currently visible or not.
- Deselecting rules by file name deselects all the rules included in that file.
- To select all the rules, click on the *Select All* button.
All the rule files and rule names in the browser are selected.
- To deselect all the rules, click on the *Deselect All* button.
All the rule files and rule names in the browser are deselected.
- To select or deselect all rules in a file, click the check box corresponding to the file name.
If the check box of the file name is currently checked, it is unchecked. All rules within that file are also unchecked. If it is not currently checked, it is selected. All rules within that file are also checked.
- To select or deselect a single rule click the check box corresponding to the rule name.
If the rule is currently checked, it is unchecked. If it is not currently checked, it is checked.

Getting Help on a Rule

When a rule is highlighted using mouse, the rule name will be displayed in the *Rule Selected* field. To read help information about this rule, click on the *Rule Help* button. A window opens with reference information on the rule.

Customize EMControl Variables

In a design, you may use default variables or, based upon the requirements, edit default variables to create user-defined variables. The `emc_param.par` file contains the default definitions provided by Cadence for all user-definable EMControl variables. Your particular PCB design may require that you edit the default variable values, or choose to use the default values as provided.

There are three layered parameter files that EMControl uses, the default parameter files, the site parameter files, and the custom parameter files. The variables defined in latter files will override the variables defined in previous ones. For example, the variable x in `emc_custom.par` with the value 10 will override the value 20 defined in the site parameter file and the value 100 in the default parameter file.

You can customize both the rule variables and the global variables.

- [Customizing EMControl Rule Variables](#)
- [Customizing EMControl Global Variables](#)

Customizing EMControl Rule Variables

The rule variables are specific to every rule. To customize the rule variables:

1. Select a rule.

All rule variables will be displayed in the *Rule Parameters* section.

2. Select the variable from *Rule Parameters* section.
3. Customize the value as needed in the edit box.

The new value will be stored in the EMControl Customize parameter file, `emc_custom.par`, for local use.

4. Click on the *Load Default* button if you wish to keep the default value.
5. Click on the *Make Default* button if you wish to keep the new settings as site default.

This writes the new values into the site parameter file.

Note: You need write permissions for the site parameter file in order to change the default settings for a site.

Important

The data type for variables is one of `STRING`, `STRING LIST`, `NUMERIC`, and `NUMERIC LIST`. When you customize the variables, please pay attention to the data type otherwise they will fail to change.

Customizing EMControl Global Variables

EMControl provides some global variables for all rules to use.

To Customize EMControl Global Variables

1. Click the *Global* button in the EMControl main window.

This brings up the form illustrated in figure 5-4.

Figure 5-4 Customizing EMControl Global Variables

Parameter	Value
BYP_CAP_SENS_DIST	300.0,350.0,400.0,450.0,500.0
CRITICAL_TO_MHATTAN_LEN_RATIO	1.1,1.2,1.3,1.4,1.5
DESIGN_MARGIN	30.0
EMC_COMP_CLASSES	"CLASS1","CLASS2","CLASS3","CLASS4","CLASS5"
EMC_NET_CLASSES	"CLASS1","CLASS2","CLASS3","CLASS4","CLASS5"
EMI_REGULATION	"FCC_CLASS_B"
EXPOSED_LENGTH	7500.0
GRID_X	0.0
GRID_Y	0.0
MAX_PEAK_XTALK	0.05,0.1,0.15,0.2,0.25
MAX_SIMULATION_CYCLE	2
MyLocalParam	"ME","You","HE"
PERFORM_COLORING	"YES"
PULSE_DUTY_CYCLE	"100MHz:50% 75MHz:50% 50MHz:50%"

Buttons: Load Default, Make Default, OK, Help

When the form is displayed, all of the global variables are listed with their current values.

2. Select a variable from list.

3. Customize the value as needed in the edit box.

The new value is stored in EMControl Customize parameter file, `emc_custom.par`, for local use.

4. Click on *Load Default* if you wish to keep the default settings.

5. Click on *Make Default* if you wish to keep the new settings.

This writes the new value into the site parameter file.

6. Click *OK* to close the dialog box.

Note: You need write permissions for the site parameter file in order to change the default settings for a site.

Important

The data type for variables is one of `STRING`, `STRING LIST`, `NUMERIC` and `NUMERIC LIST`. When you customize the variables, please pay attention to the data type otherwise you will fail to change.

Audit EMControl Rules

Cadence recommends that you verify whether the property requirements have been met for the selected rules before you execute rule checking. EMControl generates an Audit Report whenever you execute an Audit.

The Audit Report describes the properties that have not been properly configured for the EMC rules selected for design checking. The Audit Report reflects the contents of the ASCII file called `emcrc_verify.msg`.

To generate an Audit Report:

1. Select the rules using the methods described in [Select Rules to be Checked](#) on page 64.
2. Click the *Audit* button.
3. The Audit Report is displayed in the EMC Report window.

Execute EMControl Rules

To execute EMC rules, click the *Execute* button in the EMControl main window. EMControl starts checking the current design against the rules selected. The progress of rule execution will be available in the Allegro PCB SI CWI.

The following files are created in the EMC design-specific working directory during design checking:

- `emcrc_execute.msg`
- `emcrc.mkr`
- `emcrc.log`

The result files are updated automatically and contain the violation messages that are generated during the most recent run.

To abort EMC Rule execution,

- Enter *Control-C* in the layout Command window to abort EMControl checking.
EMControl checking is terminated.
- or -
- Click *Stop* in the Allegro PCB SI Status window.



If you abort EMC rule execution, the EMControl files will be left in an unknown state.

After the EMC Rule Checking,

- a confirmation window is displayed with "No Violations" if no violations are found.
- a Marker window is displayed with violations.

See Chapter 6, "Resolving EMC Rule Violations in Your Design."

View Results

To view the results of EMControl verification and rule checking, you can:

- View the list of violation messages found during rule checking.
- Highlight the object in your layout that corresponds to a particular violation.
- View the details of a violation, including a description of the rule that is violated.
- Open a report that summarizes the results of either property auditing or design checking.
- Open a results text file to view descriptive details for each violation message.

For detailed information on viewing and assessing violations in the design, see [Chapter 6, “Resolving EMC Rule Violations in Your Design.”](#)

EMControl enables you to

- Click the *Tag Report* button to [View the Log File for Automatically Property Tagging](#).
- Click the *Audit Report* button to [View Audit Reports](#)
- Click the *Check Report* button to [View Checking Reports](#)
- Click the *Marker Report* button to view the Marker Report.

See [Chapter 6, “Resolving EMC Rule Violations in Your Design.”](#)

View the Log File for Automatically Property Tagging

When you automatically attach critical frequency class properties to objects in a design, EMControl creates the EMC AutoPropertyTag log file (`emcrc_propedit.log`) in the EMControl design-specific working directory and displays the file.

The log file contains the following sections:

- [Header Information](#)
- [Information About Critical Nets](#)
- [Information about Components on Critical Nets](#)
- [Information About Critical Regions](#)
- [Summary of Tagging](#)

Header Information

The EMC AutoPropertyTag Log file contains the following header information:

- The name of the design for which the report is generated
- The date and time the report is generated
- The settings for the run:
- The names of critical frequency classes used
- The slew rate associated with each class
- Whether Supersede Existing Properties is on or off

An example of a log file header is illustrated in the following figure.

Figure 5-5 AutoPropertyTag Header Information

```
|-----+  
| Auto Property Tagging Log |  
| Drawing      : G:\sig_tutor.brd |  
| Date/Time   : May 11 09:26:52 2000 |  
|-----+
```

*** Settings for this run ...

Slew Rate values (V/ns) :

CLASS1 : 5.000

CLASS2 : 2.500

CLASS3 : 1.667

CLASS4 : 1.250

CLASS5 : 1.000

Supersede Existing Properties : ON

Information About Critical Nets

The Checking Nets section of an EMC AutoPropertyTag log file contains the following information:

- The name of the net

- The maximum slew rate on the net
- The critical frequency class name value for the `EMC_CRITICAL_NET` property if it is attached to the net

Figure 5-6 illustrates an example from the Checking Nets section from an EMC AutoPropertyTag log file.

Figure 5-6 Information About Critical Nets

```
Checking Extended net ( NET2A )
Maximum slew rate for this xNet is 5.000000 V/ns
Existing EMC_CRITICAL_NET = CLASS1 is correct

Checking Extended net ( NET3 )
Maximum slew rate for this xNet is 5.000000 V/ns
Existing EMC_CRITICAL_NET = CLASS1 is correct

Checking Extended net ( NET4 )
Maximum slew rate for this xNet is 3.333330 V/ns
Changing EMC_CRITICAL_NET from CLASS1 to CLASS2
```

Information about Components on Critical Nets

The Checking Components section of an EMC AutoPropertyTag log file contains the following information for every component:

- The name of each component
- The names and critical frequency classes for all critical nets that connect to the component
- The number of nets of each critical class.
- The critical frequency class name value for the `EMC_CRITICAL_IC` property attached to the component (this is the highest value of the `EMC_CRITICAL_NET` property attached to the nets connected to the component)

Figure 5-7 illustrates an example of the Checking Components section of an EMC AutoPropertyTag log file.

Figure 5-7 Information about Components on Critical Nets

*** Checking Components ...

Critical Net(s) connected to U1 :

```
[ NET4:CLASS2 NET3:CLASS1 VCC:CLASS1 GND:CLASS1 ]  
[ CLASS1 - 3 CLASS2 - 1 ]  
Adding EMC_CRITICAL_IC = CLASS1
```

Critical Net(s) connected to U2 :

```
[ VCC:CLASS1 GND:CLASS1 NET3:CLASS1 NET4:CLASS2 NET2A:CLASS1 ]  
[ CLASS2 - 1 CLASS1 - 4 ]  
Adding EMC_CRITICAL_IC = CLASS1
```

Information About Critical Regions

The Checking Regions section of an EMC AutoPropertyTag log file contains the following information:

- A list of components in the room with critical frequency class name values attached
- The total number of components tagged with each critical frequency class
- The critical frequency class name value for the EMC_CRITICAL_REGION property attached to the room.

Figure 5-8 illustrates an example taken from the Checking Regions section of an EMC AutoPropertyTag log file.

Figure 5-8 Information About Critical Regions

*** Checking Regions ...

Critical Component(s) in ROOM1 :

```
[ ZC5:CLASS2 ZB5:CLASS2 ZC6:CLASS2 ZB6:CLASS2 ZC7:CLASS2 ZB7:CLASS2 ]  
[ CLASS2 - 6 ]  
Existing EMC_CRITICAL_REGION = CLASS2 is correct
```

Summary of Tagging

The summary of tagging section contains a summary of the tagged critical components, critical nets, and critical regions. An example of the Summary of Tagging section is illustrated in figure 5-9.

Figure 5-9 Summary of Tagging

Autotag log summary

Critical nets tagged

CLASS1:

NET2A

NET3

CLASS2:

NET1A

NET1

NET4

Critical components tagged

CLASS1:

U2

U1

Critical regions tagged

CLASS1:

room1

clock_room

View Audit Reports

Audit Report information assists you in locating and correcting rule violations. The Audit report contains the following sections:

- Audit Report Header Information
- Audit Report Rule-Specific Information
- Audit Report Summary Information

Audit Report Header Information

The upper portion of an Audit Report

- Lists the complete path of the checked design
- Lists the names of the rules that you check

Figure 5-10 is an example of a report header.

Figure 5-10 Audit Report Header Information

```
*****
Design Name: D:\MyOwn\mytesting\cds_routed.brd
Rules Checked:
    single_diff_mode_EMI
    sum_diff_mode_EMI
    pwr_gnd_plane_separation
    bypass_cap_per_plane_square
    filtered_IO_signals
    no_critical_net_thru_IO_comps
    gnd_under_clock
    comp_not_conn_dist
    comp_to_conn_dist
    gnd_screw_between_clock_and_conn
    central_clock
    filters_to_clean_gnd
    bypass_critical_IC
    bypass_cap_type
    bypass_fast_sw_trans
    bypass_drvr_rcvr_bidir
    critical_IC_loop_area
    critical_IC_3caps_C_2C_4C
    clock_spectral_content
    max_pwr_gnd_resistance
    critical_net_via_count
    critical_net_via_pin_ratio
    critical_net_card_edge_dist
    critical_net_exp_length
    critical_net_man_ratio
    max_critical_net_xtalk
    shield_clock_nets
    critical_net_hole_dist
    critical_net_return_path
    critical_net_termination
    critical_net_ringing
    bypass_pwr_trace
    pwr_gnd_trace_width
    return_path_near_signal_via
```

Audit Report Rule-Specific Information

The body section of the Audit Report describes the short message (if it exists) and the advisor messages for each violation. These messages list:

- The severity level of the violation
- The name of the rule
- Information about the rule violation

Figure 5-11 is an example of the body section of an Audit Report.

Figure 5-11 Audit Report Rule-Specific Information

```
*****
WARNING (filtered_IO_signals)
All IO signals must be filtered.
Connector 'J1' has unfiltered IO signals.

The following IO nets running to connector 'J1'
need to be filtered:
    'A1'
    'A2'
    'A3'
    'A4'
    'A5'
    'A6'
    'A7'
    'A8'
    'A9'
    'VCC'
    'A10'
    'A11'
    'A12'
    'A13'
    'A14'
    'A15'
    'D0'
    'D1'
    'D2'
    'D3'
    'D4'
    'D5'
    'D6'
    'D7'
    'D8'
    'D9'
    'D10'
    'D11'
    'D12'
```

EMControl User Guide

Performing EMControl Rule Checking

Audit Report Summary Information

The bottom section of the Audit Report provides a summary of the total number of rule violations and the number of violations recorded for each rule, by severity.

Figure 5-12 is an example taken from the body of an Audit Report.

Figure 5-12 Audit Report Summary Information

```
*****
Violations: WARNING = 2
*****

Rule Based Summary
*****
RULENAME          INFO    OVERSIGHT    WARNING    ERROR    Fatal    TOTAL
single_diff_mode_EMI      0         0         0         0         0         0
sum_diff_mode_EMI        0         0         0         0         0         0
pwr_gnd_plane_separation  0         0         0         0         0         0
bypass_cap_per_plane_square 0         0         0         0         0         0
filtered_IO_signals       0         0         1         0         0         1
no_critical_net_thru_IO_comps 0         0         0         0         0         0
gnd_under_clock          0         0         0         0         0         0
comp_not_conn_dist       0         0         0         0         0         0
comp_to_conn_dist        0         0         1         0         0         1
gnd_screw_between_clock_and_conn 0         0         0         0         0         0
central_clock            0         0         0         0         0         0
filters_to_clean_gnd     0         0         0         0         0         0
bypass_critical_IC       0         0         0         0         0         0
bypass_cap_type          0         0         0         0         0         0
bypass_fast_sw_trans     0         0         0         0         0         0
bypass_drvr_rcvr_bidir   0         0         0         0         0         0
critical_IC_loop_area     0         0         0         0         0         0
critical_IC_3caps_C_2C_4C 0         0         0         0         0         0
clock_spectral_content    0         0         0         0         0         0
max_pwr_gnd_resistance    0         0         0         0         0         0
critical_net_via_count    0         0         0         0         0         0
critical_net_via_pin_ratio 0         0         0         0         0         0
critical_net_card_edge_dist 0         0         0         0         0         0
critical_net_exp_length   0         0         0         0         0         0
critical_net_man_ratio    0         0         0         0         0         0
max_critical_net_xtalk    0         0         0         0         0         0
shield_clock_nets        0         0         0         0         0         0
critical_net_hole_dist    0         0         0         0         0         0
critical_net_return_path  0         0         0         0         0         0
critical_net_termination  0         0         0         0         0         0
critical_net_ringing      0         0         0         0         0         0
bypass_pwr_trace         0         0         0         0         0         0
pwr_gnd_trace_width      0         0         0         0         0         0
return_path_near_signal_via 0         0         0         0         0         0
conn_in_low_freq_regions  0         0         0         0         0         0
nets_over_clean_gnd     0         0         0         0         0         0
decouple_emc_regions     0         0         0         0         0         0
fence_off_emc_regions    0         0         0         0         0         0
TOTAL                   0         0         2         0         0         2
*****
```

View Checking Reports

The results of verification and EMC checking are automatically saved to the EMControl design-specific working directory that you specify in the EMControl system environment.

The results from the *Execute* command are stored in the `emcrc.mkr` and `emcrc_execute.msg` files. Having a separate directory for each check run allows convenient access to the results of previous runs. User can use these directories to compare the results between subsequent check runs.

The report results from the *Audit* command are stored in the `emcrc_verify.msg` file.

Reset Design After Rule Checking

EMControl enables user to clean the design after EMControl rule checking.

To reset design,

- Click the *Reset Design* button in the EMControl main window.

Resolving EMC Rule Violations in Your Design

Overview of Rule-Checking

When EMC rule checking is complete, you can display the results from the EMC main window. EMControl displays the list of messages for all the violations that were found, and lets you view the objects (pins, nets, or components) in your design that correspond to each violation. EMControl can display up to 1000 messages in the violations list.

EMControl uses colored markers to highlight the objects that are in violation of your selected rules. As you select each violation message, the corresponding marker is displayed in a contrasting color. When more than one object is found for a violation, a pop-up menu lets you select which object to highlight.

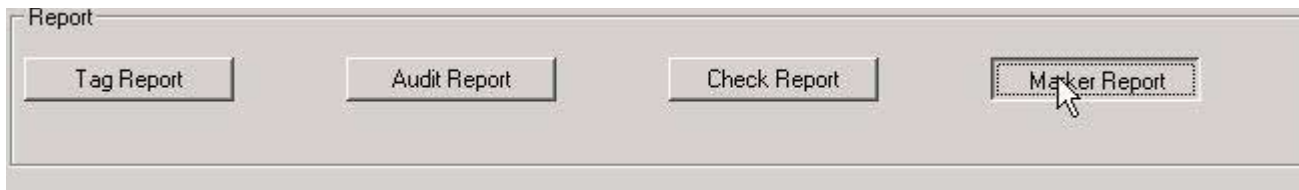
As you examine each violation, you can correct it within your design. When you have resolved all violations, you can check the design again, repeating the process as often as necessary.

EMControl also creates a markers file called `emcrc.mkr` when it finds rule violations. The file contains detailed information about each violation. EMControl uses the contents of this file when you request a display of results from the EMControl main window as figure [6-1](#).

Viewing the Results of Rule-Checking

This section shows you how to locate violations in your design. In addition to viewing violations in the design, you can check the `emcrc_execute.msg` or `emcrc.mkr` file to determine where violations were found in your design, or you can read a formatted report.

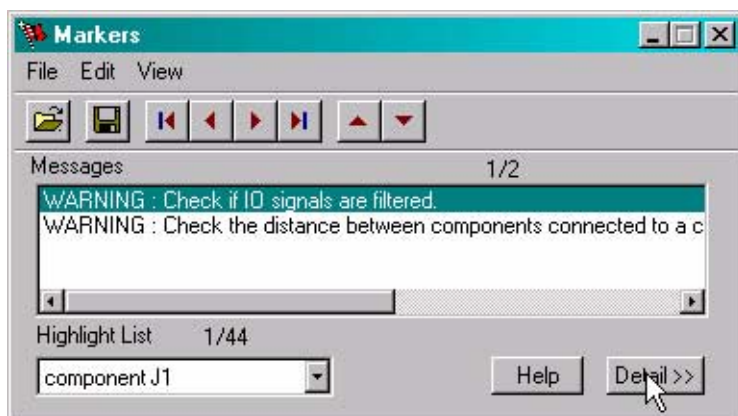
Figure 6-1 Viewing the Results of Rule-Checking



To view the results of design checking,

- Click *Marker Report* in the EMControl main window as shown in figure [6-1](#).
The Markers dialog box is displayed.

Figure 6-2 EMControl Marker Window



Viewing a Violation

1. Click on the violation message in the *Messages* list of Markers dialog box.

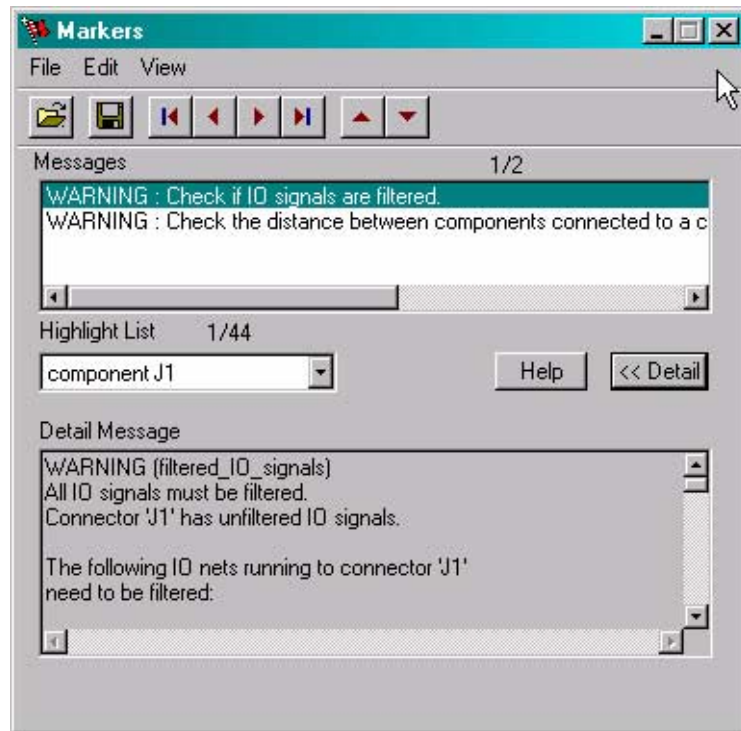
The selected message is highlighted in the list.

Note: The left-right arrow icons in the markers toolbar can be used to traverse the list of violations.

2. Click *Detail>>*.

The details of a violation message appears in the markers *Detail Message* window.

Figure 6-3 Marker Detail Message Windows



Highlighting Objects

1. Select the violation message in the Markers dialog box.

The first object associated with it is automatically highlighted in the Design window.

2. Select an object from the *Highlight List* combo box that lists all the violating objects of a violation.

The selected object is highlighted in the Design window.

Note: You can also use the up-down arrow icons in the toolbar to traverse and highlight the violating objects of a message.

Filtering Violations

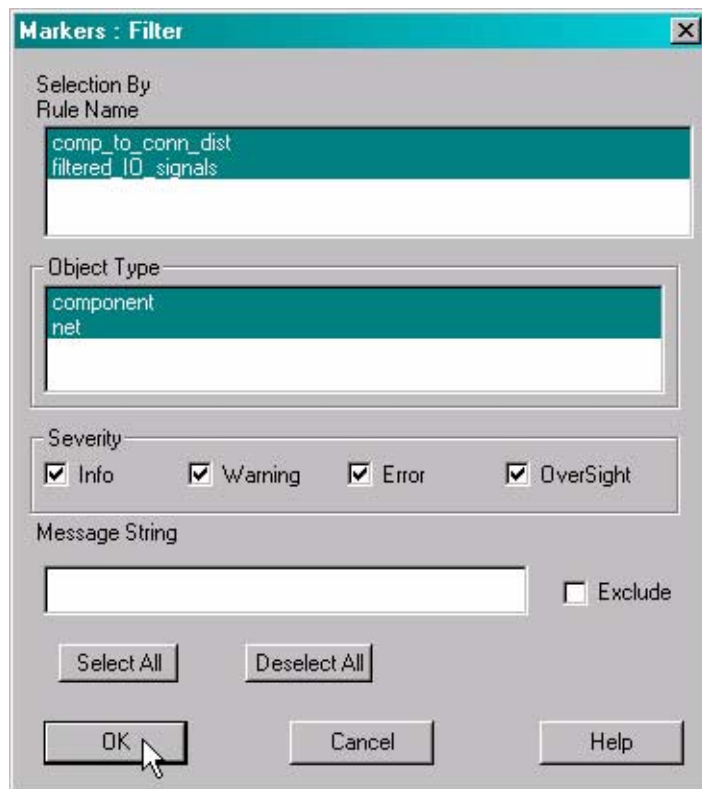
The EMC Filters dialog box lets you control which violation messages are displayed in the messages browser. You can customize the display of messages in the browser by rule name, object type, and severity level. You can also select to display, or not display, messages that match a particular regular expression.

To open the Filters dialog box,

- Select *View – Filter Options...* from the Markers menu.

The *Filter* dialog box appears.

Figure 6-4 Marker Filter Window



Filtering Messages by Rule Name

You can display violations by the name of the rule associated with them. All rule names are displayed in a list box at the top of the EMC Filters dialog box.

To control the display of violations by rule name, select (highlight) only the names of the rules for which you want to display violations. Violations are displayed for the rule if its name is highlighted in the box. By default, all rules are selected.

- ❑ Use *Select All* to highlight all rule names.
- ❑ Use *Deselect All* to remove highlighting from all rule names.
- ❑ Click on a selected rule name to deselect it.

- ☐ Click on a deselected rule name to highlight and select it.

Filtering Messages by Object Type

You can display violations by object type. For example, you might want to view only violations on components or on pins.

To control the display of violations for objects of a particular type, select (highlight) only the object types for which you want to display violations. Violations are displayed for the object type if its name is highlighted in the box. By default, all object types are selected.

- ☐ Use *Select All* to highlight all object types.
- ☐ Use *Deselect All* to remove highlighting from all object types.
- ☐ Click on a selected object type to deselect it.
- ☐ Click on a deselected object type to highlight and select it.

Filtering Messages by Severity Level

You can display violations by severity level. For example, you might want to view only messages for violations labeled `ERROR`.

To control the display of violations by severity level, check the box to the left of one or more Severity definitions.

- ☐ Violations are displayed for a severity if its box is checked.
- ☐ No violations are displayed for a severity if its box is unchecked.

Filtering by Regular Expression

You can use the Message String field to specify a UNIX regular expression to use as a filter criterion for the display of violations. For example, you might want to display only messages that contain `ground` or to exclude from display all messages that refer to `output pins`.

To filter by regular expression,

1. Type a regular expression in the *Message String* field.
2. If you want the regular expression to define messages to exclude from display, check *Exclude*.

Examples

1. To search for messages that contain `ground` anywhere in the message, type the following in the Message String field: `ground`
2. To filter out of the messages browser any messages that contain output pins, type the following in the Message String field and check *Exclude*: `output pins`
3. To search for messages that contain `output` and `pins` (in order, but not necessarily together), Type the following in the Message String field: `output.*pins`

Applying Your Filtering Selections to the Browser Display

To apply your filtering definition,

- Select *OK* in the EMC Filters dialog box.

The dialog box closes. The messages browser in the EMC Results dialog box is updated to reflect your selections. If none of the violations pass your filtering criteria, a message informs you that all markers are filtered out.

Cross-Probing Multiple Violations

The Markers dialog box lists multiple violations.

1. Click the first violation to highlight it.
2. Select *Detail>>* in the Markers dialog box to bring up a detailed description of the highlighted violation.
3. Click *Highlight List* in the Markers dialog box.

This opens a combo box that lists the offending elements corresponding to the selected violation. Select one of the elements from the combo box. The selected object is highlighted in the Design window.

You can use the toolbar up or down icons to traverse through the list of violations. As you select each message, the detailed violation description in the *EMC Advisor* dialog box is automatically updated. The first element in the *Highlight List* of each violation is highlighted as and when you traverse. Toolbar icons are provided to traverse the *Highlight List* also.

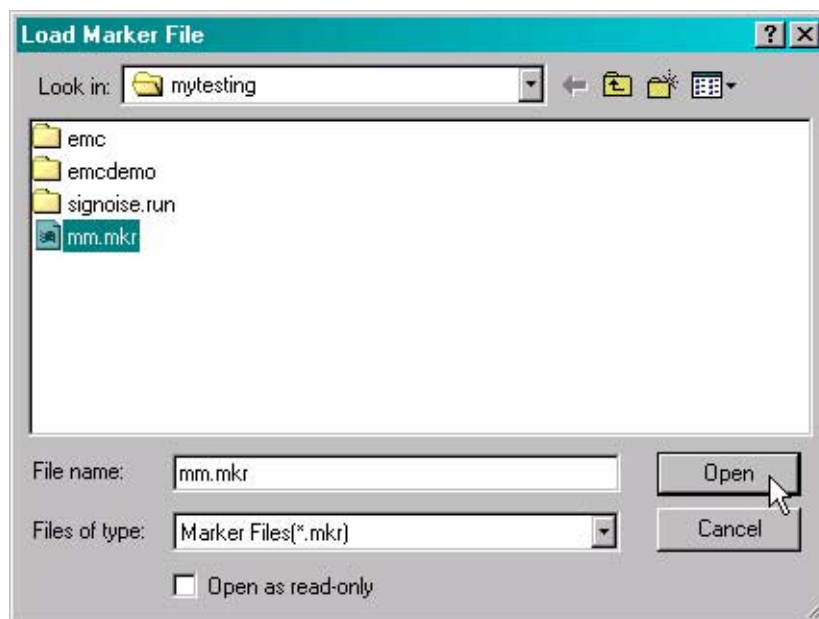
The numeric designator near the *Highlight List* indicates whether there are multiple objects associated with the current message or not. The second number on the button specifies how many objects are there for this violation, and the first number indicates which object of the sequence is currently highlighted in the design. You can use up or down icon to Markers toolbar to traverse *Highlight List*.

4. Select *Close* in the Markers dialog box when you have finished viewing violations.

Loading a Markers File

1. Select *File – Load* from the Markers menu to invoke the Marker File browser.
2. Select the markers file using the file browser.

Figure 6-5 Load a Marker File



3. Click *Open*.

The main markers window is loaded with messages from the specified markers file.

Note: The corresponding design must be loaded in Allegro PCB SI when you load a marker file.

Hiding a Violation Message

You can remove a violation from the messages browser in the Markers form. You might choose to remove a violation from view if you have determined that the violation is actually acceptable in your design or if you have corrected the violation.

To remove a violation in the messages browser,

1. Select the violation message to remove.

The message is highlighted.

2. Click *Edit – Delete Marker*.

The message is hidden from the messages browser, and the associated object is dehighlighted in the layout. The display in the upper right corner of the EMC Results form is updated to reflect the change in the total number of messages in the browser.

Note: This removes the violation only from the messages browser. The violation is not deleted from the markers file. To permanently remove from the markers file the violations that you have hidden from the browser display, see [“Saving a Markers File.”](#)

Saving a Markers File

You can update a markers file to permanently remove any violations that are currently hidden from the messages browser.

To permanently remove hidden messages,

- Select *File – Save* in the Markers form.

The updated markers file is saved.

Note: All violations that you removed from display using *Edit – Delete Marker* are permanently erased from the saved file.

Reading an Execute Report

The following sections describe the format of the Execute report.

Header Information

The upper portion of an Execute report

- Tells you which design was checked
- Lists the names of the rules that you checked

Figure [6-6](#) is an example of a report header:

Figure 6-6 Execute Report Header Information

```
*****
Design Name: G:\sig_tutor.brd
Rules Checked:
    pwr_gnd_plane_separation
    bypass_cap_per_plane_square
    single_diff_mode_EMI
    sum_diff_mode_EMI
    bypass_critical_IC
    bypass_cap_type
    bypass_fast_sw_trans
    bypass_drvr_rcvr_bidir
    critical_IC_loop_area
    critical_IC_3caps_C_2C_4C
    filters_to_clean_gnd
    comp_not_conn_dist
    comp_to_conn_dist
    gnd_screw_between_clock_and_conn
    central_clock
    gnd_under_clock
    max_pwr_gnd_resistance
    clock_spectral_content
    bypass_pwr_trace
    pwr_gnd_trace_width
    critical_net_termination
    critical_net_ringing
    decouple_emc_regions
    fence_off_emc_regions
    nets_over_clean_gnd
    conn_in_low_freq_regions
*****
```

Rule-Specific Information

The body of the Execute report describes the short message (if one exists) and the advisor message for each violation.

These messages list

- The severity level of the violation
- The name of the rule
- Information about the violation

Figure 6-7 is an example of rule-specific information. The rule is `gnd_under_clock`:

Figure 6-7 Execute Report Rule-Specific Information

```
WARNING (gnd_under_clock)
Clock generators must have symmetrically placed ground shapes
(grounded grids) on the same ETCH subclass as the footprint.

Clock generator 'U1' does not have a ground shape under
it on ETCH/TOP .

WARNING (gnd_under_clock)
Clock generators must have symmetrically placed ground shapes
(grounded grids) on the same ETCH subclass as the footprint.

Clock generator 'U2' does not have a ground shape under
it on ETCH/TOP .
```

Summary Information

The bottom of the report provides a summary of the number of violations recorded by severity for each EMC rule, as shown in figure 6-8.

Figure 6-8 Execute Report Summary Information

```
*****
Violations: WARNING = 7
            ERROR  = 2
*****
Rule Based Summary
*****
RULENAME          INFO OVERSIGHT WARNING ERROR Fatal TOTAL
pwr_gnd_plane_separation  0          0          0          0          0          0
clock_spectral_content    0          0          6          0          0          6
bypass_pwr_trace         0          0          1          0          0          1
pwr_gnd_trace_width      0          0          0          2          0          2
critical_net_termination  0          0          0          0          0          0
critical_net_ringing      0          0          0          0          0          0
decouple_emc_regions     0          0          0          0          0          0
fence_off_emc_regions    0          0          0          0          0          0
nets_over_clean_gnd     0          0          0          0          0          0
conn_in_low_freq_regions  0          0          0          0          0          0
TOTAL                 0          0          7          2          0          9
*****
```

Rule Development

Overview

This chapter explains what user need to know to write and compile custom rules for use during EMControl rule checking.

EMControl uses rules that have been written using the Cadence Advanced Rule Language (ARL). The source rule file (ARL file) has suffix `.arl`, and the compiled file has suffix `.rle`.

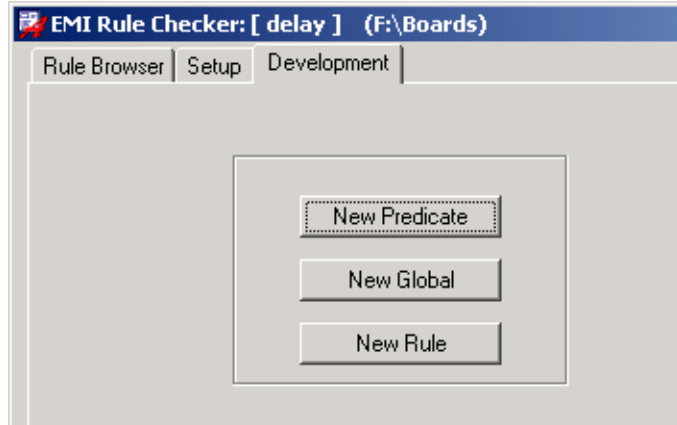
Cadence supplies a set of rules, but user can also write his own rules. For this, user may use the standard predicates and parameters or define new ones, as per requirement. Parameters are like variables, which have certain default values but you can also edit them. Predicates are analogous to functions in other programming languages.

Once you have written a rule and compiled it, you might want to make it accessible to the users in your site. For this, you need to make the parameters, the predicates used in the rule and the compiled rule file (`.rle`) visible.

- To make the rule file visible, The `.rle` file should be selected by [Setting Up Rule Files](#).
- To make the predicates visible, the mapping file (`.env`) containing the predicates used in the rule must be selected by [Setting Up Mapping Files](#).
- To make the parameters visible, the parameter file containing rule parameters in the rule must be selected by [Setting Up Rule Parameter Files](#).

The EMI Rule Checker dialog box provides the interface illustrated in figure [7-1](#).

Figure 7-1 EMI Rule Checker Dialog Box - Development Tab

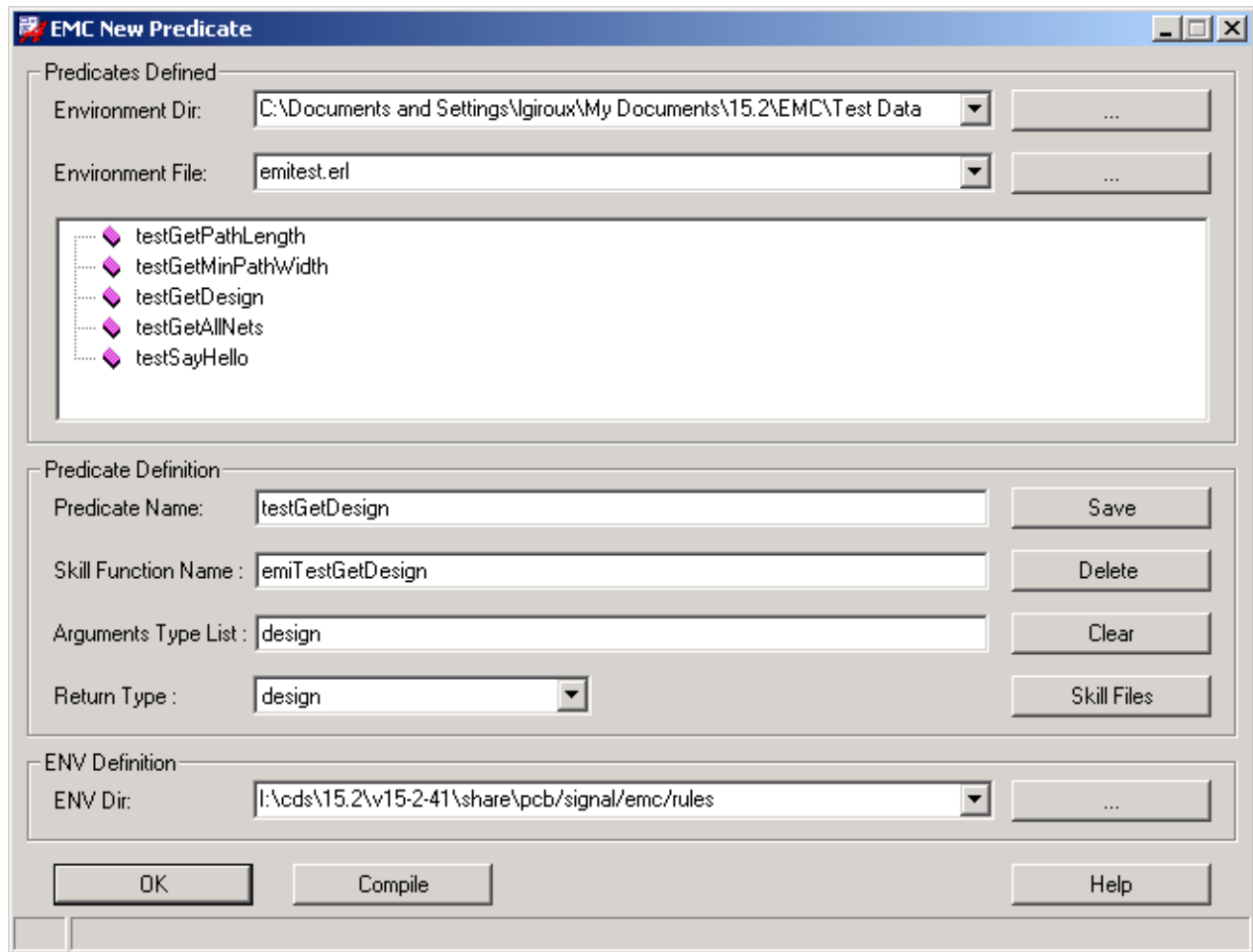


Click a button to:

- Develop New Predicates
- Develop New Parameters
- Develop New Rules

Develop New Predicates

Figure 7-2 EMC New Predicate Dialog Box



The dialog box is titled "EMC New Predicate". It is divided into three main sections: "Predicates Defined", "Predicate Definition", and "ENV Definition".

Predicates Defined: This section contains two input fields: "Environment Dir:" with the value "C:\Documents and Settings\lgiroux\My Documents\15.2\EMC\Test Data" and a browse button "..."; and "Environment File:" with the value "emitest.erl" and a browse button "...". Below these is a list box containing five items, each preceded by a purple diamond icon: "testGetPathLength", "testGetMinPathWidth", "testGetDesign", "testGetAllNets", and "testSayHello".

Predicate Definition: This section contains four input fields and four buttons. The "Predicate Name:" field contains "testGetDesign", with a "Save" button to its right. The "Skill Function Name:" field contains "emiTestGetDesign", with a "Delete" button to its right. The "Arguments Type List:" field contains "design", with a "Clear" button to its right. The "Return Type:" field contains "design" and has a dropdown arrow, with a "Skill Files" button to its right.

ENV Definition: This section contains one input field: "ENV Dir:" with the value "I:\cads\15.2\v15-2-41\share\pcb\signal\emc\rules" and a browse button "...".

At the bottom of the dialog box are three buttons: "OK", "Compile", and "Help".

EMControl User Guide

Rule Development

The following sections describe the options for the dialog box in figure 7-2:

<i>Environment Dir</i>	Specifies the path where your mapping files are located. You can use the <i>browse(...)</i> button to select the path or input a new path.
<i>Environment File</i>	Specifies the source mapping file (<code>.erl</code>) from the drop-down list, or the <i>browse(...)</i> button. You can also input a new mapping file with the suffix <code>.erl</code>
	The predicates defined tree control displays the predicates defined in the file.
<i>Predicate Name</i>	Denotes the current predicate name which is being edited.
<i>SKILL Function Name</i>	Denotes the SKILL function name which is mapped to the current predicate.
<i>Argument Type List</i>	Specifies the argument type list of the SKILL function.
<i>Return Type</i>	Specifies the return type of the SKILL function.
<i>Save</i>	Saves the current predicate definition to the mapping file.
<i>Delete</i>	Deletes the current predicate definition from the mapping file.
<i>Clear</i>	Clears the current predicate definition.
<i>Skill Files</i>	See Setting Up SKILL Files .
<i>Compile</i>	Compiles the source mapping file to an <code>env</code> file. The compiled <code>env</code> file has the <code>.env</code> suffix.
<i>ENV Dir</i>	Specifies the path where the compiled mapping file will be located. You can use <i>browse (...)</i> to select another one or input a new path in the field directly.
<i>OK</i>	Closes the dialog box.

The EMC New Predicate dialog box enables you to:

- [Add a New Predicate](#)
- [Delete a Predicate](#)
- [Edit an Existing Predicate](#)

Add a New Predicate

To add a predicate from the EMC New Predicate dialog

1. Select the environment (`.erl`) file where predicate will be saved.

The Predicates window displays a list of the predicates currently defined in the file.

2. In the *Predicate Definition* section, submit the details of the predicate to be added. Give its name, underlying Skill function, the arguments' types and its return type.

Note: The underlying Skill function needs to be loaded in Allegro PCB SI before the corresponding predicate is defined.

3. Click *Save*.

The specified environment file gets updated with the changes.

Note: A warning message is presented if you click *OK* or *Compile* before saving the predicate definition.

4. Click *Compile*.

The environment (`.erl`) file is compiled to produce the mapping file (`.env`).

Note:

- You need to have write permissions to add a predicate to an environment file.
- A warning message is presented if you click *Compile* before saving the predicate definition.

Delete a Predicate

To delete a predicate from the EMC Skill Predicate dialog

1. Select the environment file where predicates are to be deleted.

The Predicates Defined tree control displays the predicates defined in the file.

2. In the *Predicates Defined* section, select the predicate to delete.

3. Click *Delete*.

The specified environment file gets updated with the changes.

4. Click *Compile*.

The environment (`.erl`) file is compiled to produce the mapping file (`.env`).

Note: You need to have write permissions to delete a predicate from an environment file.

Edit an Existing Predicate

To edit a predicate from the EMC Rule Developer dialog

1. Select the environment (`.erl`) file where predicates are to be edited.

The predicates defined tree control displays the predicates defined in the file.

2. Select the predicate to be edited.

The details of this predicate appear in the *Predicate Definition* section

3. Edit the details of the predicate, the name of the predicate, its underlying Skill function, the arguments' types and its return type.

Note: The underlying Skill function needs to be loaded in Allegro PCB SI before the corresponding predicate is defined.

4. Click *Add*.

The specified environment file gets updated with the changes.

5. Click *Compile*.

The environment (`.erl`) file is compiled to produce the mapping file (`.env`).

Note: You need to have write permissions to add a predicate to an environment file.

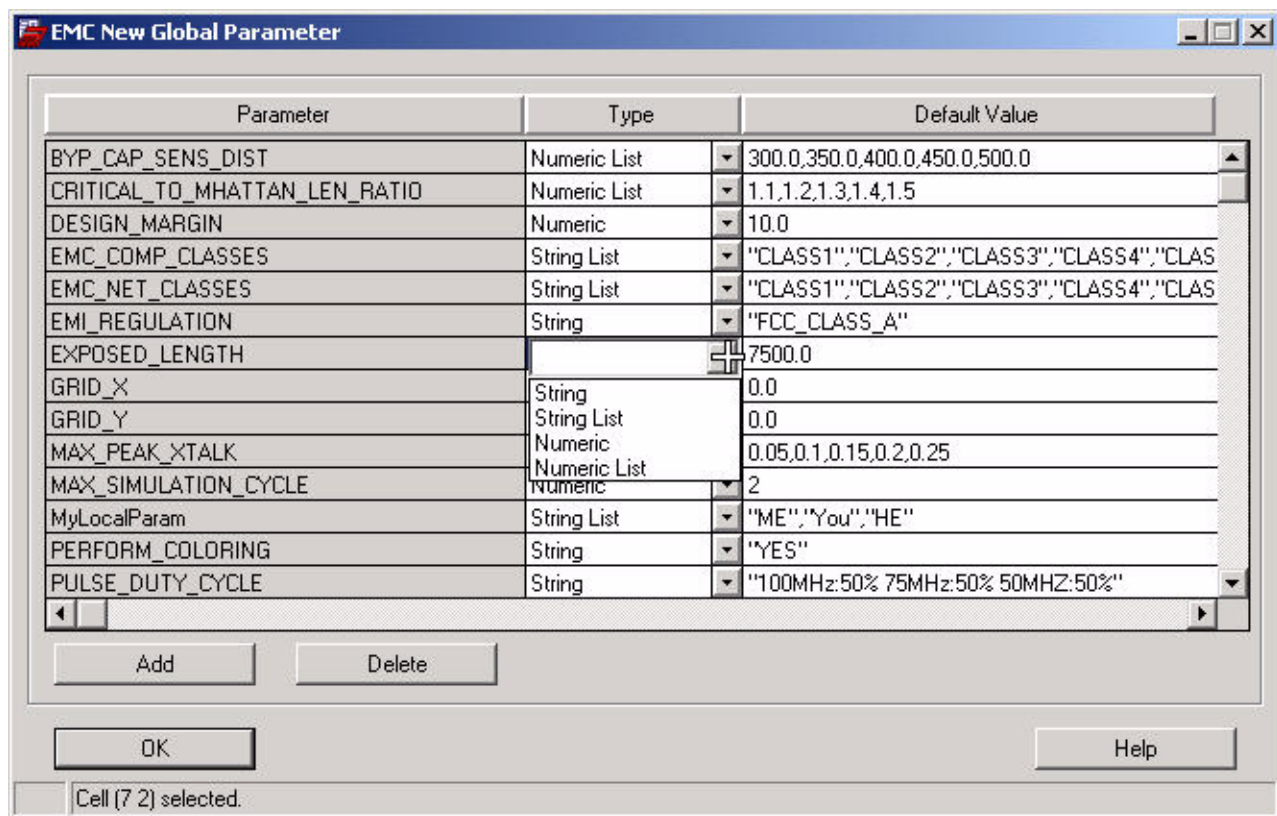
Develop New Parameters

EMControl enables users to add, delete, and edit global and rule parameters.

To access the global parameters

- Click the *New Global* button in EMC main window.

Figure 7-3 Develop New Parameter



The EMC New Global Parameter dialog box enables you to:

- Add a Global Parameter
- Delete a Global Parameter
- Edit a Global Parameter

To access the rule parameters, click the *New Rule* button and see [Create Rule Parameters](#)

Add a Global Parameter

To add a global parameter

1. Click *Add*.

A popup window is displayed for you to input the global parameter name, type and the default value. A new row appears in the parameter grid with the new parameter, its type, and its default value. See [Adding a Parameter](#).

2. Click *OK* to close the dialog box.

Delete a Global Parameter

To delete a global parameter

1. Select the parameter row to be deleted.
2. Click *Delete*.
3. Click *OK* to close the dialog box.

Edit a Global Parameter

To edit a global parameter

1. Select the parameter row to be edited.
2. Customize the parameter type by the type drop-down list.

The parameter type should be `STRING`, `STRING LIST`, `NUMERIC`, or `NUMERIC LIST`.

3. Customize the parameter default value.

Note: The default parameter value must use the correct type.

4. Click *OK* to close the dialog box.

Develop New Rules

EMControl enables user to create/customize their own rules.

To access the rule development form,

- Click *New Rule* button in EMControl main window.

Figure 7-4 Develop New Rules

EMC New Rule

Rule | Audit

Source Dir: D:\Cadence\PSD_14.2\share\pcb\signal\emc\rule_src

Source File: emc_bypass.arl

Rules Defined:

- bypass_cap_type
- bypass_fast_sw_trans
- bypass_drvr_rcvr_bidir
- critical_IC_loop_area
- critical_IC_3caps_C_2C_4C
- decouple_emc_regions
- fence_off_emc_regions

Rule Selected: critical_IC_3caps_C_2C_4C

Parameter File: emc_param.par

Parameter	Type	Default Value
CAP1_DIST_FROM_PWR_PIN	Numeric List	300.0,350.0,400.0,450.0,500.0
CAP2_DIST_FROM_PWR_PIN	Numeric List	200.0,250.0,300.0,350.0,400.0
CAP3_DIST_FROM_PWR_PIN	Numeric List	100.0,150.0,200.0,250.0,300.0

Add Delete Edit Compile Global

Compiled File Directory: D:\Cadence\PSD_14.2\share\pcb\signal\emc\rules

OK Help

The EMC New Rule dialog box enables you to:

- Create a Rule File
- Create Rule Parameters
- Compile a Rule File
- Create a Rule Help File
- Create an Audit Rule File

Create a Rule File

To create a rule file from the EMC Rule Developer dialog

1. Specify the ARL file path in the Source Dir.

You can use the *browse (...)* button to select a path or input a new one.

2. Specify the ARL file name in the Source File.

You can use the *browse (...)* button to select a file name or input a new one.

All rules in the source file will be listed in the rule tree view, and all rule parameter files will be listed in the *Parameter File* drop-down list.

3. Click *Edit*.

The file that you specify is opened in the text editor specified by the `EDITOR` environment variable. If a new rule file is being written, a new file opens.

4. When you finish working in the text editor, save the rule file and quit the editor.
5. Click the *Refresh* button to update the Rules display in the dialog box.

Create Rule Parameters

EMControl enables user to create the rule parameters.

To create a rule parameter

1. Select a rule parameter file from *Parameter File* drop-down list.
2. Select a rule from rule tree view.

All existing rule parameters with type and value are listed in parameter grid.

3. Click *Add* button to add a new parameter
4. Select one parameter then edit the type and value directly.
5. Select one parameter then click the *Delete* button to delete it.
6. Click the *Global* button to customize the global parameter.

See [Develop New Parameters](#) on page 95.

Compile a Rule File

After you edits and save the rule file, you can compile the file using the EMC Rule Development dialog box.

To compile a rule file

- Click the *Compile* button.

EMControl compiles the file and creates the compiled file in the specified *Compiled File Directory*. The file name extension of the compiled rule file is *.rle*.

In case of any errors or warnings, the EMC Compilation Log window comes up showing the errors and warnings encountered during the compilation process.

If the file is compiled correctly without any errors, the message “0 Messages” is displayed in the Console window, and the compiled file is saved in the path specified in the *Compiled File Directory* field. You can change the path by using the *browse (...)* button or by inputting a new path directly.

Create a Rule Help File

Cadence recommends that you create a help file for each rule that you write. You should include critical information that describes the function of the rule in the help file.

You view the help information through the EMC *Rule Browser* window and use it to:

- Determine whether a particular rule is relevant to the current design-checking task.
- Learn what properties and other variables are required by the rule.
- Attach relevant properties to objects in the design or modify default variable values, if needed.

To create a help file

1. Select a rule from rule tree view.

The rule name you selected is displayed in the *Rule Selected* field.

2. Click *Rule Help* button.

A text editor displayed with help information for you to create the help file.

Note: A help file must have the name `<rule_name>.hlp`, where `rule_name` is the name of the rule (or rule file) for which this file provides help. The required extension is `.hlp`.



Tip

To make the file available for display in EMControl, place it in an appropriate directory location. You can do one of the following:

- ☐ Set the environment variable `EMC_HELP_PATH` to the path location of the help file.

For example:

```
setenv EMC_HELP_PATH /usr1/abc/pcb/emc/help
```

- ☐ Place the file in the following default directory for help files:

```
<your_install_dir>/share/pcb/signal/emc/help
```

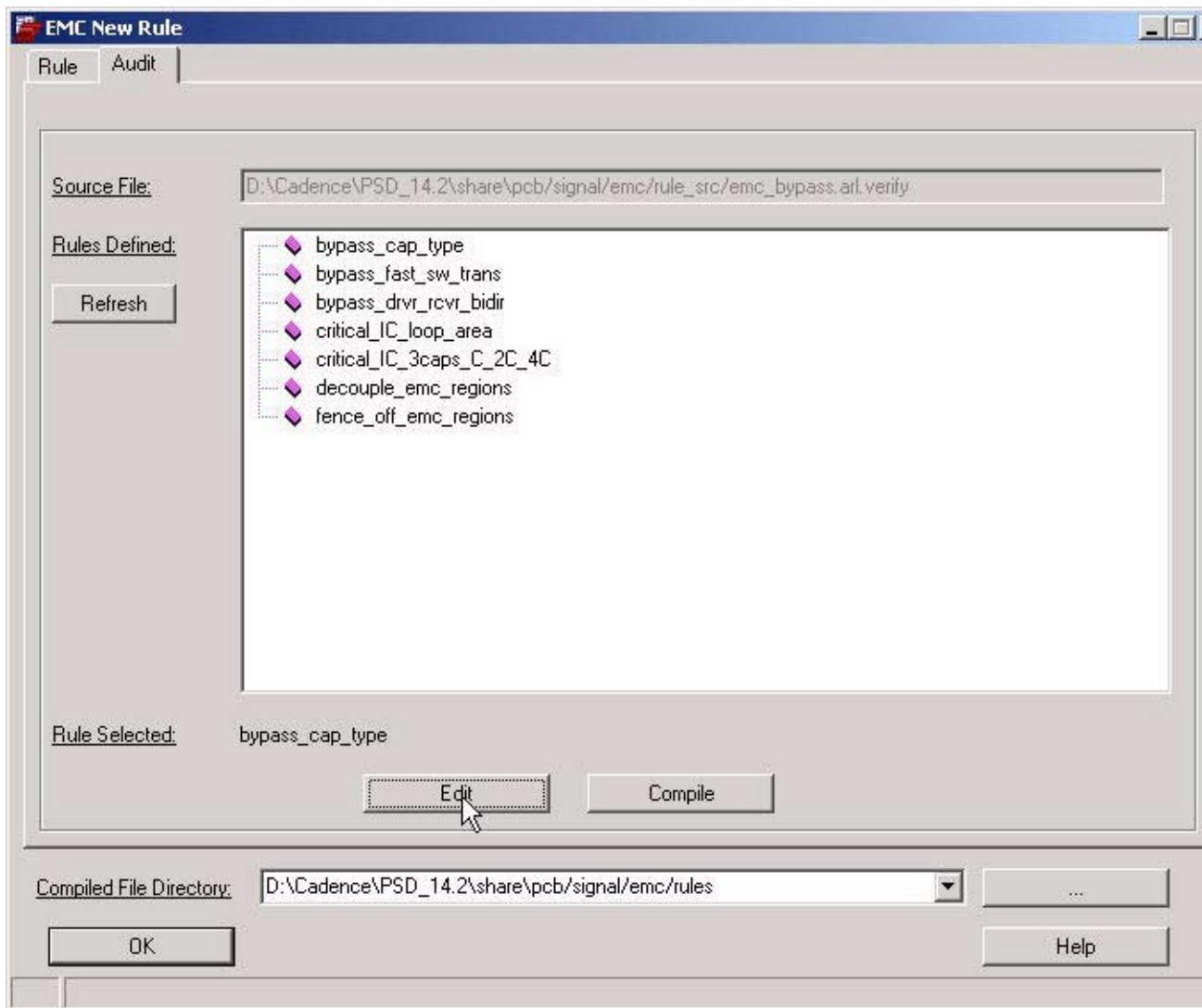
EMControl looks for help files in this directory when `EMC_HELP_PATH` is not set.

- ☐ Place the help file in the compiled rule file paths specified in [Setting Up Rule Files](#) on page 33. The default help file path is the same path for this rule file compiled path.

Create an Audit Rule File

EMControl also enables you to create an Audit rule file while you develop a rule. An audit rule is used to verify whether the properties and variables and other design configurations are set correctly before you run rule checking.

Figure 7-5 Create an Audit Rule File



To develop an audit rule

1. Specify the source rule file in rule development tab.

The audit rule file is automatically displayed in *Source File* in the Audit tab, with the suffix `.arl.verify` in the same location of rule file. All of the rules in the audit rule file are displayed in the audit rule tree view.

2. Click the *Edit* button to edit the audit rule file.
3. Save the audit rule file, then click the *Refresh* button in the Audit tab to update the Rules display in the dialog box.
4. Click the *Compile* button in the Audit tab to compile the audit rule file.

If no errors found, the compiled audit rule file will be in the same directory as the compiled rule file, with the `.rle.verify` suffix.

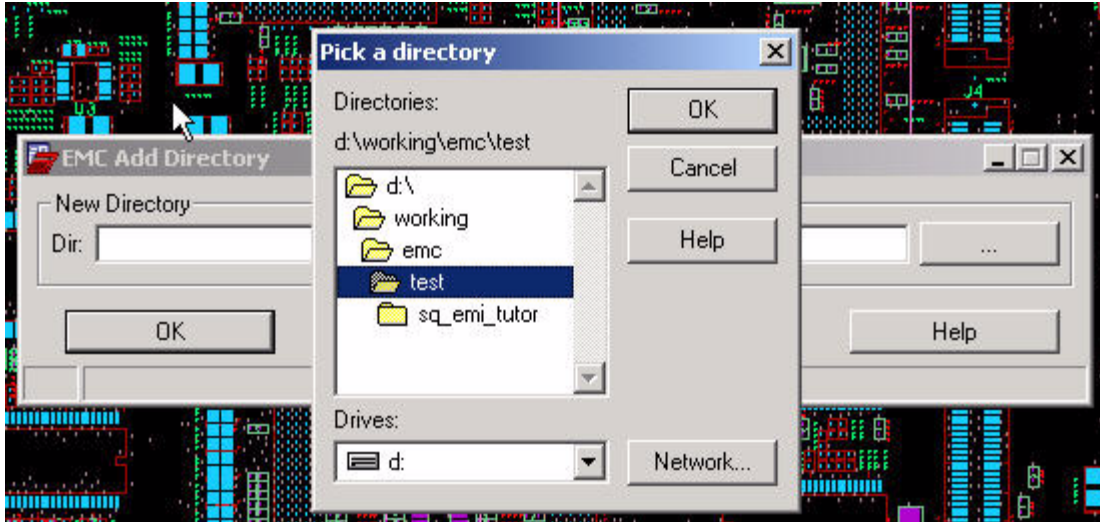
Running new rules

Once you have written and compiled a new rule file, the compiled rule file will be included in the *Rule Browser* Window automatically. You can also select or deselect the rule files by Setting Up Rule Files.

See Chapter 5, "Performing EMControl Rule Checking" to run this rule.

Adding a Directory

Figure 7-6 Adding a Directory

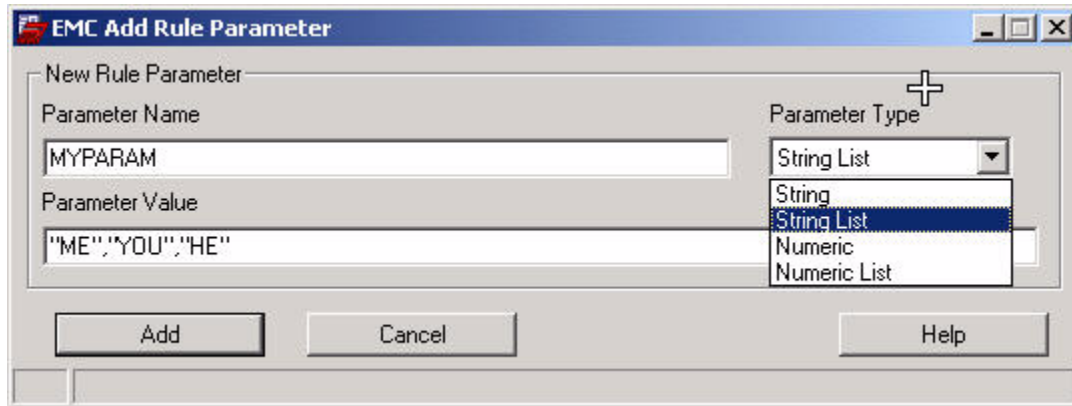


To add a directory

1. Input the path in the text field.
- or -
2. Use the *browse (...)* button to select.
3. Click *OK* to close the dialog and return the directory.
4. Click *Cancel* to close the dialog and discard the input.

Adding a Parameter

Figure 7-7 Adding a Parameter



To add a rule or global parameter

1. Enter a name in the *Parameter Name* text box.
2. Click the *Parameter Type* down arrow and select a data type from the drop-down menu.
3. In the *Parameter Value* text box, enter a default value appropriate for the parameter type selected.
4. Click *OK*.

The rule or global parameter is added as specified.

ARL Training Guide

Introduction

EMControl is a Allegro Design Entry HDL Rules Checker based tool. It consists of the Rules Checker core engine Advanced Rule Checker (ARC), the EMControl rule set, and the environment which defines the interaction between ARC and Allegro PCB.

ARC is an execution engine which executes rules written in the Advanced Rule Language (ARL). In order to utilize the maximum potential of ARL-based tools like EMC, it may be necessary to write customized rules that are focussed on a customer's unique design environment. With the recent enhancements to the core ARL engine, many more customer rule requests are now possible. This training guide will start with the basics of the ARL language and then move into the intricacies of more complex rule-writing.

Language Highlights

ARL is a language suitable for expressing design rules. The major features of the language which distinguish it from other general purpose languages are highlighted below.

- Simple Condition - Action semantics.
- Implicit looping through design objects
- Ability to accept parameters at run-time. This allows the user to customize the rule according to his/her design needs
- Ability to specify the objects to be highlighted by the markers utility.
- Automatic iteration over function (predicate) arguments.

EMControl Objects

ARL is currently used by many different tools at Cadence. The Rules Checker core engine, which executes rules written in ARL, do not have any internal information about the design

on which it is checking the rules. (e.g. The Rules Checker core engine does not have any knowledge of Allegro PCB elements).

Rules Checker understands about the target design (e.g. Allegro PCB board) from the environment file provided. The environment file defines the objects in the design that can be manipulated by ARL. It also defines a set of functions (called predicates) which is used to interface with Allegro PCB.

The EMControl environment contains 8 different objects:

Design	The board design upon which EMC is run.
Component	Components in the design.
Net	Nets in the design.
Pin	Pin elements in the design.
Via	Via elements in the design.
Shape	Shape, rectangle and filled-rectangle elements in the design.
Polygon	Polygons in the layout X_Y plane.
Drc	DRCs in the design.

Any element in the design which belongs to the above types can be directly referenced in a rule.

EMControl Predicates

Predicates facilitates the interaction between ARC and Allegro PCB. They are analogous to functions in other general purpose languages. A predicate takes one or more arguments and returns a list as result.

e.g.

component

`component(design)`

The predicate `component()` takes a design object and returns all the components in the design.

pin

```
getPin(component)
```

The predicate `getPin()` takes a component object and returns the list of pins of the component.

value

```
name (net)
```

The predicate `name()` takes a net object and returns a string representing the net name.

Predicates internally call C or SKILL functions which compute the desired result by accessing the design database. The complete list of predicates for EMControl is given in the user guide.

Getting a Feel for the Language

In order to get an introduction to ARL we will dissect a few simple rules that demonstrate the fundamentals.

Example 1

Consider a simple case. Suppose you want to list all the nets in a design individually. In pseudo-code we could write it like:

```
for each net in the design
  print the name of the net
endfor
```

In ARL it is coded as:

```
RuleDefine                                /* start of rule declaration */
  Rule print_net                          /* start of rule <rulename> */
    net1                                /* condition statement */
    Message(Info,net1,"?net1")/* message statement */
  EndRule                                /* end of rule */
EndRuleDefine                             /* end of rule declaration */
```

Basic Language Constructs

There are various parts to a rule. The keywords `RuleDefine` and `EndRuleDefine` indicate the start and end of the rule declaration area of a text file. The keywords `Rule` and `EndRule` are part of the rule declaration syntax that specify the logical breakup of separate rules. Every rule must be enclosed by its own `Rule` and `EndRule` set of keywords. This differs from the `RuleDefine` and `EndRuleDefine` which are only required once per text file, but allowed as often as once per rule. A rule name can consist of alphanumeric characters, the underscore and the `$` sign. The first character cannot be a number.

Note: The language is case insensitive, so `ruledefine` and `RuleDefine` are equivalent. The general syntax that you will see followed here will mix upper and lower case in an attempt to clarify the intended meaning of the Function.

Inside the rule declaration constructs exist the remaining 2 sections of the rule. These sections are the rule conditional statements and the message statement. As you may predict, the conditional statements determine whether the rule reports the information that is supplied in the Message statement. In the above example the conditional section consists of 1 line, `net1`. As you will see later in this section, the conditional section is normally a logical grouping of statements that get evaluated to `True` or `Nil` (`False`) according to the predicates called and the data the predicates are supplied.

Variables and Base Objects

Every Rules Checker rule contains exactly 1 Base Object. A base object of a rule is a variable which cannot derive its value from any other variable in the conditional section of the rule.

Example 1:

```
RuleDefine
  Rule base_obj
    component1 := component(design1) AND
                /*component1 is derived from design1 */
    net1 := net(design1) /* net1 is derived from design1 */
    Message(Info,"?design1");
  EndRule
EndRuleDefine
```

In the above example `design1` cannot derive its value from any other variable. Therefore the base object of the rule is `design1`.

Example 2:

```
RuleDefine
  Rule base_obj2
    component1 := getCompsConnToNet(net1) AND
                /* component1 is derived from net1 */
    component2 := hasProperty(component1, "EMC_COMP_TYPE")
                /* inst2 is derived from inst1 */
    Message(Info,component2 ,"Components having prop EMC_COMP_TYPE are
?component2");
  EndRule
EndRuleDefine
```

In example 2 the base object is `net1` as `component1` is derived from it and `component2` is derived from `component1`.

The base object also determines the number of times the rule is executed on the existing design. The EMC environment contains seven different base objects. The Net object is one

from this group. Since we have selected Net as our base object, this rule will be executed by the Rules Checker engine 1 time per net in the design. Therefore the maximum number of error messages generated by the rule is equal to the total number of nets in the design. For example, if a design has 10 nets then this rule may result in 0 to 10 messages.

Base Objects and Implied Looping

Base Objects constitute 1 main area of implied looping. In this rule we have not explicitly told the engine to iterate over all nets. But by selecting Net as our base object the engine understands that it must evaluate the rule for every net in the design. In order to understand the implied looping execution, think of the engine working like the following pseudo-code:

```
net1 := First net in the design
While (!end of net list) do
{
    Execute the rule
    print message if condition evaluates "True"
    net1 := Next net in design
}
```

As you can see, the number of messages printed can be anywhere between 0 and the number of nets in the design.

Later in the design we will investigate the rationale behind selecting a base object. In the meantime here is another example to reinforce the idea of Base Object looping

Example:

```
List all the nets of the design in one message.
foreach design
    print all nets in design
endfor
RuleDefine
    Rule print_net11
        net1 := net(design1)
        Message(Info,net1,"?net1");
    EndRule
EndRuleDefine
```

In this example the base object is Design. Within an execution in the EMC environment there is only 1 item that can be assigned to the Design object. This item is the board that was requested as the target of the EMC run. Since there is only 1 value for the base object, this rule will only be executed 1 time.

Predicate Calls

A predicate call in the Rules Checker language is similar to a function call in other languages. It takes 1 or more arguments and returns a value. Before writing or understanding rules in a

specific environment it is necessary to have a list of predefined predicates available for that environment.

The rule in example 2 calls the function, Net(), and passes it the base object variable, Design. By looking up the definition of the Net() function you find out that it accepts a parameter of type Design, and returns a list of nets in that design. Therefore after the execution of the Net() predicate, the variable, net1, will consist of a list of nets in the design.

Variable Typing

This leads us into the discussion of variable types and how variables are assigned and compared. The Rules Checker variables are used to store values that you compute. There are two types of variables -

- Object Variables: Holds an object (objects are defined in the EMC environment)
- Non Object Variables: Assigned numerical values, strings etc.

Object variables are identified by their names. Object variable names must start with the object name that they will be assigned. So in Example 1 the return value of the predicate, Net(), is a net object, therefore the variable name must begin with "net". Any string may be appended to the object type to create a variable name. Some valid Net variables follow:

"net1", "net2", "net_VCCI", "net_ground", "net_all" etc.

The rule compiler will report errors in the compilation of the rule if this variable naming requirement is not met.

ARL Operators

ARL provides a rich set of arithmetic and logical operators. Logical operators are used to join multiple boolean results to create complex logical expressions. Examples of these operators are AND, OR, and XOR. From [Exercise 3](#) you can predict, this rule will only report information to the user if the total number of signals in the design equals 3. Each of the conditional statements must evaluate to a TRUE, or non-null value since they were joined together with the AND operator. If not the Message construct will not be executed.

The following set of operators can be used in expressions. These operators are grouped according to precedence, from highest to lowest.

Operator	Description
isNull, abs,exp	negate, isInputArgumentNull, absolute, exponent

EMControl User Guide

ARL Training Guide

Operator	Description
*, /, mod, rem	multiply, divide, modulus, remainder
+, -	add, subtract
and, or, xor	logical operators
{item, item, ...}	one of a list of items
==, /=, <, <=, >, >=	equal to, not equal to, less than, less than or equal to, greater than, greater than or equal to

Exercises

Exercise 1

Consider a design with 10 nets and 2 components. How many messages will the following rules generate?

1.

```
RuleDefine
  Rule print_net
    net1
    Message(Info, "?net1");
  EndRule
EndRuleDefine
```

2.

```
RuleDefine
  Rule print_component
    component1
    Message(Info, "?component1");
  EndRule
EndRuleDefine
```

3.

```
RuleDefine
  Rule print_all
    net1 := net(design1) AND
    component1 := component(design1)
    Message(Info, "?inst1 ?sig1");
  EndRule
EndRuleDefine
```

Exercise 2

Consider a design which has 10 components and 0 nets. How many messages will the above rules generate now?

Exercise 3

What is the basic object in the following rule?

1.

```
Rule ex1
  component1 := getCompsConnToNet(net1) and
  net2 := getNetOf(getPin(component1)) and
  vall := count(net2)
  Message(Error, "?net2 ?vall");
EndRule
```

Exercise 4

Write a rule to list all the components in the design individually. The number of messages should equal the number of the components in the design.

Exercise 5

Write a rule to list all the components in the design in one message.

Exercise 6

Write a rule to print a count of all components and nets in the design

List Manipulation

The Rules Checker rule language provides lists as its only form of data structure. Every variable type in the language can be used to create a list of 1 or more elements. Up to this point in the training guide we have demonstrated only basic list creation techniques using the implied looping over a list of base objects by the Rules Checker engine. The language also supports other basic forms of list creation. This section will investigate these language characteristics and the operations supplied for manipulating lists.

What are Lists

A Rules Checker list is a collection of one of the following:

- Object Elements
- Non Object Elements

Lists are homogeneous lists i.e. they can contain only one type of object. For example, you cannot have a list that consists of components and nets. Nor can you have a list that contains strings and integers.

Example 1:

```
RuleDefine
  Rule list_def
    component1 := component(design1) AND
    val_type := getPropertyValue(component1, "EMC_COMP_TYPE")
    Message(Info,"Components ?component1 , comtypes : ?val_type");
  EndRule
EndRuleDefine
```

In the above example `component1` is a list of components (object elements) and `val_type` is a list of non-object elements (strings in this case). Note that `getPropertyValue()` is passed a list of components. The predicate is called for each component in the list `component1` and the results are put in another list which gets assigned to `val_type`.

The lists are created by predicates. In the previous section we discussed how lists of Base Objects were created by the Engine for use in evaluating each selected rule. This is 1 type of implied list manipulation that does not require any special understanding by the programmer. It should be understood that the length of the Base Object list determines the number of times the rule is executed on the design database. This base object list can be displayed by using the `Message()` command on every item in the list. In the following example `net1` is assigned each element of the base object list, one at a time, for each execution of the rule.

Example 1:

```
RuleDefine
  Rule print_net
    net1
    Message(Info,net1,"Name : ?net1");
  EndRule
EndRuleDefine
```

List Manipulation Routines

The difference between a list routine and other predicates is that while the list routines operate on the full list, other predicates operate on one element of the list at a time.

Example 1:

```
RuleDefine
  Rule ListEg4
    shapes := shape(design1) AND
    shapes_rect := isRectangle(shapes) AND
    shapes_no_rect := remove(shapes, shapes_rect) AND
    val := count(shapes_no_rect)
    Message(INFO, shapes_no_rect, "No. of non-rectangle
                                shapes = ?val");
  EndRule
EndRuleDefine
```

The intent of this rule is to use `count()` to compute the count of all non-rectangle shapes in the design.

In the above rule `isRectangle()` will be called implicitly once per element in the list `shapes`. But `remove()` and `count()` being list routines will be called only once.

For example consider a design with 10 shapes. In this case `isRectangle()` will be executed 10 times (once for each element in the list `shapes`) while `remove` and `count` will be executed only once in this line.

(Note: There is an interesting bug in the above rule! If the design does not have any rectangle elements, the rule will not report non-rectangle shapes. In this case, the rule does not give a violation)

Here are some of the predicates which operate on lists. Each of the following predicates except `append` returns the result. The input list is not modified. So in the following discussion saying `remove(list1, list2)` removes the elements in `list2` from `list1`. This means that the `remove` predicate returns `list3`, which contains all elements of `list1` which are not contained in `list2`. As an exception, `append(list1, list2)` attaches `list2` to `list1`. So it modifies its input argument. Here is a table of the basic ARL list manipulation routines:

Function	Description
<code>car(list)</code>	Returns the first element
<code>cdr(list)</code>	Returns the list without the first element
<code>nth(list, i)</code>	Returns the <i>i</i> th element
<code>remove_i(list, i)</code>	Returns the list without the <i>i</i> th element
<code>min(list)</code>	Returns the smallest element of the list of non objects
<code>max(list)</code>	Returns the largest element of the list of non-objects
<code>sum(list)</code>	Returns the sum of the list of integers/floats
<code>last(list)</code>	Returns the last element of the list

EMControl User Guide

ARL Training Guide

Function	Description
<code>index(list, element)</code>	Returns the index into the list if element exists
<code>count(list)</code>	Returns the length of the list
<code>concat(list1, list2)</code>	Returns a new list that is the concatenation of list1 and list2
<code>union(list1, list2)</code>	Returns a unique list that is the union of list1 and list2
<code>intersection(list1, list2)</code>	Returns a unique list that is the intersection of list1 and list2
<code>remove(list1, list2)</code>	Returns list1 such that it contains no elements of list2
<code>unique(list1)</code>	Returns a list that contains no duplicate items
<code>sort(list1)</code>	Returns a sorted list
<code>concat_str(list1)</code>	Returns a string value that is formed by concatenating the strings from list1
<code>append(list1, list2)</code>	Modifies list1 by appending the values from list2

Here are some examples of how these list manipulation functions work:

Assume a list of Vowels like the following.

Vowels = (a e i o u)

Note that here the parentheses are not recognized as a syntax to specify lists. They are used for clarification only.

What will be the output of the following?

- ☐ `car(Vowels)` (a)
- ☐ `cdr(Vowels)` (e i o u)
- ☐ `nth(Vowels 3)` (i)
- ☐ `car(cdr(Vowels))` (e)
- ☐ `cdr(car(Vowels))` nil
- ☐ `index(Vowels, "a")` (1)
- ☐ `index(Vowels, "p")` nil
- ☐ `count(Vowels)` (5)

Note: A single element return value is these examples actually constitutes a list containing one element.

Foreach Construct

The Foreach construct provides a mechanism for iterating over all members in a list. A foreach construct is generally used to extract a subset of a list. Here is the syntax of the foreach construct:

```
foreach(list, condition)
```

Note: The return value of any Foreach construct is always a new list. Since the Foreach construct is really just another conditional line in a rule, it evaluates to true if the list that is returned is non-null.

The first parameter of the Foreach construct is a list. This list is also used inside the condition statements as the variable. An example best explains this.

Example:

```
List all components that have more than two pins.
RuleDefine
    Rule comps_pins
        component1 := component(design1) AND
        component2 := foreach(component1,
                                count(getPin(component1)) > 2
                                )
        Message(ERROR, "Components that have more than two pins are",
                "?component2");
    EndRule
EndRuleDefine
```

In this example you can see that component1 is the list that is used in the Foreach construct. It is also the variable name used inside the condition. component2 will be a subset of component1. It will contain those elements of component1 that satisfy the condition section of the foreach construct.

Saving Intermediate Results Within a Foreach construct

Often you would like to collect the intermediate results while doing computation inside a foreach construct. This is especially helpful for debugging rules as you develop them. The value of the variable used inside a foreach construct will change for each iteration of the loop. The append() predicate provides a convenient way to accumulate the results in one variable.

Suppose you want to find all cline segments over a shape that intersect with the shape boundary at the non-90 degree intersection angles of cline segments over a shape. The rule for this will be something like:

EMControl User Guide

ARL Training Guide

```
RuleDefine
Rule append_usage
    nets_clineseg := getClineSegsInArea(getPolygon(shape)) AND
    nets_clineseg_non90 :=
        foreach(nets_clineseg ,
valAngle := getIntersectAngle(shape, nets_clineseg) AND
valAngle /= 90 AND
        )
        Message(INFO, shape, "The none-90 degree cline segs",
            "are", "?nets_clineseg_non90");
EndRule
EndRuleDefine
```

Now if you want to find all the non-90 degree angles aswell, you can use `append()` in the above rule as illustrated below:

```
RuleDefine
Rule append_usage
    nets_clineseg := getClineSegsInArea(getPolygon(shape)) AND
    nets_clineseg_non90 :=
        foreach(nets_clineseg ,
valAngle := getIntersectAngle(shape, nets_clineseg) AND
valAngle /= 90 AND
            append(val_non90_angles , valAngle)
        )
        Message(INFO, shape, "The none-90 degree cline segs",
            "are", "?nets_clineseg_non90");
EndRule
EndRuleDefine
```

If we print `valAngle` in the message, we'll get the value for the last iteration. The `append()` predicate accumulates all non-90 degree angles in `val_non90_angles`. `Append()` differs from other predicates in some characteristics. They are -

- `Append()` creates its first argument. So it must be an identifier. Passing something else, e.g. a predicate call to `append()` will result in compilation error.
- As a corollary to 1, `append` can create a variable only once. Trying to use `append` twice on a variable will result in an error. The same variable cannot be used as the first argument of two different appends. Please note that iterating over the same `append` call is allowed using `foreach`, as in the above example. Actually that is what gives `append` the power!

If Construct

ARL provides an If conditional construct. Like the `Foreach` construct, `If` construct also returns a value which may be assigned to variables.

The syntax of the `If` construct is:

```
if(condition, expression1, expression 2)
```

If the condition evaluates to TRUE expression 1 is returned. Otherwise expression 2 is returned. The following example illustrates the use of If:

```
RuleDefine
  Rule If _prop
    valProp := if(hasProperty(component1, "EMC_COMP_TYPE"),
                  getPropertValue(component1, "EMC_COMP_TYPE"),
                  "NO_PROP"
                )
    Message(INFO, "Component ?component1 have comptype ?valProp");
  EndRule
EndRuleDefine
```

If the component has the `EMC_COMP_TYPE` property, `valProp` is assigned its value. Otherwise `valProp` gets the value `NO_PROP`.

expression1 and expression2 can contain assignments as well. In such cases the same set of variables should be bound in both expressions.

Exercise

Assuming `component1` is a list of components and `net1` is a list of nets in the design. Which of the following conditions are valid

1. `component1 = concat(component2, net1)`
2. `val2 := concat("a", "b")`
3. `val3 := concat("a", 1)`
4. `component1 := component(design1) and`
`net1 := net(design1) and`
`val1 := name(component1) and`
`val2 := name(component1) and`
`val3 := concat(val1, val2)`
5. `component1 := net1`
6. `val1 := 2 and`
`component1 := component(design1) and`
`val1 == component1`

Dissection of an existing rule

In this section we will discuss the implementation of some of the existing EMC rules. The source code of all the rules will be available in the customer installation. Understanding the existing rules will help the user to get a better insight into the ARL language and its power in specifying design rules. It will also help the user to make modifications in the rules so that they can be tailor-made to suit his/her EMC needs.

Rule critical_net_via_count

Let us first look at the simple routing rule critical_net_via_count. The rule checks that all critical nets have not more than EMC_VIA_COUNT vias on them.

EMC_VIA_COUNT is a parameter used in this rule which specifies the maximum number or vias allowed for each class.

```
#define EMC_VIA_COUNT "7 8 9 10 11"

RuleDefine
Rule critical_net_via_count

valClass := upperCase(getPropertyValue( net1, EMC_CRITICAL_NET)) AND
/* Get the net class into valClass. Only critical nets get selected */

valPosClass := upperCase(getAllSubstrings( EMC_NET_CLASSES, {""," "}, {" ", ""}))
AND
/* Get the list of all net classes in valPosClass */
valIndex := index(valPosClass, valClass) AND/* Get the index of the selected net's
class in valPosClass */

valViaCntAll := atoi(getAllSubstrings( EMC_VIA_COUNT, {""," "}, {" ", ""})) AND
/* Get the list of all via counts specified in parameter EMC_VIA_COUNT */

valViaCnt := nth(valViaCntAll, valIndex) AND
/* Get the via count corresponding to the selected net's class .
For example a CLASS3 net will have valViaCnt = 9*/

vall := count( getViasOnNet( net1 ) ) AND
/* Get the via count for the net into vall */

vall > valViaCnt
/* Compare if the via count vall is greater than the maximum allowed number
valViaCnt */

Message( ERROR,
    net1,
    "Check the number of vias per net.", "\n\nThe number of vias per net should
    be kept at a",
    "minimum in order to improve reliability. The maximum number of ",
    "vias allowed = ?valViaCnt, vias found on net ?net1 = ?vall "
);

/* Report the violation. Highlight the net and report the number of vias found */
```

```
endRule
endRuleDefine
```

Rule conn_in_low_freq_regions

Now we will study the implementation of a more complex placement rule conn_in_low_freq_regions. The rule checks for 2 things.

- All connectors should be placed in the lowest frequency regions
- A lowest frequency region containing a connector should not have any higher frequency components placed in it.

```
RuleDefine
```

```
Rule conn_in_low_freq_regions
    /* CHECK 1 */
    isEmcRegion(shape_emc_region) AND
    /* Select emc regions only */

    valRegionClass := upperCase(getPropertyValue( shape_emc_region,
        EMC_CRITICAL_REGION)) AND
    /* Get the EMC_CRITICAL_REGION property value for the selected region */

    valAllClasses := getAllSubstrings( EMC_COMP_CLASSES, {""," "}, {" ", ""})
    AND
    /* Get the list of all EMC region classes into valAllClasses */

    valLowestFreqClass := last(valAllClasses ) AND
    /* Get the lowest frequency class */

    valRegionClass /= valLowestFreqClass AND
    /* The rule proceeds further only if the EMC region is not of lowest frequency
    */

    valLayer := if( getSubclass(shape_emc_region) == "BOTTOM_ROOM" ,
        "BOTTOM", "TOP") AND
    /* get the layer (TOP or BOTTOM) of the EMC region */
    components := getCompInArea(getPolygon(shape_emc_region)) AND
    /* get all the components in the area of the EMC region */

    components_region := foreach(components,
        getLayer(components) == valLayer) AND
    /* Select only those components which are in the same layer as that of the
    EMC region */

    components_connector := foreach(components_region,
        isConnector(components_region))
    /* Find out all the connector components in the EMC region */

    Message(WARNING, shape_emc_region components_connector,
        "Checks for connectors in Low Frequency Region",
        "\nConnectors should be placed in the lowest frequency region.",
        "Connector(s)\n\t?components_connector",
        "lies in region ?shape_emc_region of frequency class ?valRegionClass.",
        "This connector should be placed in a ?valLowestFreqClass region"
```


EMControl User Guide

ARL Training Guide

```
);

/* Report violation. Highlight the EMC regions and offending connectors */
/* CHECK 2 */
isEmcRegion(shape_emc_region) AND
/* Select emc regions only */

valAllClasses := getAllSubstrings( EMC_COMP_CLASSES, {""," "}, {" ", ""}) AND
/* Get the list of all EMC region classes into valAllClasses */

valRegionClass := upperCase(getPropertyValue( shape_emc_region,
EMC_CRITICAL_REGION)) AND
/* Get the EMC_CRITICAL_REGION property value for the selected region */

valRegionClass == last(valAllClasses) AND
/* The rule proceeds further only if the EMC region of lowest frequency */

valIndex := index(valAllClasses, valRegionClass) AND
/* Get the index of the EMC region class in list valAllClasses */

valLayer := if( getSubclass(shape_emc_region) == "BOTTOM_ROOM" ,
"BOTTOM","TOP") AND
/* get the layer (TOP or BOTTOM) of the EMC region */

components := getCompInArea(getPolygon(shape_emc_region)) AND
/* get all the components in the area of the EMC region */

components_region := foreach(components,
getLayer(components) == valLayer) AND
/* Select only those components which are in the same layer as that of the
EMC region */

components_connector := foreach(components_region,
isConnector(components_region)) AND
/* Find out all the connector components in the EMC region */

components_high_freq := foreach(components_region,
valCompClass := upperCase(getPropertyValue(components_region,
EMC_CRITICAL_IC)) AND
index(valAllClasses, upperCase(valCompClass)) < valIndex )

/* Get a list of all higher frequency components in the EMC region. This is
found out by comparing the indices of the component class and region class
*/

Message(WARNING, shape_emc_region components_high_freq,
"Checks for Higher Frequency components in Low Frequency Region",
"\nLowest frequency regions containing connectors should not contain",
"any higher frequency components.\n",
"The lowest frequency region ?shape_emc_region containing
connector(s)",
"\t?components_connector",
"contain the following higher frequency component(s):",
"\t?components_high_freq"
);

/* Report violation. Highlights the lowest frequency EMC region and the higher
frequency components contained in it */
```

```
EndRule  
EndRuleDefine
```

Laboratory Exercises

Basic Rules

Exercise 1

Write a rule to report all nets having EMC_CRITICAL_NET as CLASS1. The base object should be net.

Exercise 2

Keeping the same base object, modify the rule in problem 1 so that it reports all nets having EMC_CRITICAL_IC as CLASS1 and connected to a connector.

Exercise 3

Write a rule to report the number of pins and number of vias on a net.

Exercise 4

Modify the rule in exercise 2 to list the connector names also.

Exercise 5

Write a rule which for each IC component lists all other IC components which are within a distance DIST and is connected to the component. (DIST is a parameter used in the rule).

Modifying existing EMControl rule (conn_in_low_freq_regions)

Some times the user may want to make minor changes in the rule logic to adapt it to suit his/her design guidelines. Here is an exercise which captures this.

Exercise 1

Modify the `conn_in_low_freq_regions` rule so that the rule reports a violation whenever a higher frequency component is found in a region. (eg: CLASS1 component in CLASS2 region should report an error)

Exercise 2

Modify the `conn_in_low_freq_regions` rule so that there is an exact match between component class and region class. For example the rule should give a violation if a non-CLASS1 component is found in a CLASS1 region.

Custom EMC rule writing

Exercise 1

Write a rule which checks for the capacitance per unit area between power and ground planes of a board. The rule should give a message if the capacitance exceeds the parameter `LAYER_CAPACITANCE`. Assume the dielectric constant of the dielectric material to be 4.5. The default value of `LAYER_CAPACITANCE` is 100 f Farad/square mil

EMControl User Guide

ARL Training Guide

EMControl Rules

Overview

This appendix describes the rules included with EMControl. These rules are grouped in the following categories:

- [Placement Rules](#)
- [Bypass Rules](#)
- [DC Routing Rules](#)
- [Signal Routing Rules](#)
- [Signal Quality Rules](#)
- [Default Rules](#)
- [Advanced Rules](#)

Rule descriptions in this appendix appear alphabetically grouped within the categories described above. Each rule description contains the following information:

- A brief description of the rule
- Properties used by the rule
- Variables used by the rule
- Information about data (such as properties and variables) required by the rule
- The default severity level of the rule

The EMC expert at your site can best determine when and where to use each rule to check your design.

The uncompiled, ASCII versions of the rules are contained in the following files:

- `emc_placement.ar1`

- `emc_bypass.arl`
- `emc_dc_route.arl`
- `emc_sig_route.arl`
- `emc_sig_qual.arl`
- `emc_default_rules.arl`

The path locations for these files are provided in [Chapter 2, “Installing and Accessing EMControl.”](#). Information on editing these rule files and writing new rules is contained in [Chapter 7, “Rule Development.”](#).

Many rules in this appendix use variables. You can modify the default values associated with these variables by functions in GUI. For more information on variable defaults, parameterized variables, and the `emc_param.par` file, see [Chapter 5, “Performing EMControl Rule Checking.”](#)

Many rules described in this appendix depend on certain properties being attached to objects in your design. For information about attaching properties to your design before running EMControl, see [Chapter 4, “Setting Up the EMControl Design Environment.”](#)

Placement Rules

The following rules are included in the `emc_placement.rle` file:

- [central_clock](#)
- [conn_in_low_freq_regions](#)
- [gnd_screw_between_clock_and_conn](#)

central_clock

Checks whether a clock generator is located at the center of the region formed by the IC components that it drives.

The rule first identifies all clock nets. Clock nets are defined as nets connected to the clock generator at pins with `PINUSE = OUT`. It then identifies all IC components connected to the clock nets.

The sum of the distances from each individual IC component to the clock generator is calculated and compared with the sum of the distances from each IC component to all other

IC components on the net and the clock generator. If the first sum is greater than the second sum, a violation is reported.

Properties Used

EMC_COMP_TYPE = CLOCK_GEN

PINUSE = OUT

Variables Used

None

Required Data

Attach the EMC_COMP_TYPE property set to CLOCK_GEN to all clock generators.

You do not have to attach the PINUSE property to the pins. EMControl reads the PINUSE property directly from the layout database. (PINUSE is a hidden property that is usually present.) You can, however, overwrite the automatic property setting by attaching PINUSE as you would any other property.

Default Severity

Info

conn_in_low_freq_regions

Checks whether all connectors are placed in the lowest frequency region available on the board.

This rule also checks whether any higher frequency components (as identified by their EMC_CRITICAL_IC property value) are present in the lowest frequency regions that contain connectors.

Properties Used

EMC_CRITICAL_IC

EMC_CRITICAL_REGION

Variables Used

None

Required Data

Attach the EMC_CRITICAL_IC property to critical ICs.

Attach the EMC_CRITICAL_REGION property to critical frequency regions.

Default Severity

Error

gnd_screw_between_clock_and_conn

Checks whether a grounded screw exists between a clock generator and a connector.

The area searched for a ground screw is a rectangular area with a length equal to the distance from the center of the clock generator to the center of the connector and a width equal to the length multiplied by the GND_SCREW_SENS_DIST_RATIO.

Properties Used

EMC_COMP_TYPE = CLOCK_GEN

EMC_COMP_TYPE = GND_SCREW

Variables Used

GND_SCREW_SENS_DIST_RATIO

Required Data

Attach the EMC_COMP_TYPE property set to CLOCK_GEN to all clock generators.

Attach the EMC_COMP_TYPE property set to GND_SCREW to all ground screws.

If required, you can edit the value of the GND_SCREW_SENS_DIST_RATIO variable before using this rule to check your design. GND_SCREW_SENS_DIST_RATIO specifies the ratio between the length and width for the rectangular area being searched for a ground screw.

Default Severity

Error

Bypass Rules

The following rules are included in the `emc_bypass.rle` file:

- bypass_cap_type
- bypass_drvr_rcvr_bidir
- bypass_fast_sw_trans
- critical_IC_3caps_C_2C_4C
- critical_IC_loop_area
- decouple_emc_regions
- fence_off_emc_regions

bypass_cap_type

Checks that all bypass capacitors are one of the types specified by ALL_BYPASS_CAP_TYPE.

Properties Used

EMC_CPOMP_TYPE = BYPASS_CAP

TOL

VALUE

Variables Used

ALL_BYPASS_CAP_TYPE

Required Data

Attach the EMC_COMP_TYPE property set to BYPASS_CAP to all bypass capacitors.

The TOL and VALUE properties identify the tolerance and capacitance values of the capacitor.

If required, you can edit the value of the ALL_BYPASS_CAP_TYPE variable before using this rule to check your design. ALL_BYPASS_CAP_TYPE specifies the allowed bypass capacitor types. Use the following syntax to specify a bypass capacitor type:

```
"<device type>:<cap value>:<tolerance>"
```

For example:

```
#define ALL_BYPASS_CAP_TYPE "CAP-1:82UF:5%" , "CAP-2:0.01uf:5%"  
, "CAP-3:0.01uf:3%" , "CAP-2:10uf:4%" "CAP-4:1.0uf:2%"  
, "CAP-5:5uf:5%"
```

Default Severity

Warning

bypass_drvr_rcvr_bidir

Checks that all line drivers, line receivers, and bidirectional transceivers are bypassed. These components must have their bypass capacitors at a specified distance from the power pin. The rule identifies the power pin as having the PINUSE="POWER" property attached.

Properties Used

EMC_COMP_TYPE = LINE_DRIVER

EMC_COMP_TYPE = LINE_RECEIVER

EMC_COMP_TYPE = BIDIR_TRANS

EMC_COMP_TYPE = BYPASS_CAP

PINUSE = POWER

Variables Used

BYP_CAP_SENS_DIST

EMC_BYPASS_CAP_PWR_PIN_DIST

Required Data

Attach the EMC_COMP_TYPE property set to LINE_DRIVER to all line drivers.

Attach the EMC_COMP_TYPE property set to LINE_RECEIVER to all line receivers.

Attach the EMC_COMP_TYPE property set to BIDIR_TRANS to all bidirectional transceivers.

Attach the EMC_COMP_TYPE property set to BYPASS_CAP to all bypass capacitors.

You do not have to attach the PINUSE property to the pins. EMControl reads the PINUSE property directly from the layout database. (PINUSE is a hidden property that is usually present.) You can, however, overwrite the automatic property setting by attaching PINUSE as you would any other property.

If required, you can edit the default values for EMControl variables before using this rule to check your design:

- BYP_CAP_SENS_DIST specifies the bypass capacitor sensitive distance. (This variable is parameterized.)

The rule selects the maximum value of this variable.

- EMC_BYPASS_CAP_PWR_PIN_DIST specifies the maximum distance between the power pin and the bypass capacitors.

Default Severity

Error

bypass_fast_sw_trans

Checks that all fast-switching transistors are bypassed properly.

Properties Used

EMC_COMP_TYPE = FAST_SWITCH_TRANSISTOR

EMC_COMP_TYPE = BYPASS_CAP

Variables Used

BYP_CAP_SENS_DIST

Required Data

Attach the EMC_COMP_TYPE property set to FAST_SWITCH_TRANSISTOR to all fast switching transistors.

Attach the EMC_COMP_TYPE property set to BYPASS_CAP to all bypass capacitors.

If required, you can edit the value of the BYP_CAP_SENS_DIST variable before using this rule to check your design. BYP_CAP_SENS_DIST defines the bypass capacitor sensitive distance. The rule uses the maximum value of BYP_CAP_SENS_DIST. (This variable is parameterized.)

Default Severity

Error

critical_IC_3caps_C_2C_4C

Checks the distances between bypass capacitors and critical IC.

This rule ensures that

- Three bypass capacitors are used for each critical IC power pin.
- The three capacitors have values in the ratio of 1:2:4.
- The largest capacitor is farthest from the pin within a distance of CAP1_DIST_FROM_PWR_PIN.
- The second largest capacitor is within a distance of CAP2_DIST_FROM_PWR_PIN.
- The smallest capacitor is closest within a distance of CAP3_DIST_FROM_PWR_PIN.

The rule identifies the power pin by the PINUSE="POWER" property on the pin.

Properties Used

EMC_CRITICAL_IC

EMC_COMP_TYPE = BYPASS_CAP

VALUE

PINUSE = POWER

Variables Used

CAP1_DIST_FROM_PWR_PIN

CAP2_DIST_FROM_PWR_PIN

CAP3_DIST_FROM_PWR_PIN

BYPASS_CAP_SENS_DIST

Required Data

Attach the EMC_COMP_TYPE property set to BYPASS_CAP to all bypass capacitors.

Attach the EMC_CRITICAL_IC property to all critical ICs.

The VALUE property identifies the capacitance value of the capacitor.

You do not have to attach the PINUSE property to the pins. EMControl reads the PINUSE property directly from the layout database. (PINUSE is a hidden property that is usually present.) You can, however, overwrite the automatic property setting by attaching PINUSE as you would any other property.

If required, you can edit the default values for EMControl variables before using this rule to check your design:

- CAP1_DIST_FROM_PWR_PIN specifies the distance of capacitor 1 from the power pin, for example, 300 mils. (This variable is parameterized.)
- CAP2_DIST_FROM_PWR_PIN specifies the distance of capacitor 2 from the power pin, for example, 200 mils. (This variable is parameterized.)
- CAP3_DIST_FROM_PWR_PIN specifies the distance of capacitor 3 from the power pin, for example, 100 mils. (This variable is parameterized.)
- BYP_CAP_SENS_DIST defines the bypass capacitor sensitive distance. (This variable is parameterized.)

Default Severity

Error

critical_IC_loop_area

Calculates the power and ground loop area and compares it with the area calculation based on the geometry of the package. The rule handles both routed power and ground (double-sided PCB) and unrouted power and ground (multilayered PCB) cases.

The rule works on all ICs with the property EMC_CRITICAL_IC. For every power pin of the IC, the closest ground pin and the closest capacitor are considered for area calculations.

`critical_IC_loop_area` identifies the power and ground pins as the pins having the properties `PINUSE = "POWER"` and `PINUSE = "GROUND"`, respectively.

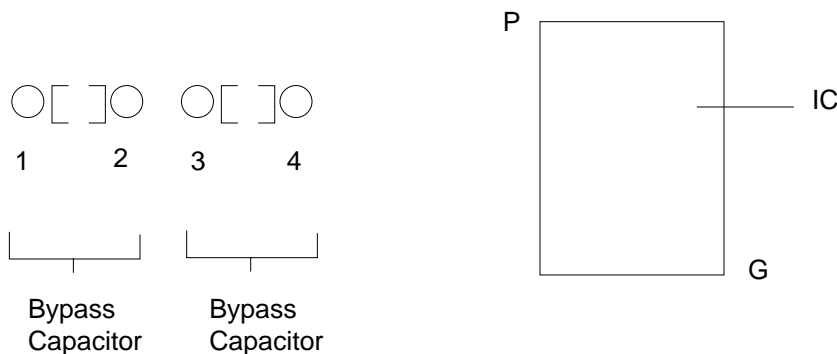
For an IC with the property `EMC_CRITICAL_IC`, the constraint area is calculated as described in the following example.

In Figure 2-1, P identifies the location on the IC of the power pin and G identifies the ground pin. P-G is taken to be `len1`, the direct distance between the power pin and the ground pin. Using `len1` as the hypotenuse, a right-angled isosceles triangle is constructed. The area of the triangle is calculated to be `Tref`.

The actual loop area is calculated differently for routed and unrouted cases:

- **Unrouted cases** — No routed connections between power and ground pins on the IC and the bypass capacitor.

Figure 2-1 Example for Calculating Loop Area



A triangle is constructed with the following vertices:

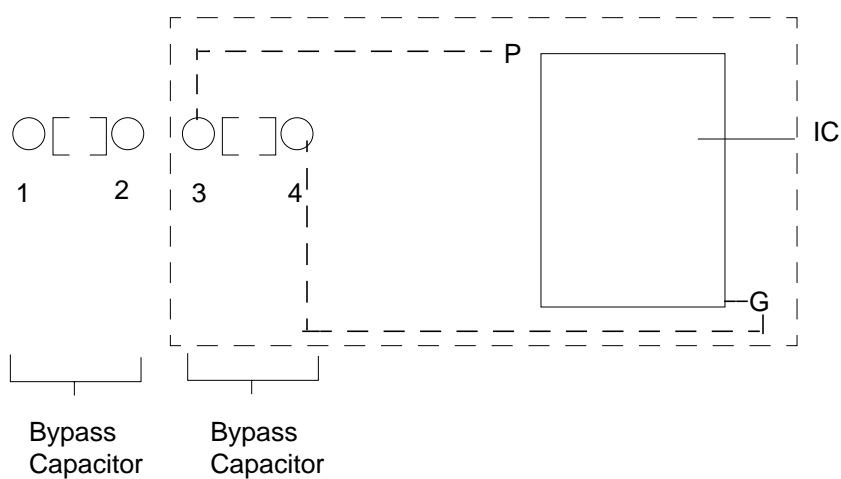
- ❑ The power pin on the IC (P)
- ❑ The ground pin on the IC (G)
- ❑ The capacitor pin that is most distant from the power pin (3)

In the above example, the triangle is P-G-3. The area of this triangle is calculated as T_{act} .

- **Routed cases** — The rule identifies the shortest etch paths from the power and ground pins of the IC to the bypass capacitor.

The etch that connects the power pin to the capacitor is called `pwr_etch`. The etch that connects the ground pin to the capacitor is called `gnd_etch`.

Figure 2-2 Example of Routed Case



In Figure 2-2, the minimum bounding box is shown as a dotted line. It encloses:

- ❑ The power pin on the IC (P)
- ❑ The ground pin on the IC (G)
- ❑ The bypass capacitor (capacitor with pins 3, 4)
- ❑ The `pwr_etch` (etch 3-P or 4-P)
- ❑ The `gnd_etch` (etch 4-G or 3-G)

The area of the minimum bounding box is calculated as T_{act} .

Two areas have now been determined: T_{act} and T_{ref} . The ratio of these areas is calculated. A violation is reported if the ratio exceeds the value specified by the `LOOP_AREA_COEFFICIENT` variable.

Properties Used

EMC_COMP_TYPE = BYPASS_CAP

EMC_CRITICAL_IC

PINUSE = POWER

PINUSE = GROUND

Variables Used

LOOP_AREA_COEFFICIENT

BYP_CAP_SENS_DIST

Required Data

Attach the EMC_COMP_TYPE property set to BYPASS_CAP to all bypass capacitors.

Attach the EMC_CRITICAL_IC property to all critical ICs.

You do not have to attach the PINUSE property to the pins. EMControl reads the PINUSE property directly from the layout database. (PINUSE is a hidden property that is usually present.) You can, however, overwrite the automatic property setting by attaching PINUSE as you would any other property.

If required, you can edit the default values for EMControl variables before using this rule to check your design:

- LOOP_AREA_COEFFICIENT defines the maximum allowable ratio of the actual area (calculated as T_{act}) and the reference area (calculated as T_{ref}). (This variable is parameterized.)
- BYP_CAP_SENS_DIST defines the bypass capacitor sensitive distance. (This variable is parameterized.)

Default Severity

Error

decouple_emc_regions

Checks the boundary between adjacent EMC regions having different EMC frequency classifications for a sufficient number of bypass capacitors. An EMC region is identified by the EMC_CRITICAL_REGION property value attached to it. Bypass capacitors are identified by the property EMC_COMP_TYPE = BYPASS_CAP.

Areas of the board without an assigned EMC_CRITICAL_REGION property are considered to be part of a room having the lowest assigned frequency class. Rooms are not actually merged. When two rooms overlap, the room with the higher frequency class takes precedence.

The “adjacency” of two rooms and their common boundary is defined as follows:

- If the edges of a room and its neighbor are separated by more than the distance specified by BYP_CAP_SENS_DIST, they will not be considered adjacent. The area between them will be considered as belonging to the default room of the lowest frequency.

The area between them will be considered as belonging to the default room of the lowest frequency.

- If the edges of the two rooms are within the distance specified by BYP_CAP_SENS_DIST, the edge of the region corresponding to the higher frequency will be taken as the center of the boundary region where bypass capacitors are to be checked. No bypass checking will be done for the lower frequency region.

- A TOP EMC region considers a TOP or BOTH region as its neighbor.

A BOTTOM region considers only BOTTOM rooms as neighbors.

Properties Used

EMC_COMP_TYPE = BYPASS_CAP

EMC_CRITICAL_REGION

Variables Used

BYP_CAP_SENS_DIST

BYP_CAP_SEP_DIST

Required Data

Attach the EMC_COMP_TYPE property set to BYPASS_CAP to all bypass capacitors.

Attach the EMC_CRITICAL_REGION property to all critical regions.

If required, you can edit the default values for EMControl variables before using this rule to check your design:

- BYP_CAP_SENS_DIST defines the bypass capacitor sensitive distance. (This variable is parameterized.)
- BYP_CAP_SEP_DIST defines the minimum separation between decoupling capacitors along the boundary. (This variable is parameterized.) The BYP_CAP_SEP_DIST variable has one less value than the specified number of EMC classes.

Default Severity

Info

fence_off_emc_regions

Checks for the presence of a fence, or boundary area, along the boundaries between EMC regions having different EMC_CRITICAL_REGION property values. This rule also checks for the percentage overlap of the fence and the boundary.

Fences are identified by the property EMC_COMP_TYPE = FENCE. A TOP or BOTH region checks for fences on the TOP layer only. A BOTTOM region checks for fences on the BOTTOM layer only.

Properties Used

EMC_COMP_TYPE = FENCE

EMC_CRITICAL_REGION

Variables Used

FENCE_BOUNDARY_DIST

FENCE_BOUNDARY_RATIO

Required Data

Attach the EMC_COMP_TYPE property set to FENCE to all fences.

Attach the EMC_CRITICAL_REGION property to critical frequency regions.

If required, you can edit the default values for EMControl variables before using this rule to check your design:

- FENCE_BOUNDARY_DIST specifies the distance at which fences can be found from the boundary.
- FENCE_BOUNDARY_RATIO specifies the minimum ratio of overlap required between the fences and region boundaries.

Default Severity

Info

DC Routing Rules

The following rules are included in the `emc_dc_route.rle` file:

- bypass_pwr_trace
- filters to clean ground
- max_pwr_gnd_resistance
- nets over clean_gnd

bypass_pwr_trace

Checks that power traces are bypassed to ground. The rule operates on the power trace segments whose length is more than the value specified by `POWER_TRACE_BYPASS_DIST`. It counts the number of bypass capacitors on these segments and compares this number with the number of bypass capacitors required on the segment. The number of required bypass capacitors is determined by assuming that bypass capacitors must be equally spaced and at the distance specified by `POWER_TRACE_SENS_DIST`.

For example, if POWER_TRACE_BYPASS_DIST has a value of 500 and the length of a segment of the power trace is 1100 units, the number of bypass capacitors required for the segment is two.

Properties Used

VOLTAGE

Variables Used

POWER_TRACE_SENS_DIST

POWER_TRACE_BYPASS_DIST

Required Data

Attach the EMC_COMP_TYPE property set to BYPASS_CAP to all bypass capacitors.

Attach the VOLTAGE property to all DC nets. The most common nets requiring attachment of this property are power and ground. Ground nets must have a VOLTAGE value of zero. Power nets are identified by a nonzero VOLTAGE value.

If required, you can edit the default values for EMControl variables before using this rule to check your design:

- POWER_TRACE_SENS_DIST defines the distance between bypass capacitors and the power trace.
- POWER_TRACE_BYPASS_DIST specifies the length of the power trace for which bypass capacitors should exist.

Default Severity

Warning

filters_to_clean_ground

Checks whether the IO filter components are connected to a clean ground plane. A clean section of the ground plane is defined as a separate shape on the ground plane surrounded by a moat.

EMControl User Guide

EMControl Rules

Filter components are identified by the property EMC_COMP_TYPE = FILTER. Ground shapes are on ETCH subclass with a layer type of plane. They are part of a net with the property VOLTAGE = 0.

For each filter connected to an IO component and having more than two pins, at least one pin must connect to a clean ground shape. An error is reported if there is no connection to clean ground.

Properties Used

EMC_COMP_TYPE = FILTER

VOLTAGE

Variables Used

None

Required Data

Attach the EMC_COMP_TYPE property set to FILTER to all filters.

Attach the VOLTAGE property set to zero to all ground shapes.

Default Severity

Error

max_pwr_gnd_resistance

Checks the resistance between the power/ground pin of an IC and the voltage source, using the maximum permissible value specified by the PIN_PIN_RESISTANCE variable or the PIN_PLANE_RESISTANCE variable, as described below.

Properties Used

VOLTAGE_SOURCE_PIN

VOLTAGE

Variables Used

PIN_PIN_RESISTANCE

PIN_PLANE_RESISTANCE

Required Data

When the design has no dedicated power/ground layers, attach the VOLTAGE_SOURCE_PIN property to a voltage source pin, such as a connector pin. The VOLTAGE_SOURCE_PIN property is a flag that identifies the voltage source.

Assign the VOLTAGE property to all nonsignal nets. The most common nets requiring attachment of this property are power and ground. Ground nets must have a VOLTAGE value of zero. Power nets are identified by a nonzero VOLTAGE value.

If required, you can edit the default values for EMControl variables before using this rule to check your design:

- PIN_PIN_RESISTANCE specifies the maximum pin-to-pin resistance.

The PIN_PIN_RESISTANCE variable is used for designs in which no dedicated plane layers exist and one of the pins has the VOLTAGE_SOURCE_PIN property attached.

- PIN_PLANE_RESISTANCE specifies the maximum pin-to-plane resistance.

The PIN_PLANE_RESISTANCE variable is used for designs that have dedicated plane layers.

Default Severity

Error

nets_over_clean_gnd

Checks the routing of nets over clean ground shapes. All nets routed over a clean ground shape should enter the clean ground region at right angles. The nets should have a minimum etch path over the clean ground shape.

This rule also checks whether any components other than filters or connectors are present above the clean ground shape.

Clean ground shapes are isolated regions on the ground plane. They will be separated shapes in the ground plane surrounded by a moat, or void area.

Properties Used

EMC_COMP_TYPE = FILTER

VOLTAGE

Variables Used

CRITICAL_TO_MHATTAN_LEN_RATIO

Required Data

Attach the EMC_COMP_TYPE property set to FILTER to all filter components.

Attach the VOLTAGE property set to zero to all ground nets.

If required, you can edit the default value for the CRITICAL_TO_MHATTAN_LEN_RATIO variable. CRITICAL_TO_MHATTAN_LEN_RATIO defines the ratio of the critical net length to the manhattan length.

Default Severity

Error

Signal Routing Rules

The following rules are included in the `emc_sig_route.rlc` file:

- critical net man ratio
- critical net via count
- filtered IO signals
- max critical net xtalk

critical_net_man_ratio

Checks the length of all critical nets. In order to minimize radiation, critical nets should be routed in inner layers and they should have minimum exposed etch length. The length of

critical nets must not exceed the percentage of their manhattan length specified by CRITICAL_TO_MHATTAN_LEN_RATIO.

Properties Used

EMC_CRITICAL_NET

Variables Used

CRITICAL_TO_MHATTAN_LEN_RATIO

Required Data

Attach the EMC_CRITICAL_NET property to all nets considered critical from the EMC standpoint.

If required, you can edit the default values for EMControl variables before using this rule to check your design:

CRITICAL_TO_MHATTAN_LEN_RATIO specifies the ratio of the critical net length to the manhattan length. (This variable is parameterized.)

Default Severity

Error

critical_net_via_count

Checks the number of vias for each critical net. The number should be kept to a minimum to improve reliability, and should not exceed the value specified by EMC_VIA_COUNT.

Properties Used

EMC_CRITICAL_NET

Variables Used

EMC_VIA_COUNT

Required Data

Attach the EMC_CRITICAL_NET property to all nets considered critical from the EMC standpoint.

If required, you can edit the EMC_VIA_COUNT variable to change the default definition for the maximum number of vias allowed for each critical net. (This variable is parameterized.)

Default Severity

Error

filtered_IO_signals

Checks that all signals running to an IO connector are filtered.

Properties Used

EMC_COMP_TYPE = FILTER

Variables Used

None

Required Data

Attach the EMC_COMP_TYPE property set to FILTER to all filter components.

Default Severity

Warning

max_critical_net_xtalk

Checks the coupling from critical nets to neighboring IO nets, and compares the crosstalk with the maximum value specified by the MAX_PEAK_XTALK variable.

This rule requires Crosstalk table to be generated before executing this rule. Note that Geometry window should be equal to XTALK_WINDOW to get desired results. The Geometry

window can be set through the Signal Analysis dialog box, *Preferences > InterconnectModels*.

For more information, see topic *Generating a Crosstalk Table* in the *SPECCTRAQuest Simulation and Analysis Reference Guide*.

Properties Used

EMC_CRITICAL_NET

Variables Used

XTALK_WINDOW

MAX_PEAK_XTALK

Required Data

Attach the EMC_CRITICAL_NET property to all nets considered critical from the EMC standpoint.

If required, you can edit the default values for EMControl variables before using this rule to check your design:

- XTALK_WINDOW specifies how far the crosstalk algorithms look for neighboring nets to determine the crosstalk value.
- MAX_PEAK_XTALK specifies the maximum crosstalk in mV that can come from any individual EMC_CRITICAL_NET neighbor before a violation is reported. The rule uses the larger value of either forward or backward crosstalk to determine whether a violation exists. (This variable is parameterized.)

Default Severity

Error

Signal Quality Rules

The following rules are included in the `emc_sig_qual.rle` file:

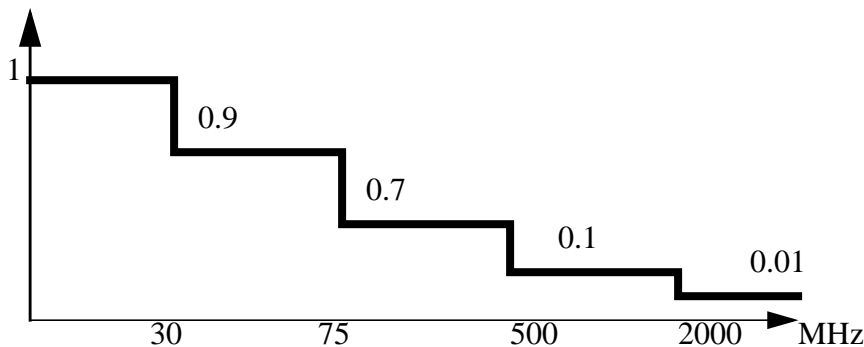
- clock spectral content

- critical net ringing
- critical net termination
- single diff mode EMI
- sum diff mode EMI Default Rules

clock_spectral_content

Checks the frequency content on clock nets and compares the spectrum with the MAX_CLOCK_SPECTRAL_CONTENT variable. MAX_CLOCK_SPECTRAL_CONTENT defines a staircase limit for the spectrum Figure 2-3.

Figure 2-3 MAX_CLOCK_SPECTRAL_CONTENT



When you execute the clock_spectral_content rule, EMControl starts a SigNoise pulse simulation for each clock generator output net in the design and supplies SigNoise with the pulse frequency and duty cycle for the clock signal stimuli as specified by the PULSE_DUTY_CYCLE variable. The number of cycles simulated is determined by the MAX_SIMULATION_CYCLE variable.

EMControl reports a warning message for each net that violates the limits defined by the MAX_CLOCK_SPECTRAL_CONTENT variable. For example, Figure 2-3 defines the following limit:

```
MAX_CLOCK_SPECTRAL_CONTENT = "0MHz:1V 30MHz:0.9V 75MHz:0.7V 500MHz:0.1V  
2GHz:10mV"
```

This limit specifies that the strength of all clock signals must be:

- Less than 1.0 V from DC to 30 MHz
- Less than 0.9 V from 30 MHz to 75 MHz

EMControl User Guide

EMControl Rules

- Less than 0.7 V from 75 MHz to 500 MHz
- Less than 0.1 V from 500 MHz to 2 GHz
- Less than 0.01 V for frequency higher than 2 GHz

When you select a warning message in this list, EMControl does the following:

- Highlights the net in the design.
- Opens the SigNoise waveform window (SigWave) and displays the waveform for the first load on the selected net. SigWave performs a Fast Fourier Transform (FFT) on the waveform and displays the results.

Selecting a different clock signal in the message list repeats the process for the selected net. Selecting another load from the SigWave pop-up menu updates the waveform and spectrum displays.

Properties Used

EMC_CRITICAL_NET

EMC_COMP_TYPE = CLOCK_GEN

PINUSE

Variables Used

PULSE_DUTY_CYCLE

MAX_CLOCK_SPECTRAL_CONTENT

MAX_SIMULATION_CYCLE

Required Data

Attach the EMC_CRITICAL_NET property to all nets considered critical from the EMC standpoint.

Attach the EMC_COMP_TYPE property set to CLOCK_GEN to all clock generators.

You do not have to attach the PINUSE property to the pins. EMControl reads the PINUSE property directly from the layout database. (PINUSE is a hidden property that is usually

present.) You can, however, overwrite the automatic property setting by attaching PINUSE as you would any other property.

If required, you can edit the default values for EMControl variables before using this rule to check your design:

- PULSE_DUTY_CYCLE defines the pulse frequency and duty cycle for the respective EMC_CRITICAL_NET properties.
- MAX_CLOCK_SPECTRAL_CONTENT defines a staircase limit for the frequency content of critical clock nets.
- MAX_SIMULATION_CYCLE defines the number of cycles SigNoise simulates.

Default Severity

Warning

critical_net_ringing

Checks the ringing of critical nets by verifying that overshoot and undershoot do not exceed a specified percentage of the voltage swing.

This rule requires that SigNoise be run in the background.

For each net with the EMC_CRITICAL_NET property, the critical_net_ringing rule verifies that the maximum overshoot and undershoot obtained by SigNoise simulation is less than the value of the ratio defined by MAX_OVER_UNDERSHOOT multiplied by the smallest voltage swing due to driver pins on the net.

For example, if the value of MAX_OVER_UNDERSHOOT is 0.15 for a net that has a 0 V to 5 V swing, the maximum amount of overshoot or undershoot will be determined as follows:

$$0.15 * 5V = 0.75V$$

In this example, any value greater than 0.75 V is considered to be a violation.

Properties Used

EMC_CRITICAL_NET

Variables Used

MAX_OVER_UNDERSHOOT

Required Data

Attach the EMC_CRITICAL_NET property to all nets considered critical from the EMC standpoint.

If required, you can edit the default value for the MAX_OVER_UNDERSHOOT variable before using this rule to check your design. MAX_OVER_UNDERSHOOT specifies the ratio used to calculate the maximum permitted overshoot and undershoot. (This variable is parameterized.)

Default Severity

Error

critical_net_termination

Checks for the termination of critical nets. Critical signals must be terminated when the driver output rise or fall time is less than twice the propagation delay.

This rule requires data produced by SigNoise.

Reports ERROR if the critical net is not terminated.

Reports WARNING if the critical net contains discrete components.

Properties Used

EMC_CRITICAL_NET

Variables Used

None

Required Data

Attach the EMC_CRITICAL_NET property to all nets considered critical from the EMC standpoint.

Default Severity

Error

single_diff_mode_EMI

Checks for excessive differential mode EMI produced by signal nets.

This rule extracts linear parameters of the device models assigned to the pins of devices. Therefore, it is important to ensure that the correct models are assigned to pins and that the appropriate pins and that the appropriate libraries are loaded in SigNoise.

The differential mode emission from a trace is estimated by estimation of the current flowing through the trace and applying the loop antenna algorithm. A linear model approximation is used for the representation of the drivers and loads. The computed differential mode emission level is compared with a user-selected emission standard. A design margin value can also be specified by the user.

If the emission level of a signal net exceeds the standard minus the design margin level, the net is identified as having failed the emissions test. The peak emission level in six frequency octaves is displayed in the report. The signal nets will be colored depending on the emission level. This is useful for a quick visual identification of hot spots.

Essential requirements for the analysis

- Signal net must contain at least 2 IC pins
- At least one pin in the net must be of the output type
- Clock generators must have EMC_COMPONENT_TYPE property set to CLOCK_GEN
- BUS_NAME property should be set for all data signals
- The clock frequency must be set using the PULSE_PARAM property

Properties Used

EMC_COMPONENT_TYPE (for clock generators)

PULSE_PARAM

BUS_NAME (for data signals)

PINUSE

Variables Used

EMI_REGULATION

DESIGN_MARGIN

GRID_X

GRID_Y

USER_LEVEL

RESET_DESIGN

Required Data

Attach property EMC_COMP_TYPE = CLOCK_GEN for all clock generator components. Specify signal frequency using the PULSE_PARAM property. Assign BUS_NAME property to all data nets which need to be analyzed.

If required, you can edit the default values for the following EMControl variables before using this rule to check your design.

- ☐ EMI_REGUALTION defines the EMI regulation against which the design is to be checked. This variable can take the following values:

FCC_CLASS_A FCC_CLASS_B CISPR_CLASS_A CISPR_CLASS_B VC
CI_CLASS_1 VCCI_CLASS_2 USERDEF_CLASS

- ☐ DESIGN_MARGIN defines the design margin in db
- ☐ GRID_X and GRID_Y defines the dimensions of the ground grid in inches for two layer boards.
- ☐ USER_LEVEL defines the user defined emission level in dBmicroVolts per meter. This variable is used only if EMI_REGUALTION is USERDEF_CLASS.

- ❑ RESET_DESIGN is used for resetting the board after a rule execution is complete. To reset the design, set RESET_DESIGN to YES and run the rule. RESET_DESIGN should be set to NO during normal execution of the rule.

Default Severity

WARNING

sum_diff_mode_EMI

Checks an entire design for total EMI emissions.

This rule extracts linear parameters of the device models assigned to the pins of devices. Therefore, it is important to ensure that the correct models are assigned to pins and that the appropriate libraries are loaded in SigNoise.

The sum_diff_mode_EMI rule does the following:

- ❑ Calculates the differential mode EMI for each signals as described for the single_diff_mode_EMI rule.
- ❑ Sums the EMI values for individual nets.
- ❑ Compares the total EMI to the appropriate EMI_REGULATION value minus the DESIGN_MARGIN.
- ❑ Reports a violation if the total emission exceeds the criteria set by EMI_REGULATION.
- ❑ Colors the signals nets on the board as per their emission levels.

Essential requirements for the analysis.

- ❑ Signal net must contain at least 2 IC pins
- ❑ At least on pin in the net must be of the output type
- ❑ Clock generators must have EMC_COMPONENT_TYPE property set to CLOCK_GEN
- ❑ BUS_NAME property should be set for all data signals
- ❑ The clock frequency must be set using the PULSE_PARAM property

Properties Used

EMC_COMPONENT_TYPE (for clock generators)

PULSE_PARAM

BUS_NAME (for data signals)

PINUSE

Variables Used

EMI_REGULATION

DESIGN_MARGIN

GRID_X

GRID_Y

USER_LEVEL

RESET_DESIGN

Required Data

Attach property EMC_COMP_TYPE = CLOCK_GEN for all clock generator components. Specify signal frequency using the PULSE_PARAM property. Assign BUS_NAME property to all data nets which need to be analyzed.

If required, you can edit the default values for the following EMControl variables before using this rule to check your design.

- ☐ EMI_REGULATION defines the EMI regulation against which the design is to be checked. This variable can take the following values:
FCC_CLASS_A FCC_CLASS_B CISPR_CLASS_A CISPR_CLASS_B
VCCI_CLASS_1 VCCI_CLASS_2 USERDEF_CLASS
- ☐ DESIGN_MARGIN defines the design margin in dB
- ☐ GRID_X and GRID_Y defines the dimensions of the ground grid in inches for two layer boards.

EMControl User Guide

EMControl Rules

- ❑ **USER_LEVEL** defines the user defined emission level in dBmicroVolts per meter. This variable is used only if **EMI_REGULATION** is **USERDEF_CLASS**.
- ❑ **RESET_DESIGN** is used for resetting the board after a rule execution is complete. To reset the design, set **RESET_DESIGN** to **YES** and run the rule. **RESET_DESIGN** should be set to **NO** during normal execution of the rule.

Default Severity

WARNING

Default Rules

The following rules are included in the `emc_default_rules.rle` file:

- bypass cap per plane square
- bypass critical IC
- comp not conn dist
- comp to conn dist
- gnd under clock
- pwr gnd plane separation
- pwr gnd trace width
- critical net card edge dist
- no critical net thru IO comps
- critical net hole dist
- serial resistor position

bypass_cap_per_plane_square

Checks the distributed bypassing of power and ground planes. Square plane regions with area `MIN_AREA_PER_BYPASS_CAP` square user units without a bypass capacitor is reported. This rule is included in the `emc_pwr_gnd_dist.rle` file.

Properties Used

`EMC_COMP_TYPE = BYPASS_CAP`

Variables Used

`MIN_AREA_PER_BYPASS_CAP`

Required Data

Attach the `EMC_COMP_TYPE = BYPASS_CAP` property to all bypass capacitors.

If required, you can edit the default values for the EMControl variables before using this rule to check your design:

MIN_AREA_PER_BYPASS_CAP defines the plane area per bypass capacitor.

Default Severity

WARNING

bypass_critical_IC

Checks bypassing of high-current, high-speed integrated circuits (ICs). It ensures that all critical ICs have connected to them at least the number of bypass capacitors specified by MIN_BYPASS_CAPS. The capacitors used must be one of the types specified by the variable CRITICAL_IC_BYP_CAP_TYPE.

Properties Used

EMC_CRITICAL_IC

EMC_COMP_TYPE

TOL

VALUE

Variables Used

BYPASS_CAP_SENS_DIST

CRITICAL_IC_BYPASS_CAP_TYPE

MIN_BYPASS_CAPS

Required Data

Attach the EMC_CRITICAL_IC property to all critical ICs.

Attach the EMC_COMP_TYPE property set to BYPASS_CAP to all bypass capacitors.

The TOL and VALUE properties identify the tolerance and capacitance values of the capacitor.

EMControl User Guide

EMControl Rules

If required, you can edit the default values for EMControl variables before using this rule to check your design:

- **BYP_CAP_SENS_DIST** defines the bypass capacitor sensitive distance. (This variable is parameterized.)
- **MIN_BYPASS_CAPS** defines the minimum number of bypass capacitors that must be associated with a critical IC. (This variable is parameterized.)
- **CRITICAL_IC_BYP_CAP_TYPE** identifies the bypass capacitor types that are allowed for a critical IC.

This variable is parameterized. Use the following syntax to specify the bypass capacitor type for each class:

`"<device type>:<cap value>:<tolerance>"`

For example:

```
#define CRITICAL_IC_BYP_CAP_TYPE "CAP-1:82UF:5%!CAP-3:0.01uf:3%"  
, "CAP-2:10uf:4%" , "CAP-4:1.0uf:2%!CAP-5:5uf:5%" , "CAP-5:5uf:5%"  
, "CAP-5:5uf:5%"
```

Note: '!' is used as a separator between capacitor types allowed for the same class.

Default Severity

Error

comp_not_conn_dist

Checks for a minimum distance between an IO component (a connector) and the components not connected to it. This rule ensures that any component not connected to an offcard net is more than the **COMP_CONN_MIN_DISTANCE** from the IO component. It also ensures that components not connected to the IO component are more than the **COMP_COMP_MIN_DISTANCE** from any component connected to the IO component.

Properties Used

Voltage

Variables Used

COMP_CONN_MIN_DISTANCE

COMP_COMP_MIN_DISTANCE

Required Data

You can edit the values for the EMControl variables before using this rule to check your design:

- COMP_CONN_MIN_DISTANCE specifies the permissible minimum distance between components that are not connected to the IO component and the IO component itself.
- COMP_COMP_MIN_DISTANCE specifies the minimum distance between components that are not connected to the IO component and components that are connected to it.

Default Severity

Warning

comp_to_conn_dist

Checks for a maximum distance between an IO component (a connector) and the components connected to it, excluding passive elements and discrete components.

This rule ensures that all components connected to an offcard net are within the distance specified by the EMC_COMP_CONN_DISTANCE of the IO component connected to the offcard net.

Properties Used

Voltage

Variables Used

EMC_COMP_CONN_DISTANCE

Required Data

You can edit the EMC_COMP_CONN_DISTANCE variable to modify the value for the maximum distance between components and the IO component.

Attach the voltage property to all non-signal nets that are connected to the component being tested. The most common nets requiring attachment of this property are power and ground. Ground nets must have a VOLTAGE value of zero. Power nets are identified by a nonzero VOLTAGE value.

Default Severity

Warning

gnd_under_clock

Checks for a symmetrically placed grounded grid under each clock generator. Grounded grids are shapes that have the same subclass as that of the clock generator footprint (that is, either TOP or BOTTOM depending on where the clock generator is placed). They will be part of the ground net.

For each clock generator component on the board, the rule checks to ensure that at least one ground shape exists under the clock generator. It reports an error if it does not find a ground shape.

The rule also reports errors if the ratio of overlap between the ground shapes and package geometry area of the clock generator is less than 0.5 or if there is not at least one ground shape under each quadrant of the package geometry.

Properties Used

EMC_COMP_TYPE = CLOCK_GEN

VOLTAGE

Variables Used

None

Required Data

Attach the EMC_COMP_TYPE property set to CLOCK_GEN to all clock generators.

All ground shapes must be part of a net with the property VOLTAGE = 0.

Default Severity

Warning

pwr_gnd_plane_separation

Checks for a separation between power and ground planes. The rule checks that the distance between a power plane and a ground plane (on the Z axis) is not less than MIN_PWR_GND_SEPARATION and not more than MAX_PWR_GND_SEPARATION.

Properties Used

None

Variables Used

POWER_PLANE_NAME

GROUND_PLANE_NAME

MIN_POWER_GND_SEPARATION

MAX_POWER_GND_SEPARATION

Required Data

If required, you can edit the values for EMControl variables before using this rule to check your design:

- POWER_PLANE_NAME specifies the default name of the power plane.

For example:

```
#define POWER_PLANE_NAME "VCC"
```

- GROUND_PLANE_NAME specifies the default name of the ground plane.

For example:

```
#define GROUND_PLANE_NAME "GND"
```

- MIN_PWR_GND_SEPARATION identifies the minimum power to ground separation distance on the Z axis.

- MAX_PWR_GND_SEPARATION identifies the maximum power to ground separation distance on the Z axis.

Default Severity

Error

pwr_gnd_trace_width

Checks the power and ground trace widths. The rule verifies that the power and ground traces are at least three times the nominal line width or the width defined by GND_PWR_TRACE_WIDTH, whichever is greater. This increases signal integrity and minimizes simultaneous switching noise. EMControl uses the VOLTAGE property to identify the power and ground traces.

Properties Used

VOLTAGE

Variables Used

GND_PWR_TRACE_WIDTH

Required Data

Attach the VOLTAGE property to power and ground. Ground nets must have a VOLTAGE value of zero. Power nets are identified by a nonzero VOLTAGE value.

You can edit the GND_PWR_TRACE_WIDTH variable to specify the minimum power and ground trace width to use.

Default Severity

Error

critical_net_card_edge_dist

Checks the distance from the critical net to the card edge. The rule verifies that the critical nets do not come within the distance specified by EXT_NET_EDGE_CRITICAL_DIST of the

reference plane copper edge. If critical nets are buried, they must not come within the distance specified by INT_NET_EDGE_CRITICAL_DIST of the reference plane copper edge.

Properties Used

EMC_CRITICAL_NET

Variables Used

EXT_NET_EDGE_CRITICAL_DIST

INT_NET_EDGE_CRITICAL_DIST

Required Data

Attach the EMC_CRITICAL_NET property to all nets considered critical from the EMC standpoint.

If required, you can edit the default values for EMControl variables before using this rule to check your design:

- EXT_NET_EDGE_CRITICAL_DIST specifies the minimum distance between critical nets on external layers and copper edge. (This variable is parameterized.)
- INT_NET_EDGE_CRITICAL_DIST specifies the minimum distance between critical nets on internal layers and copper edge. (This variable is parameterized.)

Default Severity

Error

no_critical_net_thru_IO_comps

Checks for the routing of critical nets through connectors. Critical nets must not be routed through connector footprints.

Properties Used

EMC_CRITICAL_NET

VOLTAGE

Variables Used

None

Required Data

Attach the EMC_CRITICAL_NET property to all nets considered critical from the EMC standpoint.

Attach the VOLTAGE property to all nonsignal nets. The most common nets requiring attachment of this property are power and ground. Ground nets must have a VOLTAGE value of zero. Power nets are identified by a nonzero VOLTAGE value.

Default Severity

Error

critical_net_hole_dist

Checks the spacing between critical nets and mounting holes.

Properties Used

EMC_COMP_TYPE = MOUNTING HOLE

Variables Used

MIN_NET_SLOT_DIST

Required Data

The rule automatically identifies the mounting holes represented by stand-alone pins, vias and connector mounting holes.

Attach the EMC_COMP_TYPE property set to MOUNTING_HOLE if any other component is to be treated as a mounting hole.

If required, you can edit the default values for the EMControl variables before using this rule to check your design:

MIN_NET_SLOT_DIST defines the minimum spacing allowed between a critical net segment and a mounting hole. (This variable is parameterized.)

Default Severity

ERROR

serial_resistor_position

Serial resistor termination became more and more popular in modern design. In this case, the serial resistor should be placed close to the driver pin. But it's difficult for the layout engineer to identify the driver pin one by one.

This rule checks if the serial resistor is close to the driver pin in a point-to-point net.

Properties Used

EMC_COMP_TYPE = RESISTOR

PINUSE

Variables Used

SERIES_TO_DRIVER_DISTANCE

Required Data

Attach the EMC_COMP_TYPE property set to RESISTOR to all serial resistors in a point-to-point net.

You do not have to attach PINUSE property to the pins. EMControl reads the PINUSE property directly from the layout database. (PINUSE is a hidden property that is usually present.) You can, however, overwrite the automatic property setting by attaching PINUSE as you would any other property in SpecctraQuest.

SERIES_TO_DRIVER_DISTANCE: specifies the max length of the serial resistor to the driver pin.

EMControl User Guide

EMControl Rules

Default Severity

Error

Advanced Rules

The following rules are included in the `emc_advance_rules.rle` file:

- bypass_plane_split
- critical_net_return_path
- critical_net_via_pin_ratio
- return_path_near_signal_via
- shield_net
- vias_per_plane_square
- plane_overlap
- ic_power_sequence
- bulkcap_decap_trace
- ceracap_decap_trace
- gap_between_split_plane
- matched_via_number_diffpair
- symmetry_of_via_position_of_diffpair
- termination_of_diffpair

bypass_plane_split

Checks whether critical net segments run over the split plane. If the segment runs over the split plane and there is no bypass capacitors or direct trace within a certain distance to bypass the split plane, the segment is highlighted and an error message is reported.

Properties Used

EMC_CRITICAL_NET
EMC_COMP_TYPE
VOLTAGE

Variables Used

RETURN_PATH_LINE_WIDTH_RATIO
BYPASS_CAP_EFFECT_DISTANCE
DIRECT_TRACE_EFFECT_DISTANCE

Required Data

The EMC_CRITICAL_NET property identifies which net is the critical net and checks it.

The EMC_COMP_TYPE property identifies whether the component is a bypass capacitor or not.

The VOLTAGE property identifies whether the net is a power / ground net or not.

RETURN_PATH_LINE_WIDTH_RATIO defines the minimum ratio of the return path width and the critical net segment width.

The variable BYPASS_CAP_EFFECT_DISTANCE defines the effective scope of the bypass capacitor (within the scope of the critical signal) that can run over the split plane without errors. The default value is 100.

The variable DIRECT_TRACE_EFFECT_DISTANCE defines the effective scope of the direct trace (within the scope of the critical signal) that can run over the split plane without errors. The default value is 100.

Default Severity

ERROR

critical_net_return_path

Checks that critical nets have continuous plane layers above and below to provide return path for differential currents. The minimum return path window size is specified by the variable RETURN_PATH_LINE_WIDTH_RATIO if the segment is longer than RETURN_PATH_NECK_LENGTH, otherwise the variable RETURN_PATH_NECK_RATIO is used.

The rule also checks that the return path does not have any interruptions due to pins or vias passing through the planes.

Properties Used

EMC_CRITICAL_NET

Variables Used

RETURN_PATH_LINE_WIDTH_RATIO

RETURN_PATH_NECK_LENGTH

RETURN_PATH_NECK_RATIO

Required Data

Attach the EMC_CRITICAL_NET property to all nets considered critical from the EMC standpoint.

If required, you can edit the default values for the EMControl variables before using this rule to check your design.

RETURN_PATH_LINE_WIDTH_RATIO defines the minimum ratio of the return path width and the critical net segment width. This variable is parameterized. If the cline segment length is longer than the parameter RETURN_PATH_NECK_LENGTH, this rule uses RETURN_PATH_NECK_RATIO to calculate the return path width instead of RETURN_PATH_LINE_WIDTH_RATIO.

Default Severity

WARNING

critical_net_via_pin_ratio

Checks the via-to-pin ratio on all critical nets. Restricting the number of vias allowed for each net can help enforce EMC.

The rule calculates the ratio of vias to pins and determines whether the ratio exceeds the value specified by MAX_VIA_PIN_RATIO. When the maximum allowed ratio is exceeded, the rule records a violation for the net. The advisor message for a rule violation reports the PINUSE property value for each pin on the net.

Properties Used

EMC_CRITICAL_NET

Variables Used

MAX_VIA_PIN_RATIO

Required Data

Attach the EMC_CRITICAL_NET property to all nets considered critical from the EMC standpoint.

If required, you can edit the MAX_VIA_PIN_RATIO variable to change the default definition for the maximum via-to-pin ratio. (This variable is parameterized.)

Default Severity

Error

return_path_near_signal_via

Identifies critical signals that jump the reference planes. It also identifies the vias (jumping vias) at which signals jump their reference planes and ensures a return path close to the jumping vias. A return path in close proximity results in small loop area, hence lesser EMI.

Specify the maximum distance between jumping via and the return capacitor or via with local parameter MAX_RETURN_VIA_DIST.

The capacitor should be of the same or better type (EMC_CRITICAL_IC) than the class of the jumping signal (EMC_CRITICAL_NET).

Note: In practice, a capacitor can be connected to reference planes through vias. These are the vias that are tested for proximity to the jumping via. The distance with the capacitor is not considered. The lead-traces of the capacitor are also not checked for their length.

The property VOLTAGE attached to the nets of the plane is used to determine the type of interconnect required. Absence of this property results in the inability of the rule to determine whether a capacitor or a via should be the interconnect between the planes. In such a case, both capacitor and via are considered valid interconnects. In the absence of both, the rule prescribes putting an `interconnect`.



Attaching the EMC_CRITICAL_IC property to a capacitor causes the capacitor to be treated as an IC in some rules.

Properties Used

EMC_CRITICAL_IC

EMC_CRITICAL_NET

EMC_COMP_TYPE

Variables Used

MAX_RETURN_VIA_DIST

Required Data

If required, you can edit the MAX_RETURN_VIA_DIST parameter to change its default value.

Attach EMC_CRITICAL_IC property to the capacitor and EMC_CRITICAL_NET property to the net.

Attach property EMC_COMP_TYPE with value BYPASS_CAP to the capacitor.

It is recommended that you attach property VOLTAGE to the nets of the plane.

Default Severity

Warning

shield _net

Checks whether grounded guard traces exist for all nets that need to be shielded and that they lie parallel to the nets.

Guard traces should have the following characteristics:

- ☐ They should be part of the ground net and in the same layer as the net to be shielded.
- ☐ They should exist on at least one side of the shielded net within the distance GUARDING_DISTANCE.

- ❑ They should connect to the ground plane at regular intervals of λ /GUARD_TRACE_VIAS_PER_LAMBDA through vias. Lambda is the wavelength associated with the signal.
- ❑ They should be terminated with ground vias on both sides of the etch.

Properties Used

EMC_NET_TYPE = SHIELDED_NET

VOLTAGE

Variables Used

GUARD_TRACE_VIAS_PER_LAMBDA

GUARD_TRACE_VIAS_PER_1000_USERUNIT

GUARDING_DISTANCE

WAIVE_DISTANCE

Required Data

Attach the EMC_NET_TYPE property set to SHIELDED_NET to all nets that need to be shielded.

Ground shapes are on ETCH subclass with a layer type of plane. They are part of a net with property VOLTAGE = 0.

If required, you can edit the default values for the EMControl variables before using this rule to check your design:

- ❑ GUARD_TRACE_VIAS_PER_LAMBDA defines the number of vias found on a guard trace per lambda length, where lambda is the wavelength for the net.
- ❑ If EMControl can not get the lambda for net for some reasons, the variable GUARD_TRACE_VIAS_PER_1000_USERUNIT will be used instead of variable GUARD_TRACE_VIAS_PER_LAMBDA.
- ❑ GUARDING_DISTANCE defines the maximum distance from the shielded net to the guard traces.

- ❑ WAIVE_DISTANCE defines the distance between the shielded net and the component pin, which is not considered as error.

Default Severity

Error

vias_per_plane_square

This rule is used to check for a minimum number grounding vias per unit area on each ground plane of a multi-layer PCB.

Many modern-day PCBs have complex layering stack-ups that include multiple ground planes on different layers. In order for all ground planes to share a common electrical potential, they need to be connected together through vias, which short each ground-plane together. However, for high-speed systems, in order to maintain a common electrical potential between ground planes at high frequencies, there needs to be some minimum number of vias per unit area across the entire area of each ground plane.

This rule checks the number of vias per unit area across each ground plane on the PCB. All areas of the PCBs which are greater-than or equal-to user-defined sub-area and which have less-than the minimum number of required vias or have more-than the maximum number of required vias will be reported.

The ground shape will be splitted into the following Grids as figure 2-4:

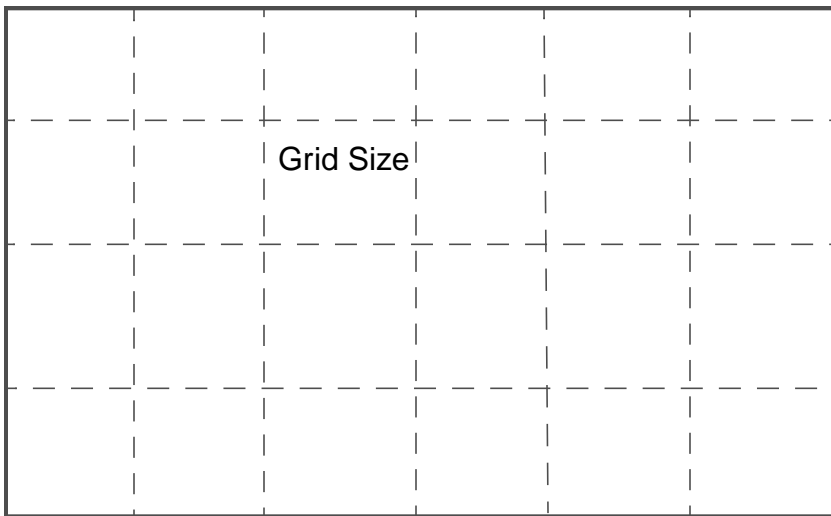


Figure 2-4 Split the gound plane shape

Here the grid size is specified by the MINIMUM_GROUNDVIA_AREA which is the area of the grid, so the grid side length will be the square root of the grid area.

For each grid on ground shape, if the number of vias contained in any grid is less than the minimum required specified by MINIMUM_VIA_NUM_PER_UNIT or more than the maximum required specified by MAXIMUM_VIA_NUM_PER_UNIT, that grid will be reported as error.

Properties Used

VOLTAGE

Variables Used

MINIMUM_GROUNDVIA_AREA

MINIMUM_VIA_NUM_PER_UNIT

MAXIMUM_VIA_NUM_PER_UNIT

Required Data

The VOLTAGE properties identify the voltage of net on the shape. The ground plane shape has a net which voltage is 0.0

MINIMUM_GROUNDVIA_AREA defines the minimum area where the number of required vias are calculated.

MINIMUM_VIA_NUM_PER_UNIT defines the minimum number of required vias per minimum area .

MAXIMUM_VIA_NUM_PER_UNIT defines the maximum number of required vias per minimum area .

Default Severity

ERROR

plane_overlap

Checking for the overlapping of different power/ground plane shapes with different power supply. In PCB design, power plane/shape is most way to provide the power. When two power/ground plane/shape with different power supply, for example, V+33 and V+18, are overlapped with each other, the power noise from these two power system will crosstalk each other and raise the issues on power integrity.

This rule checks the overlapping of two power/ground plane shapes with different power supply. Any overlapping of two plane shapes which has no other power/ground plane shape covering the overlapping will be reported as WARNING for use to confirm.

Properties Used

VOLTAGE

EMC_POWER_GND_NET_MATCH

Variables Used

NONE

Required Data

The VOLTAGE property identify the power/ground net in design. Any net with VOLTAGE property and the value 0.0 of VOLTAGE property is Ground Net, and the value greater than 0.0 is Power Net. Any plane shape which attached on the power net is power shape, and attached on the ground net is ground shape.

The EMC_POWER_GND_NET_MATCH property identify that which ground net a power net matches. The power/ground net which is matched belong to the same power supply and are allowed to overlap with each other. The power/ground net which is not matched belong to the different power supply and are not allowed to overlap with each other, otherwise the crosstalk will impact on each other and raise the power integrity problem. User can specify the EMC_POWER_GND_NET_MATCH property through EMControl product. Please click on the design configuration button on EMControl main form to set this property.

Default Severity

WARNING

ic_power_sequence

In some cases, some IC chip require that the power should go through decoupling capacitor, then go into the IC chip power pin. This rule check how the power connect to the IC chip power pin according to above requirement.

Properties Used:

EMC_CRITICAL_IC = CLASS1-5

EMC_COMP_TYPE = BYPASS_CAP

Variables Used

NONE

Required Data:

Attach the EMC_CRITICAL_IC property set to CLASS1-5 to all IC chips that have the power sequence requirement.

Attach the EMC_COMP_TYPE property set to BYPASS_CAP to all ceramic decoupling capacitor.

Default Severity

Error

bulkcap_decap_trace

This rule is to checking for the width and length of leading trace for BULKCAP capacitor to ensure that the inductance by the leading trace is acceptable.

This rule is also to checking for the number of leading traces and the number of vias on the leading traces for BULKCAP capacitor to ensure that the inductance by the leading trace is acceptable.

This rule checks the width and length of leading trace for BULKCAP capacitor. Any leading trace of BULKCAP capacitor which width is less than the user-defined width or which length is longer than the the user-defined length will be reported as error.

This rule checks the number of the leading traces and number of vias on the leading traces for BULKCAP capacitor. Any pins of BULKCAP capacitor which leading trace number is less than the user-defined number or the total number of vias on the leading traces is less than the the user-defined length will be reported as error.

Properties Used

EMC_COMP_CAP_TYPE

Variables Used

BULKCAP_LEADING_WIDTH

BULKCAP_LEADING_LENGTH

BULKCAP_LEADING_NUM

BULKCAP_LEADING_VIA_NUM

Required Data

The EMC_COMP_CAP_TYPE properties identify the BULKCAP capacitor in design. This property should be set as BULKCAP for BULKCAP capacitor. User can refer to the following information to identify the BULKCAP capacitor:

1. The class of the capacitor is DISCRETE
2. The leading name of component reference name, for example, the name with leading character "SC"
3. The package type of the component, for example, the "STC???"

Here the option 1 is required, and the options (2) and (3) is optional. User can specify the BULKCAP capacitor from EMControl Manual Property form.

BULKCAP_LEADING_WIDTH defines the minimum width of the leading trace of a BULKCAP capacitor.

BULKCAP_LEADING_LENGTH defines the maximum length of the leading trace of a BULKCAP capacitor.

BULKCAP_LEADING_NUM defines the minimum number of the leading traces of a BULKCAP capacitor.

BULKCAP_LEADING_VIA_NUM defines the minimum via number on the leading traces of a BULKCAP capacitor.

Default Severity

ERROR

ceracap_decap_trace

This rule checks the width and length of leading trace for CERACAP capacitor. Any leading trace of CERACAP capacitor with a width that is less than the user-defined width or with length that is longer than the user-defined length is reported as an error.

Properties Used

EMC_COMP_CAP_TYPE

Variables Used

CERACAP_LEADING_WIDTH

CERACAP_LEADING_LENGTH

Required Data

The EMC_COMP_CAP_TYPE properties identify the CERACAP capacitor in design. This property should be set as CERACAP for CERACAP capacitor. User can refer to the following information to identify the CERACAP capacitor:

1. The class of the capacitor is DISCRETE
2. The leading name of component reference name, for example, the name with leading character "C"
3. The package type of the component, for example, the "C0805"

Here the option 1 is required, and the options (2) and (3) is optional. User can specify the CERACAP capacitor from EMControl Manual Property form.

CERACAP_LEADING_WIDTH defines the minimum width of the leading trace of a CERACAP capacitor.

CERACAP_LEADING_LENGTH defines the maximum length of the leading trace of a CERACAP capacitor.

Default Severity

ERROR

gap_between_split_plane

Checking for the gap between two splitted power/ground plane shape in the same plane. In PCB, the power/ground plane is splitted into several plane/shape with different power/ground supply. And the gap between them must be large enough to decrease the crosstalk.

This rule check the gap between any two splitted power/ground plane shape in the same plane. Any power/ground plane shape which gap between them is less than the user-defined value will be reported. If such two shapes have the same voltage, then a warning will be invoked. If such two shapes have different voltage, then an error will be invoked.

Properties Used

VOLTAGE

Variables Used

MIN_SEPARATION_OF_SPLITPLANE

Required Data

The VOLTAGE property identify the power/ground net in design. Any net with VOLTAGE property and the value 0.0 of VOLTAGE property is Ground Net, and the value greater than 0.0 is Power Net. Any plane shape which attached on the power net is power plane shape, and attached on the ground net is ground plane shape.

The MIN_SEPARATION_OF_SPLITPLANE properties identify the minimum distance between two plane shape.

Default Severity

ERROR

WARNING

matched_via_number_diffpair

This rule checks and compares the numbers of vias between the two traces of a differential pair. The goal is to ensure that two traces of every differential pair have the same number of vias.

Properties Used

DIFFERENTIAL_PAIR

Variables Used

None

Required Data

The DIFFERENTIAL_PAIR property identifies the differential pairs of the board and is checked by this rule.

Default Severity

ERROR

symmetry_of_via_position_of_diffpair

This rule checks if the vias on the two traces of one differential pair are at symmetrical locations. The goal is to ensure that the via positions of two traces for every differential pair are symmetrical to each other.

Properties Used

DIFFERENTIAL_PAIR

Variables Used

MAX_NON_SYMMETRY_DISTANCE

Required Data

The DIFFERENTIAL_PAIR property identifies the differential pairs of the board and is checked by this rule.

Default Severity

ERROR

termination_of_diffpair

This rule checks the termination existence and position of each differential signal. The goal is to ensure that every differential pair has termination at the receiver end.

Properties Used

DIFFERENTIAL_PAIR
DIFFERENTIAL_TERMINATION_RESISTOR

Variables Used

MAX_TERMINATION_DISTANCE_AT_RECEIVER

Required Data

The DIFFERENTIAL_PAIR property identifies the differential pairs on the board and is checked by this rule.

The variable MAX_TERMINATION_DISTANCE_AT_RECEIVER specifies the maximum distance between the termination and the receiver pin.

Default Severity

ERROR

EMControl User Guide

EMControl Rules

EMControl Predicates

Overview

Predicates are the routines supplied by Cadence to perform the various tasks required by the EMControl rules. You can use the Cadence predicates to write your own rules. Predicates call functions that are written in SKILL or the C language.

This appendix describes the physical environment objects and predicates supplied with EMControl.

Physical Environment Objects and Predicates

Table C-1 lists the primary objects that exist in the layout physical environment that can be referenced by the EMControl predicates.

Table C-1 Objects Accessed by EMControl Predicates

Object	Description
design	Specifies the complete design or a user-specified portion of the design
component	Specifies an instance of a physical component in the design
via	Defines an opening in a dielectric layer that connects adjacent conductor layers
net	Specifies a signal corresponding to a logical net in the design
pin	Specifies an interface terminal of a component
shape	Specifies a shape, rectangle, or filled rectangle in the design
polygon	Specifies a polygon in the layout x-y plane

The following sections describe the predicates that allow you to access information in the layout database. Each section lists and describes the predicates that apply to one of the

object categories described in Table B-1. Within each section, predicates appear in alphabetical order.

A call to a predicate takes one or more arguments and returns a list comprised of one or more values.

General-Purpose Predicates

This section describes predicates that you can apply to values in the layout database.

append

```
list append(list1, list2)
```

Appends `list2` to `list1`, modifying `list1`. The lists can be object lists or value lists.

atof

```
value atof(string)
```

Converts the number `string` to a real number.

atoi

```
value atoi(string)
```

Converts the number `string` to an integer number.

compareProfile

```
value compareProfile(valProfile, valLimit, valFlag, valLength)
```

Compares a list of frequency and value pairs with a predefined profile limit. It returns `true` if it exceeds the limit.

- `valProfile` is the value returned from the `getSpectralContent` predicate.
- `valLimit` is a user-defined parameter.
- The value of `valFlag` varies depending on the format in which you specify `valLimit`:
 - When `valLimit` is given in dBuV format, `valFlag` = 1. For example:
`#define valLimit "30MHz:49.5dBuV 88MHz:54.0dBuV 216MHz:56.5dBuV 960MHz:60dBuV"`
 - When `valLimit` is given in V format, `valFlag` = 0 if. For example:


```
#define valLimit "0MHz:5.0V 30MHz:0.9V 75MHz:0.7V 500MHz:0.1V"
```

- The value of Length varies depending on the rule being used:
 - When used in the `spectral_content` rule, `valLength = 0.0`
 - When used in the `single_diff_mode_EMI` rule, `valLength` is the exposed trace length
 - When used in the `sum_diff_mode_EMI` rule, `valLength < 0.0`

component

```
value component(value)
```

Converts the component ID `value` to the component itself.

convertUnits

```
value convertUnits(value, unit)
```

Converts `value` into the units specified by `unit`.

count

```
value count(list)
```

Returns the number of elements in the list. The list can be an object list or a value list.

float

```
value float(value)
```

Converts the integer `value` to a real number.

format

```
value format(value, value)
```

Formats a string (the first `value`) such that its length is the integer specified by the second `value`. When the integer is less than the string length, the string is truncated at the trailing end. Otherwise, extra blank characters are inserted.

getAllSubstrings

```
value getAllSubstrings(string1, startToken, endToken)
```

Returns all substrings of `string1` that occur between `startToken` and `endToken`. The returned string does not include `startToken` and `endToken`.

For example:

```
getAllSubstrings("VCC=VEE;VDD=VSS;", "=", ";")
```

returns a list containing VEE and VSS.

getDistance

```
value getDistance(value ,value ,value ,value)
```

Finds the distance between two points, (x1,y1) and (x2,y2), where x1 and y1 are the first and second `values` and x2 and y2 are the third and fourth `values` three and four.

getShapeOutline

```
value getShapeOutline("class_name/subclass_name")
```

Returns the outline of the shape specified by the `class_name` and `subclass_name`.

getValueOneOfList

```
value getValueOneOfList(ARL_optional_list)
```

Returns the list of values that are passed as an optional list of values to `getValueOneOfList`. This predicate is for printing an optional list specified by the user in the rule message field.

For example:

```
val1 = getValueOneOfList( { "A", "B", "C", "D" } )
```

In the above example, `val1` is the list containing the values: "A", "B", "C", "D".

index

```
value index(list, value)
```

Returns the index to the first occurrence of `value` in the `list`.

integer

```
value integer(realValue)
```

Returns the integer value of the `realValue`.

isFalse

`value isFalse(boolean)`

Returns *t* if the `boolean` value is false. Returns *nil* if the `boolean` value is true.

isGroundPlane

`value isGroundPlane(value)`

Returns *t* if the plane specified by `value` is Ground plane, return *nil* otherwise.

isNull

`value isNull(object)`

Determines whether the value of `object` is null. If so, `isNull` returns *t*. Otherwise, it returns *nil*.

isPowerPlane

`value isPowerPlane(value)`

Returns *t* if the plane specified by `value` is power plane, returns *nil* otherwise.

isTrue

`value isTrue(boolean)`

Returns *t* if the `boolean` value is true. Returns *nil* if the `boolean` value is false.

matchCapType

`value matchCapType(captypel, captype2)`

Compares the capacitor types of two bypass capacitors. Returns *t* if they match. Otherwise, it returns *nil*.

The capacitor type for bypass capacitors must be in the following format:

`<device type>:<cap value>:<tolerance>`

max

`value max(list)`

Returns the maximum value of the `list`.

min

`value min(list)`

Returns the minimum value of the `list`.

nth

`value nth(list, integerValue)`

Returns the element in the `list` at the position indexed by `integerValue`.

ntoa

`value ntoa(value)`

Converts the numerical argument into a string.

skipWhiteSpaces

`value skipWhiteSpaces(string)`

Strips spaces, tabs, and linefeeds from the specified *string*, unless they are enclosed within single quotation marks (for example, `'\t'` is not stripped). Returns the resulting string.

sqrt

`value sqrt(value)`

Calculate the numerical argument into square root.

unique

`value unique(list)`

Returns a list of the elements in the input *list*. Multiple occurrences of any element are filtered out of the returned list. For example, for the following input:

```
unique(2, 3, 3, 4, 6, 2, 3)
```

`unique` returns the following list:

```
(2, 3, 4, 6)
```

upperCase

`value upperCase(string)`

Converts the input *string* to a string that has all uppercase letters.

userAlert

value userAlert(*string*)

Prompts the user message provided by *string*. Always returns *t*.

Design Predicates

This section describes predicates that you can apply to the design object.

component

component component(*design*)

Returns all the component instances in *design*.

getDiffPSegs

net getDiffPSegs(*design*)

Returns the trace segment for each differential pair that has the maximum length in the *design*. The trace segment is part of the differential pair net. Differential pair nets are nets with the same DIFFERENTIAL_PAIR property assignment.

getGroundNet

net getGroundNet(*design*)

Returns the ground net in *design*.

getLayer

value getLayer(*design*)

Returns all layer in *design*.

getLayerSeparation

value getLayerSeparation(*design*, "layer1", "layer2")

Returns the vertical layer separation distance between `layer1` and `layer2` of `design`. `layer1` and `layer2` are the names of the layers. Layer names must be enclosed in quotation marks.

getLowerLeft

`value getLowerLeft(design)`

Returns the lower left coordination of the design.

getNominalNetWidth

`value getNominalNetWidth(design)`

Returns the nominal etch width of the nets in `design`. The value is returned in user units.

getPlaneOutline

`value getPlaneOutline(design)`

Returns the outline of the design plane.

getPowerNet

`net getPowerNet(design)`

Returns the power net in `design`.

getPropertyValue

`value getPropertyValue(design, propertyName)`

Returns the value of `propertyName` attached to `design`. Returns `nil` if `propertyName` is not found.

hasProperty

`value hasProperty(design, propertyName)`

Returns `t` if the property specified by `propertyName` is attached to the design. Otherwise, it returns `nil`.

name

`value name(design)`

Returns the design (board) name.

net

net net(design)

Returns all the nets in design.

pin

pin pin(design)

Returns all the component pins in design.

shape

shape shape(design)

Returns all the shape objects in design.

splitDeisgn

polygon splitDesign(design,xinc,yinc,startx,starty)

Splits the design.

via

via via(design)

Returns all the vias in design.

Component Predicates

This section describes predicates that you can apply to an object that is a component.

getCenter

value getCenter(component)

Returns the center of the component as a list of two numbers representing the coordinate.

getComp2CompDistance

value getComp2CompDistance(component1, component2)

Returns the distance between component1 and component2.

getCompC1eqTwiceC2eqTwiceC3

value getCompC1eqTwiceC2eqTwiceC3(component, distance, *propertyValue*)

Returns all the components, in the area represented by the bounding box, that have the value for the EMC_COMP_TYPE property specified by *propertyValue*.

getCompC1eqTwiceC2eqTwiceC3 determines the bounding box area as follows:

((x1-distance, y1-distance) (x2+distance, y2+distance))

where ((x1,y1) (x2, y2)) specifies the diagonal coordinates of the bounding box of the component.

If the value of *propertyValue* is ALL, this predicate returns all of the components in the calculated area.

getCompC1eqTwiceC2eqTwiceC3 is used to find the bypass capacitors for a given component. You should use the property EMC_COMP_TYPE with an assigned value of BYPASS_CAP with this predicate.

Once all the bypass capacitors for a component are found, the predicate gets the value for the VALUE property on the capacitors, and looks for all the capacitors that satisfy the following equation:

1st cap = twice (2nd cap), 2nd Cap = twice (3rd cap)

getCompC1eqTwiceC2eqTwiceC3 returns the list of the capacitors that match the equation in a fixed order:

- The first element of the list is the C1 capacitors.
- The second element is the C2 capacitors.
- The third element is the C3 capacitors.

For example, in a case where a component has five bypass capacitors with the following values:

- C1 = 0.01UF
- C2 = 0.03UF
- C3 = 0.02UF

- C4 = 0.01UF
- C5 = 0.04UF

getCompC1eqTwiceC2eqTwiceC3 finds all the capacitors that satisfy the equation described above. For the example, these are:

- 1st Cap = C5
- 2nd Cap = C3
- 3rd Cap = C1, C4

For the example, *getCompC1eqTwiceC2eqTwiceC3* returns the following list:

(C5, C3, (C1, C4))

getCompInArea

component *getCompInArea*(*component*, *distance*, *propertyValue*)

Returns all the components with the EMC_COMP_TYPE definition specified by *propertyValue* in the area represented by the bounding box. *getCompInArea* determines the area as follows:

((*x1-distance*, *y1-distance*) (*x2+distance*, *y2+distance*))

where ((*x1*,*y1*) (*x2*, *y2*)) specifies the diagonal coordinates of the bounding box of the component.

If the value of *propertyValue* is ALL, *getCompInArea* returns all of the components in the calculated area.

getComponentType

value *getComponentType*(*component*)

Returns a string that provides the component type of *component*. The component type is returned in the following format:

<device type>:<value>:<tolerance>

getComponentType is usually used to get the bypass capacitor type.

getCompsConnToComp

component *getCompsConnToComp*(*component*, *bypassDiscreteFlag*)

Returns all the components connected to `component`. If `bypassDiscreteFlag` is `t`, `getCompsConnToComp` ignores the discrete components connected to `component` and scans further until it finds a nondiscrete component. If `bypassDiscreteFlag` is `false`, `getCompsConnToComp` stops scanning at any discrete component.

A component is considered to be connected to another component if the connecting net is a signal net. All nonsignal nets must have the VOLTAGE property assigned to them.

getDesign

design `getDesign(component)`

Returns the design of `component`.

getGroundPin

pin `getGroundPin(component)`

Returns all ground pins of `component`.

getLayer

value `getLayer(component)`

Returns TOP or BOTTOM depending on the location of the component on the board.

getNetsConnToComp

net `getNetsConnToComp(component)`

Returns all nets connected to `component`.

getNetsInArea

net `getNetsInArea(component)`

Returns all nets in the bounding box of `component`.

getPin

pin `getPin(component)`

Returns all pins of `component`.

getPolygon

polygon getPolygon(component)

Returns a polygon representing the package bounding box of the component.

getPowerPin

pin getPowerPin(component)

Returns all power pins of component.

getPropertyValue

value getPropertyValue(component, propertyName)

Returns the value of *propertyName* attached to *component*. Returns *nil* if *propertyName* is not found.

hasProperty

component hasProperty(component, propertyName)

Returns *t* if the property specified by *propertyName* is attached to the component. Otherwise, it returns *nil*.

isCompClass

value isCompClass(component, compClass)

Returns *t* if *component* has the component class specified by *compClass*. Legal values for *compClass* are IO, IC, or DISCRETE.

isConnectedToNet

component isConnectedToNet(component, net)

Returns *t* if *component* is connected to *net*. Otherwise, it returns *nil*. A connector has a component class of IO.

isConnector

component isConnector(component)

Returns *t* if *component* is a connector. Otherwise, it returns *nil*. A connector has a component class of IO.

name

value name(*component*)

Returns the reference designator of *component*.

Net Predicates

This section describes predicates that you can apply to an object that is a signal (*net*).

getClineLength

value getClineLength(*net*)

Returns the length of the cline.

getClineOfSeg

net getClineOfSeg(*net*)

Returns the parent cline of the cline segment.

getCompInArea

component getCompInArea(*netSeg*, *distance*, *propertyValue*)

Returns all the components with the EMC_COMP_TYPE definition specified by *propertyValue* in the area represented by the bounding box.

The *getCompInArea* predicate determines the area as follows:

((*x1-distance*, *y1-distance*) (*x2+distance*, *y2+distance*))

where ((*x1*, *y1*) (*x2*, *y2*)) specifies the diagonal coordinates of the bounding box of the net segment.

If the value of *propertyValue* is ALL, *getCompInArea* returns all of the components in the calculated area.

Note: The *netSeg* argument must represent a trace segment and not the complete net.

getCompsConnToNet

component getCompsConnToNet(*net*)

Returns all the components connected to *net*.

getDelayTimeUnits

value getDelayTimeUnits(*net*)

Returns the delay time units for *net*.

getDesign

design getDesign(*net*)

Returns the design of *net*.

getEtchLength

value getEtchLength(*net*, *shape*)

Returns the etch length of *net* over *shape*.

getEtchLength

value getEtchLength(*net*, *polygon*)

Returns the etch length of *net* over *polygon*.

getGroundNetMatched

net getGroundNetMatched(*net*)

Every power net should have a ground net matching it. Meanwhile, every power plane has a ground plane to match it to reduce the noise. This predicate will return the matched ground net for the argument power net.

getManhattanLen

value getManhattanLen(*net*)

Returns the manhattan length of *net*.

getManhattanLength

value getManhattanLength(*net*, *shape*)

Returns the manhattan length of *net* over *shape*.

getMaxOverUnderShoot

value getMaxOverUnderShoot(*net1*)

Returns the maximum undershoot and overshoot values for the net specified by *net1*.

getMaxPropDelay

value getMaxPropDelay(*net*, *units*)

Returns the maximum propagation delay for *net* in the units specified by *units*.

getMaxRiseTime

value getMaxRiseTime(*net*, *units*)

Returns the slew rate of the fastest signal that passes through the net in the units specified by *units*.

getMinDistance

value getMinDistance(*net*, *lineSegment*)

Returns the minimum distance between *net* and *lineSegment*. *lineSegment* can be any other net or the design outline.

getMinNetWidth

value getMinNetWidth(*net*)

Returns the minimum width of *net* in user units.

getMinPathWidth

value getMinPathWidth(*net*)

Returns the minimum width of *Cline* in user units.

getMinWaveLength

value getMinWaveLength(*net*)

Returns the minimum wavelength of the signal on the net.

getNetCapacitance

value getNetCapacitance(*net*, *units*)

Returns the net capacitance for *net* in the units specified by *units*.

getNetLenOnLayer

value getNetLenOnLayer(*net*, *layerName*)

Returns the physical length of *net* on the layer specified by *layerName*.

getNetOfSeg

net getNetOfSeg(*net1*)

Returns the parent net of the net segment *net1*.

getNetOnLayer

value getNetOnLayer(*net*, *layerName*)

Returns all the clines of *net* on the layer specified by *layerName*.

getNetSegOnLayer

net getNetSegOnLayer(*net*, *layerName*)

Returns all trace segments of the parent *net* on the layer specified by *layerName*.

getNetShape

value getNetShape(*net*)

Returns the shape that is a part of *net*. Normally, a supply net has a shape as part of the net.

getNetsInArea

net getNetsInArea(*net*, *distance*)

Returns all nets found within the specified *distance* of *net*.

getNetSourceImpedance

value getNetSourceImpedance(*net*, *resUnit*)

Returns the source impedance for `net` in the units specified by `resUnit`.

getParallelNetsToSeg

```
net getParallelNetsToSeg(net1, distance)
```

Returns all net segments that are parallel to the net segment `net1` and are within the specified `distance` from `net1`. A net segment is considered to be parallel to another if the segments are parallel for at least 85 percent of the segment length of `net1`.

getParallelTraces

```
net getParallelTraces(net, value, value)
```

Returns all cline segments that are parallel to the given cline segment (the first argument, `net`) and are located within a specified distance (the second argument, `value`) in the specified direction (the third argument, `value`). The direction can be either UL (for upper left) or LR (for lower right).

getPathsOnNet

```
via getPathsOnNet(net)
```

Returns all Clines on `net`.

getPinsOnCline

```
pin getPinsOnCline(net)
```

Returns all pins on Cline.

getPinsOnNet

```
pin getPinsOnNet(net)
```

Returns all pins on `net`.

getPinsOnSegment

```
pin getPinsOnSegment(net)
```

Returns all pins on segment.

getPropertyValue

value `getPropertyValue(net, propertyName)`

Returns the value of `propertyName` attached to the `net`. Returns *nil* if `propertyName` is not found.

getSegLength

value `getSegLength(netSeg)`

Returns the length, in user units, of the line segment specified by `netSeg`. The `netSeg` argument must represent a trace segment and not the complete net.

getTraceLengthInWindow

value `getTraceLengthInWindow(net, net, value)`

Returns the length of cline (the first argument, `net`) enclosed in a rectangular window. The rectangle has a specified length (the second argument, `net`) and a height two times the specified `value` (the third argument). The length bisects the window.

For example, specify `c1`, `c2`, and `d` as the arguments. The length of cline is `c1`. It is enclosed in a rectangular window that has `c2` as its length. `c2` also bisects the window. The height of the window is two times `d`.

getViasOnCline

via `getViasOnCline(net)`

Returns the vias on the cline.

getViasOnNet

via `getViasOnNet(net)`

Returns all vias on `net`.

getViasOnSegment

via `getViasOnSegment(net)`

Returns all vias on segment.

getXtalkBetweenNets

value getXtalkBetweenNets(*net1*, *net2*)

Returns the crosstalk contribution in mV on *net2* caused by *net1*.

hasProperty

net hasProperty(*net*, *propertyName*)

Returns the net if the property specified by *propertyName* is attached to *net*. Otherwise, it returns *nil*.

isGroundNet

value isGroundNet(*net*)

Returns *t* if *net* is a ground net.

isNetTerminated

value isNetTerminated(*net*)

Returns *t* if *net* is terminated.

isPowerNet

value isPowerNet(*net*)

Returns *t* if *net* is a power net.

isPowerGroundNetMatched

value isPowerGroundNetMatched(*net*, *net*)

Return *t* if the arguments, the power net and ground net, are matched. Otherwise *nil* is returned.

name

value name(*net*)

Returns the name of *net*.

Pin Predicates

This section describes predicates that you can apply to an object that is a pin.

getActualArea

value getActualArea(*pin*, *propertyValue*, *sens_distance*)

Returns the `actual_area` for the IC with respect to the power `pin` and the nearest bypass capacitor of the IC. The nearest bypass capacitor is identified as the nearest component within the distance specified by `sens_distance` that has the `EMC_COMP_TYPE` property value specified by `propertyValue`.

The actual area is calculated differently for routed and unrouted cases:

Unrouted Cases:

For these cases, the nearest bypass capacitor is either not connected to the power pin and ground pin or the connection is partial. A triangle is constructed, with the vertices being

- The power pin
- The nearest ground pin on the IC
- The bypass capacitor pin most distant from the power pin

The area of this triangle defines the actual area.

Routed Cases:

For these cases, the bypass capacitor is connected to the power and ground pins by etch. The etch that connects the power pin to the capacitor is called `pwr_etch`. Similarly, the etch that connects the ground pin to the capacitor is called `gnd_etch`.

A minimum bounding box is constructed. The box contains

- The power pin of the IC
- The ground pin of the IC
- The package geometry of the bypass capacitor
- The `pwr_etch`
- The `gnd_etch`

The calculated area of this bounding box defines the actual area.

getAllegroPinUse

```
value getAllegroPinUse(pin)
```

Returns the PINUSE of the specified `pin`. `getAllegroPinUse` extracts the PINUSE from the layout database. If you have explicitly attached the PINUSE property to a pin, the value of the PINUSE property overrides the PINUSE attached to the component definition.

getCompInArea

```
component getCompInArea(pin, distance, propertyValue)
```

Returns all components with the EMC_COMP_TYPE definition specified by `propertyValue` in the area represented by the bounding box. `getCompInArea` determines the area as follows:

```
((x1-distance, y1-distance) (x2+distance, y2+distance))
```

where `((x1, y1) (x2, y2))` specifies the diagonal coordinates of the bounding box of the pin.

If the value of `propertyValue` is ALL, `getCompInArea` returns all of the components in the calculated area.

getComponent

```
component getComponent(pin)
```

Returns the component of which the specified `pin` is a part.

getDistance

```
value getDistance(pin, component)
```

Returns the distance from the specified `pin` to the `component`.

getDistance

```
value getDistance(pin1, pin2)
```

Returns the pin-to-pin distance.

getLoopDistance

```
value getLoopDistance(pin, byp_cap_prop, sens_distance)
```

Returns the loop distance from the pin to the nearest bypass capacitor to the ground pin, then back to the pin.

The bypass capacitors must have the EMC_COMP_TYPE property definition specified by `byp_cap_prop` (for example, `BYPASS_CAP`), and they must be within the distance specified by `sens_distance` from the component boundary.

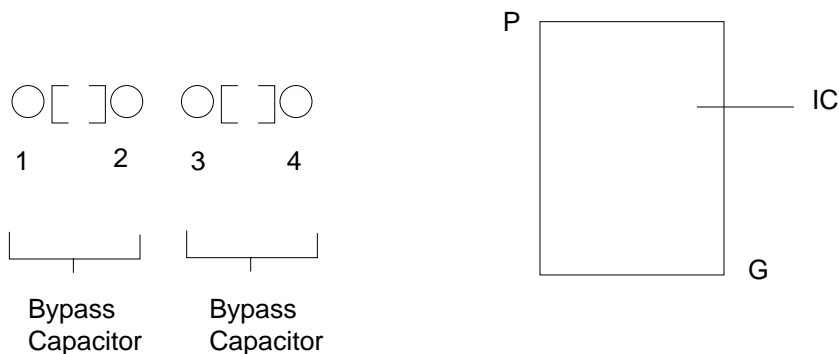
For an IC marked `EMC_CRITICAL_IC`, the loop distance is calculated as follows:

1. From the power pin of the IC to the nearest bypass capacitor pin
2. From that bypass capacitor pin to the other (second) bypass capacitor pin for that capacitor
3. From that second bypass capacitor pin to the nearest ground pin on the same IC
4. From the nearest ground pin on the same IC back to the original power pin

For example, in the following figure, P identifies the location on the IC of the power pin and G identifies the ground pin. Pin 3 is assumed to be attached to the ground plane. The pin-to-pin connections that the rule needs to calculate are P-4, 4-3, 3-G, and G-P.

Note that 1-2 is ignored because, even if it is connected to power and ground, the check is always made on the closest bypass capacitor.

Example of Checking Power-Ground Loop Distance



In the cases of P-4 and 3-G, real etch might or might not exist. In such cases, the real etch length is used for the loop distance calculations; otherwise, manhattan length is used. In the case of 4-3 and G-P, no real etch can exist in the layout, and the rule always uses pin-center to pin-center distances.

getMinRiseFallTime

value getMinRiseFallTime(*pin*, *units*)

Returns the minimum rise time or fall time, whichever is higher, of *pin* in the specified *units*.

getNetName

value getNetName(*pin*)

Returns the name of the net connected to the specified *pin*.

getNetOfPin

net getNetOfPin(*pin*)

Returns the net that *pin* is a part of.

getPin2PinResistance

value getPin2PinResistance(*Pin1*, *Pin2*)

Returns the resistance between *Pin1* and *Pin2*.

getPin2PlaneResistance

value getPin2PlaneResistance(*pin*, *planeShape*)

Returns the resistance between *pin* and the supply plane specified by *planeShape*.

getPropertyValue

value getPropertyValue(*pin*, *propertyName*)

Returns the value of *propertyName* attached to *pin*. Returns *nil* if *propertyName* is not found.

getReferenceArea

value getReferenceArea(*pin*)

Returns the reference area of the IC with respect to the specified power *pin*. The reference area is calculated as follows:

- The nearest ground pin for this power pin on the IC is identified.

- C is taken to specify the distance between this ground pin and the power pin.
- A right-angled isosceles triangle is constructed, with C as the hypotenuse.
- The area of this triangle defines the reference area.

getShapesConnToPin

shape getShapesConnToPin(*pin*)

Returns all shapes physically connected to *pin* through thermal-relief clines.

getSpectralContent

value getSpectralContent(*pinDriver*, *valLength1*, *valDuty*, *valCycle*, *valClassNum*)

Performs pulse simulations for all the drivers on the xnet that connect to *pinDriver*. Returns the result of the simulation.

- *pinDriver* is a driver pin
- *valLength1* is the trace length in meters:
 - When *valLength1* = 0, the waveform at the driver pin is transformed to the frequency domain by means of a DFT.
 - When *valLength1* > 0, the emission level caused by the net is computed using a closed-form equation and a list of frequencies.

- *valDuty* is a parameter containing the clock frequency and duty cycles.

For example, *valDuty* could be the EMControl variable PULSE_DUTY_CYCLE. For example:

```
#define PULSE_DUTY_CYCLE "300MHz:50%, 75MHz:50%, 50MHz:50%"
```

- *valCycle* is the maximum number of cycles the circuit simulator would simulate.
- *valClassNum* is the class of the net (for example, CLASS1).

getVoltageSwing

value getVoltageSwing(*pin*)

Returns the voltage swing for *pin*.

hasProperty

pin hasProperty(*pin*, *propertyName*)

Returns t if the property specified by `propertyName` is attached to `pin`. Otherwise, it returns *nil*.

isBypassCompPin

`value isBypassCompPin(pin)`

Returns t if the pin is a pin of bypass capacitor.

isGroundPin

`value isGroundPin(pin)`

Returns t if the pin connected to the ground net. Otherwise, it returns *nil*.

isLoopRouted

`value isLoopRouted(pin, byp_cap_prop, sens_distance)`

Returns t if the loop is routed. Otherwise, it returns *nil*. A loop is considered to be routed if there is real etch between the power pin and the bypass capacitor pin or between the bypass capacitor pin and the ground pin.

isPinConnected

`value isPinConnected(pin1, pin2)`

Returns t if the two pins are connected within a branch. Otherwise, it returns *nil*.

isPowerPin

`value isPowerPin(pin)`

Returns t if the pin connected to a power net. Otherwise, it returns *nil*.

matchObjectByPropertyValue

`pin matchObjectByPropertyValue(pin, propertyName, propertyValue)`

Returns t if `pin` has the specified `propertyName` attached, with the value specified by `propertyValue`. Otherwise, it returns *nil*.

name

`value name(pin)`

Returns the name of the specified `pin`

Via Predicates

This section describes predicates that you can apply to an object that is a via.

getNet

```
net getNet(via)
```

Returns Net that are connected by *via*.

getPathsConnToVia

```
net getPathsConnToVia(via)
```

Returns all Cline physically connected to *via*.

getShapesConnToVia

```
shape getShapesConnToVia(via)
```

Returns all shapes physically connected to *via* through thermal-relief clines.

getSigCnxnLayer

```
value getSigCnxnLayer(via)
```

Returns all layers that are connected by *via*.

getViaLayer

```
value getViaLayer(via)
```

Returns layers that are connected by *via*.

getViaLocation

```
value getViaLocation(via)
```

Returns the location of the via.

isGroundVia

value isGroundVia(*via*)

Returns *t* if the via is connected to the ground plane/shape.

isMountingHole

value isMountingHole(*via*)

Returns *t* if the via is a mounting hole.

name

value name(*via*)

Returns the coordinates of *via*.

Shape Predicates

This section describes predicates that you can apply to an object that is a shape.

getClass

value getClass(*shape*)

Returns the class of the shape object.

getIntersectAngle

value getIntersectAngle(*shape*, *net*)

Returns the list of angles of intersection of the shape with a cline segment.

getDistance

value getDistance(*shape*, *shape*)

Returns the minimum distance of two shapes.

getNet

net getNet(*shape*)

Returns the net of which *shape* is a part.

getPolygon

polygon getPolygon(*shape*)

Returns a polygon representing the geometry of the shape.

getPropertyValue

value getPropertyValue(*shape* , *value*)

Returns the *value* of the property of *shape*.

getShapeSegments

shape getShapeSegments(*shape*)

Returns the list of boundary segments that make up *shape*.

getSubClass

value getSubClass(*shape*)

Returns the subclass of *shape*.

getUnbypassedRegionSegments

shape getUnbypassedRegionSegments(*shape*, *value*, *value*, *value*, *value*)

Returns the boundary segments of an EMC region that do not have enough decoupling capacitors. A decoupling capacitor is identified by having the property EMC_COMP_TYPE = BYPASS_CAP attached.

- The second argument is the sensitive distance.
- The third argument is the maximum separation allowed between decoupling capacitors along the boundary.
- The fourth argument is the bypass capacitor property value (that is, BYPASS_CAP).
- The fifth argument is the critical classes of regions defined (for example, "CLASS1 CLASS2 CLASS3 CLASS4 CLASS5").

getUnfencedRegionSegments

shape getUnfencedRegionSegments(*shape*, *value*, *value*, *value*, *value*)

Returns the boundary segments of an EMC region that are not fenced properly.

- The second argument is the fence-sensitive distance.
- The third argument is the minimum overlap required between a fence and the boundary segment of the room.
- This value is given as a fraction.
- The fourth argument is the fence component type (that is, EMC_COMP_TYPE = FENCE).
- The fifth argument is the critical classes of regions defined (for example, "CLASS1 CLASS2 CLASS3 CLASS4 CLASS5").

hasProperty

shape hasProperty(*shape* , *value*)

Returns *shape* if the object has the property given by *value*. Otherwise, it returns *nil*.

isCleanGround

shape isCleanGround(*shape*)

Returns *shape* if the object is a clean ground shape. Otherwise, it returns *nil*. A clean ground shape is a plane ground shape enclosed by a void.

isEmcRegion

shape isEmcRegion(*shape*)

Returns *shape* if the object is an EMC region. Otherwise, it returns *nil*. EMC regions reside in BOARD_GEOMETRY/ [TOP/BOTTOM/BOTH]_ROOM and have segments parallel to the x or y axis of the layout plane.

isGroundPlaneShape

value isGroundPlaneShape(*shape*)

Returns *t* if the object is a plane ground shape. Otherwise, it returns *nil*. Plane ground shapes lie in ETCH class with a PLANE subclass type. They are part of a net with VOLTAGE = 0.

isPlaneShape

shape isPlaneShape(*shape*)

Returns *shape* if the object is a plane shape. Otherwise, it returns *nil*. Plane shapes lie in ETCH class with a PLANE subclass type.

isPowerPlaneShape

value isPowerPlaneShape(*shape*)

Returns *t* if the object is a plane power shape. Otherwise, it returns *nil*. Plane ground shapes lie in ETCH class with a PLANE subclass type.

They are part of a net with VOLTAGE \neq 0.

isRectangle

shape isRectangle(*shape*)

Returns *shape* if the object is an Allegro PCB rectangle element. Otherwise, it returns *nil*.

name

value name(*shape*)

Returns the name of the shape.

Polygon Predicates

This section describes predicates that you can apply to an object that is a polygon.

getArea

value getArea(*polygon*)

Returns the area of the polygon.

getCenter

value getCenter(*polygon*)

Returns the center of the bounding box of the polygon.

getClineSegsInArea

net getClineSegsInArea(*polygon*)

Returns all cline segments that lie fully or partially inside the polygon.

getCompInArea

component getCompInArea(*polygon*)

Returns all components that lie fully or partially inside the polygon.

getIntersectionPolygon

polygon getIntersectionPolygon(*polygon* , *polygon*)

Returns the intersection of the polygons.

getLength

value getLength(*polygon*)

Returns the length of the polygon bounding box.

getRectangle

polygon getRectangle(*value* ,*value* , *value* ,*value* ,*value*)

Returns a rectangular polygon with the dimensions specified by the arguments.

- The first argument is x1.
- The second argument is y1.
- The third argument is x2.
- The fourth argument is y2.
- The fifth argument is b.

The `getRectangle` predicate constructs a rectangle whose center line runs from point (x1, y1) to point (x2, y2) and whose width equals b.

getShapesInArea

shape getShapesInArea(*polygon*,*value*,*value*)

Returns all shapes in the area defined by the polygon that have a class matching the second argument and a subclass matching the third argument.

getUnionPolygon

polygon getUnionPolygon(*polygon* , *polygon*)

Returns the union of the polygons.

name

value name(*polygon*)

Returns the name of the polygon as *poly:Bbox*.

splitPolygon

polygon splitPolygon(*polygon*)

Returns a list of four polygons representing the four quadrants of the bounding box of the polygon.

