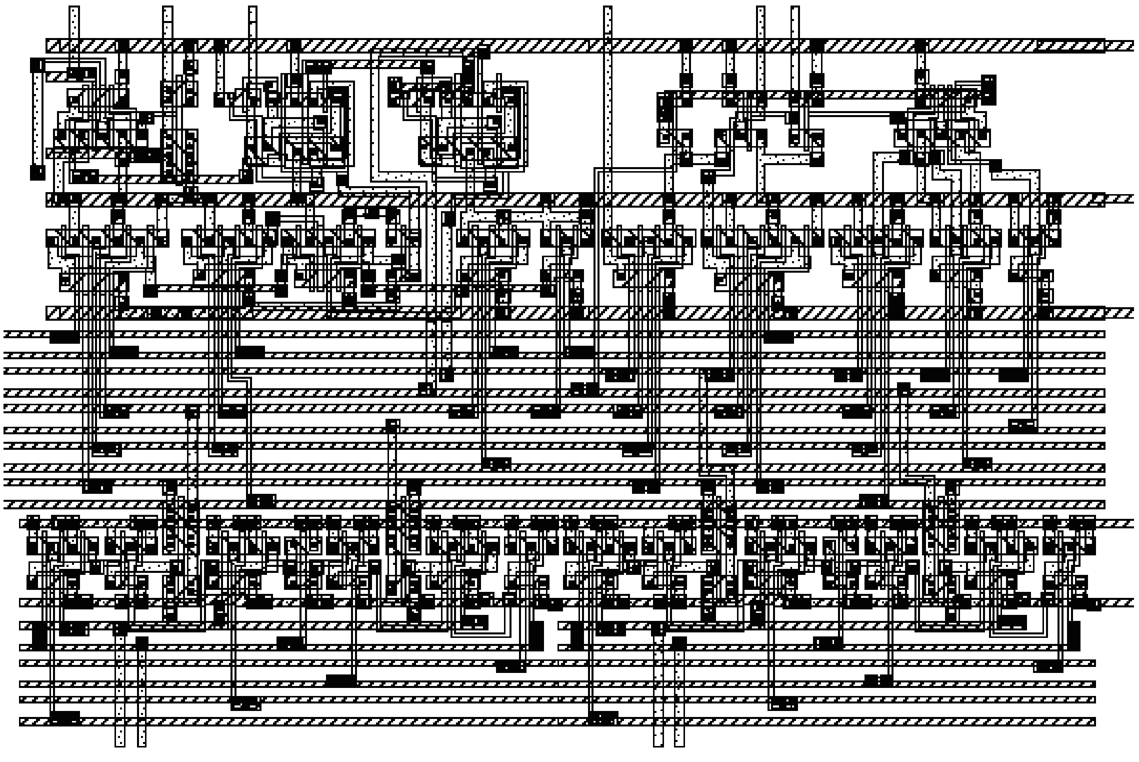
ICED32™

NLE Circuit Extractor and LVS Comparison Utility

Reference Manual



Version 1.xx

IC Editors, Inc.

Acknowledgments:

The majority of this manual was written or revised by Ference Professional Services in Sonora, CA. The original documentation was created by Richard Brandshaft and Pratapa Yeddula.

Michael Gentry of MGC, Inc. created the layout that is used on the cover and as an example in this manual. It is a section of a CMOS simulation of a 74181 4-bit ALU.

Table of Contents

Introduction

The LVS program from IC Editors, Inc. is a fast netlist-netlist comparison program. It is used to verify whether or not two netlists represent the same semiconductor circuit. Usually, one of the netlists is generated from the physical circuit layout and the other corresponds to a schematic used for circuit simulation. We refer to this type of comparison as an LVS (Layout Vs. Schematic) comparison. The LVS can also be used to compare two schematic netlists (SVS) or two layout netlists (LVL).

When you want to perform either an LVS or LVL comparison, you create the layout netlist by running the NLE (Net List Extractor) program on the data created with the ICED32™ layout editor.

The NLE creates the layout netlist from the shapes in an ICED32™ cell by using rules you define in an ASCII rules file. These rules direct the circuit recognition process. Other rules direct the optional ECC (Electrical Connections Checks) which can easily find problems such as shorts between nets and open circuits.

This manual is organized into the following sections:

"Getting Started" covers the program requirements and installation. This section also includes a brief tutorial covering the basic steps for preparing the input files and running the NLE and LVS programs.

"NLE Basics" describes how the NLE program operates. The NLE and the DRC program (available separately from IC Editors, Inc.) have many similarities. Both are rules-driven programs that manipulate layout data. Many of the rules described here are also covered in the DRC manual. If you are not familiar with the DRC, you should read this section carefully.

"NLE Circuit Recognition" covers the rules the NLE uses to build the layout netlist from the layout data.

"Running the NLE" contains details on executing the NLE rules compiler and the NLE program.

"LVS Input Files" explains the format of the input files for the LVS comparison. The LVS requires three input files: two netlist files and a control file. A netlist can be either a schematic netlist or a layout netlist.

> **"Schematic Netlists"** contains details on preparing a schematic netlist for comparison. The LVS accepts schematic netlists written in the PSPICE, HSPICE, SPICE, or CDL (Cadence[1] Circuit Description Language) formats. Your original schematic netlist file must be combined with models for each unique type of device found in the netlist.
>
> **"Layout Netlists"** describes how to combine the binary layout netlist generated by the NLE with the other information required by the LVS. Device models, similar to those required in schematic netlist, are also required in layout netlists.
>
> **"The LVS Control File"** explains the importance of each option in this file. Options in this file control the behavior of the LVS matching algorithm and determine which reports are generated. You should be familiar will all of the options in this file to insure that the LVS performs a valid comparison of the two netlists.

"Running the LVS Circuit Comparison" contains a variety of information on using the LVS effectively.

> **"Command Line Syntax"** describes the syntax used to launch the LVS program from DOS.
>
> **"Runtime Errors"** explains the format of error messages generated when there are syntax errors in the command line, or in the input files.
>
> **"Overview of Matching Algorithm"** provides a quick summary of how the LVS matches circuits.

[1] Cadence is a registered trademark of Cadence Design Systems, Inc.

"Device Transformations" describes how the LVS can optionally transform netlists before comparison. Device level circuits can be transformed into higher-level circuits before comparison. This allows circuits that are logically equivalent but physically dissimilar in each netlist to be matched. The simplest transformations include merging parallel or series devices into single devices. Control file options and device models in each netlist control the device transformations performed.

"Pad Connection Verification" describes how to verify pad connections in your layout. This is a problem often overlooked by chip designers since pads are not usually included in a schematic netlist. However, an error in a pad connection can be just as catastrophic as any other type of layout error.

"Parameter Calculation" explains how the LVS can optionally verify the size or value of matched devices. This chapter describes the calculation of device values, how tolerances are defined, and how to enable parameter checking.

"Advanced Uses of Node Labels" covers how node labels in the layout can affect the LVS comparison. Labeled nodes in the layout can be used to define points of equivalence between the netlists before the LVS proceeds with the comparison, however this is not required. Node labels can also be used to define virtual connections. Special characters in node labels can prevent transformations of circuits.

"Using a Node Correspondence File" describes how a set of node equivalences (which usually make use of labeled nets in the layout) can be used to insure that the comparison progresses after a few known points of correspondence have been matched. This is not required, however convergence may be slow if the two circuits have inherent symmetry, or if they are substantially different.

"Using Node Label Overrides" describes how to add or modify node labels in your layout netlist without re-executing the NLE.

"Symmetric Circuits" covers several features of the LVS that allow you to verify highly symmetric circuits efficiently. Some examples of symmetric circuits are included.

"LVS Output Files" includes a description of each report generated by the program. The LVS always generates a report that lists all unmatched devices and unmatched nets. Several other types of reports can be generated according to options in the control file. Tables are included which describe the data in each report and the relevant control file options that enable it and determine the name of the file. Included in this section is the **"Using the Node Outliner Commands"** chapter that describes the commands for the ICED32™ layout editor used to highlight specific devices and nets in the layout.

"The LPE Utility" describes how to translate a layout netlist into a schematic netlist. The layout netlist generated by the NLE is a binary file. One of the optional output files from the LVS is an ASCII schematic netlist generated from the layout. If your only purpose in running the LVS is to generate this file, you can run the LPE utility instead.

The **"Advanced Tutorial"** uses the examples provided on the installation diskettes to take you through all the steps in diagnosing several different types of discrepancies in an LVS comparison.

Getting Started

Program Requirements

The NLE and LVS programs may run with as little as 8 Megabytes, but run times are likely to be long.

See **Panel Processing** to reduce the amount of memory required for the NLE.

We recommend that the NLE program be run on a computer with at least 16 Megabytes of memory. The NLE does create swap files for virtual memory, however this disk swapping will slow the program down.

The LVS requires less memory in general than the NLE, however it does not support virtual memory. The amount of memory required for the LVS varies greatly with the number of devices in your netlists. For a chip with 100,000 devices we recommend that you have 64 Megabytes of memory available. Small circuits can be compared successfully with as little as 8 Megabytes of memory.

The swap files created by the NLE can grow very large. We have seen swap files over 1 Gigabyte in size for large chips. If you have limited memory for the NLE, you will need plenty of free disk space.

Both of these programs run in DOS, outside of the ICED32™ layout editor.

The NLE requires a key on your printer port. Install the key on the LPT1 port, then connect your printer cable to the key. The key should not interfere with normal printer operations.

Installation

The NLE requires a key on your printer port. Install the key on the LPT1 port, then connect your printer cable to the key.

The LVS and NLE programs and their related files are installed at the same time as the ICED32™ layout editor. Place diskette 1 in floppy drive A: or B: and type at the DOS prompt:

```
A:INSTALL
      or
B:INSTALL
```

You will need to specify the name of the installation directory. All executable files, including LVS.EXE, NLE.EXE, and ICED32.EXE will be stored here. We refer to this directory throughout this manual as Q:\ICED. Whenever you see "Q:\ICED", replace the 'Q' with the drive letter you chose during installation. Replace "ICED" with the name of the installation directory.

When the installation routine has copied all files, you will see a message about the fact that you should edit your AUTOEXEC.BAT file to add the installation directory to the DOS PATH environment variable. A copy of your AUTOEXEC.BAT file, with this simple change already made, will be stored in the Q:\ICED\SAMPLES directory.

If changes to your CONFIG.SYS file are required, the installation routine will inform you and store a modified copy of this file in the same directory.

You should look carefully at these files and either copy the modified files to the root directory of your boot disk, or make equivalent edits to your existing files.

Some useful examples are included during installation. A sample schematic netlist and a control file are located in the Q:\ICED\SAMPLES\74181\LVS directory. The files stored in Q:\ICED\SAMPLES\74181\CLEAN contain layout data that match the sample schematic file. We will be using these files in the tutorial below.

The files in Q:\ICED\SAMPLES\74181\BAD contain a version of the same layout with several errors. The **Advanced Tutorial**, beginning on page 406, demonstrates how you use the LVS program to diagnose the errors in this layout.

Quick Tutorial

To execute the LVS program, you must prepare a control file and two netlists. These netlists can be either layout netlists; created from circuit layout data, or schematic netlists; created using a circuit simulation language such as SPICE.

This tutorial will cover all steps in performing a comparison of a layout netlist to a schematic netlist. This is referred to as an LVS (Layout Vs. Schematic) comparison. The circuit used will be the ONEBIT subcircuit of the 74181 cell that is provided with the installation.

See the **Advanced Tutorial**, at the end of this manual, for examples with design errors.

This example contains no errors. The reports the LVS will generate will show you that the two netlists represent exactly the same circuit. If you wish to see how errors are reported, you can add errors to either netlist and view the results. We suggest that you copy all files to a different directory before you begin adding errors.

The flow of data to prepare the input files is shown in Figure 1.

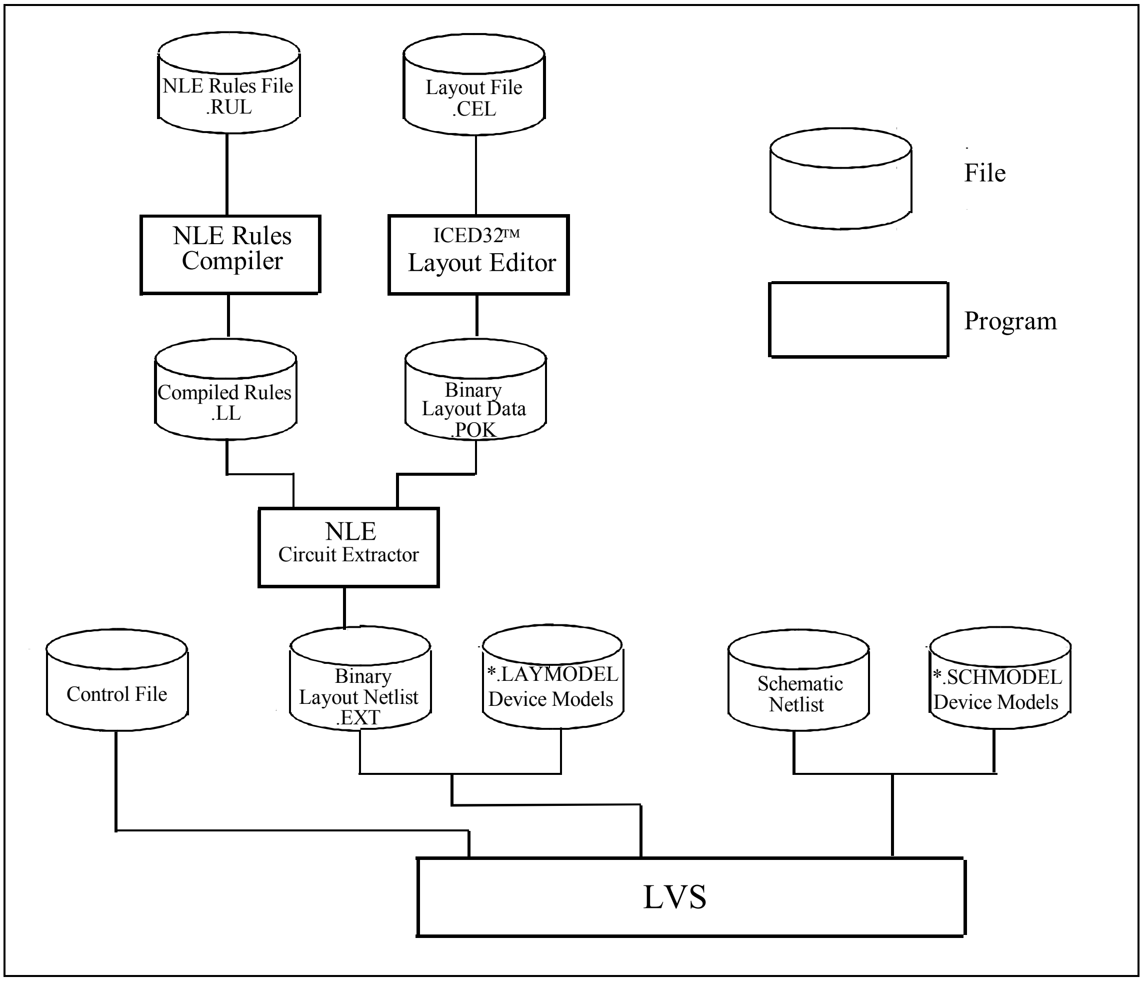


Figure 1: The flow of data in an LVS comparison.

The layout netlist is created using the NLE circuit extractor. To use the NLE, you must have two input files. The first contains layout data generated by the DRC command in the ICED32™ layout editor. The second file contains rules for electrical connection of the layers and for device recognition.

The binary output from the NLE is combined with an ASCII file that contains models of all devices included in the layout netlist. Each device model begins with the *.LAYMODEL keyword. These models define default values, tolerances, and options which control how devices can be combined into higher level devices before the LVS comparison is performed.

The schematic netlist must also have device models added to it. These models perform a similar function to the layout device models. Schematic netlist device models begin with the keyword *.SCHMODEL.

The control file is usually created by editing a sample control file provided with the LVS installation. This tutorial will show you how to perform a few simple changes to a copy of the sample control file.

We will copy the following files to a new, empty working directory before starting this tutorial:

From Q:\ICED\[2]SAMPLES\74181\LVS

CONTROL.LVS	Sample control file
TOP181.CIR	Original schematic netlist
SCHMODEL.NET	Schematic netlist models
LVS_SCH.NET	Schematic netlist including models
S181.RUL	NLE rules file for layout circuit extraction
LAYMODEL.NET	Layout netlist device models
LVS_LAY.NET	ASCII layout netlist including models

From Q:\ICED\SAMPLES\74181\CLEAN

*.CEL	Cell files for layout

[2] Remember that Q:\ICED represents the drive letter and path where you have installed ICED32™.

Before we create the new directory, we need to make the current DOS drive a drive with plenty of available space. Replace the drive letter in the following command to a disk drive with free space and type at the DOS prompt:

```
C:
```

The exact location and name of the tutorial directory is not important. Create it wherever it is convenient.

Now create the new directory with the command:

```
MD ICEDTUTR
```

Make this new directory the current DOS directory with the command:

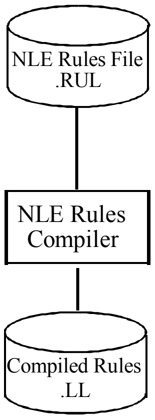
```
CD ICEDTUTR
```

Now copy the required files from your ICED directory with the following commands:

```
COPY Q:\ICED\[3]SAMPLES\74181\LVS\*.*
COPY Q:\ICED\SAMPLES\74181\CLEAN\*.*
```

[3] Remember that Q:\ICED represents the drive letter and path where you have installed ICED32™.

Preparing the Layout Netlist



The first step for creating a layout netlist is to prepare an NLE rules file for your technology. This file controls how the NLE will recognize devices and electrical connections. The NLE rules file we will be using for this example is a simple one for the CMOS technology which is supplied with the installation. See Figure 2.

```
input layer{
   2 n0_diff;
   3 p0_diff;
  41 p_well;
  46 poly_in;
  47 cont_to_poly;
  48 cont_to_active;
  49 metal1_in;
  50 via;
  51 metal2_in;
  60 n_text;
};

scratch layer{
  n_well;   p_diff;   n_diff;   metal1;  metal2;
poly;
}

metal1 = metal1_in;
metal2 = metal2_in;
poly = poly_in;

output layer{
   100 p_gate;  101 n_gate;
}

output layer{
   110 bad_p_gate;  111 bad_n_gate;
}
```

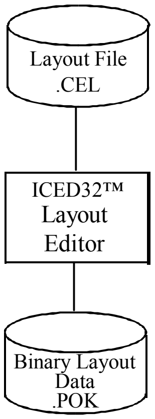
```
n_well = not p_well;

p_diff = p0_diff and not poly;
n_diff = n0_diff and not poly;
p_gate = p0_diff and poly;
n_gate = n0_diff and poly;

attach text n_text n_well;
connect p_diff p_well;
connect n_diff n_well;
connect poly metal1 by cont_to_poly;
connect metal1 p_diff by cont_to_active;
connect metal1 n_diff by cont_to_active;
connect metal1 metal2 by via;

transistor pmos id=p_gate, err=bad_p_gate{
   gate=poly;
   s$d=p_diff /poly;
   bulk=n_well;
}

transistor nmos id=n_gate, err=bad_n_gate{
   gate=poly;
   s$d=n_diff /poly;
   bulk=p_well;
}
```

Figure 2: NLE rules file for a CMOS technology, S181.RUL.

This rules file is compiled by the NLE rules compiler before the NLE circuit extraction takes place. The DOS command for rules compilation is:

```
RULESNLE  S181
```

Where RULESNLE is the name of the program and the rules file is S181.RUL. This program will create the compiled rules in a file named S181.LL. We will use this file later when we run the NLE.



Now that we have the rules for circuit extraction, it is time to prepare the layout data for extraction. This is performed in the ICED32™ layout editor with the DRC command.

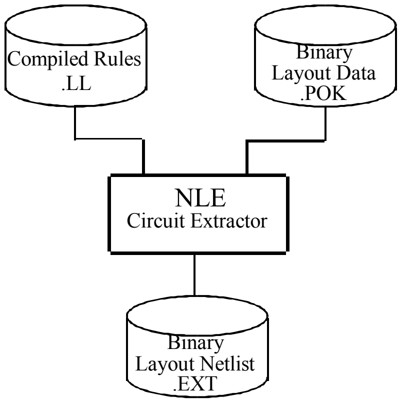
We need to launch the ICED32™ layout editor to edit our cell with the DOS command:

```
IC32  ONEBIT
```

Once in the editor, we create the binary layout data for the NLE using the DRC command without any parameters.

```
DRC
```

This will export the entire layout contained in the cell ONEBIT, and its nested cells, to the file ONEBIT.POK. Once the command is completed, we can use the **QUIT** command to terminate the editor and return to DOS.



Now we are ready to perform the NLE circuit extraction. The NLE command line is:

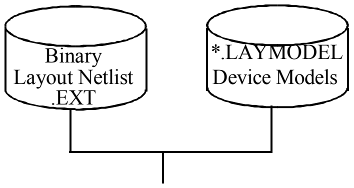
NLE S181 ONEBIT ONEBIT

This command will create several files. ONEBIT.EXT is the binary layout netlist we will be using for our LVS comparison.

All devices found by the circuit extractor must have device models defined for them. The devices in ONEBIT.EXT have model names that correspond to the model names used in the NLE rules file on page 21; PMOS and NMOS. Model statements for these devices must be added to the layout netlist. These models are created in a separate file. This file is often called LAYMODEL.NET. The model file will need one model statement for the PMOS device model, and one for the NMOS device model. The *.LAYMODEL statement is used to define device models for the layout netlist. For this example, we need only to match the device model names to the physical device types they represent. This file has already been prepared for you. See Figure 3.

```
*
*.laymodel pmos pmos
*.laymodel nmos nmos
```

Figure 3: Layout netlist model file, LAYMODEL.NET.



The model file is combined with the binary layout netlist in the file LVS_LAY.NET. See Figure 4. The model file is added to the netlist with the statement:

.include laymodel.net

The binary output from the NLE circuit extractor is added to the netlist with the statement:

***.layout onebit.ext**

```
*
.include laymodel.net
*.layout onebit.ext
.end
```

Figure 4: Layout netlist, LVS_LAY.NET.

If you are not already familiar with a DOS ASCII file editor, use the editor that comes with your DOS installation, **EDIT**.

Edit the LVS_LAY.NET file in your tutorial directory and change the *.LAYOUT statement to look like the line above. The unmodified LVS_LAY.NET file assumes that you want to compare the entire TOP181 circuit (which contains ONEBIT as a subcircuit). The *.LAYOUT statement above includes the binary layout netlist for the ONEBIT circuit we created from ONEBIT.CEL instead of a binary netlist created from the TOP181 cell.

The layout netlist is now ready for comparison. The LVS_LAY.NET file is the file we will refer to in the LVS command line.

Preparing the Schematic Netlist

The schematics for this example are provided on page 434.

We will use a schematic netlist based on a CDL file as the other netlist in our example. This file was created from the circuit schematics included at the end of this manual. A CDL file created from these schematics for the ONEBIT circuit will look like Figure 5.

```
* CDL FILE FOR SIMULATION OF ONEBIT
.SUBCKT ONEBIT A B OT1 OT2 S0 S1 S2 S3 VDD VSS
XIN1 VDD VSS B 10062 INV1
XNA2 VDD VSS B S3 A 10055 NAND3
XNA3 VDD VSS A S2 10062 10060 NAND3
XNA4 VDD VSS 10055 10060 10058 NAND2
XIN5 VDD VSS 10058 OT1 INV3
XNA6 VDD VSS 10062 S1 10063 NAND2
XNA7 VDD VSS S0 B 10065 NAND2
XNA8 VDD VSS 10063 10065 A 10066 NAND3
XIN9 VDD VSS 10066 OT2 INV3
.ENDS
.SUBCKT INV1 VDD VSS IN OUT
MN1 OUT IN VSS VSS MN W=2U L=1.0U
MP1 OUT IN VDD VDD MP W=4U L=1.0U
.ENDS
.SUBCKT INV3 VDD VSS IN OUT
MN1 OUT IN VSS VSS MN W=6U L=1.0U
MP1 OUT IN VDD VDD MP W=12U L=1.0U
.ENDS
.SUBCKT NAND2 VDD VSS IN1 IN2 OUT
MP2 OUT IN2 VDD VDD MP W=4U L=1.0U
MP1 OUT IN1 VDD VDD MP W=4U L=1.0U
MN1 OUT IN1 10091 VSS MN W=2U L=1.0U
MN2 10091 IN2 VSS VSS MN W=2U L=1.0U
.ENDS
.SUBCKT NAND3 VDD VSS IN1 IN2 IN3 OUT
MP3 OUT IN3 VDD VDD MP W=4U L=1.0U
MP2 OUT IN2 VDD VDD MP W=4U L=1.0U
MP1 OUT IN1 VDD VDD MP W=4U L=1.0U
MN1 OUT IN1 10086 VSS MN W=3U L=1.0U
MN2 10086 IN2 10087 VSS MN W=3U L=1.0U
MN3 10087 IN3 VSS VSS MN W=3U L=1.0U
.ENDS
.END
```

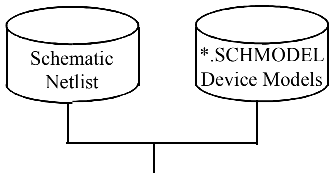
Figure 5: CDL schematic netlist for the ONEBIT circuit.

```
* .schmodel mn nmos
*.schmodel mp pmos
```

Figure 6: Schematic netlist model file, SCHMODEL.NET.

The schematic netlist provided with the installation, TOP181.CIR, represents the entire higher level circuit, TOP181. We will not have to edit this file at all to compare only the ONEBIT subcircuit. We will override the top-level subcircuit the LVS will use in the control file later.

Note that the device models used in the schematic netlist in Figure 5 are MN and MP. We need to add device model statements for these models to the schematic netlist just as we added device models to the layout netlist. Schematic netlist models are defined with the *.SCHMODEL statement. The sample file provided with the installation uses simple statements that only match the model names to the physical device type. See Figure 6.



The model file is combined with the CDL schematic netlist in the file LVS_SCH.NET. See Figure 7. The device model file is added to the schematic netlist with the statement:

.include schmodel.net

The CDL file is added to the schematic netlist with the statement:

***.schematic top181.cir**

```
*
.include schmodel.net
*.schematic top181.cir
.end
```

Figure 7: Schematic netlist file, LVS_SCH.NET.

Note that the schematic netlist is arranged in a manner very similar to the layout netlist. The format of each file has been defined to make the two netlists as similar as possible.

The schematic netlist is ready for comparison. The LVS_SCH.NET file is the file we will refer to in the LVS command line.

Preparing the Control File

Control File

The control file we will use for this example requires a few changes to the sample LVS control file supplied with the program, CONTROL.LVS. This sample file contains all available options. You will always use copies of this file, then modify the default values of options only where required. Never delete lines from this file.

Edit the following line in the "COMPARISON TYPE & FILE FORMAT" section:

TOP_LEVEL_SUBCKT_IN_SCHEMATIC_FILE = top181

The top level subcircuit needs to be "onebit" instead of "top181". This name corresponds to the name of the subcircuit we want to compare. Only this subcircuit of the larger TOP181.CIR file will be parsed by LVS.

The new line in your control file should look like this:

TOP_LEVEL_SUBCKT_IN_SCHEMATIC_FILE = onebit

We must also account for the fact that the sizes of the devices in the CDL file are given in meters instead of microns. Note that each of the device sizes in Figure 5 is followed by a "U". For example, the last device in the file, device MN3 in subcircuit NAND3, is represented with the following line:

MN3 10087 IN3 VSS VSS MN W=3U L=1.0U

As LVS parses this line, it will store the width of the device as 3.0e-6 and the length as 1.0e-6. Since the device dimensions in our layout are expressed in microns, this will cause parameter mismatches for every device. The devices would still be matched, but a parameter mismatch error message would be listed for each device.

To correct the schematic netlist device sizes so that they are expressed in microns, we can use a control file option. Under "MOSFETS" in the "INDIVIDUAL DEVICE OPTIONS" section is the option:

SCALE_MOSFET_LENGTH_AND_WIDTH = 1

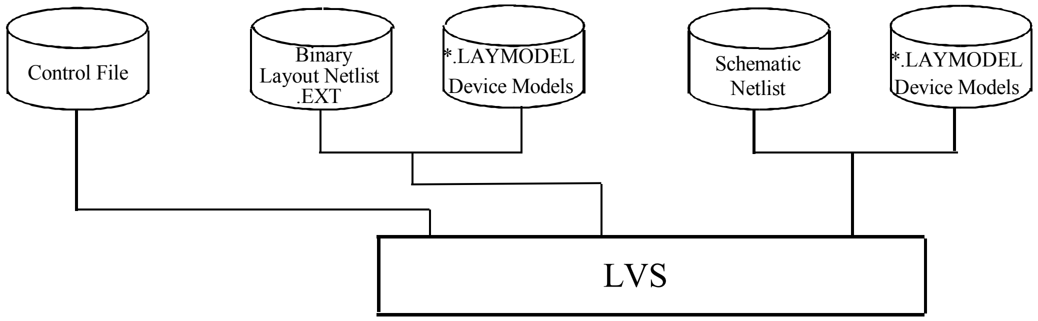
You should change the scale value so the line reads:

SCALE_MOSFET_LENGTH_AND_WIDTH = 1e6

This will multiply each length or width parameter by 10^6, or 1,000,000. This will convert all sizes in the schematic netlist to microns.

We will leave the remainder of the control file intact. Do not delete any lines, even those referring to device types not used in this example.

Running the Program



Now that we have our three input files prepared, we can execute LVS to perform the comparison. The command line is:

```
LVS  CONTROL.LVS  LVS_SCH.NET  LVS_LAY.NET
```

The first parameter is the name of the control file. The second parameter, LVS_SCH.NET, is the name of the schematic netlist file. The third parameter is the name of the layout netlist file. **The order of these two netlist file names is important.** You can not reverse the order of the schematic and layout netlist files in a LVS comparison.

The LVS command should execute very quickly. The information it prints to your screen may scroll so quickly that you do not see it all. However, all screen output is echoed in the file "LVS.LOG" in the current directory. The contents of this file should be similar to Figure 8.

```
% lvs control.lvs lvs_sch.net lvs_lay.net

! ICED: LVS    beta version 601.40
! (C)Copyright 1995-1996 IC Editors, Inc., ALL RIGHTS RESERVED
!
! Use and sale of this software is restricted by United States and
! Canadian government export regulations. If you are in another
! country, your organization agreed to abide by these regulations
! prior to licensing this software.
!

Parsing first netlist....
Parsing second netlist....
Running flat on first netlist....
Preprocessing first netlist....
Running flat on second netlist....
  LABEL <<VDD:>> in the layout is recognised. Attached to net 2.
  LABEL <<VSS:>> in the layout is recognised. Attached to net 1.
Checking virtual connections....
Preprocessing second netlist....
Pass #1   : Devices matched: 0      of 72     Nets matched: 6      of 52
Pass #2   : Devices matched: 34     of 72     Nets matched: 44     of 52
Pass #3   : Devices matched: 72     of 72     Nets matched: 52     of 52
Pass #4   : Devices matched: 72     of 72     Nets matched: 52     of 52
Checking for suspicious matches .....
Pass #4   : Devices matched: 72     of 72     Nets matched: 52     of 52
Done .....

Printing results to output files .....
Generated <<onebit.p8k>> file.
Check <<results\results.lvs>> for summary of netlist comparison.
```

Figure 8: LVS console output, LVS.LOG

A log like this indicates that the comparison found no mismatches in the device connections of the two netlists. Note that the last line tells you where to look for the reports generated by the LVS program. Exactly which reports are generated is determined by the options in the control file.

Looking at the Results

The log file above directed us to the "results" directory to find the report files generated by the LVS. There should be at least twelve files in this directory, but the number and content may vary slightly with your version of the LVS. Most of the reports list the names of the input files, the date of the run, and LVS version information at the top of the report.

Refer to **LVS Output Files** to see which control file options control report file names.

Here is a list of the report files created in our example:

RESULTS.LVS	This report is a summary of the comparison results. The number of devices and nets before and after pre-processing is listed, along with the number matched. The number of devices with parameter mismatches is also listed here. Also included is a summary of totals of devices and nets in various categories, such as how many devices were filtered by preprocessing and how many nets are unconnected. This is followed by a summary of devices of each device type. Then, the report lists the control file options used. This includes a list of the names of all other reports.
UNMATCH.LVS	All devices and nets that were not matched in the two netlists will be listed here. Since all devices matched in our example, this report lists no devices or nets.
MATCH.LVS	Every matched device is listed here. The device name in each netlist is provided along with the device coordinates in the layout, the model name, and parameter values. If you did not change the SCALE-_MOSFET_LENGTH_AND_WIDTH option in the control file, as discussed on page 28, the device size mismatches would be indicated with the string "** PARAMETER ERROR **" above each device with a device size mismatch. (This information would also be listed in the PARAM.LVS report listed below.)

After the list of matched devices is a list of matched nets. The netname in each netlist is provided. The two nets labeled in the layout are listed at the bottom.

PARAM.LVS	All parameter mismatches, such as device size mismatches, are listed here. Since our example has no device size mismatches, this report lists no devices. If you execute the example again after changing the size of a single device in the schematic netlist, you will see the device listed in this report.
EQUIV.LVS	This file lists all labeled nets in the layout. Matched nets will include the name of the corresponding net in the schematic netlist. You can look in this file to see that net "VDD" in the schematic netlist was matched to the net labeled "VDD:" in the layout netlist.
FILTER.LVS	If devices had been filtered out of either netlist due to options in the control file, they would be listed here. No devices were filtered, so this report lists no devices.
COLLAPSE.LVS	This report lists details on circuits that have been collapsed by the LVS into pseudo devices. This example did not allow the generation of pseudo devices, so the file contains no useful information for this run.
NETDEG.LVS	This report lists nets and their degrees. The degree of a net is defined as the number of devices to which it connects. A summary is provided of how many nets of each degree are in each netlist. No details on specific nets are listed since we used the control file option PRINT_ALL_NETS_WHOSE_DEGREE_GREATER-_THAN = 40, and none of the nets in our example has a degree this large.

If you want to see details on some specific nets, set the control file option PRINT_ALL_NETS_WHOSE_DEGREE_GREATER_THAN to 10 and run the example again. VDD and VSS will now be listed since they are the only two nets with a degree greater than 10.

NETONE.LVS	This report lists floating nets. A floating net connects to a single device or to no devices. Since no nets in our example meet these criteria, no nets are listed.
SMETRIC.LVS	This report lists devices and nets that were forced to match to allow the netlists to be compared. This report should list no forced matches since the circuit is not very symmetric.
SPICE.LVS	This file is a flat (i.e. no subcircuits) spice netlist created from the layout. If this type of output is not desired, it can be turned off with the GENERATE_SPICE_NETLIST_FROM_THE_EXTRACTOR_OUTPUT = NO option in the control file.
LABELS.LVS	This file lists the node numbers of all labeled nets in the layout netlist. You can edit this file to add or modify node labels in the layout netlist for the next time you run the LVS. Thus, you will not have to execute the NLE circuit extractor again to add labels to the layout netlist.

Symmetric circuits and forced matches are defined on page 365.

Now we can use the outliner commands in the ICED32™ layout editor to find devices and nets listed in the reports. These commands are the easiest way to locate errors in your layout.

```
          SCHEMATIC                    |          LAYOUT
                                       |
                                       |
# :1                                   | # :1
XIN1.MN1                               | 284
X :0            Y :0                   | X :64        Y :32
MODEL :MN    TYPE :NMOS                | MODEL :NMOS        TYPE :NMOS
LENGTH :1      WIDTH :2                | LENGTH :1    WIDTH :2
                                       |
```

Figure 9: Fragment of matched devices report.

The MATCH.LVS file should contain a report for the first device matched which looks like Figure 9. To find this specific device in the layout, open the cell with the ICED32™ layout editor using the following DOS command:

See **Using the Node Outliner Commands** for more details on how to use these commands.

```
IC32  ONEBIT
```

In the editor, type the following commands:

```
@NODES
N0  284
```

If you get the message "Insufficient memory" when you try to execute the N0 command, you will have to reserve memory for it. See page 389 for instructions.

where you have replaced 284 with the actual number in your report under the LAYOUT column. (The command is "N0" with a zero not the letter 'O'.) The device should light up on your screen and blink for a few seconds. If you do not see a white rectangle blinking, it may be outside of the current view window. Try the following commands:

```
VIEW ALL
BLINK
```

If you fail to see any nodes blinking, it may be that you have launched ICED32™ with the COLORS=8 command line parameter. These commands require COLORS=16.

The matched devices report will list the matched nets at the end of the file. This section of the report should contain lines that look similar to Figure 10. To locate this net in the layout, type the following commands:

```
ND
N0  VDD:
```

```
SCHEMATIC  |LAYOUT

# :3       | # :3
VDD        | 2
           | LABEL :VDD:
```

Figure 10: Fragment of matched devices report listing a matched net.

The N0 command creates shapes on layer 250 of the current cell. You usually want to delete the shapes on this layer before using N0 again. The ND command above deletes the outline shapes created by the previous use of N0. The second N0 command then outlines all shapes on the net "VDD:". If you did not execute the ND command, both node 284 and node VDD: would be outlined.

This concludes the tutorial. You can continue to experiment with this example by editing the netlists or control file as you learn more about the LVS.

NLE Basics

The NLE is a **N**et **L**ist **E**xtractor utility that performs circuit recognition on the layout data in an ICED32™ cell. This utility runs in DOS outside of the ICED32™ layout editor. The netlist the NLE generates is in a binary form. This layout netlist can be compared to a schematic netlist with the LVS utility.

In addition to generating the layout netlist, the NLE can optionally perform electrical connection check (ECC) tests to find problems like shorts and opens before you run the LVS utility.

This chapter, which describes the basics of how the NLE works, contains mainly information that is also covered in the DRC (Design Rules Checker) manual. The NLE uses many functions identical to those used by the DRC to process the layout data. The layer generation rules are almost identical in both programs, as are the algorithms that divide the layout into panels for processing.

The rules unique to NLE circuit recognition are covered beginning on page 93. See the table on the next page to see which rules are supported by the NLE and which are supported by the DRC.

The DRC program uses the layer generation algorithms in combination with size, spacing, and shape rules to verify that your design meets technology specific design criteria like spacing groundrules. The NLE program uses the same layer generation algorithms in conjunction with electrical connection and device recognition rules to perform circuit recognition.

Rule	**Use**	**Supported by DRC**[4]	**Supported by NLE**	**Page #**
ALL_DANGER	Control hierarchical processing	Yes*	No	
ALL_SAFE	Control hierarchical processing	Yes*	No	
AND	Boolean AND of two layers	Yes	Yes	63
ASPECT_RATIO	Classify shapes by relative dimensions	Yes*	Yes	78
Assignment Rule	Copy layer or inverse of layer	Yes	Yes	62
ATTACH TEXT	Move text labels to another layer	No	Yes	141
BLOAT	Expand shapes	Yes	Yes	67
BLOAT_ANGLE	Define angle for BLOAT rule	Yes	Yes	69
BORDER	Explicitly define panel overlap	Yes	Yes	89
BOUNDS	Classify shapes by size	Yes*	Yes	77
BRIDGE	Recognize air bridges	Yes*	Yes	80
CAPACITOR	Recognize capacitor devices	No	Yes	130
COMMENT	Define polygon id label prefix	No	Yes	143
CONNECT	Electrically connect layers	Yes*	Yes	96
CONST	Define constant value	Yes	Yes	46
DANGER_CELL	Control hierarchical processing	Yes	No	
DEVICE	Recognize devices	No	Yes	113
DISJOINT	Allow virtual connections in ECC	No	Yes	155
ERROR_LAYER	Set layer number for ECC error shapes	No	Yes	154
IN_CELL	Classify shapes in certain cells	Yes*	Yes	66
INCLUDE	Allow rules file nesting	Yes	Yes	47
INPUT LAYER	Define input layer	Yes	Yes	53
IS_BOX	Classify shapes by size	Yes*	Yes	75
ISLANDS	Find holes	Yes*	Yes	74
LABEL	Indicate that layer contains ECC labels	No	Yes	142
MAXIMUM ANGLE	Find sharp points in notches	Yes	No	

[4] * indicates that rule is not supported in DRC version 2.xx, but is supported in version 3.xx.

Rule	**Use**	**Supported by DRC**	**Supported by NLE**	**Page #**
MINIMUM ANGLE	Find sharp points	Yes	No	
MINIMUM AREA	Find small shapes	Yes	No	
MINIMUM NOTCH	Find small notches	Yes	No	
MINIMUM SIDE	Find shapes with at least one small side	Yes	No	
MINIMUM SPACE	Find shapes too close together	Yes	No	
MINIMUM WIDTH	Find shapes with small width	Yes	No	
NO_ECC	Turn off ECC checking	No	Yes	153
NO_SAVE	Restrict layers in node outliner file	No	Yes	144
OFF_GRID	Find vertices not on resolution grid	Yes*	No	
OR	Boolean OR of two layers	Yes	Yes	64
OUTPUT LAYER	Define layer for output	Yes	Yes	57
OVERLAPPING	Find shapes with common area	Yes	Yes	73
PADSIZE	Define size for UNCONNECTED rule	No	Yes	156
PANELX	Define maximum panel size in X direction	Yes	Yes	88
PANELY	Define maximum panel size in Y direction	Yes	Yes	88
PATH_LAYER	Set layer number for path trace of shorts	No	Yes	154
SAFE_CELL	Control hierarchical processing	Yes	No	
SAVE	Restrict layers in node outliner file	No	Yes	144
SCRATCH LAYER	Define temporary layer	Yes	Yes	59
SHRINK	Shrink shapes uniformly	Yes	Yes	68
SNAP	Relocate vertices on resolution grid	Yes	No	
STAMP	Electrically connect poor conductors	No	Yes	102
TOUCHING	Find touching shapes on different layers	Yes*	Yes	71
TRANSISTOR	Recognize transistor devices	No	Yes	125
UNCONNECTED	Verify that shapes are not electrically connected to any other node	No	Yes	155
XOR	Boolean exclusive OR	Yes	Yes	65

Figure 11: Table of NLE rules vs. DRC rules

We suggest that you avoid performing design rules checks for device layout in your NLE rule set. Design rules can be more completely checked using the additional rules in the DRC, and you will save time every NLE run if you keep your NLE rule set simple.

The rules for the NLE can be written with any ASCII text editor. The rules file is then compiled with the NLE rules compiler. The compiled rules file is used by the NLE circuit extractor along with the binary layout data file created by the DRC command in the ICED32™ layout editor.

NLE Rules Syntax

The syntax restrictions for NLE rule statements vary greatly from rule to rule. You must read the rule statement descriptions to determine the syntax of each rule. However, there are some general syntax restrictions which all rules have in common. We will cover these syntax issues here so we don't have to repeat them too often.

Each rule has a keyword that is considered the name of the rule. The underscore character '_', present in many of the keywords, is optional and can be left out of the keywords. The underscore is included for readability only and is stripped during preprocessing.

Example: **NO_ECC**
NOECC

Both of these ways of typing the NO_ECC rule are equally valid.

The only exception to this use of case in the rules file is when you type a node name in a rule and the USE_CASE rule is present.

NLE rules are case-insensitive. This means that you can type the rule set in upper case, lower case, or any combination of the two. All text is transformed into upper case as it read by the rules compiler.

Each rule should be typed on one line. There are several exceptions to this restriction which are indicated in the rule descriptions. The rules that may be typed over several lines use curly brackets '{}' (a few rules use parentheses '( )') to enclose the text on the extra lines.

You can add comments on lines of their own, or at the end of any line. The comment indicator is the exclamation mark '!'. Any text encountered after the exclamation mark, up to the end of the line, is ignored by the rules compiler.

The syntax of individual NLE rules is described in this manual using the notation described in the ICED32™ Reference Manual with one exception. Since parentheses are used so frequently in NLE rules, we will **not** use them to indicate a choice between keywords. Instead, where a choice between keywords or parameters is allowed, we will indicate this with smaller text listing the choices near the rule syntax heading.

The syntax headings use the following notation:

KEYWORD	Bold type in the syntax section will be used to indicate the rule name keyword.
parameter value	Lower case italic type will be used to indicate where a value should be entered in a rule statement. The value could be a number or a string. The valid values for the parameter will be indicated in the description.

CONST *const_name* = *const_value*

The above line is used to indicate the syntax for the CONST rule on page 46. This rule is used to assign a value to a named constant that can be used in other rules instead of typing in the parameter value. When you type a CONST rule, substitute a string for *const_name* and a number for *const_value.*

```
CONST  MY_VAL  =  2.45
```

[KEYWORD] Square brackets indicate that the keyword or parameter is optional. Do not type the brackets in the rule.

result_layer = [NOT] *layer1* **AND** [NOT] *layer2*

This is the syntax description for the AND rule. The NOT keywords are optional. The parameters *result_layer, layer1* and *layer2* should all be replaced with layer names when the rule is typed. If the second optional NOT keyword is used, as in the following rule:

```
SRC_DRN = DIFF  AND  NOT  POLY
```

the inverse of layer POLY will be used in the Boolean AND operation rather than layer POLY itself.

...	Three dots at the end of a line of sample code, or in the syntax section, indicate that the line is continued on the next line. Three dots will also precede the continuation on the next line. When you type the rule, type it all on one line without the dots. Three dots in the middle of a line in a syntax description mean that several additional parameters are allowed but are not explicitly specified in the syntax description. **PINS =** ***pin_layer_1*** **[,** ***pin_layer_2*** **[...,** ***pin_layer_n*** **]]** This is part of the syntax description for the DEVICE rule. You may have up to 25 pin layers specified after the PINS keyword, but it would be rather verbose to list all 25 parameters. The dots take the place of the missing parameters.

Two miscellaneous NLE rules are best described here in the syntax section. The CONST rule mentioned above is used to create named constants which you can use in other rules as parameter values. The INCLUDE rule allows you to refer to a file of NLE rules which will be inserted into the rules file containing the INCLUDE rule.

CONST *const_name = const_value*

or

CONST {

const1_name = const1_value

⋮

constn_name = constn_value

}

You can use a CONST rule to define a certain number as a constant that you can refer to by name in other rules rather than typing the number itself. The *const_value* must be a real number. You may **not** use exponential notation (e.g. 1.478E-9) when typing *const_value*. You may not use layer names or other strings for *const_value*.

Example: **CONST M1_EXPANSION_VALUE = .246**
M1 = BLOAT (M1_IN, M1_ EXPANSION_VALUE)

The CONST rule above defines the string "M1_EXPANSION_VALUE" as a constant with the value .246. When this constant is used in the BLOAT rule, the rules compiler will substitute ".246" for the string "M1_EXPANSION_VALUE".

You can have many CONST rules in your rule set. This allows you to define technology dependent parameters together in one place where they are easy to find and edit. When the CONST rule is not used, it will be difficult to update an old rule set with new values since they will be scattered through the rule set.

If you have multiple constants to define, you can use the multiple line syntax to define all of them with a single CONST rule. Place each constant definition on it's own line. Surround the lines with curly brackets.

Example:

```
CONST {
        M1_EXPAND =     .246
        M2_EXPAND =     .246
        MIN_DEV_W =     1.9
        MIN_DEV_L =     4.5
}
```

INCLUDE [*dir_path*\]*file_name*

This rule allows you to nest rules files. An INCLUDE rule in one rules file will result in another rules file being inserted at that point. The *file_name* parameter is used to specify the name of the file. The file extension (if any) must be included in *file_name*. You may optionally supply a directory path with the file name.

Example:

```
INCLUDE  MOSCONST.RUL
```

This rule will cause the text in the file MOSCONST.RUL to be added to the current rules file at the point where the INCLUDE rule is found. Since no *dir_path* parameter is used, the file MOSCONST.RUL must exist in the current directory.

You **cannot** use the INCLUDE rule in the middle of another rule. You may nest rules files up to 10 deep with the INCLUDE rule.

Layer Processing

All layers in an NLE rule set must be defined before they are used in a rule. The layer definition rules (INPUT LAYER, OUTPUT LAYER, and SCRATCH LAYER) are used to define layers.

Input layers (defined with the INPUT LAYER rule) correspond to layers in the input ICED32™ cell. Only layers in an INPUT LAYER rule will be read in from the ICED32™ cell. Other layers in the cell are ignored.

Shapes can be created by the NLE on output layers (defined with the OUTPUT LAYER rule). Shapes on output layers will be created in a file you can later read into an ICED32™ cell. This can be useful for diagnosing problems with your rule set as well as finding errors in your layout.

The layer names used in the ICED32™ cell are ignored by the NLE. Both the INPUT LAYER and OUTPUT LAYER rules use only the layer number in the ICED32™ cell to identify the layer. This layer number is associated with an NLE layer name in the INPUT LAYER and OUTPUT LAYER rules. The NLE layer name is used throughout the rest of the NLE rules to represent the layer.

Only shapes in the current panel are merged. To learn more about panels, see page 83.

One of the first preprocessing steps the NLE performs on your layout data is to convert all shapes to polygons. All touching polygons on the same layer are then merged into single polygons. This is done before any rules are processed.

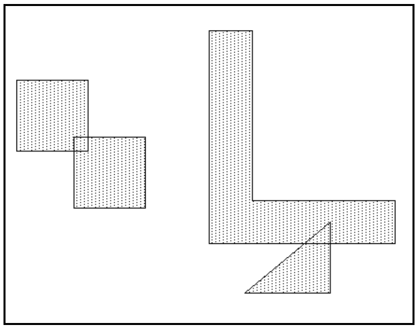


Figure 12: Layer before NLE preprocessing.

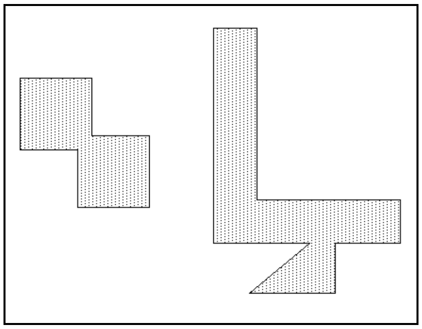


Figure 13: Layer after NLE preprocessing.

The two boxes on the left in Figure 12 are converted into the polygon on the left in Figure 13. The wire on the right in Figure 12 is converted to a polygon and merged with the triangle to create the polygon you see on the right in Figure 13.

The topology of the original shapes is not used by the NLE. The topology and dimensions of the merged shapes will be used by the program.

Some shapes cannot be represented in the ICED32™ editor as simple polygons. For example, the ICED32™ editor will not allow shapes with holes or shapes with more than 199 vertices. However, the NLE does not have these limitations. When the NLE creates polygons like these, they are processed internally without modification. It is only at output that these shapes are modified to be valid ICED32™ polygons.

Panel boundaries are explained in detail in Panel Processing on page 83.

Shapes that cross panel boundaries are broken at the boundaries during the NLE run. If a shape crosses the vertical or horizontal panel boundary at a skewed angle, there may be a tiny displacement of the vertex coordinates where the shape is cut by the boundary to keep the vertices on grid. Shapes that have been cut at the panel boundaries are not merged before output.

The NLE rules compiler will optimize layer processing rules to minimize the number of NLE passes required. The compiler will also remove rules that are redundant or unused. Let us say that you write several rules to generate a layer that you use in a specific rule. You then remove the rule that uses that layer. You do not need to search backward to remove the rules which created the layer. The compiler will do this automatically to optimize your rule set.

Layer Definition Rules

As mentioned above, a layer referred to in an NLE rule must be defined before it is used. Layer definitions are usually grouped together at the top of the file, but this is not required.

NLE layer names may be up to 30 characters long. The first character must be a letter. The remaining characters can be letters, digits, or any of the following special characters: '_', '~', '$', '.', or '#'.

Layers are referred to by name in the NLE rules. ICED32™ layers are referred to by number. All layers in an ICED32™ cell which will be processed by the NLE must be defined with INPUT LAYER rules which associate the ICED32™ layer number with the NLE layer name.

To determine the ICED32™ layer number from the ICED32™ layer name, use the LAYER or TEMPLATE commands in the editor.

Layers which will be created by the NLE to be viewed or used in the ICED32™ layout editor must be defined in OUTPUT LAYER rules which associate the NLE layer names with ICED32™ layer numbers. At the end of the NLE run, all shapes on output layers will be written to a command file that can be read into an ICED32™ cell.

If an OUTPUT layer has shapes stored on it by one of the rules which stores shapes on an error layer, the number of shapes on the layer will be reported in the "Error Layer Outputs" section of the NLE log.

The layer names in the ICED32™ cell are completely ignored by the NLE.

Any shapes that are created to highlight ECC (Electrical Connection Checks) errors will be created on special output error layers. You do not need to explicitly mention these layers in an OUTPUT LAYER rule. The ECC creates shapes on ICED32™ layers 98 and 99 by default. You can change these error layer numbers with the ERROR_LAYER and PATH_LAYER rules. Shapes on these layers will also be reported as errors in the NLE log.

Any layers you use to create or modify shapes, which are not output layers, must be defined in a SCRATCH LAYER rule. These layers are defined only by name since they are never output to an ICED32™ cell as numbered layers.

Use the assignment rule to copy a layer. See page 62 for details.

Layers defined in an INPUT LAYER rule cannot be modified. If you want to modify the shapes on an input layer, you must first copy the input layer to a layer defined in an OUTPUT LAYER or SCRATCH LAYER rule.

INPUT LAYER *iced_layer_number_1* ...
... [+ *iced_layer_number_2* [... + *iced_layer_number_5*]] ...
... [[NOT] INCELL *cell_name*] ...
... *nle_layer_name* ...
... [NOT *nle_not_incell_layer_name*] ...
... [TEXT *iced_text_layer_number*] ...
... [ID *iced_id_layer_number*]

If you need to copy an input layer to more than one NLE layer, you can use the assignment rule to copy the NLE layer.

All layers in an ICED32™ cell that will be used in your NLE rule set must be defined in an INPUT LAYER rule. The only required parameters for the INPUT LAYER rule are *iced_layer_number_1* and *nle_layer_name.* The *iced_layer_number* parameters correspond to the layer numbers in the ICED32™ cell. (The layer names used in the ICED32™ cell are ignored by the NLE.) A specific *iced_layer_number* can be referred to only once in your set of INPUT LAYER rules.

The shapes on NLE layers created with the INPUT LAYER rule **cannot** be modified by other rules. If you need to modify an input layer, see page 62 for an example of how to get around this problem.

The *nle_layer_name* is the name of the layer you will use in succeeding rules. The name does not need to be identical to the layer name in the ICED32™ cell. A specific *nle_layer_name* can appear only once in your set of INPUT LAYER rules.

Example: **INPUT LAYER 2 M1**

When this rule is used, all components on layer 2 in the ICED32™ cell, and its subcells, will be copied to layer M1 in the NLE database. The layer name M1 is what you use in other NLE rules to refer to this layer.

If you want to combine shapes on several ICED32™ layers into one NLE layer, specify several *iced_layer_number* parameters separated with plus signs ('+'). You can combine up to five ICED32™ layers into one NLE layer.

Example:

```
INPUT LAYER        2 + 12 + 22    M1
```

This rule will combine the ICED32™ layers 2, 12, and 22 into the NLE layer M1.

The INCELL options, [[NOT] INCELL *cell_name*] and [NOT *nle_not_incell_layer_name*], are used to classify components on an input layer by whether or not they are in specific subcells.

Example:

```
INPUT LAYER  2  INCELL  INDUCTOR_CELL  INDUCTOR_M1
```

The IN_CELL rule, which also classifies shapes by cell, **will** include shapes in subcells of the specified cells.

This rule will copy all components on layer 2 contained in instances of cell INDUCTOR_CELL (but **not** its subcells) to NLE layer INDUCTOR_M1.

The NOT keywords are used to indicate that the layer contains only shapes on layer *iced_layer_number* which are **not** contained in the specified cells. Only one NOT keyword is allowed.

You would use the first optional NOT keyword to restrict *nle_layer_name* to shapes that are **not** contained in the cells indicated after the INCELL keyword. You would use the second optional NOT keyword when you need one NLE layer for shapes in the cells, and another NLE layer for those shapes which are not in the cells. When the second optional NOT keyword is used, *nle_layer_name* is restricted to shapes which **are** contained in the cells indicated after the INCELL keyword, and layer *nle_not_incell_layer_name* will contain shapes which are **not** in the indicated cells.

Example:

```
INPUT LAYER  2         INCELL INDUCTOR_CELL    IND_M1
INPUT LAYER  2   NOT   INCELL INDUCTOR_CELL    M1        !Error
```

These 2 statements together would cause a compiler error since each *iced_layer_number* can occur in only one INPUT LAYER rule. You can achieve the desired result with the following single statement:

Example:

```
INPUT LAYER  2  INCELL INDUCTOR_CELL  IND_M1  NOT M1
```

Layer 0 in an ICED32™ cell is used to store subcell bounding boxes. Ordinary shapes are never stored on that layer. In an INPUT LAYER statement, layer 0 can be used to store a rectangle that covers a subcell. This may be useful in some types of layer processing.

Example:

```
INPUT LAYER   0  INCELL INDUCTOR_CELL  IND_MASK
INPUT LAYER   2  M1_IN
M1 =          M1_IN  AND  NOT  IND_MASK
IND_M1 =      M1_IN  AND       IND_MASK
```

When this set of statements is used to classify layer M1 instead of the example above, there is an important side effect you must be careful with. In this example, the processing on M1 is performed after the cell is flattened hierarchically. Shapes on M1 in subcells of INDUCTOR_CELL (or in the main cell, or any other cell) which happen to be located within the bounding box of INDUCTOR_CELL will also be classified as IND_M1. This can be desirable or not, depending on how your design is organized.

Whenever you remove material from a conductive layer, you must be careful not to prevent real shorts from being found. See page 100 for important examples of how real design errors can be hidden by using INCELL or layer 0 processing.

The *cell_name* in the INCELL parameter can contain wildcard characters ('*'). A vertical bar, '|' can be used as well to indicate a list of valid cell names. More than one '|' delimiter can be used. Do not use any blanks when entering the *cell_name* parameter.

Example:

```
INPUT LAYER   2   INCELL   IND*|*NH   IND_M1
```

This input layer specification will copy to layer IND_M1 all components on layer 2 contained in cells which begin with the string "IND", or which end in the string "NH".

The ATTACH TEXT rule can also be used to move text components to a design layer. See page 141.

The optional [TEXT *iced_text_layer_number*] parameter is used to identify a layer (other than *iced_layer_number*) which will be used to label components on *iced_layer_number.* The layer *iced_text_layer_number* should contain text components that will be interpreted as node labels for components on *iced_layer_number*.

When the TEXT keyword is not used, text components on the design layer *iced_layer_number* will be interpreted as node labels for that layer. **If the TEXT keyword is used, text components on layer *iced_layer_number* will be ignored.**

Example: **INPUT LAYER 2 M1 TEXT 102**

See page 138 for important information on how to use node labels and polygon id labels.

This statement will cause text components on layer 102 of an ICED32™ cell to be moved to NLE layer M1. These text components will then be used to label nodes on layer M1. All text components on layer 2 will be ignored.

The optional [ID *iced_id_layer_number*] parameter is also used to move text components to an NLE layer. However text components on *iced_id_layer_number* will be interpreted by the NLE as polygon id labels rather than node labels. A polygon id label is used to identify only single polygons rather than entire electrically connected nodes. These identifiers are used only in ECC error messages. They are not used in the netlist generation.

When you need to define many input layers, you can list several input layers in one INPUT LAYER rule. Separate the layers with semicolons (';').

Example: **INPUT LAYER 1 A; 2 B TEXT 102 ID 202; 3 C**

When an input layer definition is split over more than one line, you must surround the layer definition with curly braces {}. If you type one layer definition on each line, semicolons are not required.

Example:

```
INPUT LAYER {
        1 A
        2 B  TEXT 102  ID 202
        3 C
        }
```

OUTPUT [ERROR] [OUTLINE] **LAYER** *iced_layer_number nle_layer_name*

To see how to read these layers into the ICED32™ layout editor, see page 189.

Output layers will be included in the command file the NLE generates. This file can be used to create shapes in an ICED32™ cell. Use OUTPUT LAYER rules to define layers you wish to inspect using the ICED32™ layout editor. Use SCRATCH LAYER rules to define all other layers the NLE will create or modify.

The only required parameters for the OUTPUT LAYER rule are the *iced_layer_number* and the *nle_layer_name.* The *iced_layer_number* will be the number of the layer created in the ICED32™ cell when you execute the command file created by the NLE.

The *nle_layer_name* is the name of the layer used in the other NLE rules. The name will not be used in the ICED32™ cell. Only the layer number is preserved as you import the shapes into ICED32™.

The data is created after all rules are executed at the conclusion of the NLE run, regardless of where the rule appears in the rules file.

Example: **OUTPUT LAYER 101 GATE**

To assign a name to a layer in the ICED32™ layout editor, use the LAYER command.

This example will cause the NLE to copy all shapes on layer GATE in the NLE database to a command file which can create the shapes in an ICED32™ layout editor session. The layer number in the ICED32™ cell will be 101. (The layer in the ICED32™ cell will **not** automatically have the name GATE. Whatever name was assigned to this layer number, if any, will remain the name of the layer.)

You can use the same *iced_layer_number* for both an input layer and an output layer, but you will receive a warning from the compiler.

You can output more than one *nle_layer_name* to one *iced_layer_number.* In this case, shapes from several NLE layers will all be created on one ICED32™ layer.

Example:

```
OUTPUT LAYER  10  POLY
OUTPUT LAYER  10  RESISTOR_POLY
```

This pair of rules will cause the NLE to generate a command file that can be used to create shapes on layer 10 in the ICED32™ editor. All shapes on both NLE layers POLY and RESISTOR_POLY at the conclusion of the NLE run will result in shapes on layer 10 in the ICED32™ cell. The name assigned to layer 10 in the ICED32™ cell (if any) will be used as the name of the layer.

The device recognition rules will automatically classify layers specified with the ERR keyword as error layers. See page 115.

The optional ERROR keyword will cause error messages to be printed to your screen and to the log file if shapes on the indicated layer exist at the end of the NLE run. The total number of shapes on all error layers will be reported in the "Error Layer Outputs" section of the log file.

By default, the NLE will generate an ADD POLYGON command in the output command file for each shape on an output layer. The optional OUTLINE keyword can be used to change this behavior. Use the OUTLINE keyword to generate wire components that follow the outlines of the polygons rather than the polygon components themselves.

You can use semicolons and curly braces to allow more than one layer definition in one OUTPUT LAYER rule. The syntax is the same as that used in the INPUT LAYER rule. See page 56 for details.

The *iced_layer_number* 0 is treated differently than other output layers. Commands that create shapes on layer 0 will **not** be included in the output command file. Instead, the layer is treated as a scratch layer. This feature is used to facilitate diagnosing problems.

Let us say that you have an intermediate layer you need to look at occasionally to diagnose problems with device recognition. This layer is really a scratch layer and is not usually output. However you do want to include it in the output file occasionally. You should define this layer as an output layer with the layer number 0. When you do want to see this layer in the output, simply edit the layer number to a number other than 0 and the layer will be included in the output. This is much easier than editing the rules file to move the layer back and forth from an OUTPUT LAYER statement to a SCRATCH LAYER statement.

SCRATCH LAYER *nle_layer_name*

See previous page for hint on easily switching a layer from a scratch layer to an output layer

All layers used in the rules file must be defined before they are used in a rule. If a layer is not defined with the INPUT LAYER or OUTPUT LAYER rules, you must define it as a scratch layer using this rule.

The use of semicolons and curly braces to allow more than one layer definition in one statement is the same as their use in the INPUT LAYER rule. See page 56 for details.

Examples:

```
SCRATCH LAYER  SRC_DRN;  GATE;  POLY_WIRE;

SCRATCH LAYER  {
        SRC_DRN;
        GATE;
        POLY_WIRE;
}

SCRATCH LAYER  {
        SRC_DRN
        GATE
        POLY_WIRE
}
```

All three of these examples are exactly equivalent. The semicolons are not required when you use curly braces to split the SCRATCH LAYER rule across several lines.

Layer Generation Rules

Text components (i.e. labels) will be copied or moved by Boolean operations in the same manner as polygons.

Layer generation rules create polygons on an output or scratch layer based on the contents of existing layers.

Boolean operations (the XOR, OR, and AND rules as well as the NOT feature of the assignment rule) create layers from logical combinations of other layers.

The INCELL rule is used to classify a layer based on the ICED32™ cell in which it is contained.

The SHRINK or BLOAT rules are used to resize layers.

The TOUCHING and OVERLAPPING rules create layers based upon how shapes on a given layer relate to shapes on other layers.

The ISLANDS rule copies holes or disconnected polygons to a new layer.

The IS_BOX, BOUNDS, and ASPECT_RATIO rules segregate shapes on a given layer by shape or size.

The BRIDGE rule is used to recognize which shapes on a given layer form air bridges. (The BRIDGE rule is utilized primarily by users of the Gallium Arsenide technology.)

All layers used in these rules must be defined using layer definition rules. These rules are described beginning on page 51. You cannot use a layer defined as an input layer as the *result_layer* on the left of the '=' in any of the layer generation rules.

The *result_layer* will always be cleared of its previous contents and replaced with the result of the operation. The *result_layer* can be the same layer as one of the layers to the right of the '='. The following is a valid rule:

Example: **SUBSTRATE = SUBSTRATE AND NOT PWELL**

Several Boolean operations cannot be combined into a single rule (other than the use of the NOT keyword). Complex Boolean processing must be broken down into separate rules.

Example:

```
POLY_IN =           POLY_WIRES  OR        DEV_POLY
RESISTOR_POLY =     POLY_IN     AND       RESISTOR_MASK
POLY =              POLY_IN     AND NOT   RESISTOR_MASK
```

Parentheses are **not** allowed in Boolean expressions. "C = (NOT A) OR B" may seem like the natural way to write a rule, but it will generate syntax errors. In an NLE rule, the NOT keyword always applies only to the layer it precedes.

Example: **C = NOT A AND NOT B**

This rule will be interpreted by the NLE compiler as:

C = (NOT A) AND (NOT B)

The Assignment Rule: *result_layer* = [NOT] *layer1*

This rule is used to copy a layer or to create the inverse of a layer.

Example: **M1 = M1_IN**

This rule will copy all polygons on layer M1_IN to layer M1. This can be useful if M1_IN is an input layer that cannot be modified. The new layer, M1, can be modified as required.

The optional NOT keyword will create a layer which is the inverse of *layer1*.

Example: **NWELL = NOT PWELL**

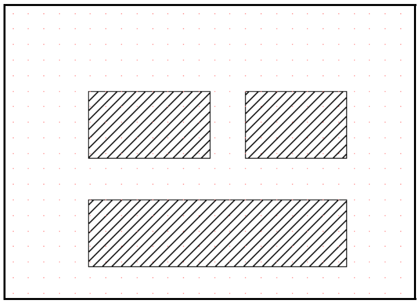


Figure 14: Layer PWELL

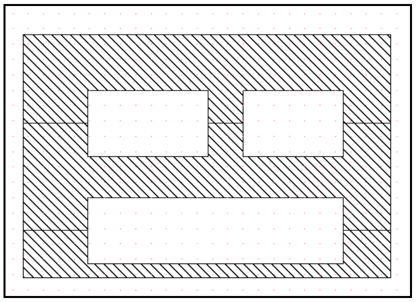


Figure 15: Layer NWELL = NOT PWELL

A bounding box is the smallest rectangle, square with the axes, which encloses the design.

This rule will create the inverse of the PWELL layer. The outer boundary of the inverse layer is the slightly bloated bounding box of your design. When the NWELL layer is used by other rules in the NLE, it will remain one large polygon with holes in it. If the NWELL layer is an output layer, before the NLE can output the layer as ICED32™ components, the shape must be divided into several polygons. Polygons with holes are not valid components in ICED32™. The somewhat arbitrary boundaries of the polygons (where the NWELL shape is cut to create valid polygon shapes) will have no effect on processing in the NLE.

result_layer = [NOT] *layer1* **AND** [NOT] *layer2*

This rule will create on *result_layer* the intersection of all shapes on layers *layer1* and *layer2*.

Example: **C = A AND B**

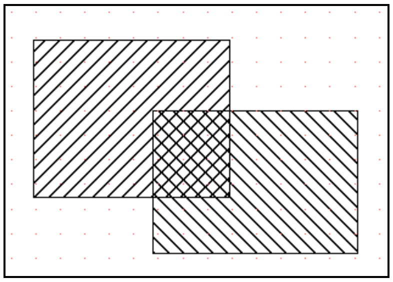


Figure 16: Polygons on layers A and B

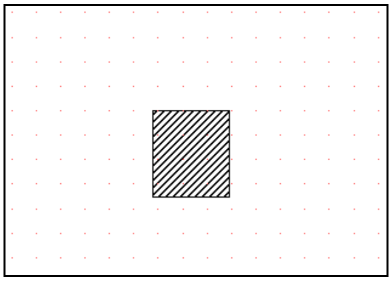


Figure 17: C = A AND B

The optional NOT keyword will perform the operation with the inverse of the layer.

Example: **C = A AND NOT B**

This rule will perform a Boolean AND of the inverse of layer B with layer A. In other words, layer B is used to etch layer A.

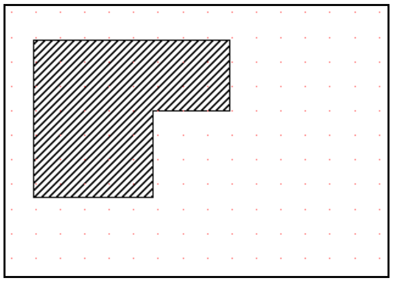


Figure 18: C = A AND NOT B

result_layer = [NOT] *layer1* **OR** [NOT] *layer2*

This rule will create the union of all shapes on layers *layer1* and *layer2*.

Example: **C = A OR B**

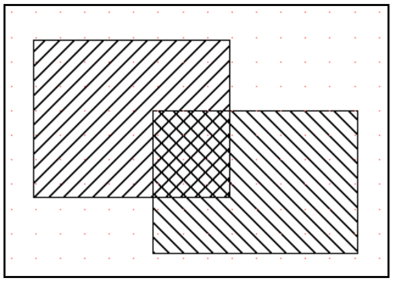


Figure 19: Polygons on layers A and B

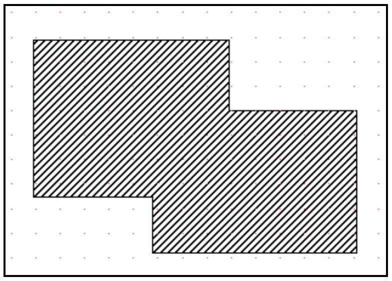


Figure 20: C = A OR B

The optional NOT keywords will perform the operation with the inverse of the layer instead of the original layer.

result_layer = [NOT] *layer1* **XOR** [NOT] *layer2*

XOR stands for "exclusive or". Use the XOR rule to create the union of the two layers and then subtract their intersection.

Example: **C = A XOR B**

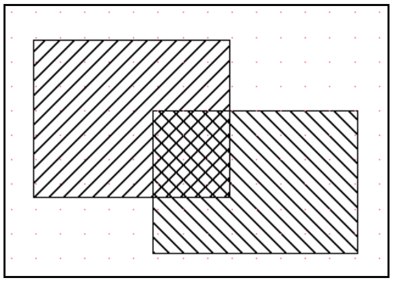


Figure 21: Polygons on layers A and B

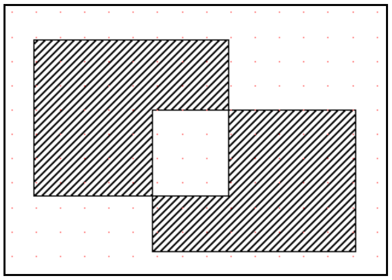


Figure 22: C = A XOR B

The optional NOT keywords work in exactly the same manner as they do in the AND rule.

result_layer = *layer1* **IN_CELL** *cell_name*

The INCELL keyword of the INPUT LAYER rule **will not** include shapes in subcells of the specified cells.

This rule will classify shapes on a layer by whether or not they are contained in specific cells. It works in a similar manner to the IN_CELL parameter of the INPUT LAYER rule. The INCELL rule processes the data differently in two ways:

> *layer1* can be any layer in the NLE database
>
> and
>
> *layer1* shapes in subcells of the specified cells **will be** included on *result_layer*.

There are no optional NOT keywords in this rule. For other details on the use of the INCELL rule, including using wildcards to specify several cells with the *cell_name* parameter, see page 54.

Whenever you remove material from a conductive layer, you must be careful not to prevent real shorts from being found. See page 100 for important examples of how real design errors can be hidden by using INCELL processing.

result_layer = **BLOAT** *(layer1, offset_val)*

Use the BLOAT rule to expand polygons on *layer1* and store them on *result_layer*. All sides of the polygons will be shifted outwards in a parallel manner by *offset_val*. *offset_val* must be a positive real number in ICED32™ user units.

Example: **A = BLOAT (B, 1.2)**

The LVS *.LAYMODEL device models can adjust the dimensions of devices for shrinking or bloating without the use of this rule in the NLE run.

Note that the parentheses and comma are required in the BLOAT rule.

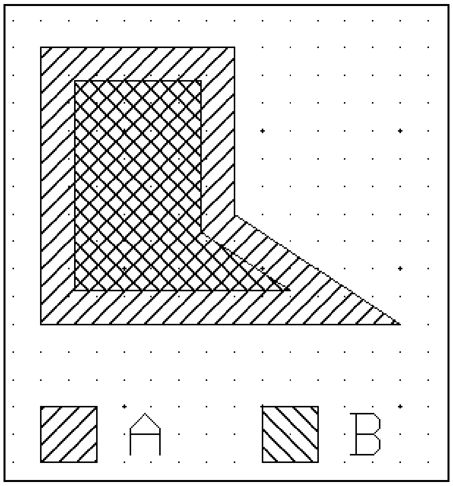


Figure 23: A = BLOAT (B, 1.2)

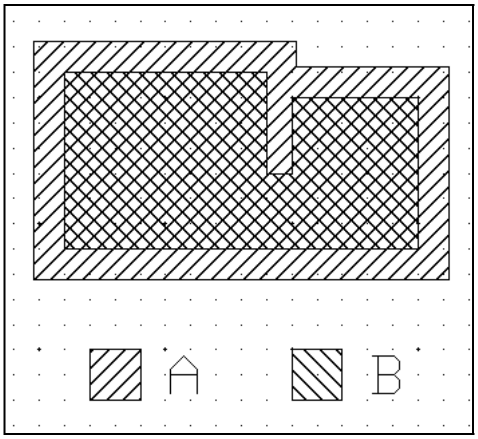


Figure 24: Note that the notch in layer B disappears after bloating.

Bloating can remove features of complex polygons. Notches or holes can disappear.

If you are using BLOAT on polygons with acute angles, you should refer to the BLOAT_ANGLE rule on page 69 for important information on the side effects of bloating sharp angles.

result_layer = **SHRINK** *(layer1, offset_val)*

Use the SHRINK rule to store on *result_layer* polygons on *layer1* which have been shrunk by *offset_val*. All sides of the polygons will be shifted inwards in a parallel manner by *offset_val*. *offset_val* must be a positive real number.

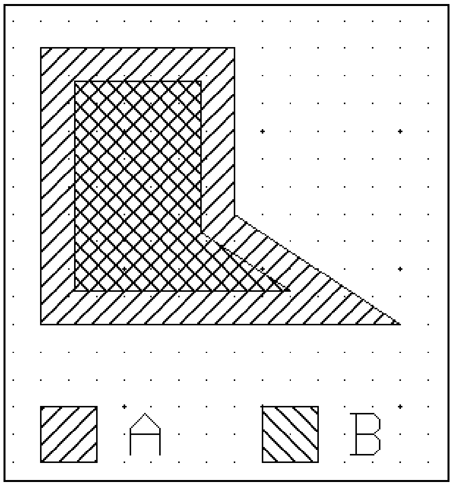


Figure 25: B = SHRINK (A, 1.2)

Example: **B = SHRINK (A, 1.2)**

The LVS *.LAYMODEL device modifier parameters can adjust the dimensions of devices for shrinking or bloating without the use of this rule.

Note that the parentheses and comma are required in the SHRINK rule.

Polygons can change shape significantly when being shrunk. Thin sections that become a width of zero or less will simply disappear. Small polygons with either dimension less than twice *offset_val* will disappear entirely.

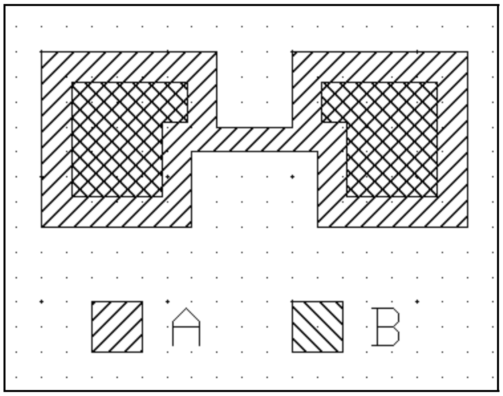


Figure 26: A single polygon on layer A becomes two polygons on layer B after shrinking.

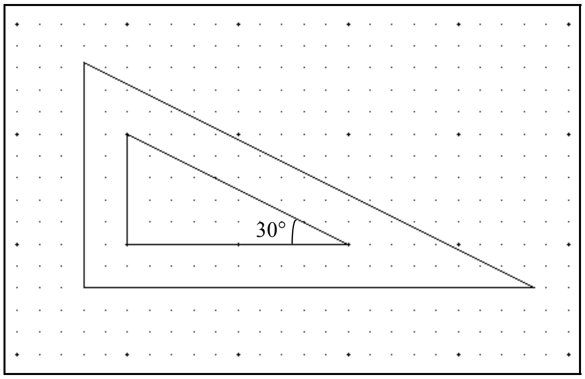
The NLE processes a SHRINK operation as a BLOAT of the inverse of a layer. When you shrink a shape with an acute angle notch, you are really bloating a shape with an acute angle. The bloat of an acute angle can result in significant distortion of your shape. This is why the default behavior of the NLE blunts acute angles before shrinking or bloating.

If you are using SHRINK on polygons with angular notches, you should refer to the BLOAT_ANGLE rule (covered next) for important information on the effects that acute angles can have on this rule.

BLOAT_ANGLE = *bloat_angle*

If you do not have acute angles in your design, you should not use the BLOAT-_ANGLE rule. The default of 45° will prevent excessive run times.

This rule is important only if your layout contains polygons with sharp points. It controls how shapes with sharp points are bloated. Points with angles more acute than *bloat_angle* will be blunted before they are bloated. The *bloat_angle* parameter must be a real number in the range 1:45. When the BLOAT_ANGLE rule is not used, the NLE uses a default of 45° for the bloat angle.



To see why bloating sharp points can be a problem, see Figure 27. The inner triangle has a sharp point with an acute angle of 30°. If this bloat is not constrained, the bottom dimension of the polygon will more than double when it is bloated by an *offset_val* of 2.

Figure 27: Unconstrained bloat of a 30° angle.

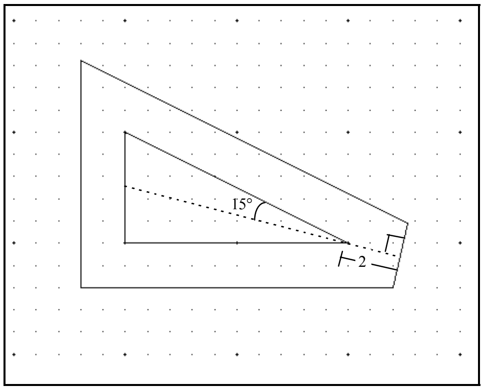


Figure 28: Constrained bloat of a 30° angle.

To avoid this type of expansion for relatively small bloats, the NLE defaults to constraining bloats on any angle less than 45°. The bloated shape is cut by a line perpendicular to the line that bisects the acute angle. The cut will be made at a distance equal to the bloat *offset_val* along the bisecting line. It is as though the point at the acute angle is blunted by an infinitesimal line segment before the bloat.

If this is not how you want your acute angles bloated, you must use the BLOAT_ANGLE rule in your rule set. Set *bloat_angle* to a small enough angle to remove the constraint for critical polygons. However, you should be aware that as the bloat angle gets smaller, the NLE run time gets longer. This is due to panel processing and borders which is a subject not covered here. To understand how the bloat angle affects run times, see **Panel Processing** on page 83.

The bloat angle affects the SHRINK rule as well, since a shrink is really processed as a bloat of the inverse of a layer.

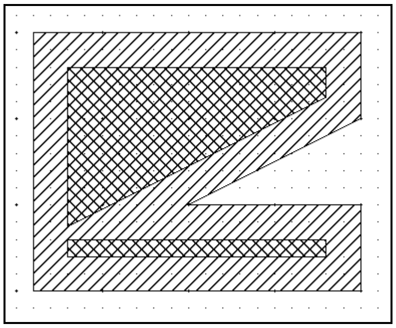


Figure 29: Unconstrained SHRINK of polygon with acute angle notch.

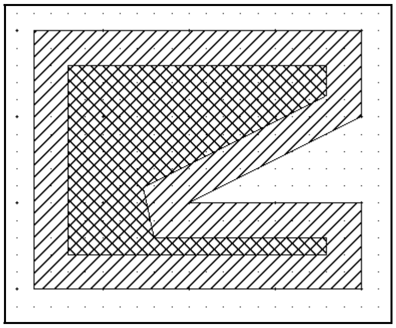


Figure 30: Constrained SHRINK of same polygon.

You can use the BLOAT_ANGLE rule more than once in a rule set.

Example:

B = BLOAT (A, 2)
BLOAT_ANGLE = 10
C = BLOAT (A, 2)
BLOAT_ANGLE = 45
D = BLOAT (A, 5.1)

You can see the bloat angle used for each BLOAT rule in the rules compiler log.

In the example above, a bloat angle of 45° will constrain the bloats that create layers B and D. A bloat angle of 10° will constrain the bloats that create layer C. However, since one of the layers uses such a small bloat angle, the run time will be much longer than if all layers used the default.

result_layer = *layer1* [NOT] **TOUCHING** [*n1* [:*n2*]] *layer2* [NOT = *result_layer2*]

This rule is used to classify polygons on *layer1* based on whether or not they touch polygons on *layer2* along a finite line or area. Polygons touching only at a point, as shown in Figure 31, are **not** considered to be touching.

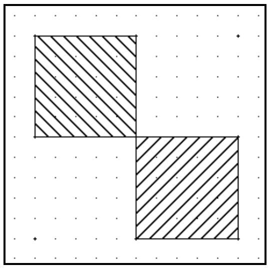


Figure 31: Polygons that do NOT touch.

Example: **C = A TOUCHING B**

In this example, layer C will contain all polygons on A which touch at least one polygon on layer B

Only one optional NOT keyword can be used in the TOUCHING rule.

Example: **D = A NOT TOUCHING B**

In this case, layer D will contain all polygons on layer A which do not touch any polygons on layer B.

Example: **C = A TOUCHING B NOT = D**

Layer C will contain all polygons on layer A that touch at least one polygon on layer B. Layer D will contain all remaining shapes on layer A, i.e. all shapes not touching layer B.

The optional *n1* and *n2* parameters can be used to specify how many polygons on *layer2* the polygons on *layer1* must touch. Use *n1* alone to specify an exact number. Use both *n1* and *n2* to specify a range.

Example: **C = A TOUCHING 2 B**

In this case, layer C will contain all polygons on layer A that touch exactly two polygons on layer B.

Example: **C = A TOUCHING 2:4 B**

When you use this rule, layer C will contain all polygons on layer A which touch exactly two, three, or four polygons on layer B.

result_layer = *layer1* [NOT] **OVERLAPPING** [*n1* [:*n2*]] *layer2* [NOT=*result_layer2*]

This rule is used to classify polygons on *layer1* based on whether or not they overlap polygons on *layer2* with a finite area. Polygons touching only at a point, or sharing only an edge, are **not** considered to be overlapping. All shapes that overlap also touch (see the previous rule).

Example: **C = A OVERLAPPING B**

In this example, layer C will contain all polygons on A which overlap with a finite area at least one polygon on layer B

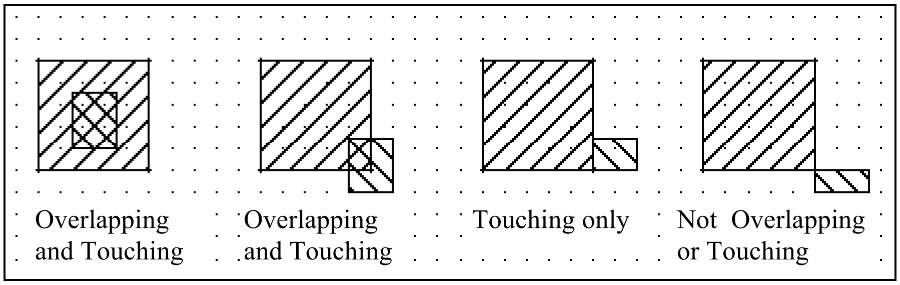


Figure 32

Only one optional NOT keyword can be used in the OVERLAPPING rule. The NOT keywords and the *n1* and *n2* parameters work in exactly the same manner as they do in the TOUCHING rule. See page 71 for examples.

result_layer = **ISLANDS** (*layer1*)

If you want to insure that a shape is not **electrically** connected to other shapes, see the UNCON-NECTED rule on page 155.

This rule is used to find holes or unconnected polygons on a specific layer. All shapes on *layer1* that are not connected to the upper left polygon on *layer1* will be copied to *result_layer*. By connected, we do not mean electrical connections through use of the CONNECT rule. For this rule, connected means shapes on one layer which touch other shapes on the same layer.

To find holes in a layer, you use this rule to find islands in the inverse of the layer.

Example:

NOT_A = NOT A
B = ISLANDS (NOT_A)

This pair of rules will result in polygons on layer B created for all holes in layer A. Note that the parentheses are required in the ISLANDS rule.

result_layer = [NOT] **IS_BOX** (*layer1*, *size1* [, *size2* [..., *sizen*]]) [NOT=*result_layer2*]

See the BOUNDS rule (covered next) for a similar rule for non-rectangular shapes.

This rule is used to classify polygons on *layer1* based on whether or not they are rectangles in a range of sizes. To be recognized by this rule, rectangles must be square with the axes (i.e. the sides must be vertical and horizontal).

(Remember that all shapes on the same layer are merged by the NLE. Rectangles that touch another shape on the same layer will be merged during preprocessing. When a rectangle is merged with touching shapes, the resulting shape may no longer be rectangular.)

The syntax of each *sizen* parameter is:

(***xmin*** **[:** ***xmax*****],** ***ymin*** **[:** ***ymax*****])**

To allow the dimensions of the rectangles to be in a range, specify both the minimum dimension and the maximum dimension separated by a colon (':'). To specify an exact dimension, type only the minimum value. When the maximum value is not included, it is assumed to be equal to the minimum. Each dimension must be a positive real number. The units of each dimension are the user units in the ICED32™ cell.

You can enter up to ten *sizen* parameters. You must enter at least one.

Example: **B = IS_BOX (A, (10,5))**

This rule will collect on layer B all rectangles on layer A which are 10 units wide in the x-direction and 5 units high in the y-direction.

Note that orientation is important. To collect non-square rectangles which may be in either orientation you **must** specify two sizes.

Example: **B = IS_BOX (A, (10,5), (5,10))**

This rule will collect on layer B all rectangles on layer A which are 10 units wide by 5 units high in either orientation.

Example: **B = ISBOX (A, (10:12, 5:7))**

Here, layer B will consist of all rectangles on layer A which are from 10 to 12 units wide in the x-direction and from 5 to 7 units high in the y-direction.

(Note that the underscore in the IS_BOX keyword is optional. The underscore character is simply ignored when it is present. This is true of all keywords.)

Example: **B = IS_BOX (A, (10:12, 6.4))**

This rule will collect on layer B all rectangles on layer A which are from 10 to 12 units wide in the x-direction and exactly 6.4 units high in the y-direction.

The optional NOT keywords are used to restrict the output layer to all shapes which do not meet the size criteria. Only one optional NOT keyword is allowed.

Example: **C = NOT IS_BOX (A, (10, 2), (11, 3))**

This above rule will collect on layer C all shapes on layer A which are **not** rectangles 10 units wide and 2 high or 11 units wide and 3 high.

When typing this rule, you may start a new line between sizes. You cannot split a single *sizen* parameter between lines. (The final optional NOT keyword must be on the same line as the closing parentheses.)

Example:

```
B =  IS_BOX ( A,
                 ( 10, 2 ), ( 2, 10 ),
                 ( 5, 1 ),   ( 1, 5))  NOT = C
```

This above rule will collect on layer B all shapes on layer A which are rectangles 10 units wide and 2 high or 5 units wide and 1 high in either orientation. Layer C will consist of all other shapes on layer A, including non-rectangular shapes.

result_layer = [NOT] **BOUNDS** (*layer1*, *size1* [, *size2* [..., *sizen*]]) [NOT=*result_layer2*]

This rule is very similar to IS_BOX, (see the previous rule) except that the size criteria applies to the bounding box of any shape rather than to the dimensions of only rectangles. The bounding box of a shape is the smallest rectangle, square with the axes, which will enclose the shape. The bounding box of a rectangle square with the axes has the same dimensions as the rectangle itself.

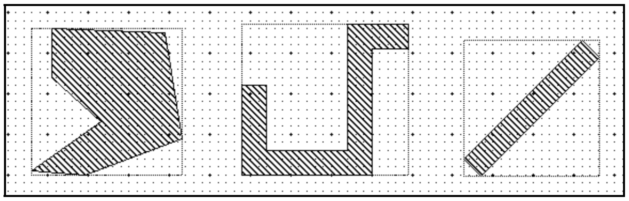


Figure 33: Bounding boxes of non-rectangular shapes.

Example: **B = BOUNDS (A, (10,5))**

This rule will collect on layer B all shapes on layer A which have bounding boxes 10 units wide in the x-direction and 5 units high in the y-direction.

The syntax of the *sizen* parameters, and the use of the optional NOT keywords, is exactly the same as the IS_BOX rule. See that rule (starting on page 75) for details.

result_layer = [NOT] **ASPECT_RATIO** (*layer1*, *max_size*, *size1* [,*size2* [...,*size10*]])...
... [NOT = *result_layer2*]

This rule is used to classify polygons on *layer1* based on their aspect ratios. An aspect ratio is the ratio of the dimensions of the bounding box. For example, if you have a bounding box 10 units wide (in the x-direction) and 5 units high (in the y-direction), it would have an aspect ratio of:

$$\frac{10}{5} = \frac{2}{1}$$ or 2 to 1.

See page 83 for a thorough explanation of panels and borders.

The required *max_size* parameter is used specify the maximum size of a bounding box guaranteed to be classified correctly. This relates to the problem of panels and panel borders. The NLE verifies large designs one panel at a time. Shapes which cross the edge of a panel must lie within a border around the panel to be verified correctly with a rule. Usually the border is determined by calculating the reach of each rule. The reach is the minimum border required for a rule to guarantee it will process shapes which cross the border properly. Since no maximum dimension is included in this rule (only the ratio of dimensions) there is no way for the rules compiler to calculate the reach. You must specify the reach explicitly with *max_size*.

Specifying too large a value for *max_size* may slow processing. However, you should be aware that shapes with a bounding box dimension larger than *max_size* may be classified incorrectly when the NLE uses panel processing.

The *sizen* parameters are specified in the same manner as the IS_BOX and BOUNDS rules. The first real number of each pair is the relative dimension in the x-direction while the second is the relative dimension in the y-direction.

Example: **B = ASPECT_RATIO (A, 20, (10, 1))**

This rule will collect on layer B all shapes on layer A which have bounding boxes with an aspect ratio of exactly 10 to 1. (10 in the x-direction to 1 in the y-direction.) Shapes with a bounding box dimension larger than 20 user units may be incorrectly classified.

To expand the above rule to include shapes which have the same ratio, but which are longer in the y-direction, you need to add a *size2* parameter that specifies a 1 to 10 aspect ratio.

Example: **B = ASPECT_RATIO (A, 20, (10, 1), (1, 10))**

You can specify up to 10 *sizen* parameters. You may specify a range instead of an exact ratio for *sizen*. The syntax for this is the same as the IS_BOX and BOUNDS rule. You specify a range of valid ratio values in the form *min*:*max*

The NOT keyword is used to collect all shapes on *layer1* which do not meet the aspect ratio criteria.

Example: **B = ASPECT_RATIO (A, 20, (5:6, 1), (7, 1:2)) NOT = C**

Layer B will consist of all polygons on layer A whose bounding boxes have aspect ratios between 5 to 1 and 6 to 1 or between 7 to 1 and 7 to 2. Layer C will consist of all shapes on layer A that do not meet the criteria.

BRIDGE {

BRIDGE = *bridge_layer*

POSTS = *post_layer*

LENGTH = *min_length* [: *max_length*]

WIDTH = *min_width* [: *max_width*]

IS_BRIDGE = *result_layer*

NOT_BRIDGE = *result_layer_2*

$\}$ must use one, can use both

[L/W =*min_ratio* [: *max_ratio*]]

[POINT_TOLERANCE = *tolerance_1*]

[POST_TOLERANCE = *tolerance_2*]

}

The BRIDGE rule is used to find air bridges. If you don't know what an air bridge is, it is unlikely that you will ever need this rule. It is of interest primarily to users of the gallium arsenide technology.

The BRIDGE, POSTS, LENGTH, and WIDTH keywords are required. At least one of the IS_BRIDGE or NOT_BRIDGE keywords must be used. You can use both. The other parameters are optional conditions that must be met for the shapes on *bridge_layer* to be considered air bridges.

To be a valid air bridge, a polygon must meet the following conditions:

The polygon on *bridge_layer* must be rectangular. The rectangle does not need be square with the axes.

The polygon on *bridge_layer* must share opposite end-sides with polygons on *post_layer*, one at each end. Each end-side of the air bridge must be coincident with a side of the post. The post may be wider than the bridge, but the entire end-side of the bridge must be touching the post.

The bridge must fall within a certain range of lengths, widths, and aspect ratios (length/width). The length is the distance between end-sides shared with a post. The width is the distance between the other two sides.

When the IS_BRIDGE keyword is used, the *result_layer* will contain all shapes that meet the air bridge criteria. When NOT_BRIDGE is used, *result_layer_2* will contain all shapes on *bridge_layer* which are not air bridges.

When entering the LENGTH and WIDTH parameters, you can enter either a single size or a range. To enter a range, use a colon (':') to separate the maximum value from the minimum value.

Example:

```
INPUT LAYER       5  METAL;  6 POST
OUTPUT LAYER      38 BRIDGE_OUT
SCRATCH LAYER     BRIDGE_IN

BRIDGE_IN = METAL AND NOT POST
BRIDGE {
        BRIDGE = BRIDGE_IN
        POSTS = POST
        WIDTH = 2
        LENGTH = 5:20
        IS_BRIDGE = BRIDGE_OUT
}
```

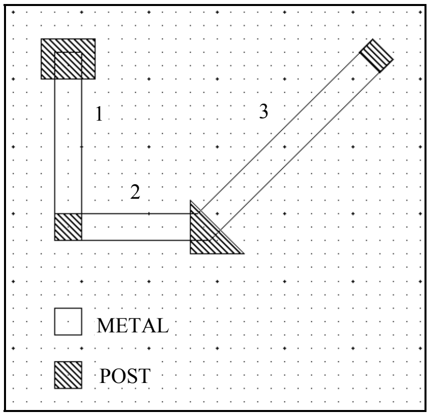


Figure 34: 3 Air bridges.

This set of rules will recognize bridges from 5 to 20 units long where layer METAL is crossed by shapes on layer POST. Note that the AND rule which etches METAL with the POST layer is required. The layer BRIDGE_OUT will contain rectangles for bridges 1 and 2 as shown in Figure 34.

Candidate 3 is a special case since it is not square with the axes. You cannot predict the exact length or width of air bridges that are not square with the axes due to vertex approximations. Unless all air bridges are horizontal or vertical, enter a range of lengths and widths. To modify the above rule to recognize candidate 3 as a valid air bridge, change the WIDTH parameter to:

```
        WIDTH = 1.99 : 2.01
```

Use the optional L/W keyword to add an additional length to width ratio constraint. You can enter a single ratio by using only *min_ratio*, or specify a range by using *max_ratio* as well. Either ratio can be entered in fraction form (e.g. "5/3") or as a single number in decimal form (e.g. "1.6667").

Example: **L/W = 5/1 : 6/1**

Add this parameter to the BRIDGE rule to restrict the valid bridges to those with aspect ratios between 5 to 1 and 6 to 1. Adding this parameter to the BRIDGE example above will result in only bridge 1 (see Figure 34) on the BRIDGE_OUT layer.

The optional POINT_TOLERANCE parameter defines the spacing tolerance for the corners of the air bridge. This accounts for small round-off errors in air bridge corners where the air bridge is not square with the axes. Each corner of the bridge must be within *tolerance_1* units in both the X and Y directions of where it would be if the bridge were exactly a rectangle.

Use the optional POST_TOLERANCE parameter to allow a small overlap or misalignment between the post sides and the bridge layer sides. The point at one end of a *bridge_layer* side must be within *tolerance_2* units in both the X and Y directions of the equivalent point on the post edge. However, if the bridge shape does not touch the post shape, the bridge shape will not be considered a bridge. The touching criterion (i.e. the bridge shape must touch 2 shapes on the post layer) must be met before the BRIDGE rule will examine the other criteria to determine if the shape is a bridge.

You can enter more than one parameter on a line if you separate the parameters with commas.

Panel Processing

Panel processing can also isolate shorts. See page 192 for an example.

The NLE can process small designs as a unit; however, larger designs may need to be divided into panels and processed one panel at a time. If you do not specify the panel size with the PANELX and PANELY rules, the NLE will first attempt to process the entire design as a unit. This can be very expensive in terms of memory requirements and running time. If the NLE runs out of memory, it will begin the entire process again after dividing the design into panels half the size of the entire design. This process may be repeated with smaller and smaller panels. This type of thrashing may waste considerable time.

Since the memory requirements are so high when the NLE processes a large design as one unit, the NLE may be forced to swap data to disk. Disk swapping will result in long run times.

If you have a small amount of memory on your computer (less then 16Meg), then dividing your design into panels may allow the NLE to run to completion when it has run out of memory trying to process the entire design as one panel. Even if you have a large amount of memory on your computer, dividing the design into panels may speed up the NLE run by over an order of magnitude.

For example, a chip that took over 8 hours to process as a single panel took only and hour and a half to process when divided into "reasonable" panels. When you have long run times, you should use the PANELX and PANELY statements to divide the design into smaller panels.

One indication that the NLE will run faster if you specify smaller panels, is when the log file from your first run reports that the NLE is swapping data to disk. The NLE reports at the end of the log file the size of the scratch file, the number of times it was used, and the percentage of processing time spent on swapping. If these numbers are large, trying a smaller panel size will probably result in a shorter running time. (In the testcase mentioned above, the log stated that 81% of the 8 hours was spent on disk swapping.)

If you will be running the NLE many times on your design, you should experiment with different panel sizes to find an optimum panel size. This can speed up the NLE processing time dramatically.

The optimum panel size varies greatly depending on the size of your design, the dimensions of your shapes, and on the type of rules you are processing. However, a rough rule of thumb, if most of your shapes and rules involve dimensions on the order of a few ICED32™ units, is to use panels on the order of 300 by 300 units. (If you have less than 32 Megabytes of memory in your computer, you may want to start with panels smaller than this.)

The NLE attempts to divide the design into roughly equal panels. The dimensions you specify with the PANELX and PANELY rules are really the maximums rather than the exact dimensions used. If you specify PANELX = 100 and PANELY = 200 and your chip is 190 by 489 units, the chip will be divided into six 95 by 163 unit panels.

During preprocessing, touching shapes on the same layer are merged. When multiple panels are used, only shapes from a single panel are merged. If there was no overlap of panels, many rules would miss errors due to touching polygons not being merged, or because nearby shapes would not be considered.

In order for shapes near or crossing a panel border to be processed correctly, the NLE must include a border around all sides of each panel. Shapes in the border area will be processed at least twice (at least four times near the corners of panels). Very small panels or very large borders will result in some shapes being processed many times. However, borders that are too small may allow errors to go undetected.

Add the SHOW-_BORDER option to the NLE command line to see how the border is calculated by the NLE.

The panel border is calculated by the NLE based on the reach of each layer as determined by the rules. Reach is defined as the minimum border distance that will insure that no polygon is incorrectly classified by the rules that generate that layer.

Rule	**Reach of *result_layer***
AND	max (Reach(*layer1*), Reach(*layer2*))
ASPECT_RATIO	*max_size* parameter
Assignment Rule	Reach(*layer1*)
BLOAT	Reach(*layer1*) + $\frac{\textit{offset\_val}}{\sin(\textit{bloat\_angle} / 2)}$
BOUNDS	Reach(*layer1*) + max (*sizen* dimension)
BRIDGE	0
CAPACITOR	0
CONNECT	0
DEVICE	0
IN_CELL	Reach(*layer1*)
IS_BOX	Reach(*layer1*) + max (*sizen* dimension)
ISLANDS	0
OR	max (Reach(*layer1*), Reach(*layer2*))
OVERLAPPING	0
SHRINK	Reach(*layer1*) + $\frac{\textit{offset\_val}}{\sin(\textit{bloat\_angle} / 2)}$
TOUCHING	0
TRANSISTOR	0
XOR	max (Reach(*layer1*), Reach(*layer2*))

Figure 35: Reach calculation for each rule.

Rules that involve a touching criterion have a reach of 0 due to the way the NLE processes touching shapes. Rules that involve dimensions and rules that involve changing the dimensions of shapes (like the BLOAT rule) require a reach to insure that shapes are processed correctly. The reach affects only passes in which the non-zero reach rules are processed.

Each layer is initially assigned a reach of 0. Rules may increase this reach. The reach of a *result_layer* is often greater than the reach of the layers used to create it. The border is defined as the maximum reach of all layers plus one extra NLE internal unit (much smaller than an ICED32™ user unit) for an extra safety factor.

The ASPECT_RATIO rule cannot compute reach, so you must specify it explicitly in the rule using the *max_size* parameter.

See the BLOAT-_ANGLE rule on page 69 for more details on bloating acute angles.

The SHRINK and BLOAT rules can increase reach dramatically if you allow bloats of acute angles in your design.

For example, look at the small polygon with the 30° angle in Figure 36. When this polygon is bloated by 2 without constraints, the bottom dimension expands from 10 to more than 20.

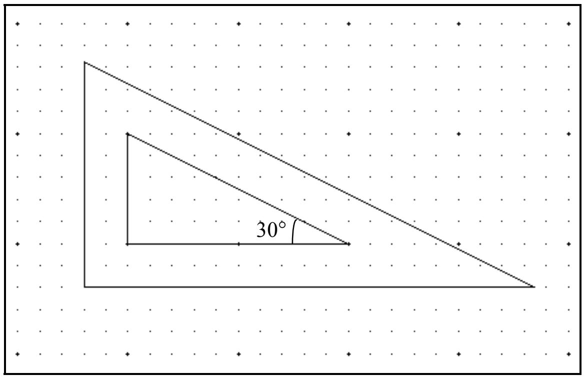


Figure 36: Unconstrained bloat of a 30° angle.

If you do not have acute angles in your design, you should always use the maximum BLOAT-_ANGLE of 45° (the default) to avoid excessive reaches.

The reach of a bloated layer is defined as:

Reach(*layer1*) + *offset_val* / sin (α / 2)

Where α is the *bloat_angle* parameter defined with the BLOAT_ANGLE rule. If the reach of layer1 is 0, the *offset_val* is 2, and the *bloat_angle* is 30, the reach of the bloated layer will be:

$0 + 2 / \sin (30 / 2) = 7.27$

The reach increases dramatically as the bloat angle decreases. If the bloat angle is set to 1 for the example above, allowing unconstrained bloats of angles as small as 1°, the reach of the bloated layer goes up to 229.

The BOUNDS and IS_BOX rules add a reach to the *result_layer* equal to the amount of the maximum dimension you are verifying. If we use a BOUNDS rule with a maximum dimension of 10 units on a bloated layer with a reach of 7.27, the reach of the *result_layer* created by the BOUNDS rule is now 17.27.

If you use the layer created by the BOUNDS rule in other rules, the reach may go up even more. A reach this large is required to be absolutely sure that no polygons are improperly processed with these rules. However, this reach may be excessive for the other layers. Many polygons will be processed multiple times due to the large border.

The panel border used by the NLE is reported in the log file. Search for the phrase "Panel Border". You can use the SHOW_BORDER option on the NLE command line to report the reach and border calculations performed by the NLE. You can sometimes rewrite rules to reduce the reach. If you are including a few rules not required for circuit recognition or ECC checks which are resulting in a large reach, you may want to move them to a different rule set you run less often.

If you are an NLE expert, and your rule set creates a border that you know is excessive; you can override the border calculated by the NLE with the BORDER rule. However, **if you don't know exactly what you are doing, you can easily prevent real errors from being found by tampering with the border.**

If you have a large border and a small panel size, your run may be aborted with the message, "Panel is too small to subdivide further -- check aborted". This means that the border is at least one half the size of a panel. You will need to reduce the border by rewriting your rules or increase the panel size. You can modify the border explicitly with the BORDER rule, but remember that you can corrupt the validity of the NLE tests by doing this.

In no case will border or panel processing cause a device to be recognized twice.

PANELX [=] *panel_x_dimension*

and

PANELY [=] *panel_y_dimension*

To understand how panels are used, you should read **Panel Processing** on page 83.

These rules can be used to set the maximum panel size the NLE will use. You should use these rules if your design is large and your NLE run has an excessive run time. The NLE always defaults to attempting to process the entire design as one panel when you do not use these rules in your rule set.

Even when your run time is acceptable, you can force the NLE to process the design in panels to isolate problems like shorts. (See page 192.)

The NLE reports the amount of time spent disk swapping near the end of the log file. If the NLE is spending a majority of the processing time in disk swapping, you should try reducing run time by using the PANELX and PANELY rules to force the NLE to process your design in smaller portions.

Both panel dimensions should be positive real numbers in user units. Since the NLE will divide your design into roughly equal panels, the actual size of your panels will probably be somewhat smaller than the values you set with these rules.

Example:

PANELX = 300
PANELY = 300

Let us say that your design is 720 user units in the x-direction and 580 user units in the y-direction. When the above rules are used to set the panel size, the design will be divided into six 240 by 290 panels.

BORDER [=] *border_dimension*

It is very important to read **Panel Processing** on page 83 before using this rule.

Use this rule to override the panel border calculations that the NLE performs and set the panel border directly. This rule should be used only when you need to set the panel border to a smaller dimension than that calculated by the NLE. **Make sure that you know what you are doing before you use this rule.**

It is dangerous to set the panel border to a value smaller than the value calculated by the NLE. **You can cause real errors to be missed since the reach of some layers will now be greater than the border.**

Hierarchical Processing

Future versions of the NLE will perform circuit recognition in a true hierarchical manner. (Device recognition and electrical connections would be performed once in a cell then used over and over when many copies of the cell exist in the main cell.) However, this version of the NLE flattens nested cells in the layout so that one main flat cell is used for circuit recognition.

Cells which occur in the INCELL parameter of an INPUT LAYER rule or in an INCELL rule will remain nested during some of the NLE preprocessing passes. However these cells are flattened before circuit recognition or ECC checking is performed.

The ALL_SAFE, DANGER_CELL, etc. rules which are used in the DRC are not used by the NLE since all processing is performed on flattened data.

NLE Circuit Recognition

Circuit recognition in the NLE is performed through a combination of:

Layer generation rules	that combine input layers and classify shapes by certain criteria,
CONNECT rules	that define how shapes on different layers form electrically connected nets,
	and
Device recognition rules	that identify device nodes and the nets which connect to the terminals or pins.

The layer generation rules were already covered beginning on page 60. The CONNECT rule is covered next. The device recognition rules are covered beginning on page 105.

You can label nets or device nodes in your design that will allow shorts and opens to be found by the NLE. Labels make the layout netlist easier to use when you execute the LVS. The details on how to add these labels are covered beginning on page 138. The few rules that control the ECC checks (used mainly to find opens and shorts) are covered beginning on page 152.

Electrical Connections

Electrical connections between layers are formed by the NLE using the CONNECT rule.

The nets are formed by assigning node numbers. Initially, each polygon in the NLE database is assigned a unique node number. As new polygons are formed from other layers, the new polygons are assigned new node numbers. When the CONNECT rules are processed, two touching polygons on layers which are defined as electrically connected will both be assigned the lower node number of the pair. Eventually, all polygons in an electrically connected net are assigned the same node number. (You can use this node number to highlight the entire node in the ICED32™ layout editor using the node outliner commands after the NLE run.)

See page 389 to learn how to use the node outliner commands.

Layers which are used in CONNECT rules form groups. All layers that are connected to each other are collected into a single group. When layers form more than one group, there is no way to electrically connect a shape on a layer in one group to a shape on a layer in separate group. This is almost always a mistake.

The number of groups is reported in the log file created by the rules compiler. If you have more than one group, you should look carefully at the log file where the layers in each group are listed to be sure that you are not forgetting to connect some layers.

You can specify electrical connections to layers that are poor conductors with the STAMP rule. This rule insures that each shape on a poor conductor layer electrically connects to exactly one node. A shape on a poor conductor layer cannot short together two shapes on conductive layers.

The STAMP rule is described beginning on page 102.

CONNECT *layer1 layer2* [BY *layer3*]

See the STAMP rule to form electrical connections to layers that are poor conductors.

The CONNECT rule will form electrical connections between touching shapes on the given layers. When the BY keyword is not used, shapes on *layer1* which touch shapes on *layer_2* are considered to be electrically connected for purposes of building the netlist and for ECC shorts and opens checking. All shapes that are electrically connected will be considered the same node and will be assigned the same node number.

The touching criterion for the CONNECT rule is the same as that used for the TOUCHING rule. Two shapes are considered touching if they share a finite area or if their edges share a finite length. Shapes that touch only at a point are not considered electrically connected.

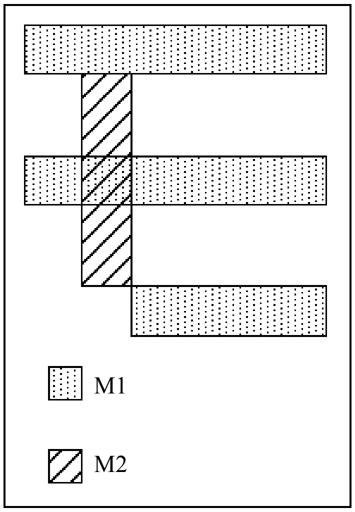


Figure 37: The top two M1 wires will be electrically connected to the M2 wire.

Example: **CONNECT M1 M2**

Do not electrically connect layers that are used as device id layers. These electrical connections are formed by the device recognition rules.

When this CONNECT rule is used, any shape on M1 which overlaps or shares a finite portion of an edge with a shape on M2 will be considered electrically connected to the shape on M2. If this rule is executed on the shapes in Figure 37, the shape on M2 and the top two wires on M1 would all be electrically connected and stamped with the same node number. The bottom wire on M1, which touches M2 only at a point, would be a separate node.

The BY keyword is used to simulate connections between layers which are formed by vias or other contact layers. When the BY keyword is used, the touching criterion changes. For a shape on *layer1* to be electrically connected to a shape on *layer2*, the shape on *layer1,* the shape on *layer2,* and a shape on *layer3* must all share a common area.

Mere touching or overlapping of these layers is not enough to connect the shapes.

Example: **CONNECT M1 M2 BY VIA**

When the above rule is used to connect the M1 and M2 layers shown in Figure 38, only the M1 wire with the label "THREE" will be connected to the vertical M2 wire. Wire "ONE" overlaps the M2 wire and both touch the via shape, but the via shape does not overlap the common area where the metal layers overlap. Wire "TWO" fails to connect for the same reason even though the via overlaps both wires. Wire "FOUR" does not overlap the M2 wire at all, so it does not connect to it even though the via shape overlaps both.

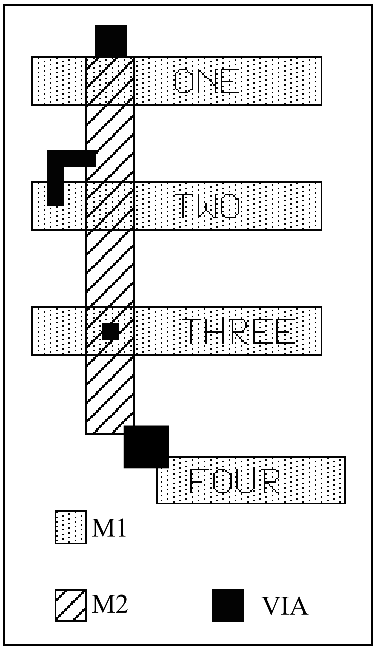


Figure 38: Only M1 wire THREE is connected to the vertical M2 wire.

You combine layer generation rules with CONNECT rules to simulate the fabrication process and electrical connectivity. You may need to process conductive layers carefully before adding the CONNECT rules.

Example:

```
INPUT LAYER       1 N_PLUS;  2 P;  3 N;  10 M1;  8 CONTACTS;
SCRATCH LAYER     EMITTER;  COLLECTOR;  BASE;
SCRATCH LAYER     N_AND_N_PLUS;  CONT_TO_BASE;

N_AND_N_PLUS =    N                   AND    N_PLUS
EMITTER =         N_AND_N_PLUS        AND    P
COLLECTOR =       N_AND_N_PLUS        AND    NOT    P
BASE =            P

CONT_TO_BASE =    CONTACTS            AND    NOT    EMITTER

CONNECT     COLLECTOR   N
CONNECT     M1   COLLECTOR     BY     CONTACTS
CONNECT     M1   EMITTER       BY     CONTACTS
CONNECT     M1   BASE          BY     CONT_TO_BASE
```

Refer to Figure 39 and Figure 40 as we discuss this example. The layer generation and connection rules for NPN transistors demonstrate how you must be careful not to short layers. (We have left out the buried layer to simplify the discussion.)

The order in which the layers are laid down should be considered when you write your rule set. The P layer in a NPN transistor prevents the N_PLUS layer from contacting the N layer. The N_PLUS layer prevents contacts from connecting M1 to the P layer.

If you created the COLLECTOR layer as follows:

```
COLLECTOR = N  AND  N_PLUS
```

then the shapes that make the emitter will also wind up on the COLLECTOR layer. In this case, the N layer will short the collector and the emitter. You must be careful to separate the COLLECTOR layer from the EMITTER layer by using the P layer.

You must classify the contacts that are over the P or BASE layer because the emitter is also over the P layer. Contacts over the emitter do not connect to the P layer, since the N_PLUS layer is in between. If the BASE layer is connected to M1 by CONTACTS, the emitter contact will short to the base since the emitter is on top of the BASE layer.

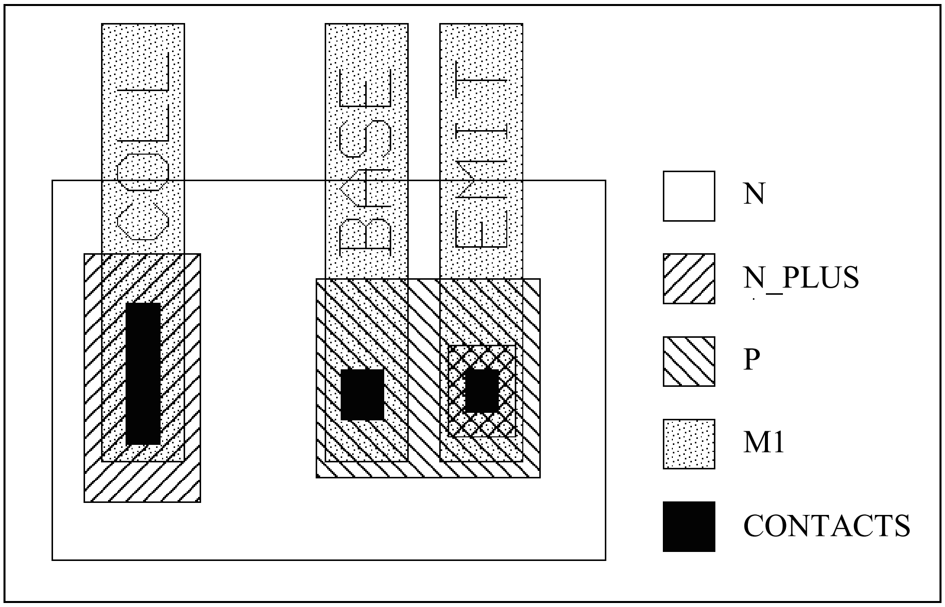


Figure 39: Simplified layout for a NPN device.

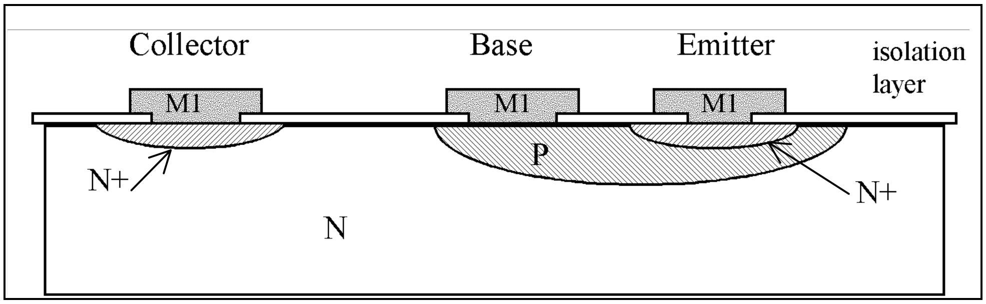


Figure 40: Simplified cross section of a NPN transistor.

Whenever you remove material from conductive layers to recognize a device that is really composed of conductive material, you must use caution to avoid hiding real design errors. The problem is that the area must be removed from the conductive layer to prevent shorts between the terminals of the device, but this may prevent genuine shorts from other pieces of conductive material from being found.

There are four primary methods for changing the area that represents a device from a conductive layer to a device id layer:

1) Use the AND rule to remove the device area from the conductive layer.

Example:

```
GATE =      DIFF AND POLY
SRC_DRN =   DIFF AND NOT POLY
```

In this example, the DIFF layer will not be used as a conductive layer in the CONNECT rules. Instead, the SRC_DRN layer is used as the conductive layer.

2) A shape on a dummy layer is added to the design. This layer is then used to etch the conductive layer, or the TOUCHING rule is used to find shapes that touch the dummy layer and these are removed from the conductive layer. (See an example on page 131.)

3) The IN_CELL rule (or the IN_CELL keyword of the INPUT LAYER rule) changes the all shapes on a conductive layer contained in certain cells to a different layer. In this case, shapes in the main cell which travel over the same area will remain on the conductive layer. (See an example on page 118.)

4) IN_CELL processing is used to save layer 0 (which represents the bounding box of a cell) in certain cells to a scratch layer, which is then used to remove area from the conductive layer. In this case, shapes in the main cell which travel over the same area will be removed from the conductive layer.

When methods 2, 3, or 4 are used, you should add rules to a separate rule set that test for the presence of accidental shorts and other potential problems. These rules can be added to the DRC rule set, or to a separate rule set you run less often than the NLE rule set you use for circuit extraction. The addition of extra rules to your circuit extraction rule set will add to your run time.

Method 3 is usually used when the device id layer is nested in a cell on a conductive layer. Shorts in the main cell on the same layer will not be found in most circumstances. You should add a rule like the following to test for shorts:

Example: **RES_ERROR = POLY TOUCHING RES_POLY**

Do not use an AND rule to test for shorts, or errors where shapes on POLY share only an edge with RES_POLY will not be found.

(If your device rule is written with the conductive layer as one of the touching layers, you may be able to avoid the extra test since the device rule will then test for how many shapes on the conductive layer touch the device id shape. Device id shapes that touch too many shapes on the conductive layer will not be recognized as valid devices.)

When you use method 2, you should consider the dummy layer as important as your design layers. You should add rules to a separate rule set which verify that the shape is in the right place, is the correct size, does not overlap the wrong shapes, etc. Mistakes on the dummy layer may cause problems just as severe as mistakes on the design layers.

Accidental connections may not be found if you fail to use caution when removing area from a conductive layer.

You cannot modify a layer after it is used in a CONNECT rule. This restriction is enforced by the rules compiler. If a layer could be modified after being used in a CONNECT rule, there would be no way to guarantee that the electrical connections made by the CONNECT rule would be valid by the time the device recognition passes and ECC checks are run.

You do not have to add extra rules to connect nodes that cross panel boundaries. The rules compiler will generate CONNECT rules which form electrical connections between shapes on the same layer where they cross panel boundaries.

STAMP *layer1* BY *stamping_layer* MULTI = *error_layer1* [NONE = *error_layer2*]

The STAMP rule is used to form electrical connections to layers that are poor conductors. Shapes on *layer1* which touch a shape on *stamping_layer* will be assigned the node number of the shape on *stamping_layer.* However, even when the shape on *layer1* touches other nodes on the *stamping_layer,* the NLE will not assign the node number of the *layer1* shape to shapes on the *stamping_layer.*

In other words, *layer1* is treated as a non-conductive material which can be "stamped" with a node number, but it cannot "stamp" any conductive layers. Electrical connections on the *stamping_layer* do not pass through *layer1*.

The NLE keeps track of how many times a shape on *layer1* is assigned a node number by shapes on the *stamping_layer.* If the shape on *layer1* is "stamped" by more than one node, it will be copied to an error layer. You can optionally use the NONE keyword to collect on a different error layer all shapes on *layer1* which are not "stamped" at all.

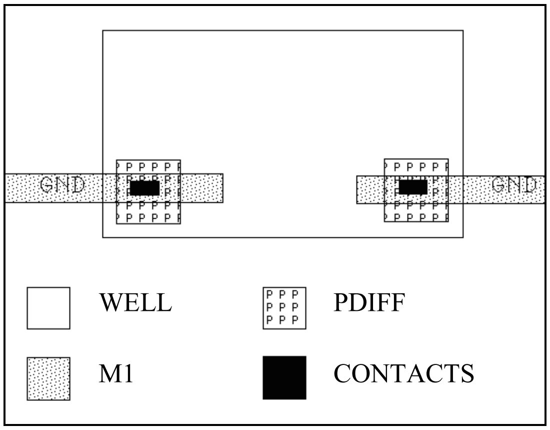


Figure 41: Open on GND node that connects only through WELL layer.

We can demonstrate the importance of not using the CONNECT rule for poor conductors with Figure 41. Let us assume that the GND wire on the right connects to the metal GND bus and from there to a pad on the chip. However, the GND wire on the left does not connect to the bus. You meant to connect these two wires, but a gap exists by accident.

If you use the STAMP rule to connect PDIFF to WELL, the WELL layer will be assigned the same node number as one of the wires, but the other wire will not be assigned the same node number. However, if you use a CONNECT rule to connect the WELL layer to the PDIFF layer, the open will **not** be found since the NLE will assign the same node number to all three shapes.

The STAMP rule can be used to verify that every shape on *layer1* is electrically connected to exactly one node. Any shapes on *layer1*, which touch more than one node on the *stamping_layer* will be copied to *error_layer1*. All shapes on *layer1* which do not connect to any nodes on *stamping_layer* can optionally be copied to *error_layer2* by using the NONE keyword. **The error layers should be defined with an OUTPUT LAYER rule so that the presence of shapes on those layers will be reported as errors in the NLE log.**

Example:

```
INPUT LAYER       1 NDIFF; 2 POLY; 3 PWELL; 4 PDIFF;
INPUT LAYER       10 M1; 8 CONTACTS;
SCRATCH LAYER GATE; SRC_DRN;
OUTPUT LAYER    101 OVER_STAMPED_WELL;

GATE =          NDIFF  AND   POLY
SRC_DRN =       NDIFF  AND   NOT POLY

CONNECT         M1  PDIFF        BY  CONTACTS
CONNECT         M1  SRC_DRN      BY  CONTACTS

STAMP  PWELL  BY  PDIFF  MULTI= OVER_STAMPED_WELL
```

If this set of rules is run on the layout shown in Figure 41, the WELL shape will be copied to the OVER_STAMPED_WELL since it will be stamped by two different nodes on layer PDIFF. The two GND net fragments will not be shorted together and will be recognized by the NLE as two separate nets.

Device Recognition

See page 100 for hints on removing the device id layer from conductive layers.

When using the NLE for device recognition, the general definition of a device is a polygon on an *id_layer* that touches a certain number of polygons on other layers. For most devices, the touching criterion is the same as that for the TOUCHING rule. (See page 71.) The only time the touching criterion is more restrictive, is when you need to extract a width for the device. (This will be discussed in more detail later.)

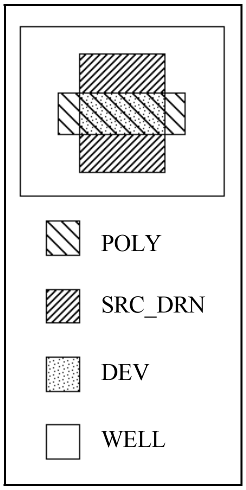


Figure 42: Valid CMOS device

In the case of the CMOS device shown in Figure 42, the *id_layer* is the DEV layer. We can write a DEVICE rule that defines CMOS devices as shapes on layer DEV that must touch:

1 polygon on layer POLY

2 polygons on layer SRC_DRN

and

1 polygon on layer WELL

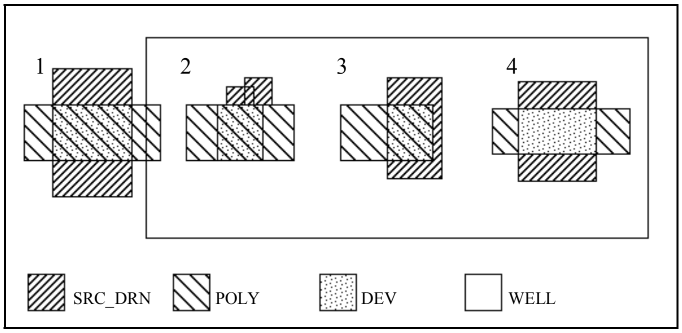


Figure 43: Invalid CMOS devices

While the shape on layer DEV in Figure 42 does meet the criteria for a CMOS device, the shapes in Figure 43 do not. DEV shape number 1 does not touch a shape on WELL. Number 2 fails since both shapes on layer SRC_DRN are joined by the NLE preprocessing into one shape, therefore the shape on DEV touches only one shape on SRC_DRN. DEV shape 3 also touches only one shape on layer SRC_DRN. Number 4 is invalid since the DEV shape touches two shapes on layer POLY.

Usually, the only criterion for recognizing shapes on the *id_layer* as devices is that they touch the right number of shapes on other layers. The manner in which they touch is not part of the criteria used by NLE device recognition. (When width is extracted for a device, additional constraints are added to the simple touching criterion. Details on this follow later.)

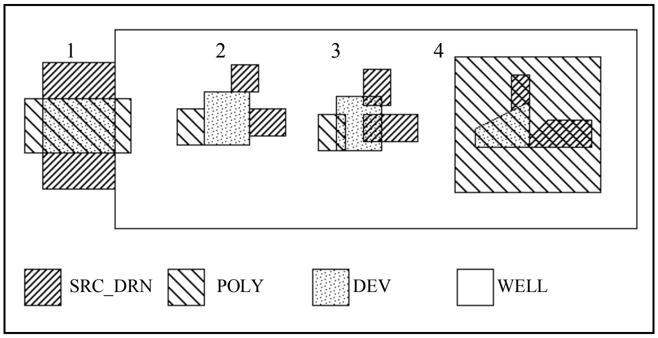


Figure 44: Valid but unusual CMOS devices

Look at the devices in Figure 44. DEV shape number 1 is valid even though it is not covered or even overlapped by layer WELL. Device number 2 is valid despite the fact that the DEV shape does not share entire sides with the SRC_DRN shapes. Device number 3 is valid even though the shapes overlap. Device 4 meets the touching criterion and will be recognized as a device even though it certainly does not meet technology requirements for a CMOS device.

It is dangerous to assume that the NLE device recognition will insure any relationship, other than simple touching, between the shape on *id_layer* and the other layers. If your technology requires more stringent requirements for devices than a simple touching rule, you should write rules that test these requirements. You should usually put these rules in a different NLE rule set that you run separately to avoid long run times every time you run the NLE to extract the netlist. (To create better technology rules tests, you should perform these checks using the DRC, the Design Rules Checker available from IC Editors, Inc.)

To insure that the SRC_DRN layer does not overlap the *id_layer,* or shapes on layer POLY, you could build it with rules similar to:

```
DEV =         DIFF  AND      POLY
SRC_DRN =     DIFF  AND NOT  POLY
```

If you want to insure that entire shapes on the *id_layer* are covered by shapes on layer WELL, you have to verify this with another rule similar to:

ERROR_DEV = DEV AND NOT WELL

However, this type of test does not belong in the NLE rule set you run every time you perform circuit recognition. It is best that you test groundrules in a separate rule set **before** running the NLE, so that devices which are designed incorrectly can be found and corrected before circuit recognition. Device design errors may lead to in incorrect device sizes or circuit recognition problems that are difficult to diagnose with the LVS.

You **cannot** modify a layer **after** it is used in any of the device recognition rules. This restriction is enforced by the rules compiler. The circuit recognized by the device recognition rules could be made invalid if you were allowed to modify device layers afterwards.

See **Electrical Connections** on page 95 for details on how electrical connections are formed from the propagation of node numbers through touching shapes on layers which appear in CONNECT rules.

Be careful about how you use the CONNECT rules to define electrical connectivity. Write the CONNECT rules after the layer generation rules that create the device id layers. Then write the device recognition rules after the CONNECT rules.

The CONNECT rules can not be used to short the device id layers to other layers. This is enforced by the rules compiler. All relationships between the device id layers and the other layers should be defined by the device recognition rule. The node numbers of each shape on a device *id_layer* should be unique and distinct from the node numbers of the terminals of the device.

Example:

```
INPUT LAYER  {
        1       DIFF
        2       POLY
        10      WELL
        3       WIRES
        6       CONTACTS
}
SCRATCH LAYER {
        SRC_DRN
        GATE
}
GATE =     DIFF  AND       POLY;
SRC_DRN = DIFF  AND NOT POLY;

CONNECT   WIRES  SRC_DRN   BY  CONTACTS
CONNECT   WIRES  POLY      BY  CONTACTS
```

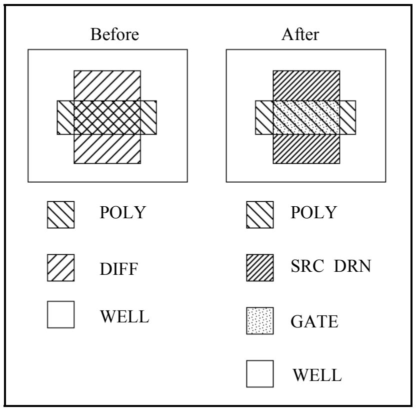


Figure 45: Device layers before and after layer processing.

The example above is a typical set of layer generation and connection rules for NMOS device recognition. GATE will be used as the device recognition *id_layer*. Note that the GATE layer is not used in a CONNECT rule.

See the corresponding NMOS device recognition rule on page 115.

The POLY layer does not have the DIFF layer subtracted from it. The POLY layer remains a conductor even where it is crossed by a shape on DIFF. The POLY layer is not connected to GATE in a CONNECT rule, the electrical relationship will be defined by the device recognition rule.

Note also that the SRC_DRN layer is used in a CONNECT rule instead of DIFF. Once the GATE and SRC_DRN layers are generated from DIFF, the DIFF layer will not be used for any further processing. If DIFF was considered a conductive layer, the source and drain of each NMOS device would be shorted.

If you are recognizing more than one type of device, the order in which you write the device rules usually does not matter. Shapes on *id_layer* recognized as one device type are not removed from consideration for other device rules. However, when a single polygon on *id_layer* is used to form two different devices, you will get an error message.

The other layers the *id_layer* has an electrical relationship to must be listed in the device rule layer definition lines and in a pin (i.e. terminal) list which defines how connections to these other layers are stored as the pins of the device. The node numbers of the shapes on the other layers will be listed as the pins of the device in the binary layout netlist that is passed to the LVS.

It is possible, but almost always an error, to have one shape on *id_layer* form two different devices. See the NO_DUP-_ID_CHECK command line option to see how the NLE tests for this problem.

It is highly recommended that you write the layer processing rules so that each unique device type has its own *id_layer*. Then you can verify that all shapes on each *id_layer* form valid devices by using the [ERR = *error_layer*] parameter in the device rules.

You must use an OUTPUT LAYER rule to define *error_layer*, if you want the presence of shapes on the layer to be reported in the log as errors. If *error_layer* is defined with a SCRATCH LAYER rule, shapes on *id_layer* which do not form valid devices will still be copied to *error_layer*, but no errors or warnings will be mentioned in the log file. You can use *error_layer* in other rules, including other device recognition rules. If the *error_layer* is a scratch layer, and you do not use it in some subsequent rule, the rules compiler will warn you when you compile the rules file.

If you do not use the [ERR = *error_layer*] parameter in your device rules, shapes on *id_layer* which do not form valid devices will simply be ignored, and usually no warning messages will be created. However, if you add the LOGBAD keyword to the NLE command line, the log file will have a message printed for each shape on *id_layer* that does not form a valid device. See page 175 for details.

See page 243 for details on setting default values in a device model.

The NLE does not extract the dimensions of devices recognized by the general-purpose DEVICE rule unless you use the WIDTH or AREA keywords. If the dimensions of the device are not recognized by the NLE, and the value of the device is not specified with a label (see page 346), the NLE will record a value of 0 for the device in the layout netlist. When this is the case, the default value in the corresponding device model will be used as the value of the device by the LVS.

See the descriptions of the device rules below to learn when width or area will be extracted.

If the NLE is instructed to recognize area, the area and perimeter of the polygon will be recorded in the layout netlist. The LVS will use the area and perimeter in combination with device characteristic parameters in the device model to calculate the value of the device.

When you instruct the NLE to recognize device width and length, the NLE must be able to recognize 2 end-sides of the polygon on *id_layer*. End-sides are defined as the sides of the *id_layer* polygon that touch polygons on an *end_layer*. Figure 46 demonstrates this concept. The *id_layer* is GATE and the *end_layer* is SRC_DRN. The width is the average length of the two end-sides.

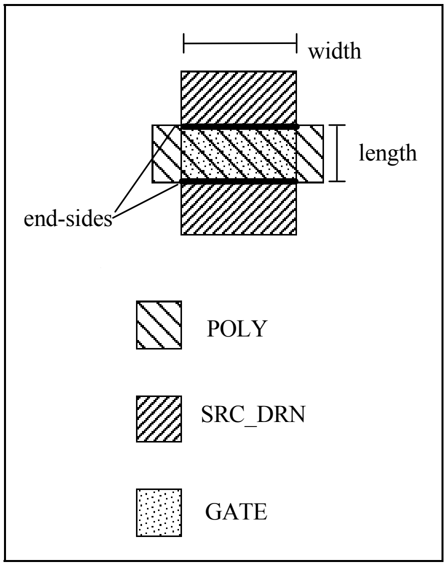


Figure 46: Width recognition of device.

To enable end-sides to be recognized, the touching rule is more restrictive when the width and length are extracted. Three additional constraints are imposed:

Exactly two polygons on *end_layer* must touch the polygon on *id_layer*. (If you define 2 different end layers, the polygon on *id_layer* must touch exactly 1 polygon on *end_layer_1* and 1 polygon on *end_layer_2.*)

The polygon on *id_layer* must not overlap the polygons on the *end_layer* (or end layers).

The entire end-sides of the polygon on *id_layer* must be touching sides of the polygons on *end_layer.*

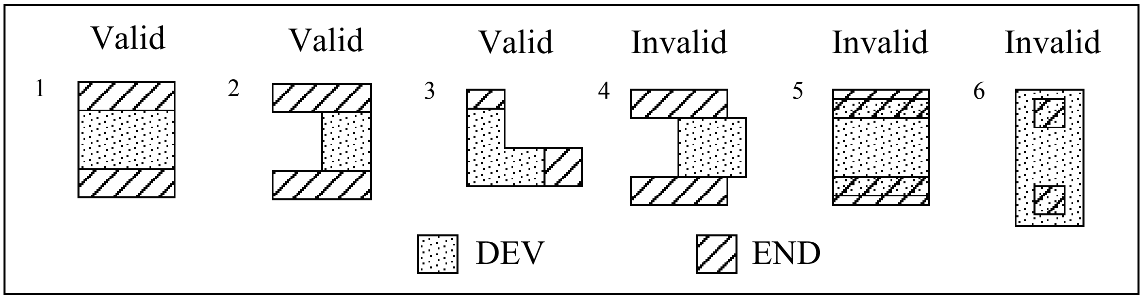


Figure 47: Examples of valid and invalid devices when device width and length are extracted.

See page 118 to learn how to extract a width for devices like number 6.

Devices 1, 2, and 3 in Figure 47 meet the above criteria. Device 4 is invalid because the entire end-sides of the polygon on the DEV layer are not touching the sides of the polygons on layer END. Device 5 is invalid because the polygon on layer DEV overlaps the polygons on layer END. While device 6 is more complex, the problem is similar to device 4. There is no way to determine end-sides of the polygon on DEV that are coincident with sides of polygons on END.

When width is recognized, the device dimensions are calculated as follows:

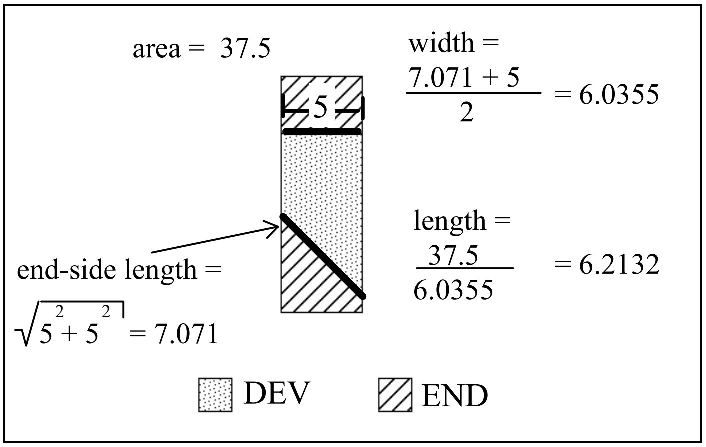
width = (sum of the length of the end-sides of the *id_layer* polygon) / 2

length = area of *id_layer* polygon / width.

If the polygon on *id_layer* is rectangular, both end-sides are equal, and width is equal to the end-side length.

If you have very unusual non-rectangular devices, the NLE may calculate the dimensions incorrectly. You can prevent the LVS from using the dimensions of these devices by using the REST=NO parameter on the device model. See page 241.

The polygon on *id_layer* does not need to be rectangular to extract a length and width. No warning message will be reported for non-rectangular devices. The width will be calculated as the average side length instead of the length of one of two equal sides.



area = 37.5
5
end-side length =
$\sqrt{5^2 + 5^2} = 7.071$
width =
$\frac{7.071 + 5}{2} = 6.0355$
length =
$\frac{37.5}{6.0355} = 6.2132$
DEV
END

Figure 48: Length and width calculation for non-rectangular device.

You can adjust the width of non-rectangular devices for the LVS with the BENDS keyword in the layout device model.

The non-rectangular transistor shown in Figure 49 has two end-sides that are 25 units each. The width of the device will be 25. Since the area of the device is 50, the length will be:

$$\frac{50}{25} = 2$$

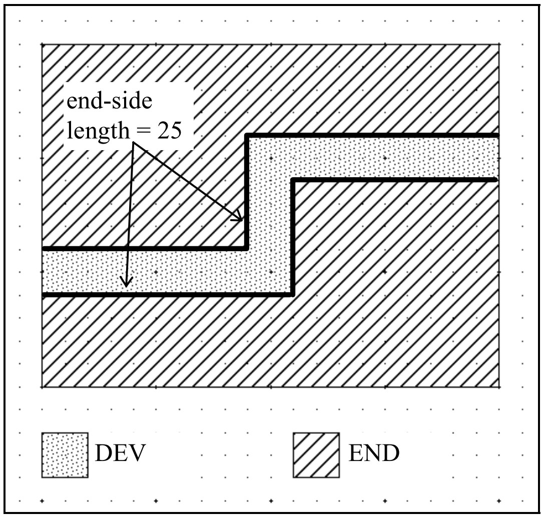


Figure 49: Non-rectangular transistor

There are 3 device recognition rules. The general-purpose DEVICE rule can be used to recognize a wide variety of combinations of layers, especially when combined with layer processing to restrict the *id_layer*. The TRANSISTOR and CAPACITOR rules are really just shorter ways to write DEVICE rules for these types of devices. Both the TRANSISTOR and CAPACITOR rules will perform the same processing as a DEVICE rule, they are just easier to write.

The General Purpose DEVICE Rule

DEVICE *model_name* ID = *id_layer* [/AREA] [ERR = *error_layer*] {

layer_n number_touching/NODE — multiple lines OK

or

layer_n number_touching/POLYGON — multiple lines OK

or

end_layer [2]/WIDTH [/EPS=*spacing_tolerance*] — at most 1 line

or

end_layer_1 1/WIDTH [/EPS=*spacing_tolerance*]
end_layer_2 1/WIDTH — } at most 1 pair

or

active_layer [1]/AREA — at most 1 line

} when used, only one dimension specification is allowed

PINS = *pin_layer_1* [, *pin_layer_2* [..., *pin_layer_n*]]

}

You should read **Device Recognition**, beginning on page 105, to see how device recognition depends only on which shapes touch each other.

A device is defined as a polygon on *id_layer* touching the right number of objects on other specified layers. The objects can be counted one of two ways: by polygon or by node.

When you use the POLYGON keyword, every physically separate polygon on *layer_n* is counted as an object. The NODE keyword is used when you want to count separate polygons as the same object if they are electrically connected (according to the electrical connection rules used in the rule set). The NODE keyword can be useful when you want to exclude or identify devices with shorted terminals.

You can use multiple lines to define several different layer specifications. If you prefer to write the DEVICE rule all on one line, you must use semicolons (';') as delimiters between the layer specifications. When each layer specification is on a separate line, no semicolons are required. The order of the layer specification lines is unimportant. The order of the terminals is determined by the parameters after the required PINS keyword.

For an example of the use of the NODE keyword, see page 135.

In a single DEVICE rule, you can use several layer specification lines with the NODE or POLYGON keywords. In addition, you can add a layer specification using the AREA or WIDTH keywords to extract device dimensions. Only one dimension specification is allowed. Details on dimension specifications will be covered a little later.

You can use the NUMBER_OF-_PINS_FOR-_*device* LVS control file option to compare devices with different numbers of terminals in the two netlists.

The PINS keyword is **required**. You must specify at least one pin layer. Specify layers used in the layer specification lines. There is no upper limit for the number of pins for a device. Each *layer_n* can be mentioned up to *number_touching* times in the PINS statement. For example, if you have defined the device as touching 2 shapes on the SOURCE_DRAIN layer, you can mention SOURCE_DRAIN twice in the terminal list defined with the PINS keyword.

The nodes electrically connected to the polygons on the indicated layers will be listed as the terminals for the device in the layout netlist. If a layer is mentioned more than once in the PINS list, the order of the terminals connected to that layer is arbitrary.

You do not need to list every layer used in the device rule as a *pin_layer*. Nodes connected to polygons on layers not in the list after the PINS keyword will be ignored.

Example:

```
INPUT LAYER  {
        1 DIFF
        3 POLY
        10 WELL
}
SCRATCH LAYER   SRC_DRN;  GATE;

GATE =          DIFF     AND          POLY
SRC_DRN =       DIFF     AND NOT      POLY

DEVICE   NMOS   ID = GATE {
                POLY        1/POLYGON
                SRC_DRN     2/POLYGON
                WELL        1/POLYGON
                PINS =      SRC_DRN, POLY, SRC_DRN, WELL;
}
```

This set of rules will recognize typical NMOS devices. An NMOS device is defined as a polygon on the *id_layer* GATE that touches:

1 polygon on layer POLY

2 polygons on layer SRC_DRN

and

1 polygon on layer WELL

These NMOS devices will have 4 terminals. The terminals will be stored in the order indicated. It is arbitrary which polygon on SRC_DRN is used for terminal 1 and which is used for terminal 3.

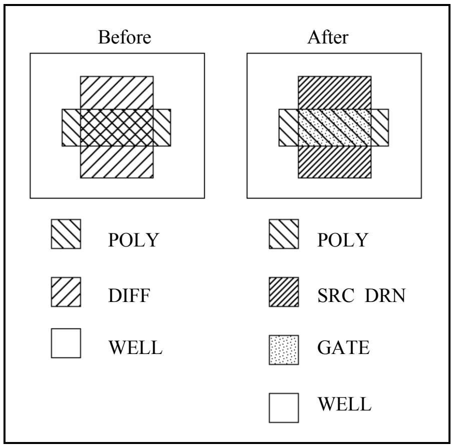


Figure 50: Device layers before and after layer processing.

See page 109 for more details on using the ERR keyword.

The optional [ERR = *error_layer*] parameter is used to store all shapes on *id_layer* which do not meet the device specifications. If *error_layer* is defined by an OUTPUT LAYER rule, rather than a SCRATCH LAYER rule, all shapes on *error_layer* will be reported as errors in the summary in the log file.

If we add the [ERR=*error_layer*] parameter to the device rule in the previous example, the first line would look like:

Example:

DEVICE PMOS ID=GATE, ERR=BAD_GATE {

See page 189 for details on using the ICED32™ editor to locate shapes on error layers.

In this case, the invalid device shown in Figure 51 will be flagged as an error. The shape on GATE will be copied to the BAD_GATE layer in the output command file. This is due to the fact that this GATE shape touches only one SRC_DRN shape.

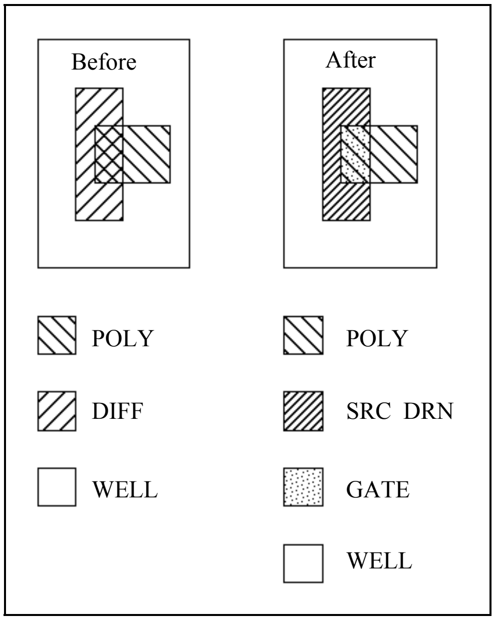


Figure 51: Invalid PMOS device before and after layer processing.

You can add a tolerance to relax these constraints. Read about the EPS keyword below.

When you want to recognize device width and length, you must use the WIDTH keyword. In order to calculate device width, the NLE must be able to recognize 2 end-sides of the polygon on *id_layer*. End-sides are defined as the sides of the *id_layer* polygon where it touches the edges of polygons on an *end_layer*. See page 110 for details on the restrictions added to the touching rule to allow end-sides to be recognized when the WIDTH keyword is used.

You can use the LOFFSET and WOFFSET keywords in the LVS device models to account for technology shrinking or bloating of device dimensions.

You can use the 2/WIDTH option on one layer specification to indicate that the shape on *id_layer* should share 2 end-sides with separate polygons on a single *end_layer*. You can instead define 2 different end layers by using the 1/WIDTH option on exactly two layer specification lines. (See an example on page 117.)

If you use the *end_layer* [2]/WIDTH syntax for one layer specification, the optional '2' before the slash is for readability only. Even when no number is provided, the rules compiler assumes that the 2 is present if only one WIDTH keyword is used in the DEVICE rule. In this case, the *id_layer* polygon must touch 2 polygons on *end_layer*.

Layer specifications using the WIDTH keyword can be combined with any number of layer specifications using the POLYGON or NODE keywords.

Example:

```
DEVICE NMOS ID=GATE, ERR=BAD_GATE {
        POLY        1/POLY
        SRC_DRN     2/WIDTH
        PWELL       1/POLY
        PINS=SRC_DRN, POLY, SRC_DRN, PWELL;
}
```

The *error_layer* must be defined as an output layer for shapes on that layer to be reported as errors in the log file.

The DEVICE rule above will recognize NMOS devices and extract their length and width. Polygons on layer GATE that do not form valid devices will be copied to layer BAD_GATE and will be reported as errors in the NLE log file. Note that the POLYGON keyword can be shortened to POLY.

The following example demonstrates a device with two different end_layers. If the source and drain of a NMOS device are on different layers, you must use the 1/WIDTH syntax on both of these layer lines to identify the two end-layers.

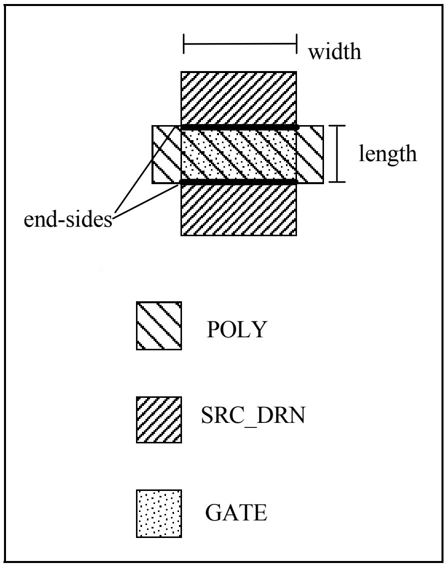


Figure 52: Width recognition of device.

Example:

```
DEVICE  SD_NMOS  ID=GATE {
    POLY      1/POLY
    SOURCE  1/WIDTH
    DRAIN    1/WIDTH
    WELL      1/POLY
    PINS=    SOURCE,  POLY,  DRAIN,  WELL
}
```

The next two examples demonstrate different methods of recognizing resistors. There are two common problems for recognizing resistors. The first problem is that resistors are often formed from conductive material. If the shape that forms the resistor is not removed from the conductive layer, the resistor shape will short the terminals.

The other problem is that the resistor shape usually extends around the contacts that form the terminals of the device. If the NLE cannot find end-sides of the resistor shape, it cannot extract a width.

Look at Figure 53. This resistor has a typical layout. A POLY_IN rectangle has a contact at each end. However, the NLE cannot extract a width for the POLY_IN rectangle because the CONTACTS shapes are covered by POLY_IN. There are no end-sides coincident with an end-layer.

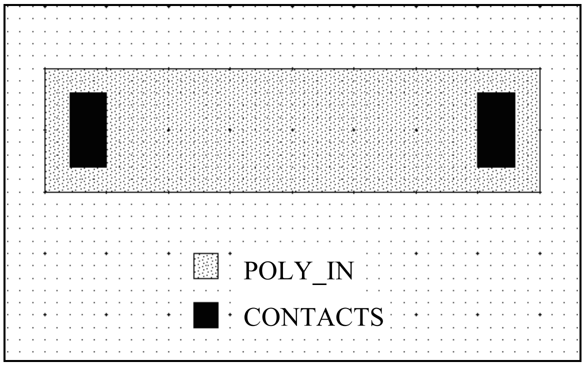


Figure 53: Cell RESCELL

The first method to recognize this resistor requires no extra shapes on non-design layers to allow end-sides to be formed. The IN_CELL rule will be used to separate layer POLY from the resistor layer. This method makes use of the fact that the contacts are exactly 2 units away from the edges of the POLY_IN layer.

Example:

```
INPUT LAYER         3 POLY_IN;  6 M1;  8 CONTACTS
OUTPUT LAYER        100 POLY;   102 POLY_RES;
SCRATCH LAYER       POLY_RES_TEMP;  CONT_BLOAT

CONT_BLOAT =        BLOAT (CONTACTS, 2.0)

POLY_RES_TEMP = POLY_IN   IN_CELL  RESCELL
POLY_RES =          POLY_RES_TEMP   AND   NOT   CONT_BLOAT
POLY  =             POLY_IN   AND   NOT   POLY_RES

CONNECT   POLY    M1   BY   CONTACTS

DEVICE  PRES  ID=POLY_RES
            POLY      2/WIDTH
            PINS =    POLY, POLY
}
```

When the above set of rules is run on a copy of RESCELL added to a design cell with metal wires, the layers used for the CONNECT and DEVICE rules will look like Figure 54.

You use the R_CONTACT and OHMS_PER-_SQUARE keywords in the layout netlist device models to allow the LVS to calculate resistance from the resistor dimensions.

Note that the rule set removes the resistor layer, POLY_RES, from the POLY layer. This prevents the resistor from forming a short between NETA and NETB.

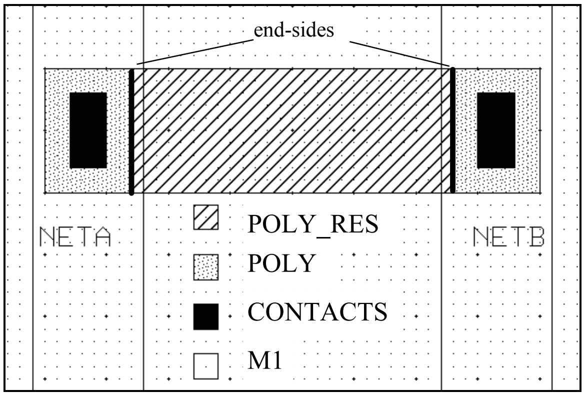


Figure 54: Cell RESCELL added to main cell after layer processing.

The shape on POLY_RES now has end-sides the NLE can recognize for device dimensions. Since this resistor is rectangular, the length of the resistor is the distance between the end-sides. (Note that you may want to correct the length for the LVS. See below.)

When the contacts are bloated by 2.0, they cover the ends of the POLY_IN rectangle exactly, leaving an exact rectangle of POLY_RES when they are used to etch the POLY_RES_TEMP layer.

See page 235 for complete details on how to use the LOFFSET keyword.

Since each end-side of the POLY_RES shape is now 2 units distant from the contacts, you must correct the resistor length in the LVS device model to get accurate device dimensions. The LOFFSET keyword is used to correct device length before the value of the resistor is calculated. The value supplied with the LOFFSET keyword is added to the length the NLE reports. Use a value of 4 to account for the fact that the NLE will measure the length 4 units too short (2 units on each end).

```
*.LAYMODEL   PRES   RES    LOFFSET=4.0
*+                  R_CONTACT=10   OHMS_PER_SQUARE=1000
```

The other method for resistor recognition avoids the length miscalculation, but you must add a shape on a dummy layer to RESCELL. Add this shape carefully, since it will affect device dimensions. An error on this layer may cause errors in your layout netlist.

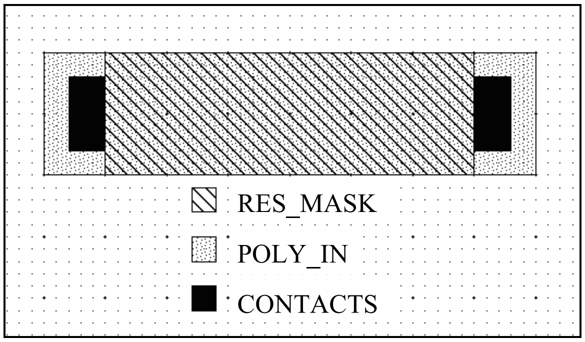


Figure 40: Cell RESCELL with shape on RES_MASK layer

The following rule set uses the shape on RES_MASK to separate the POLY layer from the POLY_RES layer.

Example:

```
INPUT LAYER         3 POLY_IN;  6 M1;  8 CONTACTS; 50 RES_MASK
OUTPUT LAYER        100 POLY;   102 POLY_RES;

POLY_RES =  POLY_IN AND          RES_MASK
POLY  =     POLY_IN AND NOT  RES_MASK
CONNECT     POLY   M1   BY  CONTACTS

DEVICE PRES ID=POLY_RES
    POLY   2/WIDTH
    PINS =  POLY, POLY
}
```

You should verify non-design layers that affect device recognition as carefully as design layers.

This rule set is much simpler, but since a shape on a non-design layer is used, errors on that layer will create false errors in the layout netlist. For example, if you stretch the design layers in the resistor, but forget to stretch the RES_MASK shape, the NLE dimensions of the modified resistor will remain unchanged, even though the true dimensions have changed.

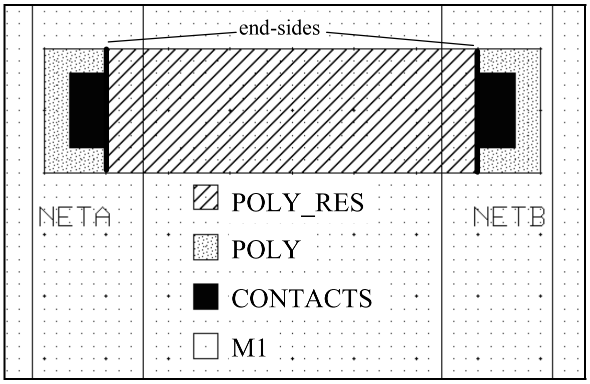


Figure 55: Modified RESCELL added to main cell after layer processing.

See page 100 for more information on removing area from conductive layers.

When you need to move the area that represents a device from a conductive layer to a device *id_layer*, you must be careful to avoid preventing shorts from being found. Picture a piece of POLY wire passing across a resistor in error. This would form a short between the resistor and the POLY wire. Either of the methods for resistor recognition we just covered would prevent the netlist from indicating the short, since the POLY wire would touch the POLY_RES layer, not the POLY layer. However, the error would still be caught. The POLY_RES shape would now touch the wrong number of shapes on POLY. The POLY_RES shape would not form a PRES device. If you do not use the ERR keyword in the DEVICE rule, no error message would be generated, but the device would be missing from the layout netlist and the LVS would fail to match the circuit. It may take some investigative work at this point to find the problem in the layout.

To insure that such obvious layout problems are not so difficult to find, you should always add the ERR keyword to your device rules. Always be very cautious when removing material from a conductive layer to insure that accidental shorts will be reported.

If you do not consider the length of the centerline to be the length of a device with bends, use the BENDS_CR keyword in the LVS device model.

The polygon on *id_layer* does not need to be rectangular to extract a length and width. No warning message will be reported for non-rectangular devices. (If the end-sides are not equal, the width will be calculated as the average end-side length instead of the length of one of two equal end-sides.)

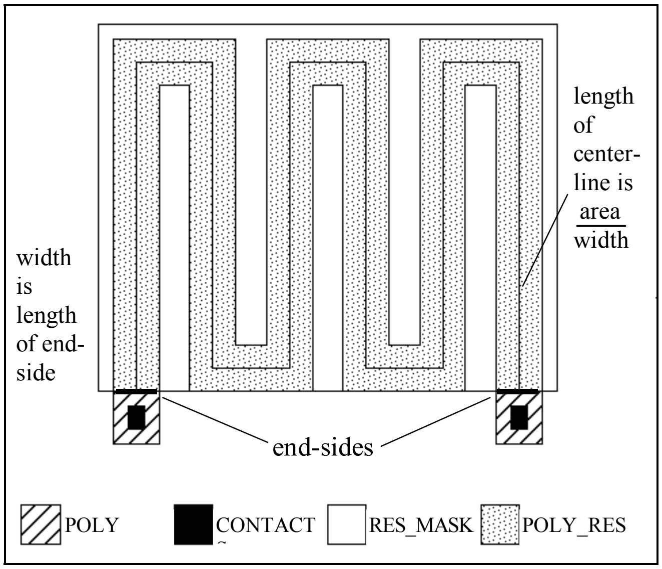

length of center-line is $\frac{\text{area}}{\text{width}}$
width is length of end-side
end-sides
POLY
CONTACT
RES_MASK
POLY_RES

Figure 56: Non-rectangular resistor.

The optional [/EPS = *spacing_tolerance*] parameter used with the WIDTH keyword allows you to relax the requirement that the entire end-sides of the polygon on *id_layer* must be touching sides of the polygons on an *end_layer*. The EPS keyword is intended to ease recognition of non-rectangular devices or rectangles that are not square with the axes (non-manhattan layouts). Shapes like these may fail to meet the exact touching rule due to errors introduced through floating point calculations or vertex approximations.

When the EPS keyword is used, the end-sides of the shape on *id_layer* must be within *spacing_tolerance* ICED32™ units of the shapes on an *end_layer*.

Example:

```
DEVICE TEST_WIDTH  ID=DEV, ERR=BAD_DEV {
        END 2/WIDTH, EPS= .6
        PINS= END, END;
}
```

This rule will allow devices with small spacing problems like those shown in Figure 57 to be recognized as valid devices. However, device number 1 will not be recognized because it does not meet the primary criteria for device recognition, the DEV shape does not touch the right number of shapes on layer END. The shape on layer DEV must touch two shapes on layer END to be a candidate for device recognition by the above rule.

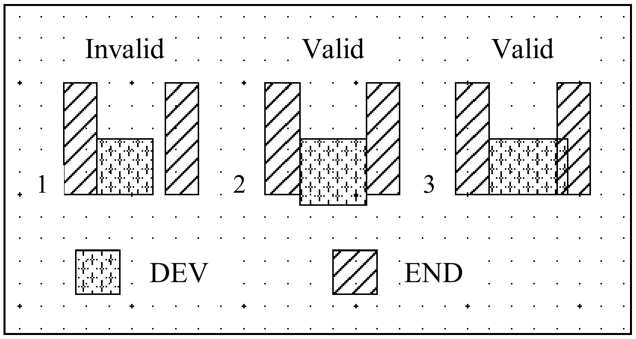


Figure 57

The AREA keyword is used to recognize the area and perimeter of devices. When you add the AREA keyword after the *id_layer* parameter, the area and perimeter of the device will be calculated from the polygon on *id_layer*. If you add the AREA keyword to a layer definition line, the polygon on that layer will be used to calculate the area and perimeter of the device.

```
DEVICE   LATERAL_BIPOLAR   ID=BASE {
         EMITTER        1/AREA
         COLLECTOR   1/POLYGON
         PINS =            COLLECTOR,  BASE,  EMITTER;
}
```

For this example, let us say that in your bipolar technology, the emitter shape in a lateral transistor does not touch the collector shape, however the base shape touches both. The area of the emitter should be recognized as the area of the device. You cannot use the emitter layer as the *id_layer* since it does not touch the collector layer. You must use the base layer as the *id_layer*, then use the AREA keyword on the emitter line to use the shape on that layer for the area of the device.

Specific Device Rules

TRANSISTOR *model_name* ID = *id_layer* [ERR = *error_layer*] {
    GATE = *gate_terminal_layer*
    S$D = *source/drain_layer* $\underbrace{\text{[/POLYGON] [/NODE]}}_{\text{must use exactly one keyword}}$
    BULK = *bulk_layer*
}

You should read Device Recognition, beginning on page 105, to understand how the NLE recognizes devices.

The TRANSISTOR rule is a shorter way to write a DEVICE rule for transistor recognition. A transistor is formed from a polygon on *id_layer* that touches:

1 polygon on *gate_terminal_layer*

2 polygons or nodes on *source/drain_layer*

1 polygon on *bulk_layer* (touch only not cover)

The definition of touch for the *gate_terminal_layer* and the *bulk_layer* is exactly the same as the TOUCHING rule. The 2 shapes on the *source/drain_layer* must meet additional criteria for width recognition. These criteria are covered on page 110. For the TRANSISTOR rule, the *source/drain_layer* is the only end layer and the shape on *id_layer* must touch exactly 2 shapes on that layer.

If your transistors cannot be described using these criteria, use the more powerful DEVICE rule to recognize them. See page 113.

You must perform the layer processing to create the *id_layer* for transistor recognition. You should not electrically connect the *id_layer* to any other layer. The only connections should be through the device.

Example:

```
INPUT LAYER {
        1 DIFF
        3 POLY
        10 WELL
}
SCRATCH LAYER   SRC_DRN;
SCRATCH LAYER   GATE;

GATE =      DIFF  AND       POLY
SRC_DRN = DIFF  AND NOT POLY
```

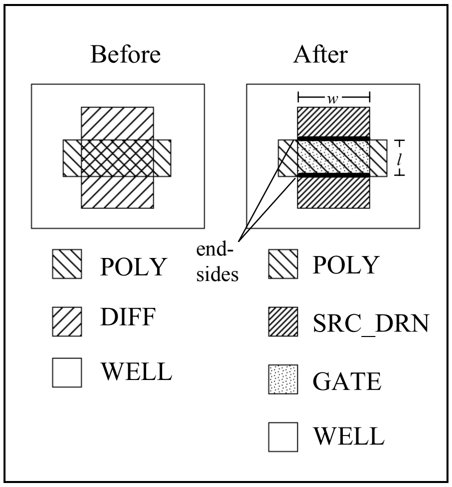


Figure 58: Layer processing before transistor recognition.

The transistor *id_layer* (GATE in this example) should be subtracted from the *source/drain_layer* (SRC_DRN). However, do not subtract the *id_layer* from the *gate_terminal_layer* (POLY). The *gate_terminal_layer* layer remains a conductor even where it crosses a diffusion shape.

Once the layer processing is performed to form the device *id_layer* and the *source/drain_layer*, the TRANSISTOR rule can be used.

Example:

```
TRANSISTOR   NMOS   ID = GATE {
        GATE =   POLY
        S$D =     SRC_DRN  /POLYGON
        BULK =   WELL
}
```

This TRANSISTOR rule is exactly equivalent to the following DEVICE rule:

```
DEVICE  NMOS  ID=GATE {
        POLY        1/POLYGON
        SRC_DRN   2/WIDTH
        WELL        1/POLYGON
        PINS =      SRC_DRN,   POLY,   SRC_DRN,  WELL;
}
```

The *id_layer* for the TRANSISTOR rule in the example above is the GATE layer. A shape on that layer which touches exactly one polygon on layer POLY, two polygons on layer SRC_DRN, and one polygon on layer WELL will be recognized as an NMOS device.

You can use the NUMBER_OF-_PINS_FOR-_*device* LVS control file option to compare 4 terminal transistors in the layout netlist to 3 terminal transistors in the schematic netlist.

You do not need to specify the PINS keyword in a TRANSISTOR rule. The NLE will automatically store the device with the terminals in the following order: *source/drain_layer* node 1, *gate_layer* node, *source/drain_layer* node 2, *bulk_layer* node. The order of the two source/drain terminals is arbitrary.

The device dimensions are extracted from the layout. To determine device width, the *id_layer* polygon must share an entire end-side with each of the two shapes on the *source/drain_layer.* The width will be the average length of the two end-sides of the polygon on GATE where it shares sides with the polygons on layer SRC_DRN. (See Figure 58 on the previous page.)

The restrictions on how the shapes must touch are the same as those required when using the WIDTH keyword of the DEVICE rule. See page 110 for examples and for details on how the dimensions are calculated from the layout. Unlike the DEVICE rule, the device dimensions will be calculated automatically despite use of either the POLYGON or NODE keywords.

The polygon on *id_layer* does not need to be rectangular. See page 112 for an example of a valid, non-rectangular transistor.

You **must** use either the NODE or the POLYGON keyword on the S$D line of the TRANSISTOR rule. The POLYGON keyword forces the NLE to recognize as transistors shapes on *id_layer* which touch exactly two polygons on the *source/drain_layer*. Even if the two polygons are shorted so that they represent the same node, the device will be recognized as a transistor.

The LVS device filters can be used to force transistors with shorted terminals to be ignored in the layout netlist.

The NODE keyword will prevent devices with a short between the source and drain from being recognized as transistors. However, not all shapes on *id_layer* which touch exactly two nodes on the *source/drain_layer* will be recognized. The restrictions enforced for width recognition still apply.

Example:

```
INPUT LAYER {
          1         DIFF
          2         POLY
          3         WELL
          4         WIRE
          5         CONT
}

SCRATCH LAYER {
          GATE
          SRC_DRN
}

GATE =        DIFF  AND          POLY;
SRC_DRN =     DIFF  AND NOT      POLY;

CONNECT    SRC_DRN    WIRE    BY
CONT;
```

For another example of the use of the NODE keyword, see page 135.

```
TRANSISTOR    TYPE1   ID = GATE {
   GATE=     POLY;
   S$D =     SRC_DRN  /NODE;
   BULK=     WELL;
}
```

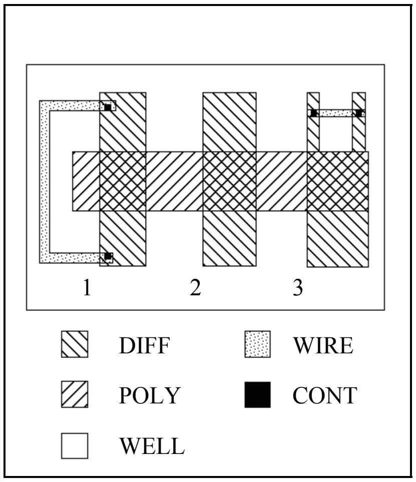


Figure 59: Potential transistors.

If the rule set above is run on the layout shown in Figure 59, only the device in the middle would be recognized as a TYPE1 transistor. Device 1 is not valid since the two shapes on the *source/drain_layer* are shorted and represent only a single node. Device 3 is not recognized even though it does touch 2 *source/drain_layer* nodes. This is due to the fact that the shape on the *id_layer* does not share exactly 2 end-sides with shapes on the *source/drain_layer.* The sharing of end-sides is required for width recognition.

If the POLYGON keyword was used on the S$D line in the TRANSISTOR rule above, device 1 in Figure 59 would be recognized even though the source and drain are shorted to form one node.

The optional [ERR = *error_layer*] parameter is used to store all shapes on *id_layer* which are not recognized as valid transistors. If you use an OUTPUT LAYER rule to define *error_layer*, the existence of any shapes on *error_layer* is reported in the NLE log file as an error. If you use a scratch layer for *error_layer,* usually no errors will be reported in the log.

Example:

```
TRANSISTOR  NMOS  ID=NGATE,  ERR=BAD_NGATE  {
            GATE =  POLY
            S$D =   SRC_DRN /POLYGON
            BULK =  PWELL
}
```

The NLE command line option LOGBAD will add to the log file the coordinates of each shape on *id_layer* that does not form a valid device. See page 175.

You should not assume that the TRANSISTOR rule is verifying all transistor design rules. For instance, the TRANSISTOR rule requires only that the *id_layer* shape touch a shape on the *bulk_layer*. Device 1 in Figure 44 on page 106, where the shape on *id_layer* is not covered by *bulk_layer,* is still a valid transistor. To insure that the *bulk_layer* covers all devices completely, you could restrict the *id_layer* to shapes that are covered by *bulk_layer*. However, this extra step will add to your NLE run time. You should verify this sort of design rule in a separate run, preferably with the DRC program available separately from IC Editors, Inc.

CAPACITOR *model_name* ID = *id_layer* [ERR = *error_layer*] {
PLUS = *plus_layer*
MINUS = *minus_layer*
}

The CAPACITOR rule will recognize polygons on *id_layer* as devices when the *id_layer* polygon touches exactly 1 polygon on *plus_layer* and 1 polygon on *minus_layer*. This rule is simply a shorter way of writing a DEVICE rule.

Example:

```
CAPACITOR   MY_CAP  ID=CAP {
          PLUS = M1
          MINUS = M2
}
```

is equivalent to:

```
DEVICE          MY_CAP  ID=CAP/AREA {
          M1  1/POLYGON
          M2  1/POLYGON
          PINS=M1, M2;
}
```

If your capacitors cannot be described using this simplified rule, use the more powerful DEVICE rule to recognize them. (See page 113.) However, you should still read the rest of this description for some important capacitor examples.

Capacitors present special problems for device recognition. The layout combinations that produce capacitors often occur in other places in the design where you do not want them to be considered capacitors. Sometimes the shapes on *id_layer* are used to create transistors. You want a transistor recognized as a capacitor only when the source and drain of the transistor are shorted. We will cover this scenario a little later. The other problem is when the same combination of shapes can represent either a capacitor, or no electrical relationship at all.

For example, you design capacitors by using large rectangles of M1 and M2 on top of each other. You want the large overlap of these layers shown in Figure 60 to be recognized as a capacitor, but you want the chance overlap of the two wires in the upper right to be ignored.

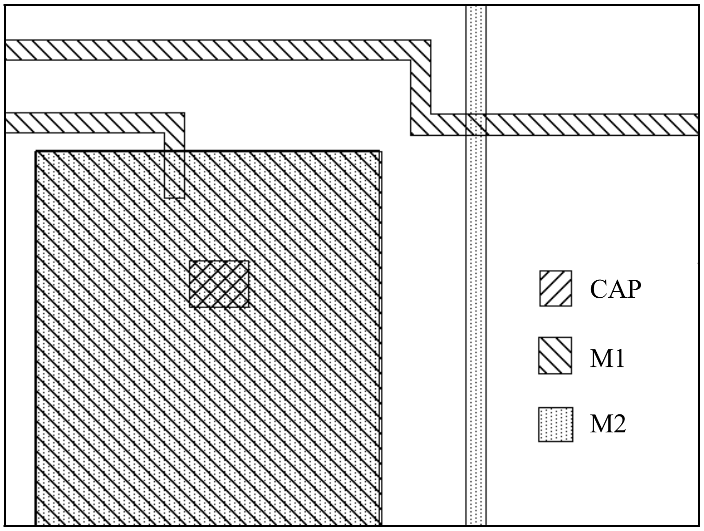


Figure 60: Genuine capacitor and mere crossing of wires.

One way to distinguish the large overlap which represents the capacitor from the small overlap of wires is to classify the overlap by size using the BOUNDS rule.

Example:

```
INPUT LAYER        2 M1; 3 M2;
SCRATCH LAYER      MAY_BE_CAP; CAP_DEV;

MAY_BE_CAP =    M1  AND M2
CAP_DEV =       NOT  BOUNDS ( MAY_BE_CAP,  (0:5, 0:5) )

CAPACITOR  MY_CAP  ID=CAP_DEV {
           PLUS =      M1
           MINUS =     M2
}
```

The rule set above will not copy the overlap of wires from the MAY_BE_CAP layer to the CAP_DEV layer. However the larger common area of M1 and M2 which represents the capacitor is copied to the CAP_DEV layer. The CAP_DEV

layer is then used as the *id_layer* for the CAPACITOR rule. No shapes on non-design layers (often called dummy layers) are required.
Another common method of capacitor recognition is to use a shape on a dummy layer to identify the genuine capacitor from the mere crossing of wires. In Figure 60, the dummy shape is on layer CAP. You would use the CAP layer as the *id_layer* in the CAPACITOR rule. The following simple rule set below will then recognize the large overlap as a capacitor and ignore the crossing of wires entirely.

Example:

```
INPUT LAYER         1 CAP;  2 M1;  3 M2;

CAPACITOR  MY_CAP  ID=CAP {
           PLUS =     M1
           MINUS =    M2
}
```

You use the C_AREA and C_PERIMETER keywords in the device model to allow the LVS to calculate the value from the area and perimeter.

However, this rule set has a problem. The area and perimeter of the shape on *id_layer* are automatically passed to the LVS to calculate the value of the device.

When the shape on *id_layer* is only a dummy shape used to identify device candidates, invalid dimensions may be passed to the LVS. You can use extreme caution when adding the *id_layer* shapes to your design so that the shapes on *id_layer* are always exactly the size of your device, or you can use clever layer processing to insure that valid dimensions are passed.

Example:

```
INPUT LAYER          1 CAP_ID;   2 M1;   3 M2;
SCRATCH LAYER        MAY_BE_CAP;   CAP_DEV;

MAY_BE_CAP =         M1   AND   M2
CAP_DEV =            MAY_BE_CAP   TOUCHING   CAP_ID

CAPACITOR            MY_CAP    ID=CAP_DEV {
        PLUS =     M1
        MINUS =    M2
}
```

Using the above example will provide you with accurate capacitor dimensions, but the extra processing of the MAY_BE_CAP layer will add to your run time. If you try to save time by creating the CAP_DEV layer with the following line:

Example: **CAP_DEV = M1 TOUCHING CAP_ID**

you may get inaccurate dimensions. This is due to the fact that the entire M1 shape will be used as the *id_layer*, instead of the overlap of M1 and M2. If your layout looked like Figure 60, the area and perimeter of the M1 wire extending from the capacitor in the upper left will be included in the value of the capacitor.

If you prefer to use the area of the *plus_layer* or *minus_layer* as the area of the capacitor without extra layer processing, you could also use the general DEVICE rule instead of the CAPACITOR rule and specify the AREA keyword on an appropriate line of the rule. The following rule requires no additional layer processing, however since the area of any M1 attached to the device will be included in the device calculation, it has the same problem as the previous example.

Example:

```
DEVICE  MY_CAP  ID=CAP_ID {
        M1  1/AREA
        M2  1/POLYGON
        PINS=M1, M2;
}
```

However, if you lay out your capacitors so that the terminals attach only through a contact or via layer, wires on the same layer will not affect the area of the M1 capacitor shape. When this is the case, the above example is an efficient method of recognizing capacitors.

There is often a trade off in how careful you must be in writing your NLE rule set verses how careful you must be in your layout. However, if you minimize NLE run time with careful layout restrictions, you may still run into trouble.

The TOUCHING rule on the previous page:

CAP_DEV = MAY_BE_CAP TOUCHING CAP_ID

is somewhat more expensive in NLE runtime than the simple Boolean rule

CAP_DEV = MAY_BE_CAP AND CAP_ID

but the effects are very different. The TOUCHING rule makes the shape or size of the dummy shape on the CAP_ID layer unimportant. If the AND rule is used instead, the size and placement of the CAP_ID shape is critical, and mistakes on this layer may cause problems that are never caught. Let say that you carefully add the CAP_ID shape to be the same size as the M2 rectangle, but months later, you edit the capacitor while the CAP_ID layer is blanked (i.e. not displayed). Now the CAP_ID shape overlaps the capacitor only slightly. The NLE will not generate any error messages. However, the area of the device passed to the LVS will now be inaccurate.

One method which will insure accuracy, but which allows you to write an efficient NLE rule set, is to create a separate NLE or DRC rule set which verifies the accuracy of the non-design shapes you add to your layout. Run this rule set less often, but be sure to run it on your final design.

For the example above, you could write the following rules in a separate rule set:

OUTPUT ERROR LAYER CAP_ID_ERR1; CAP_ID_ERR2

CAP_ID_ERR1 = CAP_ID AND NOT M1
CAP_ID_ERR2 = CAP_ID AND NOT M2

The NLE command line option LOGBAD will add to the log file the coordinates of each shape on *id_layer* that does not form a valid device. See page 175.

The optional [ERR = *error_layer*] parameter can be used to collect all shapes on *id_layer* which do not meet the criteria that the shape on *id_layer* touches exactly one shape on *plus_layer* and one shape on *minus_layer*. If *error_layer* is defined with an OUTPUT LAYER rule, the existence of shapes on *error_layer* will be reported as an error in the NLE log file. If *error_layer* is a scratch layer, errors will usually not be reported.

When your capacitors are really shorted transistors, you can recognize them by eliminating the transistors with a TRANSISTOR rule using the NODE keyword then using the *err_layer* from that rule to recognize capacitors.

Example:

```
INPUT LAYER         3 POLY;  1 DIFF;   4 WELL;
SCRATCH LAYER       SRC_DRN;  GATE; NMOS_CAP;

GATE =          DIFF     AND          POLY
SRC_DRN =       DIFF     AND NOT      POLY

TRANSISTOR   NMOS   ID=GATE,   ERR=NMOS_CAP {
          GATE =          POLY
          S$D =           SRC_DRN  /NODE
          BULK=           WELL
}

DEVICE   MY_CAP  ID=NMOS_CAP/AREA {
          POLY            1 /POLYGON
          SRC_DRN         1 /NODE
          PINS = POLY, SRC_DRN:
}
```

If you do want shapes created on a scratch error layer to be reported in the log, add the LOGBAD keyword to the NLE command line.

When the source and drain of NMOS devices are shorted, there will be only one SRC_DRN node touching the GATE, despite the fact that there may be two SRC_DRN polygons touching GATE. When the NODE keyword is used on the SRC_DRN line of the TRANSISTOR rule, shapes on layer GATE which have their sources and drains shorted will be copied to layer NMOS_CAP rather than forming valid NMOS devices. Since NMOS_CAP is defined with the SCRATCH LAYER rule, no errors will be reported in the log.

The DEVICE rule then uses these leftover shapes from the GATE layer as the *id_layer* for capacitor recognition. Since the valid capacitors have only one SCR_DRN node, they will be recognized by the MY_CAP rule that specifies 1 SRC_DRN node.

If you used a CAPACITOR rule rather than the DEVICE rule with the NODE keyword, devices that have the source and drain shorted with electrical connections through other layers would not be found. Remember that the CAPACITOR rule assumes the POLYGON keyword after the *plus_layer* and *minus_layer* parameters. Therefore, the CAPACITOR rule can only recognize devices where a single *minus_layer* polygon touches the *id_layer*. It will not recognize devices where two separate polygons on the *minus_layer* touch the polygon on the *id_layer* even though they may be the same electrical node.

Node Labels

There are three types of node labels recognized by the NLE.

You can also use text components in the layout to override the measured dimensions of a device. See page 346.

Polygon ID Labels: These are used to label individual polygons. These labels can be useful in diagnosing shorts and opens when they are reported in ECC error messages.

LVS Node Labels: The extracted LVS netlist will contain a list of these labels and the associated node numbers. Many LVS functions use these node labels as the names of the nodes. These labels may also contain special characters that control how the node is treated by LVS circuit transformations.

The DISJOINT rule can be used to allow virtual connections of separate nodes for ECC checking.

ECC Node Labels: An LVS label on a layer specified in a LABEL rule will also be used as an ECC label. These labels are used by the ECC to determine the node names of electrically connected nets. Two different labels on a single net will result in an error message about the short. Two separate nets with the same ECC node name will result in an error message about the open.

For a text component to be used as a label, it must either be created on a design layer, or explicitly moved to a design layer in the NLE rule set. The ATTACH TEXT rule is usually used to move text components to design layers. We will cover this rule later.

The TEXT keyword of the INPUT LAYER rule can prevent the LVS from using labels on a design layer.

Text components on a design layer are used as LVS labels by default, unless they are identified as polygon id labels using a special prefix specified by the COMMENT rule.

You will usually want to use the same text components as LVS labels and ECC labels. This is easily accomplished. The LABEL rule will force the NLE to use LVS labels on certain layers as both ECC labels and LVS labels.

You can label devices and/or nets. If an LVS label is on a device id layer, it will be used to label the device in the layout netlist. If a label is on a component that is part of an electrically connected net, it will be used to label the entire net.

See the ADD TEXT command in the ICED32™ Layout Editor Reference Manual for a list of justification codes.

The origin of the text component must be covered by the shape it is meant to label. The origin of the text component may be in any of nine locations on the text component, depending on the value of the justification code used to create it. The easiest way to determine the origin of an existing component is to select it. The origin will be indicated by a diamond shaped select mark. (See the middle text component in Figure 61.) The origin of the component must be within the boundary of the component. The origin of the top label in Figure 61 (justification code LB) is below the wire and will not label it. The origin of the bottom label (justification code RC) is to the right of the wire and will not label it.

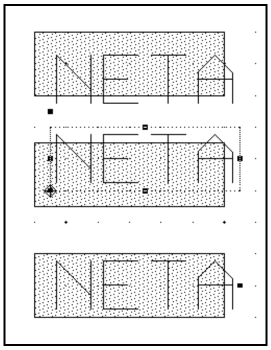


Figure 61: Only the middle label is placed correctly.

The ECC will warn you when labels are present in the layout which do not label components.

Labels are not hierarchical. **All labels should be the top-level cell**. Labels in nested cells are ignored unless you identify them as global labels. To identify a text component on a net as a global label, you must add a colon (':') as a suffix to the end of the text component. The suffix will not be stripped from the net name. The net name "VDD:" is not the same as "VDD". You cannot use global labels to label devices, only nets.

Global labels with a single colon suffix must be enabled in the LVS through the use of the RECOGNIZE_-GLOBAL_-TEXT_IN_-SUBCELLS = YES control file option.

To see why labels in nested cells should be ignored, look at the subcircuit in Figure 62. Note that two nodes are labeled "IN" and "OUT". (All of these labels use justification code CC so that the origins are in the center of the text components.) If you use this circuit 100 times in your design, you would get a long list of false error messages about shorts and

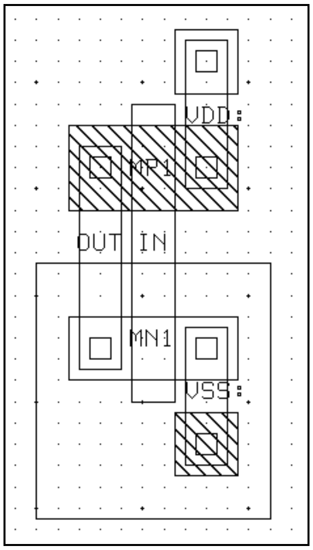


Figure 62: Subcircuit with labels

opens since 100 separate nets would be labeled "IN", and 100 separate nets are labeled "OUT".

Note that the power and ground labels in Figure 62, "VSS:" and "VDD:", use the global net suffix. These labels should be used since all of the net fragments labeled "VDD:" should be shorted in the higher level circuit. This will label the same net 100 times with the same label, which is acceptable. In fact, having so many labels on the net will help isolate shorts if they are present.

One of the important restrictions on labels is difficult to remember. All labels must be 15 or less characters. Labels created from text components which are longer than 15 characters will simply be ignored. No warning messages are produced. Let's put this in capital letters so you do not forget.

ALL LABELS MUST BE 15 OR FEWER CHARACTERS

To create text components with lower case characters in the ICED32™ layout editor, you must enable lower case text using the TEXT command.

By default, the ECC will translate all labels to upper case. To force the ECC to preserve lower case characters in labels, you must use the USE_CASE rule. This rule is described on page 143.

To preserve the case of lower case characters in labels for the LVS, you can use the LVS control file option "FORCE_ALL_LAYOUT_LABELS_TO_UPPER-_CASE = NO". If you want all labels in the layout netlist translated to upper case as the LVS reads the file, set this control file option to YES.

Rules Which Affect Labels

ATTACH TEXT *text_layer design_layer_1* [*design_layer_2* [*... design_layer_n*]]

Scratch layers are defined by the SCRATCH LAYER rule and output layers are defined by the OUTPUT LAYER rule.

This rule is used to move text components on *text_layer* to design layers.

All layers used with this rule must be scratch or output layers, since the layers are modified by this rule and you are not allowed to modify input layers. Text components will be removed from *text_layer* and added to the design layers. See page 147 for an example of how to use this rule when you need to modify input layers.

Only text components that have origins over a shape on one of the design layers are moved. All non-text components on *text_layer* will remain on *text_layer*. All text components on *text_layer* that are not over shapes on one of the specified design layers will also remain on *text_layer*. Ordinarily, you will not be warned that components remain on *text_layer*. However, if *text_layer* is an error layer, the log will report shapes left on the layer as errors. Alternatively, you can add a LABEL rule (see next rule) for *text_layer*. This will result in a detailed error message for each text component left behind on the *text_layer*.

Example: **ATTACH TEXT MY_TEXT_LAYER M1 M2 POLY**

The Boolean rules (OR, AND, etc.) and the TEXT keyword of the INPUT LAYER rule can also be used to move text components to design layers.

When this example of the ATTACH TEXT rule is used, all text components on layer MY_TEXT_LAYER which have an origin over a shape on M1 will be moved to the M1 layer. Any remaining text components on MY_TEXT-_LAYER that have an origin over a component on the M2 layer are moved to that layer. If there are any text components left on layer MY_TEXT_LAYER and they have origins over a shape on POLY, they will be moved to layer POLY. All remaining text components on TEXT_LAYER will remain on that layer. Also, all non-text components will remain on MY_TEXT_LAYER.

Only a single design layer is required. You can use multiple ATTACH TEXT rules in a rule set.

Note that the order of design layers in the ATTACH TEXT rule is significant. In the previous example, if a text component on TEXT_LAYER was placed so that its origin was over both a shape on M1 and a shape on layer POLY, it would be removed from TEXT_LAYER and moved to layer M1 before the POLY layer is processed. The text component will not label the shape on layer POLY.

[MUST] **LABEL** *layer_1* [*layer_2* [... *layer_n*]]

Labels on design layers are always passed to the LVS regardless of this rule.

The LABEL rule is used to identify layers that may contain ECC node labels. This is the only way to label nodes for ECC shorts and opens checking. You can have multiple LABEL rules, or specify all layers in the same rule.

Example:

LABEL M1 M2 POLY
LABEL SRC_DRN

For these labels to be used, the NOECC rule must **not** be used in your rule set.

The ATTACH TEXT rule is the easiest way to move text components to the layer they are meant to label.

The rules above will result in the NLE looking for labels on the M1, M2, POLY, and SRC_DRN layers to label nodes for ECC checking. All labels on other layers will be ignored by the ECC.

The text components should be on the design layers they are meant to label. If you have created your text components on a different text layer, you must move or copy them to the electrically connected design layers for the LABEL rule to label the nodes. The text components must have origins over shapes they are meant to label.

Any label that is not covered by a shape on the same layer will be mentioned in the NLE log in an "Unattached text" message.

The location of the LABEL rule in the rule deck is not important. All LABEL rules are executed at the end of the NLE run when the ECC checks are performed.

The optional MUST keyword is used to require that every polygon on the layer has a node label. This test is performed with the ECC checks.

USE_CASE

This rule is used to preserve the case of ECC node labels. By default, the ECC translates all node labels to upper case before verifying shorts and opens. If you want a net labeled "Abc" to be considered a different net than "ABC", you must add the USE_CASE rule to your rule set. The location of the rule in the rule set is unimportant.

The use of this rule also allows you to enter lower case or mixed case text in the rules file for the DISJOINT and UNCONNECTED rules. Otherwise, strings in these rules will be translated to upper case.

The use of this rule does not affect how text that creates LVS node labels is stored in the output binary layout netlist. Node labels are always stored as typed in the layout netlist. You can use the LVS control file option FORCE_ALL_LAYOUT_LABELS_TO_UPPER_CASE = YES if you want all labels in the layout netlist translated to upper case as the LVS reads the file.

COMMENT [=] *comment_string*

Use this rule to allow polygon id labels to be added on design layers rather than on a separate layer specified with the ID keyword in an INPUT LAYER rule. (See page 56.) Any label that uses comment_string as a prefix will be considered a polygon id label and not an LVS or ECC label.

If you use this rule in your rule set, use it only once.

This differentiation of labels through use of a prefix is not DRACULA[5] compatible.

[5] DRACULA is a registered trademark of Cadence, Inc.

SAVE *layer_1* [*layer_2* [... *layer_n*]]

and

NO_SAVE *layer_1* [*layer_2* [... *layer_n*]]

The node outliner commands are described in detail on page 389.

These rules are not truly label rules. They affect the generation of the node outliner file. This file is automatically generated by the NLE. The data in the file is used by the ICED32™ layout editor to generate shapes on a temporary layer that outline specific devices or entire electrically connected nets using the node outliner commands.

By default, all layers used in connection rules or in device rules will be copied to the database saved in the file. When the node outliner commands are used to generate the outline of an entire net, copies are made of every shape on the net including contacts, vias, etc.

For large chips, the size of this file may be excessive. If you want to restrict the layers that are saved in the node outliner file, you can use either of these rules.

The SAVE rule will restrict the layers saved to those layers mentioned in the rule. Be sure to include all wire layers and all device layers, or the node outlines will be broken and difficult to follow.

Example: **SAVE M1 M2 POLY GATE SRCDRN**

The NO_SAVE rule will save all layers the NLE would save by default, except for those listed in the rule. This is usually the better method to restrict layers in the node outliner file. Include layers that would not aid you in tracing a net or finding a device, such as contact and via layers.

Example: **NOSAVE CONTACTS VIA1 VIA2**

Uses of the Three Types of Labels

	Polygon ID Labels	**LVS Node Labels**	**ECC Node Labels**
Propagate through electrical connections?	No	Yes	Yes
Used by ECC?	To label individual polygons in error messages	No	To identify separate nodes for shorts and opens checking
Used by LVS?	No	To label nodes in extracted netlist	Yes, since an ECC label is always used as an LVS label.
How do I move text to a design layer and then use it as a label, when text is on a different layer than design layer?	INPUT LAYER ID keyword or ATTACH TEXT rule **and** COMMENT prefix	INPUT LAYER TEXT keyword or ATTACH TEXT rule	INPUT LAYER TEXT keyword **and** LABEL rule or ATTACH TEXT rule **and** LABEL rule
When will text components in a top-level cell on a design layer be used as labels?	Only if COMMENT prefix is used	Always, as long as the TEXT keyword is **not** used in the INPUT LAYER rule, and the COMMENT prefix is **not** used	Only when the layer is specified with the LABEL rule and the COMMENT prefix is **not** used.

The comment prefix is set with the COMMENT rule.

Any text component on a design layer that uses the COMMENT prefix will be considered a polygon id label rather than an LVS or ECC label. These labels are used to identify individual polygons rather than entire nets. These labels will be used in ECC error messages about shorts and opens in your circuit. They can help you to locate the problem by identifying which shapes are included in a net and which are not.

Example:

```
COMMENT =   @
LABEL       M1  M2
CONNECT     M1  AND M2  BY  VIA
```

The ECC checks are covered in more detail beginning on page 152.

If the rules above are used to generate the electrical connections for a circuit with a short between two nets labeled "NETA" and "NETB", the NLE ECC checks will produce error messages similar to Figure 63. If several shapes on each net have polygon id labels with a prefix of '@', these labels will not name the net, but they can be used to help isolate the short. From looking at the report, we can see that NETA shorts to NETB through a shape with the polygon id label @NETB_12. You can follow the path of the short from the coordinates listed, even if the circuit contains no polygon id labels, but it might be harder to locate the error.

```
Path from NETB to NETA----
Step 1-----------------------------------------------
   NETB at (46.5, -289), Cell NLETEST
Step 2-----------------------------------------------
   Box on layer M2[3], Cell NLETEST, x=46 <=>52,
                                 y=-287.5 <=> -289.5
Step 3-----------------------------------------------
   Box on layer M1[2], Cell NLETEST, x=34.5 <=>48.5,
                                 y=-287 <=> -290
Marked with--
   @NETB_12 at (41, -288.5), Cell NLETEST
Step 4-----------------------------------------------
   Box on layer M2[3], Cell NLETEST, x=37.5 <=>40.5,
                                 y=-280.5 <=> -291
Marked with--
   @NETA_2 at (38.5, -282.5), Cell NLETEST
Step 5-----------------------------------------------
   Box on layer M1[2], Cell NLETEST, x=38 <=>48,
                                 y=-283.5 <=> -286.5
Marked with--
   @NETA_1 at (41, -285.5), Cell NLETEST
Step 6-----------------------------------------------
   Box on layer M2[3], Cell NLETEST, x=46 <=>52,
                                 y=-284 <=> -286
Step 7-----------------------------------------------
   NETA at (47, -285.5), Cell NLETEST
```

Figure 63: Fragment of NLE log file with ECC error messages.

See page 389 for details on using the node outliner commands.

Any label without the comment prefix on a design layer that is over a shape on a net will label the entire net. These labels are stored in the extracted netlist passed to the LVS and also in the .P8K node outliner file generated by the LVS. The node outliner file can be used with the node outliner commands in the ICED32™ layout editor to highlight entire nets in the layout.

LVS labels are included in the netlist passed to the LVS only when they attach to devices in the netlist. For example, if a piece of metal is labeled correctly, but it does not connect to any devices, the labeled node will not be present in the netlist.

See Advanced Uses of Node Labels for more information.

The LVS uses these labels in many ways. You can use these labels to define forced points of correspondence between the two circuits. By adding special characters to the labels, you can prevent nets from disappearing due to device collapses. If you create a schematic netlist from the layout netlist, these labels can be used as the names of the nodes.

CONNECT rules define electrical connections.

If the labels are not created on design layers used in device rules or CONNECT rules, you must move them to the design layers. You should move the text components to the appropriate layers before the CONNECT rules.

Example:

```
INPUT LAYER    1 TEXT_IN;  2 M1_IN;  3 M2_IN;  4 VIA
SCRATCH LAYER  TEXT;  M1;  M2;

TEXT =         TEXT_IN
M1 =           M1_IN
M2 =           M2_IN
ATTACH TEXT    TEXT   M1   M2
LABEL          M1  M2
CONNECT        M1  M2  BY VIA
```

This set of rules shows how text components on layer TEXT can be used as labels for both the M1 and M2 design layers. The ATTACH TEXT rule moves the text components onto the design layers. Since the ATTACH TEXT rule modifies the TEXT, M1, and M2 layers, they cannot be input layers. This is why the input layers are copied to scratch layers before the ATTACH TEXT rule. Since no modifications can be made to a layer after it is used in a CONNECT rule, the ATTACH TEXT rule comes before the CONNECT rule.

The LABEL rule will allow these labels to be used as ECC labels. The labels will be used as LVS labels even if the LABEL rule is not present.

Once the nets are labeled with ECC labels, the ECC will list all nets with conflicting labels as shorts and all separate nets with the same label as opens in the NLE log file.

Placement and Processing of Labels

For text components to be used as labels, they must be on a design layer (or moved to a design layer in the rule set) and the origin of the text component must be over the shape it is meant to label. If the COMMENT rule is used in your rule set, labels to be used as LVS or ECC labels must not use the COMMENT prefix.

Text components can be moved to design layers with Boolean operations, the ATTACH TEXT rule, or the TEXT keyword of the INPUT LAYER rule. **If you use the TEXT keyword on the INPUT LAYER rule, text components on the design layer will be ignored.**

Boolean operations on a layer with text components will treat the text components in exactly the same manner as other components. If you perform a Boolean OR on a layer with only text components and a layer with design components, it has the effect of adding the labels to the design layer.

Example:

```
INPUT  LAYER        {
          1         M1_IN;
          101       M1_TEXT;
}
SCRATCH LAYER    M1;

M1 = M1_IN  OR  M1_TEXT
```

This example will combine the text components (and all other components) on layer M1_TEXT and all components on layer M1_IN into the new layer M1. All text components on layer M1_IN **will** remain on that layer and will be used for node labels.

When you use the TEXT keyword of the INPUT LAYER rule or the ATTACH TEXT rule, only text components are moved onto the design layer. When you use the TEXT keyword of the INPUT LAYER rule the text components on the design layer will be ignored.

Example: **INPUT LAYER 1 M1_IN TEXT 101;**

When this INPUT LAYER rule is used, all text components on layer 101 of the ICED32™ cell will be moved to the M1_IN layer. All other components on layer 101 will be ignored. All text components on the original design layer 1 will be ignored. You can use this feature to temporarily prevent all labels on a design layer from being used.

When labeling devices, you must be careful about how the layer with the text component is handled. You can easily label other shapes besides the device. Look at Figure 64. The horizontal polygon with the dashed lines is on layer POLY and other polygon is on DIFF. These layers are used to create the SRC_DRN and DEV layers as follows:

```
DEV =        DIFF AND       POLY
SRC_DRN =    DIFF AND NOT   POLY
```

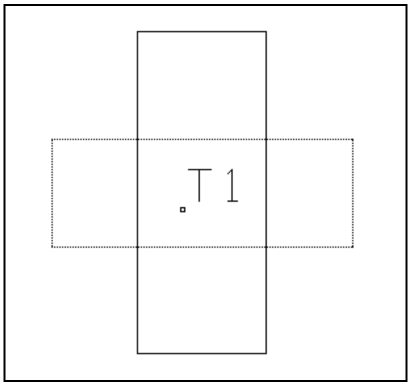


Figure 64: Using text component to label device. (The small square represents the origin of the text component.)

If the text component is on layer POLY, it will label both the shape on POLY and the shape on DEV. This means that the net that includes the POLY shape will also be labeled with the name "T1". You usually want to avoid labeling both the device and the net with the same name.

If the text is on layer DIFF, it will label the shape on DIFF (which will probably not be used in further CONNECT or DEVICE rules) and the shape on DEV. It will not label the shapes on SRC_DRN. In this case, the text component will label only the device.

A good way to label only devices, and avoid labeling nets by accident, is to add the text components on a unique layer, DEV_TEXT for example, then use the ATTACH TEXT rule to move the labels to the DEV layer.

Example: **ATTACH TEXT DEV_TEXT DEV**

ECC Rules

The NLE can optionally perform a series of tests on your design, including detection of shorts and opens. This series of tests is referred to as the ECC (Electrical Connection Checks) feature.

To disable the ECC checks you can add the NO_ECC rule (see below) to your rule set. (Or you can use the NO_ECC option on the NLE command line.)

Before performing ECC checks you should learn about node labels and the LABEL rule. See **Node Labels** on page 138.

To enable these tests, you must have node labels in your layout and you must include a LABEL rule in your rule set for each design layer with labels. The ECC uses these node labels, and the electrical connections defined by the CONNECT rule, to detect shorts and opens. If two conflicting labels exist on a net, the ECC will generate an error message about the short. If two electrically separate nodes are labeled with the same name, the ECC will issue an error message about the open.

Shapes will be created in the command file created by the NLE to aid you in locating errors like shorts and opens. See the PATH_LAYER and ERROR_LAYER rules on the following pages.

In addition to the tests for opens and shorts, the ECC also performs the following tests:

Verification that certain polygons or simple constructions are not electrically connected to any other node. (See the UNCONNECTED and PADSIZE rules below.)

Verification that all text components on layers in a LABEL rule have origins covered by a shape on the same layer.

A few rules that affect the ECC tests are covered elsewhere in this manual. These rules are covered in **Node Labels** beginning on page 138. The COMMENT rule defines a prefix that is one way to define polygon id labels. The USE_CASE rule prevents the ECC from translating all labels into upper case. The LABEL rule is used to tell the ECC which layers contain node labels to be used by the ECC tests.

Only node labels on layers defined with a LABEL rule will be used by the ECC.

See an example of the use of polygon id labels on page 146.

You can label shapes with polygon id labels. These labels will be reported in any ECC messages involving those shapes. No polygon id labels are required to perform the ECC tests. Polygon id labels on shapes in the path of a short or open will be used even when they are on layers not mentioned in a LABEL rule.

NO_ECC

Using the NO_ECC keyword on the NLE command line will have the same effect as the use of this rule in the rule set.

Use this rule to prevent the ECC checks from being performed on your design. This will save run time when you execute the NLE repeatedly after small changes to your design. However, if one of your changes causes a short or open you will not be warned of this until the LVS reports problems.

Problems like shorts can be difficult to diagnose with the LVS, however finding shorts from ECC messages is relatively easy. We do not recommend using this rule to turn off ECC checking unless you really need to reduce the time it takes to run the NLE.

PATH_LAYER [=] *iced_layer_number*

See page 189 to learn about how to add the shapes output by the NLE to your ICED32™ cell.

When the ECC finds two conflicting node labels on a net, it will outline all shapes on the path between the two labels on an error layer. These shapes will be created in your ICED32™ cell when you execute the command file output by the NLE. **By default, the layer used by the ECC is layer number 98.** This rule allows you to override this default and define the number of the error layer explicitly.

Example: **PATH_LAYER 198**

When this rule is used, the ECC will create shapes on the path of shorts on layer 198 rather than layer 98.

ERROR_LAYER [=] *iced_layer_number*

Several ECC checks will create shapes to aid you in locating errors found in your layout. These shapes will be created in your ICED32™ cell when you read in the command file created by the NLE. **By default, the layer used by the ECC for errors other than shorts is layer number 99.** You can use this rule to change the layer number used by the NLE to create these shapes.

The only other layer on which the ECC will create shapes, is the layer used to outline the path of shorts. (See the previous rule.)

Example: **ERROR_LAYER 199**

When the above rule is used, the layer used by the ECC for errors, other than path tracing through shorts, will be layer number 199.

DISJOINT *node_name*

The ECC will issue an error message when two or more separate nodes have identical node labels. This is considered an open circuit by the ECC.

You can prevent error messages for nodes left unconnected by design through the use of the DISJOINT rule. Nodes that are labeled with a node_name supplied in a DISJOINT rule will be considered virtually connected by the ECC.

This rule controls virtual connections for only ECC purposes. There are several ways to allow virtual connections of nets in the LVS. This rule has no effect on the layout netlist input to the LVS.

Example: **DISJOINT VDD:**

This rule in your rule set will suppress ECC error messages about opens between physically separate nets that are labeled with the text "VDD:". Note that if you have labeled the nets with a colon (':') suffix to identify them as global nets, you should include that suffix when typing the *node_name* in a DISJOINT rule.

You can add multiple DISJOINT rules to your rule set as required.

UNCONNECTED *nc_node_name*

When this rule is used, the ECC will issue an error message for any polygon labeled with the text nc_*node_name* which is electrically connected to another polygon. This is especially important for verifying that those chip pads that are meant to be unconnected do not short to any nets. You will receive an error message even when an unlabeled net shorts to any of the specially marked pads. Ordinarily, the NLE would have no way of realizing that a short between an unlabeled net and a large metal shape, which happens to be a pad, is an error.

Refer to **Verifying Pad Connections** to for methods to insure that pads are connected correctly.

The ECC will ordinarily find only those shorts that have conflicting labels. Unless you label every node in your design, shorts between nodes without labels will not be found by the ECC. When a short involving unlabeled nodes is between separate nodes in your circuit, you will be alerted to this in the LVS since the circuit will now be incorrect. However, when the short is between a node and an unconnected pad, which is not listed in the schematic netlist, neither the ECC nor the LVS will issue an error message. There is no way for either program to differentiate between an accidental short to a pad from an intentional one.

Accidental shorts to a pad can cause real problems. Using this rule and a few extra text components is an easy way to prevent accidental connections.

Example: **UNCONNECTED NC**

Using this rule in your rule set, and adding text components with the string "NC" on top of shapes in your design, will cause an error message to be printed if any of those shapes is electrically connected to any other shape.

The **NO_UNCON-NECTED** NLE command line option suppresses this test.

The text components must be on the layer of the polygons they are meant to label or moved to that layer during the rule set. The text components should be in the top-level cell.

You should use only one UNCONNECTED rule in your rule set. You can have many polygons labeled with node label *nc_node_name* and they will not be listed as opens.

If your pads are constructions of several electrically connected shapes, rather than simple polygons, see the PADSIZE rule below.

PADSIZE [=] *pad_dimension*

This rule affects only the ECC test for unconnected shapes as discussed in the previous rule. It is used to allow simple constructions of connected shapes to be marked as unconnected.

When this rule is not used, any short to a polygon which is marked as unconnected will cause the ECC to issue an error message. When this rule is used, the polygon marked as unconnected can short to other shapes within a box around itself without generating an error message. This prevents false errors when an unconnected pad is formed from several polygons connected to each other.

The *pad_dimension* parameter must be provided in the user units of the ICED32™ cell. Any connections from the polygon marked as unconnected, to other polygons within a box square with the axes with sides equal to *pad_dimension,* will not generate error messages. Shorts to other polygons outside of this box will generate error messages.

Example: **INPUT LAYER 6 M1 ; 7 M2; 8 VIA**

```
LABEL          M2
CONNECT        M1   M2   BY   VIA
UNCONNECTED  NC
PADSIZE 20
```

When this rule set is processed on the shapes in Figure 65, only the pad on the right will have an error message issued for it by the ECC. The M1 box which forms part of the pad on the left is electrically connected to the M2 box of the same pad, but no error message will be printed for it, since it is within a box of 20 units on a side.

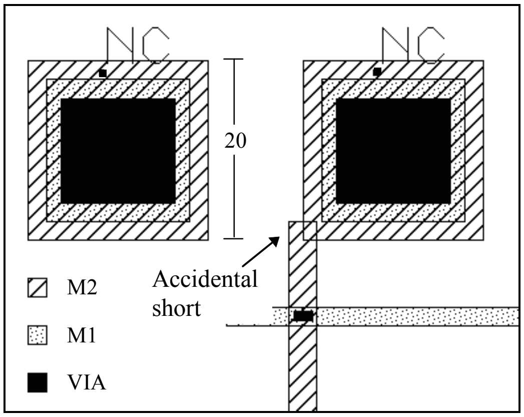


Figure 65: Two pads marked as unconnected.

In the example above, the "NC" text components are on the M2 layer. They will be processed by the ECC since the LABEL M2 rule is included. The origins of the text components are indicated by the small boxes. Note that the origin of each text component is covered by the M2 layer. This is enough to let the NLE label the polygons with the text. The entire text component does not need to be covered by the shape.

Running the NLE

Running the NLE to extract a layout netlist involves several steps.

Write the NLE rules which define layer manipulation, electrical connections, and device recognition in any ASCII text editor. It is recommended to name this file with a ".RUL" extension, but this is not required.

See Figure 1 on page 18 for a graphical representation of these steps.

Compile the rule set in DOS with the NLE rules compiler.

Create the binary layout data file for the NLE from your ICED32™ cell by using the DRC command in the ICED32™ layout editor.

Execute the NLE circuit extractor in DOS using the compiled rules file and the binary layout data file.

The DRC command is completely described in the ICED32™ Layout Editor Reference Manual. For NLE netlist extraction of your entire design, you can simply type "DRC" on the command line. The binary layout data file will be created as "*cell_name*.POK" where *cell_name* is the first 8 characters of the name of the cell you are editing.

We have already covered how to write the rule set. Next, we will describe how to run the rules compiler and finally how to run the circuit extractor.

NLE Rules Compilation

Rules Compiler Command Line Syntax

[*prog_path*\]**RULESNLE** [*rule_path*\]***rule_file_name*** ...
... [USE=*u_kbytes*] ...
... [SCRATCH_DIR=*scratch_path1* [; ... *scratch_path5*]]

The *rule_file_name* parameter is the only required parameter on the rules compiler command line. It is the name of the ASCII file containing your rule set. A file extension of ".RUL" will be added to the file name if you do not supply the file extension in *rule_file_name*.

The NLE installation routine defaults to placing the RULESNLE.EXE file in the same directory as the ICED32.EXE file. This directory should be included in the PATH statement in your AUTOEXEC.BAT file. When this is the case, you do not need to supply *prog_path* on the command line, even when the file RULESNLE.EXE is not in the current directory.

Example: **RULESNLE DEVRECOG**

To see a list of the current DOS environment variables, use the DOS command SET.

This command line typed at the DOS prompt will execute the NLE rules compiler on the file DEVRECOG.RUL. This rules file must exist in the current directory. The executable file RULESNLE.EXE must be in the current directory, or in a directory defined in the DOS environment variable PATH.

The compiler will create the compiled rule file with the same file name as the input rule set except that the file extension will be ".LL". This file will be created in the directory where the input rule file is located. In the example above, the compiled rules file will have the name DEVRECOG.LL and it will be created in the current directory.

The optional [USE=*u_kbytes*] parameter is used to restrict the amount of memory the rules compiler will use. When the USE keyword is not used in the command line, the rules compiler will use 10 Megabytes (approximately 10,000 Kilobytes). We recommend using the USE keyword only when you are executing the compiler in a DOS shell of a multitasking operating system like Microsoft Windows. Enter the maximum amount of memory the rules compiler can use in Kilobytes.

If you notice a long delay when you execute the rules compiler, the problem may be that the rules compiler is initializing much more memory than it needs. Try using the USE keyword with an initial value of 2000 to prevent the compiler from initializing too much memory on your system.

If the SCRATCH_DIR keyword is not used, a single scratch file, $NLRVIRT.000, will be created in the current directory. If the compiler completes successfully, this file will be deleted.

There are two cases where it is important to use the SCRATCH_DIR option to set the scratch directory explicitly.

You can create a DOS batch file with the rules compiler command line so that you do not need to type the required parameters each time.

If you running the NLE programs on a network, several users can use the program at the same time. If they share a scratch directory, they will corrupt each other's scratch files. **When on a network, each user should have his own scratch directory.**

If the current drive or partition has limited space, you should specify at least one scratch directory on a drive with plenty of free space.

The scratch file for the rules compiler is usually very small, so the second reason above is rarely a concern. However, the scratch file for the NLE can grow very large. The SCRATCH_DIR command line option is covered in more detail in the discussion of the NLE command line options. See page 173.

Example: **RULESNLE E:\ICED\CKTREG.RUL USE=2000 SCR=E:\NLETMP**

This command line will compile the rules in the file E:\ICED\CKTREG.RUL. The rules compiler will use no more than 2000 Kbytes, or approximately 2 Megabytes, of memory. Note that the SCRATCH_DIR keyword can be abbreviated to SCR. The NLETMP directory on the E: drive will be used to store temporary files. This directory must already exist or the rules compiler will fail with an error message.

There are two ways to force the rules compiler to terminate before it completes normally.

<Esc>	Pressing this key will halt the rules compiler after it completes the current operation. The compiler will then close all open files and delete the scratch file(s). This may take a few moments, so you should be patient and wait for the compiler to complete these tasks.
<Ctrl><C>	Pressing both of these keys simultaneously will bring the rules compiler to an immediate halt. Files will not be closed properly and the scratch file(s) will not be deleted.

If you use <Ctrl><C> to terminate the compiler, or if the compiler crashes, you should delete the scratch file(s) yourself. Remember that the scratch file(s) are created with the name $NLRVIRT.000. You may also want to run the DOS utility SCANDISK (or equivalent programs available from other vendors) to find lost chains on the disk which were left behind because files were not closed properly.

Rules Compiler Output Files

The rules compiler will create the compiled rules file with the name *rule_file_name*.LL. If a file with the name *rule_file_name*.LL already exists, it will be overwritten. This file will be created in the same directory as the source rules file. This compiled rules file will be used by the NLE program.

To suppress the warning prompt when the source rules file is not present, add the NO_RUL keyword to the NLE command line. See page 176.

You should usually leave the source rules file in its original location after you have compiled it. The location and time/date stamp of the source rules file is stored in the compiled rules file. The NLE will search for the source rules file to insure that the current source rules file has the same time/date stamp as the one used to create the compiled rules file. If the NLE cannot locate the source rules file, it will issue a warning prompt and you must reply to proceed.

The rules compiler will create a log file with the name *rule_file_name*.RLO. This file will also be created in the same directory as the source rules file. If a file with the name *rule_file_name*.RLO already exists at the start of the rules compiler program, it will be renamed to *rule_file_name*.RL1.

The rules compiler log file begins with a block of comments that include the version number of the compiler. An echo of the source rules file comes next. Any comments from the compiler about the source will be prefixed with a double dash ("--") before the comment.

If the compiler finds a syntax error as it parses the source rules file, it will stop reading the file and print an error message after the line with the error. The parameter or keyword with the problem will be indicated with double carats ("<< >>"). Only one error will be found per compile.

Warnings may be scattered through the log. All warnings will be prefixed with the string "**WARNING". All errors and warnings will be printed on the screen as well as listed in the log. Some of the warnings you may see are listed below.

Message	**Cause**
No layers have been specified as having node labels....	This message is printed when you have not used the LABEL rule to specify which layers contain node labels for ECC checking. The ECC cannot proceed with the tests it is supposed to perform unless nodes are labeled. You can use the NO_ECC rule to suppress this message.
Layer *xxxx*[*n*], which contains pins, is not connected to any other layer.	This message indicates that device terminals are located on a layer which is not used in a CONNECT rule. This means that the pins of these devices can never be connected to other devices. You may have forgotten to include the indicated layer in a CONNECT rule, or a device rule may specify a temporary layer rather than the intended layer.
Scratch layer *xxx*, set on line *n*, is never used. Action will be deleted: ...	This warning occurs when you have included a rule which creates shapes on a scratch layer, but no other rule uses that layer. The processing to create the scratch layer is unnecessary, so the rules compiler deletes the rule entirely. This situation may occur when you modify a rule set by removing a rule that used the scratch layer. The NLE will then optimize your rule set by removing the rules that create the layer. In this case you can choose to ignore the warnings, or go back and comment out the indicated rule(s). However, if you wanted to look at the shapes on that scratch layer, you should change the line that defined the layer to an OUTPUT LAYER rule instead of a SCRATCH LAYER rule.
Layer number *n* is also an input layer.	When this message is issued, you have used the same layer number as both an input layer and an output layer. The rules compiler warns you about this because if you add shapes on the output layer to your design cell you will be modifying a design layer. If you will not be using the NLE command file to add the output layer shapes to your original cell, you can ignore this warning.

Figure 66: A few of the NLE rules compiler warnings

If the source rules file contains no syntax errors, the log file will continue with a summary of the layers used in the file. The NLE layer name, ICED32™ layer number, the rules file line number which created the layer, and the layer type (INPUT, OUTPUT, or SCRATCH) will be listed for each layer.

Layers that are created, but are not used in the rule set, will not be listed in this list of layers. They are removed automatically from the rule set by the compiler.

Next, the rules log will list all constants created by the CONST rule.

The log will then list the rules exactly as they will be executed by the NLE. The rules are grouped together in passes. Each pass requires each shape in the NLE database to be interrogated by the NLE. The more passes, the longer the NLE run.

The order in which rules are executed may not be the order in which the rules are written. The rules compiler may change the order to minimize the number of NLE passes. No change made by the compiler should affect how the layers are processed.

One example of a rule generated by the compiler, is a CONNECT rule added to insure that connections which cross panel boundaries will be handled correctly.

Each NLE layer name mentioned in this list of rules will be followed by the ICED32™ layer number enclosed in square brackets ("[ ]"). The line number in the source rules file is indicated on the line after the rule enclosed in parentheses ("( )"). If the compiler has generated the rule, the word "Generated" will be used instead of the line number. Some additional information may be provided with the rule, such as the bloat angle in effect for BLOAT or SHRINK rules.

Next will be a summary of connection groups. These are groups of layers which are electrically connected through the CONNECT rule. If your log indicates more than 1 group, you may have omitted a CONNECT rule from your rule set.

Layers in the rule set which are not electrically connected to any other layer are listed under the heading "Unconnected layers". This list may contain intermediate layers, or layers which are never used. However, you may want to browse this list to insure that none of the layers that you assume are electrically connected are included in the list.

Near the end of the log file will be a report of some of the optional ECC settings. If you have used the DISJOINT, UNCONNECTED, or COMMENT rules, the settings will be listed here.

The final line in the log file from a successful compilation will always be the word "Done".

Running the Circuit Extractor

NLE Command Line Syntax

```
[prog_path\]NLE [rule_path\]rule_file_name ...
                ... [layout_path\]layout_file_name ...
                ... [output_path\]output_file_base_name ...
                ... [@opt_file] ...
                ... [USE u_kbytes] ...
                ... [SCRATCH_DIR=path_name] ...
                ... [SHORTRUN] ...
                ... [LOGEXT] ...
                ... [LOGBAD] ...
                ... [SHOW_BORDER] ...
                ... [NO_RUL] ...
                ... [LIST_RULES] ...
                ... [NO_ECC] ...
                ... [SEARCH_DEPTH = path_depth] ...
                ... [NO_OPEN] ...
                ... [NO_UNATTACHED] ...
                ... [NO_UNCONNECTED] ...
                ... [NO_DUP_ID_CHECK] ...
                ... [STARTCMD = "st_cmdstring"] ...
                ... [ENDCMD = "end_cmdstring"] ...
                ... [NO_NET_LIST] ...
                ... [NO_SAVE_CONNECTIONS] ...
                ... [NO_NODE_COUNT] ...
                ... [NO_NLE]
```

The previous page indicates the syntax for running the NLE circuit extractor. The first two input files, *rule_file_name* and *layout_file_name*, must already be prepared before you execute the program. All three required file parameters and each optional parameter are described in detail below.

The command line is typed at the DOS prompt, outside of the ICED32™ layout editor. If the directory where NLE.EXE is installed is included in the DOS environment variable PATH, or if this directory is the current directory, the *prog_path* parameter is not required.

Use blanks or commas to separate command line parameters. The underscores, '_', used in several of the option keywords are optional and are included for readability only. Several of the keywords can be abbreviated.

There are two ways to force the NLE to terminate before it completes normally. (These are the same methods used with the NLE rules compiler.)

<Esc>	Pressing this key will halt the NLE after it completes the current operation. The NLE will then close all open files and delete the scratch file(s). This may take a few moments, so you should be patient and wait for the NLE to complete these tasks.
<Ctrl><C>	Pressing both of these keys simultaneously will bring the NLE to an immediate halt. Files will not be closed properly and the scratch file(s) will not be deleted.

If you use <Ctrl><C> to terminate the NLE, or if the NLE crashes, you should delete the scratch file(s) yourself. The scratch file(s) are created with the name $NLEVIRT.000. You may also want to run the DOS utility SCANDISK (or equivalent programs available from other vendors) to find lost chains on the disk that were left behind because files were not closed properly.

The Three File Parameters

The three file name parameters are the only required parameters on the NLE command line.

The [*rule_path*\]*rule_file_name* parameter supplies the name of the compiled rules file. This file must already have been created by the NLE rules compiler described beginning on page 161. If no file extension is supplied in *rule_file_name,* a file extension of .LL will be added to the file name before the NLE searches for the file. The NLE will search for the rules file in the following directory paths in the order shown:

1) *rule_path* (if you have supplied this parameter with the file name),
2) the current DOS directory,
and
3) the DOS environment variable DRC_PATH.

DOS environment variables are set at the DOS prompt or in a batch file (e.g. AUTOEXEC-.BAT).

The [*layout_path*\]*layout_file_name* parameter must be the name of the binary layout file created by the DRC command in the ICED32™ layout editor. A file extension of .POK will be added to the file name when you do not supply it on the command line.

See an example of creating a binary layout file on page 21.

The [*output_path*\]*output_file_base_name* parameter supplies the base file name for several output files. Most of the files created by the NLE will use this parameter as the file name. The file extensions of the output files will vary. The extensions and file contents are described beginning on page 185.

Example: **NLE CKTRECOG XCHIP XCHIPOUT**

This line typed at the DOS prompt will run the NLE with the input files CKTRECOG.LL and XCHIP.POK. The output layout netlist, log file, and several auxiliary output files will be created beginning with the string "XCHIPOUT" in the current directory.

Input Redirection, Memory, and Scratch Directory Options

[*@opt_file*]

The NLE has many optional parameters and the command line can get very lengthy. Since DOS commands are limited to 128 characters, you may not be able to add all of the options you need on the command line. To solve this problem, or just to save you from repetitive typing every time you run the NLE, you can use the *@opt_file* parameter to refer to a file which contains command line options, rather than typing all options at the DOS prompt.

Example: **NLE CKTRECOG XCHIP XCHIPOUT @NLEOPT.TXT NO_ECC**

This NLE command line will cause the NLE to read the command line parameters in the file NLEOPT.TXT and execute them as though they were typed at the command line. Note that you can add other command line options after the *@opt_file* parameter. Options typed last on the command line override options in the options file.

The options file can be created with any ASCII text editor. Any end-of-line character is interpreted as a blank space. So you can type each command line option on it's own line rather than typing all options on a single line.

Comments in the options file are acceptable and will be ignored by the NLE. A comment is created as an exclamation mark ('!') followed by text. All text from the exclamation mark to the end of the line is ignored.

You can use another *@opt_file* command line option in an option file. Option files may be nested up to ten levels deep.

[USE *u_kbytes*]

This parameter limits the memory the NLE is allowed to use. It is intended primarily for use on multitasking operating systems like Microsoft Windows. If you are running the NLE on a computer without a multitasking operating system, it is best to allow the NLE to use all available memory by not adding this option to the command line.

When this parameter is not used, the NLE will allocate all available system memory. This includes your system's entire swap file.

If you see a long delay before the first NLE pass is executed, even for small designs, the NLE may be initializing much more memory than it needs. You can add this parameter to your command line to limit the amount of memory the NLE will initialize. Supply the maximum amount of memory in kilobytes.

Example: **NLE CKTRECOG XCHIP XCHIPOUT USE=20000**

This NLE command line will limit the NLE to 20,000 kilobytes (or about 20 Megabytes) of memory. This should be sufficient for small chips. If you have plenty of memory on your machine, and the other system demands are light, higher numbers for *u_kbytes* will allow the NLE to run faster.

[SCRATCH_DIR=*path_name1* [;...*path_name_n*]]

This parameter specifies the directory (or directories) for the NLE scratch file (or files). **It is critical that a scratch directory is not shared between simultaneous executions of the NLE or NLE rules compiler. This could be a problem if you are executing the NLE over a network.**

You may also want to use this option if you have limited space on the current drive and you want to make use of the space on other drives for scratch files. If you specify multiple directory paths, the NLE will create scratch files in all specified directories at the start of the run. Then if the NLE runs out of disk space while using the scratch file in the first directory, it will use the additional scratch files. The additional scratch file directories should be on other disk drives.

If you do not use the SCRATCH_DIR command line option, the NLE will create a single scratch file with the name $NLEVIRT.000 in the current directory. This file can grow very large (more than 1 Gigabyte for large chips), so make sure that there is plenty of free space on the current drive.

When you want to allow the NLE to use more than one directory to utilize space on more than one drive, specify up to five directories on different drives. Each *scratch_path* directory should already exist. The maximum number of characters in all directory paths is 82.

Example: **NLE CKTRECOG XCHIP XCHIP SCR=E:\NLETEMP;D:\NLETEMP**

This command line will result in scratch files created on the E: and D: drives in directories with the name NLETEMP. Note that the SCRATCH_DIR keyword can be abbreviated to SCR. The directory paths are separated with semicolons. If the NLE completes successfully, all temporary scratch files will be deleted automatically at the end of the run.

Logging Options

These NLE command line options control how messages are reported by the NLE during execution, and whether or not to put extra information into the log file.

[SHORTRUN]

This optional keyword on the NLE command line optimizes the on-screen messages form the NLE for short runs. If you are running the NLE on a small design which completes in a few minutes, this will prevent the quick flashing of messages which might confuse some users into thinking that error messages are being printed to the screen too quickly to be seen.

This option will not affect the contents of the NLE log file.

If your NLE run is not a very short run, use the default screen display by not adding this keyword to your NLE command line.

[LOGEXT]

If you are running the NLE on a small circuit and encounter circuit recognition problems, you can add this keyword to the NLE command line. This will add to the log file a lengthy report on each device recognized by the NLE.

If your circuit is not very small, this option will make your log file **extremely** long. The file may grow to use up a large portion of available disk space.

If you simply want to see an ASCII listing of the layout netlist, you should use the LPE utility to translate the binary layout netlist into an ASCII netlist.

[LOGBAD]

This option on the NLE command line will add a detailed message to the log file for each shape on a device recognition rule *id_layer* that does not form a valid device. This is true whether or not the rule uses the ERR keyword to collect shapes on *id_layer* which do not form valid devices.

The log will report the coordinates of each shape and the reason why the shape failed to form a valid device.

Rules File Options

These NLE command line options affect how the NLE rules file is processed and reported in the NLE log.

[SHOW_BORDER]

This command line option will add panel border calculations for each pass to the NLE log file. This can help you understand how your rule set affects the layer reach and panel border used by the NLE. If the panel border is large compared to the panel size, your NLE run may be very slow.

To learn about panels and borders, read Panel Processing beginning on page 83.

[NO_RUL]

The location of the source rules file and its time/date stamp are stored in the compiled rules file. If the time/date stamp of the source rules file (with a .RUL file extension) is different than the information stored in the compiled rules file (with a .LL file extension), the NLE will warn you, then ask you with a prompt if you want to proceed. This is to avoid a wasted run when you edit the rules file, then forget to compile it, before re-executing the NLE.

If the source rules file is not found in its original location, you will also receive an error message and a prompt. This is due to the fact that the NLE cannot tell if the source rules file is newer than the compiled rules file.

If you want to suppress this error message when the source rules file cannot be found, and avoid the prompt asking you if you want to proceed, add the NO_RUL keyword to your NLE command line.

This keyword will not prevent an error message and prompt when the source rules file is present but has a different time/date stamp than that stored in the compiled rules file.

Example: **NLE CKTRECOG XCHIP XCHIPOUT NORUL**

Note that the underscore is optional in the NO_RUL keyword. This is true of all keywords in the NLE command line.

[LIST_RULES]

Add this keyword to your NLE command line to add to the NLE log file a report of which rules were executed during each pass of the NLE. Using this keyword will also add to the NLE log most of the other information related to the rules file which was reported in the rules compiler log.

Example: **NLE CKTRECOG XCHIP XCHIPOUT LIST**

Note that the LIST_RULES keyword can be abbreviated.

ECC and Other Optional Test Keywords

The following keywords will disable certain tests that the NLE performs automatically. You can use these options to increase the speed of your NLE run, but you may prevent errors from being found.

[NO_ECC]

See page 152 to learn about ECC processing.

Use this keyword on the NLE command line to suppress all ECC (Electrical Connection Checks) processing. This will have the same effect as adding the NO_ECC rule to your rule set without forcing you recompile the source rules file.

This will speed up your NLE run, but errors like shorts and opens will not be found.

[SEARCH_DEPTH = *path_depth*]

When the ECC finds more than one unique ECC node label on a single net, it always attempts to find a path from one label to the other conflicting label. This path, including the coordinates of each polygon on the path, will be reported in the NLE log file.

See page 192 for a hint on how to isolate difficult to find shorts.

The ECC will trace through the layout building every possible path from the first label. Once it finds a conflicting ECC label on any path, it will stop tracing and then print the path with the conflicting label in the log file.

This tracing process can take considerable processing time. This is why the NLE defaults to tracing through no more than 50 polygons as it builds the paths from the first label.

If you prefer that the NLE not spend time tracing long paths between conflicting labels, add this option to your command line with the *path_depth* parameter equal to a positive integer smaller than 50. The short will still be reported in the log, but the NLE will report that "The program could not find a path through the short."

You can use a value of 0 for *path_depth.* This will prevent all path tracing through shorts.

If you do want a path traced through a short, but the default of 50 for *path_depth* is not long enough to find the path, you can use this option with a value greater than 50 for *path_depth.* This will increase run time.

We do not suggest increasing the value of *path_depth* to more than 50, unless you have a specific short reported in a previous run of the NLE that the ECC is unable to trace using the default value.

[NO_OPEN]

Use the DISJOINT rule to suppress only error messages about virtually connected nodes. See page 155.

Adding this keyword to the NLE command line will prevent the NLE from searching for unconnected net fragments labeled with identical text components. This may save time on repeated NLE runs, but open circuits will not be found.

[NO_UNATTACHED]

During the ECC tests, the NLE will print error messages for text components that have origins not covered by shapes on the same layer. The NLE assumes that uncovered text components on design layers are placed in error.

If you want to avoid this test, possibly to save time, you can use the NO_UNATTACHED keyword on the NLE command line.

[NO_UNCONNECTED]

This test is described in detail on page 155.

One ECC test finds all polygons labeled with a special node name and verifies that none of these polygons short to another net. This test can be disabled by adding the NO_UNCONNECTED keyword to the NLE command line.

[NO_DUP_ID_CHECK]

The NLE normally performs a duplicate device id polygon test on your layout. If the same polygon is recognized as a device by more than one device recognition rule, the NLE will normally warn you with an error message. If you pass a layout netlist that contains two devices with the same node number to the LVS, the LVS will abort with an error message.

This test is performed outside of the ECC checks. This test is not disabled by the NO_ECC command line keyword. However, the NO_DUP_ID_CHECK command line keyword will prevent the NLE from running this test.

Normally, you want devices that share an id polygon flagged as errors. However the duplicate id check does add more time to the NLE run. You can add the NO_DUP_ID_CHECK keyword to save time in repetitive NLE runs. However, you should perform a final NLE run without the NO_DUP_ID_CHECK keyword to find any invalid devices.

Example:

```
DEVICE DRES ID=DEV {
   SRC_DRN 2/POLYGON
   PINS=SRC_DRN, SRC_DRN;
}
DEVICE PRES ID=DEV {
   POLY 2/POLYGON
   PINS=POLY, POLY;
}
```

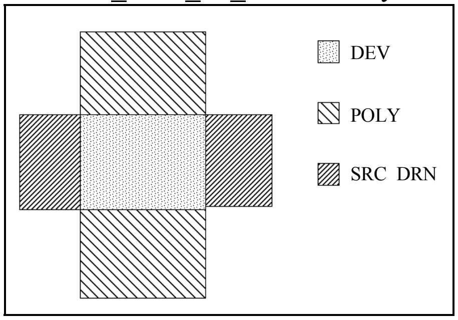


Figure 67: One shape on DEV

These rules above will result in an error message printed for the unusual device shown in Figure 67. However, if you added the NO_DUP_ID_CHECK keyword to the NLE command line, no errors or warnings would be printed.

Output File Options

These NLE command line options affect the generation of various output files.

[STARTCMD = "*st_cmdstring*"]

and

[ENDCMD = "*end_cmdstring*"]

To learn more about the command file created by the NLE see page 189.

These two command line options affect the command file created by the NLE to create shapes in the ICED32™ layout editor.

If you add the STARTCMD = "*st_cmdstring*" option to your NLE command line, the NLE will use *st_cmdstring* as the first line of the command file.

The ENDCMD = "*end_cmdstring*" option will cause the NLE to add *end_cmdstring* as the last line in the command file.

Both string parameters should be valid ICED32™ command strings. You should surround the string parameters with quotes since blanks or '@' characters (used for the *@file_name* ICED32™ command) might be misunderstood by the NLE command line parser.

Example:

```
NLE  RECOG  XCHIP  XCHIPOUT START="EDIT CELL NLEOUT"  ...
... END="EXIT;ADD CELL NLEOUT AT 0,0"
```

This NLE command line will create a command file with the name XCHIPOUT.CMD. The NLE will add the following line to the top of the file:

```
EDIT CELL NLEOUT
```

When the EDIT CELL command is executed, the ICED32™ editor will create a new cell with the name NLEOUT. All subsequent commands in the command file will create shapes in this new cell.

The command file will have the following two ICED32™ commands added on the last line:

EXIT;ADD CELL NLEOUT AT 0,0

When these two commands are executed, the ICED32™ editor will exit from the cell it has just created, then add that cell to the current cell at the coordinates 0,0.

The total effect of adding these three commands to the command file is that all shapes created by the NLE command file will be created in a subcell of the current cell, rather than as components in the current cell.

[NO_NET_LIST]

This command line option suppresses the generation of the binary layout netlist. This file is used by the LVS or LPE programs. If you do not intend to run the LVS (for comparison of the layout netlist to your schematic netlist) or the LPE (for generation of an ASCII netlist from the binary layout netlist), you can use this keyword on the NLE command line. This will save some run time if you are running the NLE only to find errors like shorts and opens.

[NO_SAVE_CONNECTIONS]

The node outliner commands are described on page 389.

Adding this keyword to your NLE command line will prevent creation of the node outliner file required by the outliner commands in the ICED32™ editor. This will save a small amount of run time and possibly a large amount of disk space.

The node outliner file has a file extension of .P9K. It is required by the node outliner commands which highlight entire electrically connected nodes. If you prevent creating it by using this keyword, you will not be able to run the node outliner commands.

To reduce the size of the node outliner file, see the SAVE and NO_SAVE rules on page 144.

If the node outliner file is missing when you execute the node outliner commands, you will not see an error message when you execute the @NODES command in the ICED32™ layout editor, but when you execute the N0, N1, or NN commands you will see the error message, "Could not open file *cell_name*.P9K."

[NO_NODE_COUNT]

The node summary file is described on page 193.

This keyword suppresses creation of the node summary file. This can save a small amount of run time and disk space.

[NO_NLE]

Adding this keyword to your NLE command line is equivalent to adding both the NO_NETLIST and NO_NODE_COUNT keywords. This will save run time and disk space if you are running the NLE for only the ECC checks.

NLE Output files

The NLE can generate several output files. The file name extensions vary from file to file, but the base part of the file name will be the same for most of the files. The base part of the binary layout netlist, the log file name, and two other auxiliary files will be the string provided in the *output_file_base_name* parameter (the third parameter) on the NLE command line.

If you do not provide an *output_path* with the *output_file_base_name* parameter, all files will be created in the current directory.

If a previous NLE run has created output files in the same directory, most of those files will be renamed with a '1' replacing the final character in the file extension before the new files overwrite them. This provides a backup of the results of your last NLE run with no effort on your part.

	File extension	Contents and use of file
NLE log file	.NLO (previous run backup .NL1)	General information about NLE run All error and warning messages Statistics on run time
NLE command file	.CMD (previous run backup .CM1)	Command file used to create shapes on output and error layers in the ICED32™ layout editor
Node summary file	.DEG (previous run backup .DE1)	Reports on nets sorted by degree Lists of the device terminals connected to each net
Binary layout netlist	.EXT (previous run backup .EX1)	Binary file used as input to the LVS or LPE utilities
Node outliner file	.P9K(file name is based on top-level cell name) (previous run renamed to .P99, then deleted at end of run)	Binary file used by node outliner commands to highlight nets in the ICED32™ layout editor.
Bad polygon command files	.ERR (file names are based on cell names, if possible) (previous run backups .ER1)	One command file is created for each input cell that contains bad polygons. The shapes are copied to an error layer.

Figure 68: NLE output files.

The NLE log file will record the file names used for each of these files.

Another file created by the NLE is the scratch file, $NLEVIRT.000. This file should be deleted by the NLE at the end of a successful run.

NLE log file

This file is where the NLE will store all error and warning messages. All of the information printed on your screen as the NLE is executing (except for the progress indicators) will be recorded in this file at the same time.

If you use the following NLE command:

```
NLE MYRULE MYCELL NLEOUT
```

The name of the NLE log file will be NLEOUT.NLO.

To include a complete listing of the rules file in the NLE log, use the LIST_RULES keyword on the NLE command line.

The file begins with some general information about the NLE run. Details on the input layout file and the input rules file are provided near the top of the log file.

If you have not included node labels in your layout, or have not included the LABEL rule in your rules file, the NLE cannot perform the ECC checks. When this is the case, the NLE will add the following warning to your log file:

To disable this message, add the NO_ECC option to the command line.

```
*****WARNING *****WARNING*****WARNING*****WARNING***
*****WARNING *****WARNING*****WARNING*****WARNING***
*****WARNING *****WARNING*****WARNING*****WARNING***
No layers have been specified as having node labels. No ECC errors will be found.
```

If bad polygons (shapes which may cause problems for mask processing software) are included in your layout, they will be mentioned next in the log. Each cell that contains bad polygons will have a command file created for it that has the bad polygons copied to an error layer. These files are described in more detail on page 193.

The NLE command line option LOGBAD will add an error message to the log for every device id polygon that fails to form a valid device.

Some details on how the layout was ungrouped (i.e. flattened) are printed next. If your layout contains cells which are not ungrouped because they are used in an INCELL rule, this will be reported.

Next is a summary of how many devices were recognized. If any devices had errors, this will be recorded here.

Next the NLE reports on the creation of the node outliner file. This file is created with a .P1K extension, but before the end of the NLE run, it will be renamed with a .P9K extension.

The results of the ECC tests are recorded next. If a polygon listed in an ECC error message is labeled with a polygon id label, this label will be included in the error message. Most ECC error messages also include the coordinates of the error. If shapes were created in the command file to help you locate the error, this will be mentioned in the error message.

At this point in the log file, the NLE will add the following line at the conclusion of a successful run:

Done.

After this line, the log file contains a summary of shapes created on output and error layers, as well as counts of ECC errors.

See page 180 for an example of a shared id polygon.

A reference to shared ID polygons may come next:

***n* devices are on shared ID polygons.**

This refers to polygons on a device id layer that form more than one device. Any number for *n* other than 0 represents a problem, and the LVS will abort if you use this layout netlist for comparison. When this line in the log indicates that you do have devices on shared id polygons, more detailed errors should be present earlier in the log.

The log ends with statistics on the scratch file and run times. If a large scratch file was required, you may be able to decrease your run time significantly by reducing the panel size. One indicator for NLE efficiency is the percentage of time the NLE spent on "Disc swaps". If more than 50% of the run time is spent swapping information into and out of the scratch file, and you are not yet breaking the design into panels, you should read Panel Processing on page 83.

NLE command file

This file will contain ICED32™ layout editor commands which create shapes to help you locate the errors the NLE has found or to see the results of layer processing.

Every shape on layers defined with the OUTPUT LAYER rule will have an ADD command in the file. These commands are created from the layer data at the end of the NLE run.

In addition, the ECC can add ADD commands to the file that will create shapes on layers defined with the PATH_LAYER and ERROR_LAYER rules. These shapes correspond with ECC error messages in the log file.

The command file will have comments in it to aid you in determining which rule of your rule file generated the shape. The comment before a block of ADD commands will contain the string 'TAG=*rule_number*'. The *rule_number* parameter is equal to the number of the rule that created the shapes. The rule numbers are listed near the bottom of the rule compiler log.

To use this file effectively, you can add the shapes in the file to your top-level cell with the *@file_name* command in the ICED32™ layout editor. However, **executing this file in your design cell will add shapes to your cell.** If you prefer to isolate the shapes generated by the NLE from your design cell, you can use several methods which allow you to add the NLE generated shapes in a way that allows you to easily delete the shapes again without affecting your original design.

You can add ICED32™ commands automatically to the NLE command file with the START_CMD and END_CMD NLE command line options. See page 181.

One simple way is to create a new cell with the ICED32™ layout editor, then execute the *@file_name* command in this new cell. You can then add the new cell to your top-level cell, or add both the new cell and your top-level design cell to a different new cell. Usually, the best way is to add your top-level cell to the new cell containing the NLE generated shapes. This will allow you to turn the display of your design shapes on and off as you view the NLE shapes.

If you use the following NLE command line:

```
NLE  MYRULE  MYCELL  NLEOUT
```

then the name of the NLE command file will be NLEOUT.CMD. You can then launch the ICED32™ layout editor to create the new cell, NEWCELL, with the following ICED32™ command line:

```
IC32  NEWCELL
```

Add the NLE shapes with the ICED32™ command:

```
@NLEOUT
```

You can then add your design cell to the new cell with the following ICED32™ command:

```
ADD  CELL  MYCELL  AT  0,0
```

The shapes the NLE creates are usually difficult to see since they are copies of shapes in your design cell which are right on top the original shapes. There are several ways to make the shapes easier to see.

One way is to temporarily turn off the display of your design cell with the command:

```
BLANK  CELL  LAYERS  0:255
```

You can turn display of your design cell back on with the command:

```
UNBLANK  ALL
```

You can use color to highlight shapes on the NLE layers with the ICED32™ LAYER command. A good color to use for the NLE layers is the HI color. This color will always have priority on your screen so that shapes drawn with that color are not hidden behind other colors. To assign the color HI to a layer generated by the NLE command file, use the command:

LAYER *n* COLOR=HI

where *n* is replaced with the layer number in the NLE command file.

You can then make this color strobe on and off with the command:

BLINK

Another way to locate shapes on one of the NLE layers is to select only the shapes on that layer with the commands:

UNSELECT ALL
SELECT LAYER *n* ALL

You can then resize the view screen to see all selected shapes with the command:

VIEW SELECT

Once you have located an NLE generated shape which points out an error in your design cell, you can edit the cell which contains the original design shape, without exiting the editor, by using any of the ICED32™ edit commands: EDIT, PEDIT, or TEDIT.

The easiest edit command to use is usually the PEDIT NEAR command. After typing this command, place your cursor on an edge of the design shape you need to modify to correct the problem, then click the left mouse button. You are now editing the cell that contains the design shape, even if it is nested several levels down. The shapes in other cells will remain on the screen but you will not be able to edit them. If you prefer that shapes in other cells are not displayed while you edit the cell with the problem, use the TEDIT NEAR command instead of PEDIT NEAR.

If your NLE run found a short between different node labels on the same net, the ECC adds to the NLE command file commands which copy the shapes on the path from one label to the other on the layer defined by the PATH_LAYER rule (by default, layer 98.)

When the short is on a net that travels over most of the chip (e.g. GND), it may be difficult to find the short since most of the net will be copied to the path layer. When this is the case, you can use panel processing to isolate the short.

Look at Figure 69. These are the shapes on layer 98 from an NLE run which found a short between VDD and VSS on a chip. It is somewhat difficult to locate exactly where the short is from these shapes.

If we perform the same NLE run again with the rules PANELX=50 and PANELY=50 rules added to the rules file, the path of the short will look like Figure 70.

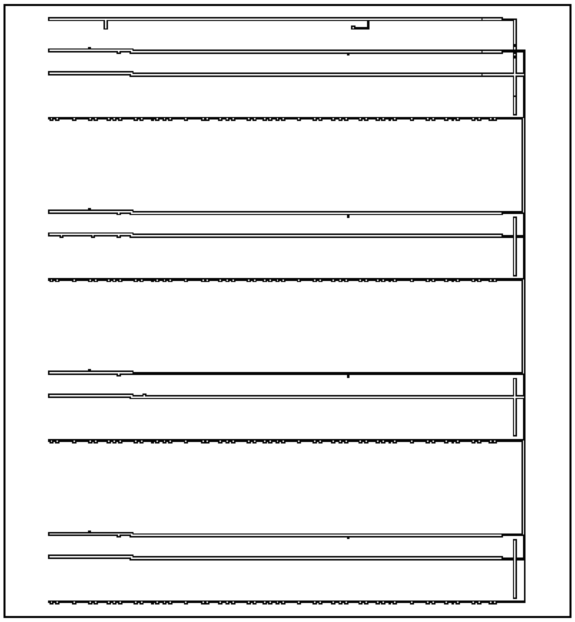


Figure 69: Path of short without panel processing.

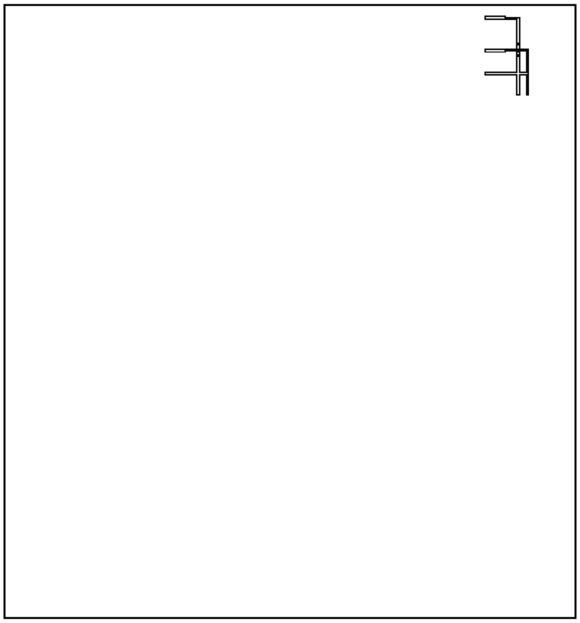


Figure 70: Path of short with panel processing.

Node summary file

The NLE command line option NO_NODE-_COUNT will suppress creation of this file.

The node summary file reports information similar to the LVS net degree report, but you can see the information before the LVS is run. This file will list the degrees of all nodes found by the circuit recognition. The degree of a net is defined as the number of device terminals to which the net connects.

The main use of this file is to quickly inspect the results of circuit recognition before you run the LVS. If you have many nodes of degree 1, and few or none of higher degrees, it is likely that your rule set is missing a CONNECT rule. If you have few nets, and each of these has a very high degree, it is likely that something is shorting your nets together. The cause of the short may be poorly written rules which short layers in error, or it may be stray pieces of conductive material which short many nets together.

See **Node Labels** on page 138 to learn how to add node labels.

If you have used node labels in your layout, the ECC will report shorts or opens in the NLE log file to provide clues to problems like these. However, this file can provide clues to major problems even when you have not used node labels. If this file alone does not provide enough clues to help you find the problem, you can add a few node labels to your layout at this point to help you see what nets are being shorted or left unconnected in error.

This report begins with a summary of how many device terminals (pins) and how many nets were found in the circuit. Then the report lists counts of nets sorted by degree.

The node outliner commands are described on page 389.

The next section lists the node numbers of nets of each degree. These node numbers can be used to highlight the nets in the ICED32™ layout editor using the node outliner commands.

The final section of the report lists each net by node number followed by the coordinates of each device terminal connected to the net.

The name of this file will be *output_file_base_name*.DEG. Before a file from a previous run is overwritten, it will be renamed *output_file_base_name*.DE1.

Binary layout netlist

Either of the NLE command line options NO_NETLIST or NO_NLE will suppress creation of this file.

This file is the input file for the LVS or LPE utilities. It is a binary file that contains a list of each device the NLE recognized along with the node numbers of the terminals. If the device rules in your rule set are written to extract device dimensions, this information will be stored with each device. LVS node labels found on devices or nets will be included in the file, in a format which the LVS or LPE can use to correspond the label to the net or device number in the netlist.

You can also get a lengthy ASCII listing of the layout netlist with the NLE command line option LOGEXT. See page 175.

Since this file is in binary form to speed the input process for the LVS, you cannot use an ASCII editor to view it. If you want an ASCII format of the file, you should run the LPE utility.

Node outliner file

This file is used by the node outliner commands in the ICED32™ layout editor to highlight devices or entire nets in your design. You can type in a node number and see the entire net blink, or select a node with the cursor and have the node number reported to you.

The NLE command line option NO_SAVE_-CONNEC-TIONS will suppress creation of this file.

The NLE generates the file *cell_name*.P9K, where *cell_name* is the same string as the *cell_name*.POK file used as the input file of the NLE. The LVS will create a companion file to the .P9K file with the name *cell_name*.P8K. The .P8K file is not required to run the node outliner commands, but it allows you to use the node names of nets (rather than node numbers only) to select them.

You can use the SAVE or NO_SAVE rules to reduce the size of this file.

The node outliner commands will expect these files to be named with the cell name. They should be located in the same directory as the cell file.

To learn how to use the node outliner commands, see page 389.

Bad polygon command files

The log file will warn you if your design contains bad polygons.

If the NLE found bad polygons or wires in your design cells, it will create separate command files with copies of these shapes to help you locate the problems. You can use these files to add copies of these problem shapes to your cells on a new layer in a similar manner as the other NLE command file described on page 189.

See the DRC manual for more information on bad polygons.

A "bad" polygon or wire is one that is likely to cause problems for mask processing software. A self-intersecting shape is an example of a bad polygon.

Unlike the other output files created by the NLE, these files are created before the design is flattened into one cell. The check for bad polygons is run as shapes are converted into polygons, before all cells are flattened.

One command file is created for each cell that contains bad polygons. The name of the command file will be *cell_name*.ERR, if that is a valid file name. If *cell_name*.ERR is not a valid file name, the NLE will create a file name based on the output file base name. (The actual names of all output files are stored in the log file.)

The coordinates used in the ADD commands contained in the bad polygon command files will be in the coordinate system of cell *cell_name*, not in the coordinates of the main cell. Run each command file while you are editing cell *cell_name*, not the top-level cell.

For example, you run the NLE on your highest level cell, MAINCELL. The cell NANDCELL is used 100 times in MAINCELL. NANDCELL contains a single self-intersecting wire. You get only a single warning message about the bad polygon, not 100 messages. The NLE will create a command file with the name NANDCELL.ERR. This command file will contain a single ADD command that creates a copy of the bad polygon on the error layer 99.

To see exactly which shape is causing the error message, you can edit the cell NANDCELL with the ICED32™ layout editor, then run the command file with the ICED32™ command:

See page 189 to learn other ways to use NLE generated command files.

```
@NANDCELL.ERR
```

This will add the shape on layer 99 to the cell. Now you can see the exact problem shape with the commands:

```
SELECT LAYER 99 ALL
VIEW SELECT
```

This will add select marks to the shape you just added and resize the view window so the shape is displayed. Once you see the shape that is causing the problem, delete the shape on layer 99 and fix the original shape.

LVS Basics

Overview

To see a diagram of how files must be prepared for the LVS program, see page 18.

The LVS is a comparison utility that verifies that two netlists represent the same circuit. It is usually used to compare a layout netlist to a schematic netlist. A layout netlist is generated by the NLE utility, which we have already covered.

The first step the LVS performs is to transform each netlist according to program options. This is done before any matching of circuits in the two netlists. These transformations include combining parallel or serial devices into single devices, filtering out shorted or unconnected devices, and making virtual connections of nets. This feature will allow circuits which are logically equivalent, but physically dissimilar, to match successfully.

The options that control these transformations, and options that control other aspects of the comparison, can be entered in three different ways:

option lines in the LVS control file,
parameters on the LVS command line,
and
device models in the netlists.

The control file contains many options. Several of these options can be overridden with parameters on the LVS command line. Parameters in device models override both of these other methods of entering options.

The control file defines default behavior for all devices of a specific type (e.g. resistors or bipolar transistors). The device models in the netlists can override this default behavior for only specific models of a device (e.g. polyamid resistors vs. diffusion resistors). For instance, you may want to merge most of your parallel resistors, but for certain resistors you want to prevent merges. You assign these resistors a unique model name in the netlist and create the device model for this model name that overrides the default defined in the control file for all resistors.

Other parameters in a device model allow the LVS to calculate a value for a device in a layout netlist based on dimensions calculated by the NLE. A tolerance for the comparison of device values can also be added to a device model.

The model name of devices in the layout netlist is defined by the *model_name* in the NLE DEVICE rule. An LVS device model must be included in the layout netlist for each unique model name in the netlist. Similarly, each unique device model name which occurs in a schematic netlist must have an LVS device model written for it which is included in the schematic netlist. When you create the device models for each netlist, you should insure that similar transformations are performed on each netlist.

You can specify that specific models can only be matched with devices using the same model, or allow any devices of the same type to match regardless of the model name used. This is defined in the control file, but can be overridden in specific device models.

Another method the LVS can use when transforming circuits in netlists is special handling of nets with certain node labels. Special character prefixes or suffixes on node labels can prevent transformations or allow virtual connections of nets.

Node labels are usually added to the layout in the ICED32™ layout editor. However you do not need to edit the layout and re-execute the NLE to modify the node labels in a layout netlist. You can add node labels overrides in the layout netlist to modify node labels.

Node labels can be used to assign forced points of correspondence between the two netlists. You usually define these points of correspondence with a node correspondence file.

Neither node labels nor forced points of correspondence are required to match the two netlists. However, forced points of correspondence based on node labels are highly recommended if your circuit is symmetric. The LVS uses a graph coloring algorithm which matches devices based on the number and types of devices each device connects to. If your design has many identical circuits, many devices will have identical properties and the LVS may be forced to match a random device from a list of identical devices in each netlist to continue it's matching algorithm.

If you provide a few forced points of correspondence, you can speed the matching algorithm considerably. This will also insure that the comparison progresses from a few known points of correspondence rather than from random nodes in a list of identical devices. However, if you assign a forced point of correspondence in error, you may prevent any circuits from matching.

All of these features will be explained in detail in the following chapters.

LVS Statement Syntax

The syntax descriptions for the statements in LVS input files use the following conventions. For the most part, these are the same conventions as those used in the NLE rules descriptions. The primary difference is that parentheses, ( )'s, are used to indicate option choices.

KEYWORD Bold type is used to indicate required keywords.

(KEYWORD A | KEYWORD B) Parentheses are used to indicate that a choice of keywords is required. The options are delimited with the | character. Exactly one of the options must be used. Do not type the parentheses.

[KEYWORD] Square brackets indicate that a keyword or parameter is optional. Do not type the brackets.

parameter value Lower case italic type is used to indicate where a value should be substituted. A value could be a number or a string. The type of value and the range is indicated in the manual.

a:z Ranges of possible values are indicated by a colon between the lowest valid value and the highest valid value.

... Three dots at the end of a line of sample code, or in a syntax description, indicate that the statement is continued on the next line. Another three dots will also preceed the continuation on the next line. The dots should not be included when the statement is typed.

Three dots in the middle of a line in a syntax description mean that several additional parameters are allowed but are not explicitly specified in the syntax description.

.VIRTUAL** ***net_name_1 net_name_2 **[...** ***net_name_n*****]**

This is part of the syntax description for the VIRTUAL netlist command. You may have multiple netnames after the *.VIRTUAL keyword. We use this syntax to indicate that you may enter as many netname parameters as necessary. The dots take the place of the missing parameters.

Q:\ICED

When this manual refers to the directory path Q:\ICED you should replace it with the drive letter and directory path where the LVS is installed.

For example, the tutorials refer to files in the directory:

Q:\ICED\SAMPLES\74181\LVS

If you have installed the LVS software on drive C: in the directory IC32, you should look for the files in the directory:

C:\IC32\SAMPLES\74181\LVS

LVS Input Files

Schematic Netlists

For an example schematic netlist file, see page 25.

The LVS can read schematic netlists written in the SPICE, PSPICE, HSPICE, or CDL[6] device level circuit simulation languages. The LVS accepts both flat and hierarchical schematic netlists. Many statements in the schematic netlist used for simulation, or other purposes, are ignored by the LVS. Statements that are not used by the LVS do not need to be deleted from the schematic netlist and are not reported as errors. Every effort has been made to reduce the effort of editing the original schematic netlist.

A hierarchical netlist uses .SUBCKT statements to define the hierarchy.

One addition you must make to the netlist is to add device models in LVS syntax for each device model in the netlist. The LVS will ignore all .MODEL statements in the schematic netlist. You must create a *.SCHMODEL statement for each device model in the netlist. These device models can be written in a separate file. This file can be combined with your original schematic netlist through the use of .INCLUDE statements.

One change you may have to make to the schematic netlist is to enclose the entire top level circuit in a .SUBCKT and .ENDS pair of statements. You specify the name of this top-level subcircuit in the control file (or in an optional parameter in the LVS command line).

These changes will be described in more detail after we cover how to write the device models for a schematic netlist.

[6] CDL stands for Circuit Description Language used by Cadence Design Systems, Inc.

The *.SCHMODEL Statement

Device models in the layout netlist are defined with the *.LAYMODEL statement described on page 229.

The *.SCHMODEL statement performs a similar role to the .MODEL statement in the SPICE language. You must create an *.SCHMODEL statement for each unique device model in the schematic netlist. The asterisk ('*') is required. It allows *.SCHMODEL statements to be present in a netlist used for simulation. The asterisk indicates a comment to programs which use the SPICE language and its derivatives. The LVS will parse *.SCHMODEL statements, but other programs will ignore them.

(The *.SCHMODEL statement is the equivalent of the *.ICEDMODEL statement in beta versions of the LVS.)

The syntax of an *.SCHMODEL statement is:

.SCHMODEL** ***model_name device_type ...

... [LTLR = *l_tolerance*] ...
... [WTLR = *w_tolerance*] ...
... [AREATLR = *a_tolerance*] ...
... [VALUETLR = *v_tolerance*] ...

Tolerance parameters

... [LOFFSET = *l_offset*] ...
... [WOFFSET = *w_offset*] ...

Control File

... [C_PERIMETER = *c_perim*] ...
... [C_AREA = *c_area*] ...
... [R_CONTACT = *r_contact*]...
... [OHMS_PER_SQUARE = *r_value*] ...

Device characteristics

... [L=*length*] ...
... [W=*width*] ...
... [M=*value*] ...
... [AREA=*area*] ...
... [VALUE=*value*] ...
... [BULK=*bulk_node_name*] ...

Default value parameters

(continued on next page)

```
... [ ALL=(YES | NO) ] ...
... [ ALLMERGE=(YES | NO) ] ...
... [ ALLCOLLAPSE=(YES|NO)] ...
... [ ALLMATCH=(YES | NO) ] ...
... [ ALLFILTER=(YES | NO) ] ...
... [ SMERGE=(YES | NO) ] ...
... [ PMERGE=(YES | NO) ] ...
... [ DSIZE=(YES | NO) ] ...
... [ CHAIN=(YES | NO) ] ...
... [ DCHAIN=(YES | NO) ] ...
... [ DMODEL=(YES | NO) ] ...          Control
... [ SERIES=(YES | NO) ] ...          file
... [ PARALLEL=(YES | NO) ] ...        overrides
... [ DCOLLAPSE=(YES | NO) ] ...
... [ MODEL=(YES | NO) ] ...
... [ PARAM=(YES | NO) ] ...
... [ OPEN=(YES | NO) ] ...
... [ ONE_CNCT=(YES | NO) ] ...
... [ TWO_CNCT=(YES | NO) ] ...
... [ SHRT=(YES | NO) ] ...
... [ GATE_NET=(YES | NO) ] ...
... [ SD_NET=(YES | NO) ] ...
... [ BASE_NET=(YES | NO) ] ...
... [ SWAP=(YES | NO) ]
```

Models can span several lines using continuation lines. See an example on page 217.

The *model_name* parameter is required. It must be identical to the device model name used in the schematic netlist. The *model_name* must not exceed 132 characters. Model names must begin with an alphabetic character. The characters '_' and '$', as well as any alphanumeric characters, can be used after the first character. Model names (as well as all other parameters) are not case sensitive. All characters are translated to upper case as the LVS reads them.

The *device_type* parameter is also required. This parameter must be one of the keywords in the table shown in Figure 72 on page 210.

Example: ***.SCHMODEL TRANSISTOR1 PMOS**

The *model_name* in this example is TRANSISTOR1. The *device_type* is PMOS. No optional parameters are used.

All parameters indicated with [ ]'s in the syntax description are optional. The order is not required to be the same as the order in the syntax description. The parameters will be read left to right. The last parameter read can override a previous one. You can use blanks for readability since they will be ignored. Use the entire string indicated (including the underscore '_') when typing parameter keywords. You can continue a line onto the next line with the "*+" continuation prefix. (See page 217 for an example.)

Tolerance Parameters

The default tolerance is non-zero to allow for truncation errors in floating point calculations.

The tolerance parameters are used to specify how closely the values of devices from each netlist must match if parameter checking is enabled. If no tolerance parameter is specified, the default is .0005, or .05%. If you want to override this default for an LVS comparison, **tolerance parameters must be added to the layout netlist device models**, not the *.SCHMODEL models. If you are performing an SVS (Schematic Vs. Schematic) or LVL (Layout Vs. Layout) comparison, you should add tolerance parameters to the models of the **second** netlist if they are required.

See Tolerance Parameters on page 233 for details on the use of these parameters.

Device Modifiers

The NLE BLOAT and SHRINK rules can modify device dimensions before device recognition.

JFET[7]
MOSFET
RESISTOR

Figure 71: Valid device categories for the LOFFSET and WOFFSET parameters.

The LOFFSET and WOFFSET parameters are used to adjust the dimensions of devices in the schematic netlist **before** any calculations or comparisons are performed. They can be used if the device values in the schematic netlist are in as-drawn dimensions, but the layout netlist has already been corrected to account for device layer shrinking or bloating.

[7] These parameters are valid only when JFET values in the device statements are defined with L and W rather than AREA.

The LOFFSET and WOFFSET parameters are more commonly used in layout netlist models. See page 235 for examples and more information.

Device Characteristics

These parameters are also more commonly used in layout netlist device models. They are used to determine how resistance or capacitance is calculated for devices that are defined by length and width rather than by value. See **Parameter Calculation** on page 343 for more details.

Default Value Parameters

If you provide default value parameter(s) for a specific model, the parameter(s) will be used as default values for each device using that model. You can override these default values in the device statements that refer to the model in the schematic netlist.

Example:

```
*.SCHMODEL  TRANSISTOR1  PMOS  L=6  W=10
*.SCHMODEL  TRANSISTOR2  PMOS

M1 1 4 12 6 TRANSISTOR1
M2 2 4 10 6 TRANSISTOR1  L=3  W=5
M3 3 5 18 6 TRANSISTOR2
M4 3 7 18 6 TRANSISTOR2  L=3  W=5
```

See **Parameter Calculation** to learn how to enable parameter checking.

This fragment of a schematic netlist shows two device models and then 4 device statements that use those models. The TRANSISTOR1 model uses default value parameters to provide a default length and width for all devices using that model. TRANSISTOR2 does not include default values. Device M1 will use the default length of 6 and width of 10. Since devices M2 and M4 define values explicitly, the defaults will be ignored. Device M3 will have no length or width assigned to it, and if parameter checking is enabled, it will never pass a parameter check.

Physical device category	***.SCHMODEL *device_type* keyword**	**Optional default value parameters**
MOSFET	NMOS PMOS	L=*length* and W=*width,* M=*multiplier*[8]

JFET	NJF
	PJF
		AREA=*area*
		L=*length* and W=*width,* M=*multiplier*[9]

GaAsFET	GASFET	AREA=*area*
BIPOLAR	NPN PNP LPNP LNPN	AREA=*area*
DIODE	D	AREA=*area*
CAPACITOR	CAP	VALUE=*value,* L=*length,* W=*width,* M=*multiplier*[10]
RESISTOR	RES	VALUE=*value,* L=*length,* W=*width,* M=*multiplier*[11]
INDUCTOR	IND	VALUE=*value,* M=*multiplier*[12]
TXLINE	TXLN	VALUE=*value*

Figure 72: Valid *.SCHMODEL *device_type* keywords.

You can indicate units of measure with the parameter values. You are responsible for making sure that the units of the *.SCHMODEL statements and the device statements in the schematic netlist are consistent with the units of the *.LAYMODEL statements in the layout netlist and the units of dimensions in the layout.

[8] Calculated device width is *width * multiplier.*
[9] L,W, and M are valid in HSPICE and CDL only.
[10] L,W, and M are valid in HSPICE only.
[11] L,W, and M are valid in HSPICE only.
[12] M is valid in HSPICE only.

A common error is to specify dimensions in the schematic netlist in meters by using a 'U' after the value (e.g. W=2.4U is translated to W=2.4e^{-6} or W=.0000024), but then the layout netlist expresses dimensions in microns (e.g. W=2.4).

See page 28 for an example of units conversion.

Make sure that the units of the device models agree with the units in the netlist. If there is a discrepancy in units between the two netlists, relate the units of one netlist to the other netlist with the SCALE_*device*_LENGTH_AND_WIDTH or SCALE_*device*_VALUE control file options.

In the HSPICE language, the M=*multiplier* parameter indicates the number of devices in parallel. When it is used with a MOSFET transistor, it will be used to multiply the width of the device during preprocessing. (If a device model specifies a WOFFSET parameter in addition to an M=*multiplier* parameter, the offset is applied before the device width is multiplied.)

Example: ***.SCHMODEL TRANSISTOR1 PMOS L=2 W=4 M=2**

This device model will be translated during preprocessing into:

***.SCHMODEL TRANSISTOR1 PMOS L=2 W=8**

When M=*multiplier* is used with a VALUE=*value* parameter, it will multiply or divide the value depending on the category of device.

Example: ***.SCHMODEL RESPOLY RES VALUE=12 M=2**

See page 245 for examples of how the M and AREA parameters are used for bipolar, JFET, GaAsFET, and diode devices.

The LVS will interpret this device model to be 2 resistors each of value 12 connected in parallel. It will translate this statement to the equivalent single device:

***.SCHMODEL RESPOLY RES VALUE=6**

Inductor values are calculated in the same manner as resistors.

Capacitors connected in parallel have their value increased. Therefore, the following device model will use the M parameter to multiply the value:

Example: ***.SCHMODEL CAPMODEL CAP VALUE=12 M=2**

and the model after preprocessing will be:

***.SCHMODEL CAPMODEL CAP VALUE=24**

The LVS will preprocess the values of device statements using M parameters in HSPICE netlists in the same manner as the device model default values in the examples above.

The [BULK=*bulk_node_name*] is a slightly different default value parameter. It is used to define the node name for the bulk layer terminal when it is not listed in a device terminal list. This parameter should be added to your device models only when you have set the NUMBER_OF_PINS_FOR_*device* control file option for the corresponding device type to a value which allows device statements with fewer terminals to have the bulk terminal added to each terminal list. See page 285 for details.

Control File Override Parameters

The remaining *.SCHMODEL parameters are used to override the device options in the control file on a per model basis. Their primary purpose is to enable or disable device transformations for only specific models of a device type.

For example, if you use the following option in the control file:

```
SWAP_GAASFET_SOURCE_DRAIN=YES
```

The LVS enables the swap of the source and drain terminals of GaAsFET devices where necessary to match devices in the two netlists. If you want this behavior for most of your GaAsFET devices, but not for a specific device model, you can disable the source/drain terminal swapping for only one model by using the SWAP=NO parameter on the device models for **both** netlists.

Example:

```
*.SCHMODEL  G_SWAP      GASFET
*.SCHMODEL  G_NOSWAP    GASFET  SWAP=NO

B1  1 2 3  G_SWAP
B2  4 5 6  G_NOSWAP
```

If these statements are used in one netlist, and devices corresponding to B1 and B2 exist in the other netlist, but with their sources and drains swapped, B1 will be matched, but B2 will not. When the G_SWAP model is used, the devices will match since the default in the control file is to allow terminal swapping for all GaAsFET devices.

You must be careful that the parameters of the *.SCHMODEL statements in one netlist are consistent with the device models in the other netlist. Control file overrides can also be used in the device models in a layout netlist. If you want to match device B2 in a layout netlist, the SWAP=NO parameter must be used in the model for that device. A device with swapping disabled will never match a device with swapping enabled, even when the terminals are not swapped. The corresponding device model in the layout netlist should be similar to:

```
*.LAYMODEL  G_NOSWAP  GASFET  SWAP=NO
```

The parameters beginning with the string ALL, override more than one option in the control file. To see which parameters to use to override specific control file options, see Figure 73. To learn what the control file options mean, see INDIVIDUAL DEVICE OPTIONS starting on page 283.

*.SCHMODEL Parameter			Control File Options Overridden
ALL	ALLMERGE	SMERGE	MERGE_SERIES_*device*S
		PMERGE	MERGE_PARALLEL_*device*S
		CHAIN	MERGE_*device*_CHAINS
		DCHAIN	MERGE_OUT_OF_ORDER_*device*_CHAINS
		DSIZE	MERGE_DISSIMILAR_SIZED_MOSFETS
		DMODEL	MERGE_*device*S_OF_DIFFERENT_MODELS
	ALLCOLLAPSE	SERIES	COLLAPSE_SERIES_LOGIC_*device*S
		PARALLEL	COLLAPSE_PARALLEL_LOGIC_*device*S
		DCOLLAPSE	COLLAPSE_DISSIMILAR_SIZED_*device*S
	ALLMATCH	MODEL	MATCH_*device*_MODELS
		PARAM	MATCH_*device*_PARAMETERS
	ALLFILTER	OPEN	IGNORE_UNCONNECTED_*device*S
		ONE_CNCT	IGNORE_ONE_TERMINAL_CONNECTED_*device*S
		TWO_CNCT	IGNORE_TWO_TERMINAL_CONNECTED_*device*S
		SHRT	IGNORE_SHORTED_*device*S
		GATE_NET	IGNORE_MOSFET_IF_GATE_PIN_IS_TIED_TO-_CRITICAL_NET
		SD_NET	IGNORE_MOSFET_IF_SOURCE_AND_DRAIN-_PINS_ARE_TIED_TO_CRITICAL_NET
		BASE_NET	IGNORE_BIPOLAR_IF_BASE_PIN_IS_TIED_TO-_CRITICAL_NET
	SWAP[13]		SWAP_GAASFET_SOURCE_DRAIN SWAP_JFET_SOURCE_DRAIN SWAP_CAPACITOR_TERMINALS SWAP_EMITTER_AND_COLLECTOR_TERMINALS

Figure 73: *.SCHMODEL parameters and the control file options they override.

[13] There is no way to disable swapping for MOSFETS, RESISTORS, or INDUCTORS.

If you use the ALLCOLLAPSE=NO parameter, it is the same as using both the SERIES=NO and PARALLEL=NO parameters. It will prevent the collapse of devices in the netlist to form pseudo devices with swappable gates. Using ALLCOLLAPSE=NO on the model statement for a MOSFET device will override the control file options COLLAPSE_SERIES_LOGIC_MOSFETS-=YES and COLLAPSE_PARALLEL_LOGIC_MOSFET=YES.

For example, if you have used the control file option COLLAPSE_SERIES-_LOGIC_MOSFETS=YES, but you want to prevent pseudo device creation for one model of MOSFET devices, you can use ALLCOLLAPSE=NO.

Example:

```
*.SCHMODEL P_DEFLT PMOS
*.SCHMODEL P_NO_CLPSE PMOS ALLCOLLAPSE=NO

M23 7 4 12 6 P_DEFLT L=6 W=5
M22 9 3 1 6 P_NO_CLPSE L=6 W=10
M21 8 2 1 6 P_NO_CLPSE L=6 W=10
```

See page 322 for details on series logic collapses.

Devices M22 and M21 are candidates for a series logic collapse into a pseudo device. However, since they use the P_NO_CLPSE model, they will not be collapsed. If devices M22 and M21 used model P_DEFLT instead, they would be collapsed.

Remember that the order of the parameters in a device model is important. The parameters are read from left to right. The parameter read last can override a previous one. If the *.SCHMODEL statement in the example above had been written:

```
*.SCHMODEL P_NO_CLPSE PMOS ALLCOLLAPSE=NO SERIES=YES
```

the SERIES=YES parameter would have overridden the ALLCOLLAPSE=NO. However, parallel collapses would still be disabled.

When using control file overrides to disable the test for matching model names (control file option MATCH_*device*_MODELS) or device values (control file option MATCH_*device*_PARAMETERS), be sure that the test is disabled in the device models of both netlists. For example: if the control file contains the option:

MATCH_CAPACITOR_PARAMETERS=YES

and you want to disable the parameter test for only one model of capacitors, the device models in both netlists must disable the test using PARAM=NO. If a test is enabled in the device model statements in **either** netlist, or by the default in the control file, the test is enabled.

Adding *.SCHMODEL Statements to the Schematic Netlist

A *.SCHMODEL statement can appear anywhere in a schematic netlist where a .MODEL statement is legal. A *.SCHMODEL statement can be defined globally outside any subcircuit, or inside a subcircuit definition. Models defined in a subcircuit take precedence over models defined globally.

For a sample schematic model file see page 25.

The preferred method is to globally declare all the *.SCHMODEL statements in a separate file. You can then create a third file which uses an .INCLUDE statement to include the model file and a *.SCHEMATIC statement to include the original schematic netlist. When you use this method, you can avoid editing the original schematic netlist. The .INCLUDE statements can include directory paths to the files if they are located in different directories.

All models in the schematic netlist must be defined before they are referenced. The LVS will report an error if a referenced model is not defined with a *.SCHMODEL statement before the reference occurs.

If you want the LVS to verify model names when matching devices, the model names chosen play an important role. If the model names of the device models in one netlist are different from the model names in the other, the LVS may end up not matching anything when the MATCH_*device*_MODELS=YES options are used in the LVS control file.

No single line should exceed 256 characters. Continuation lines (i.e. lines which begin with '+' or '*+') are allowed.

Example:

```
*.SCHMODEL  P_NO_CLPSE
*+  PMOS  SERIES=NO  L=6  W=10
```

Device Statement Restrictions

LVS commands are listed in the table on page 220.

All statements in the schematic netlist that do not begin with a '.' (commands) or a '*' (comments or LVS commands) are assumed to be device statements (also referred to as element statements). The devices recognized by the LVS are listed in Figure 74. All other device statements are ignored.

Device category	**Device prefix**
MOSFET	M
JFET	J
MESFET (treated as GaAsFET)	Z[14]
GaAsFET	B[15]
BIPOLAR	Q
CAPACITOR	C
RESISTOR	R
INDUCTOR	L
TXLINE	T[16]
Subcircuit call	X

Figure 74: Schematic device statements recognized by the LVS.

Subcircuit call statements can pass parameters. However, the [OPTIONAL: *interface_node* = *default_value*] and [TEXT: *name* = *text value*] parameters are ignored. The PSPICE syntax for passing parameters, [PARAMS: *name* = *value*], is supported.

The device terminal connections are always parsed and stored. The NUMBER_OF_PINS_FOR_*device* control file option can alter the way terminals referred to in the netlist are stored by the LVS. (See page 285.)

Most devices require a model name parameter. See Figure 75 for the only exceptions.

[14] SPICE language only.
[15] PSPICE language only.
[16] SPICE language only.

Passive devices in some schematic netlist languages do not require a device model. If your schematic netlist does not use model names in passive device statements (device statements beginning with the letter 'R', 'L', 'C' or 'T'), the LVS will supply default model names. See Figure 75 for the list of passive devices which do not require a corresponding *.SCHMODEL statement. All other devices, except for subcircuits (statements beginning with a the letter 'X'), require *.SCHMODEL statements.

Device Prefix	**Description**	**Default *model_name***
R	Resistor	RESISTOR
C	Capacitor	CAPACITOR
L	Inductor	INDUCTOR
T	Transmission line	TXLINE

Figure 75: Schematic devices with default model names.

For example, all device statements beginning with the letter 'R', without a model name, will have the same model name, "RESISTOR".

Example:

```
*.SCHMODEL  RES_POLY  RES  ALLMERGE=YES

.SUBCKT  TESTIT
R1  1  2  2.0
R2  2  3  2.0
R3  4  5  RES_POLY  4.0
R4  5  6  RES_POLY  4.0
.ENDS TESTIT
```

In the above example, the model name "RESISTOR" is assumed for resistors R1 and R2 since no model name is supplied. No *.SCHMODEL statement is required for model RESISTOR. (However, you can supply one if you desire.) Resistors R3 and R4 are defined with model RES_POLY. If this schematic netlist is run with the control file option MERGE_SERIES_RESISTORS=NO, the RES_POLY device model parameter ALLMERGE=YES will override the control file option and allow resistors R3 and R4 to merge. However, resistors R1 and R2 will not be merged.

*.SCHMODEL statements for these default model names are optional. You can add models for these default passive devices by using the model names shown in Figure 75.

In the example above, if you define a device model for RESISTOR, you could use the ALLMERGE parameter to allow the LVS to merge devices R1 and R2. You would have to add the following statement to the model file:

Example: ***.SCHMODEL RESISTOR RES ALLMERGE=YES**

The syntax requirements for supplying the value of a device vary in different schematic netlist languages. For instance, some languages allow you to specify either the length and width or the area of a capacitor, while others allow only the area. The LVS supports the published syntax of each netlist language for specifying the value or dimensions of each device, including the M=*multiplier* parameter in languages supporting its use. Other parameters are ignored.

In PSPICE, the syntax for a JFET device statement looks similar to this:

```
J1N  100  1  0  JFAST  2.0
```

where 2.0 represents the area of the device. No parameter name is allowed. However HSPICE allows the AREA= parameter to be used. In HSPICE the device statement above could be written:

```
J1N  100  1  0  JFAST  AREA=2.0
```

In either case, the LVS stores 2.0 as the area of device J1N.

Note that device statements beginning with 'V' (voltage sources) are not supported. According to Spice convention, two net nodes can be connected by specifying a zero voltage between them. For example:

```
v2 20 0 0
```

The LVS does not support this convention. The statement above would be ignored by the LVS.

Commands Supported in the Schematic Netlist

Command statements in schematic netlists are indicated by a '.' in front of the command. The command statements shown in Figure 76 are supported. All other command statements are ignored.

The LVS supports up to 10 levels of .INCLUDE or .INC statements.

All commands specific to the LVS have a '*' prefix so that these commands will be considered comments and will be ignored by other programs which read the schematic netlist.

We will not cover the use of commands defined by accepted SPICE syntax. We have already covered the *.SCHMODEL statement beginning on page 206. The other commands added for LVS purposes are covered below.

.END
.ENDS
.EOM
.GLOBAL or *.GLOBAL[17]
*.GROUND_NET
.INC or .INCLUDE
.MACRO
*.NOCOLLAPSE
.PARAM
*.PINS
*.POWER_NET
*.SCHEMATIC
*.SCHMODEL
.SUBCKT
*.VIRTUAL

Figure 76: Supported commands

[17] CDL language only.

*.GROUND_NET *gnd_name* [... *gnd_name_n*]

and

*.POWER_NET *pwr_name* [... *pwr_name_n*]

Ground nets	Power nets
GND	VDD
GND:	VDD:
!GND[18]	VCC
!GND:	VCC:
VSS	
VSS:	
0 (the number, not the letter O)	

Figure 77: Equivalent node names for ground and power nets.

The LVS will assume that nets in the top-level subcircuit with the names shown in Figure 77 are either ground or power nets. (Local nets nested in lower level subcircuits are not affected.) Different nets in each list will not be virtually connected.

The LVS treats power and ground nets differently from other nets. (For example, you can filter out bipolar NPN devices that have their bases shorted to ground. See page 300.)

If your ground or power nets have net names that are not in these lists, you can use the *.GROUND_NET and/or *.POWER_NET commands to add your net names to the lists the LVS keeps for valid names for ground or power.

Example: ***.GROUND_NET GROUND GND_X**

If you add this command to your schematic netlist, the net names "GROUND" and "GND_X" will be added to the list of nets that the LVS will consider to be ground nets. These nets must already exist in the top-level subcircuit in your netlist.

[18] HSPICE only.

The LVS will translate all nets with the name '0' to the name, "GND". If you intend that net 0 is in fact different from "GND", it will cause problems. Use a different name for separate ground nets.

***.NOCOLLAPSE** ***net_name* [... *net_name_n*]**

The *.NOCOLLAPSE command is used to prevent collapses for all devices attached to specific nets. It works in the same manner as labeling a layout net with the prefix defined by the control file option SPECIAL_CHARACTER-_FOR_NO_COLLAPSE_OF_DEVICES_CONNECTED_TO_A_NET. See Node Labels Which Prevent Device Collapses on page 351 for more information.

Example: ***.NOCOLLAPSE CLOCK1 CLOCK2**

The above command in the schematic netlist will prevent all devices connected to nets CLOCK1 or CLOCK2 from disappearing due to device merges or logic collapses. CLOCK1 and CLOCK2 will not be virtually connected. Listing both nets in one *.NOCOLLAPSE command is simply shorthand for the following two commands:

Example: ***.NOCOLLAPSE CLOCK1**
***.NOCOLLAPSE CLOCK2**

***.PINS** ***net_name*[:*padchar*] [... *net_name_n*[:*padchar*]]**

The *.PINS command is used to prevent disappearance of specific nets due to device collapses or merges. It works in the same manner as labeling a layout net with the prefix defined by the control file option SPECIAL_CHARACTER-_FOR_NO_COLLAPSE_OF_A_NET. See page 351 for more details on preventing the disappearance of nets from the netlist.

Example: ***.PINS NETA NETB**

This command in the netlist will prevent the nets "NETA" and "NETB" from disappearing from the netlist due to logic transformations.

All nets in the list of connections on the top-level subcircuit are automatically classified as pins. You do not need to add a *.PINS statement for these nets.

The *.PINS command can also be used to define pad connections of nets. See Pad Connection Verification on page 335 to learn about using the optional *padchar* specifications of the *.PINS command to specify pad connections.

*.SCHEMATIC [*dir_path*\\]*netlist_file*

You can use this command in your highest level schematic netlist file to specify the name of your original schematic netlist file. It works in the same way as an .INCLUDE statement. However, when you use this statement instead of an .INCLUDE statement, you can override the netlist file with the /S option on the LVS command line.

See page 228 for an example.

*.VIRTUAL *net_name_1 net_name_2* [... *net_name_n*]

This command will virtually connect all specified nets together as one net. The name specified as *net_name_1* will be used as the name of the net.

There are several other ways to virtually connect nets. See the list of methods on page 358.

Parameter Passing and Syntax Restrictions

You can use parameter substitution in the model file as well. See page 248 for an example.

There is a special consideration when defining parameters globally. In the PSPICE language, parameters defined on a subcircuit call have precedence over parameters defined globally using a .PARAM command. However, in the SPICE, HSPICE, and CDL languages, parameters defined globally have precedence over subcircuit parameters.

Example:

```
.PARAM val1=2
.SUBCKT  TESTIT
*
X1 CKT1 val1=4
*** IN PSPICE THE SYNTAX OF THE ABOVE STMT WOULD BE
*** X1 CKT1 params:val1=4

.ENDS TESTIT

.SUBCKT  CKT1 val1=8
*** IN PSPICE THE SYNTAX OF THE ABOVE STMT WOULD BE
*** .SUBCKT  CKT1 params:val1=8

M23 7 4 12 6 P_DEFLT L=val1 W=5
M22 9 3 1 6 P_NO_CLPSE L=6 W=10
M21 8 2 1 6 P_NO_CLPSE L=6 W=10

.ENDS CKT1
```

If the above netlist is parsed with the control file option SCHEMATIC-_FILE_FORMAT = PSPICE, the length of X1.M23 will be 8. In the other netlist languages, the length of the device will be 2.

There are two syntax features of the CDL language which the LVS does not support. The first is the use of the same node number more than once in a subcircuit definition.

Example:

```
SUBCKT  CMLT  16  14   22  22  45  46   ! Syntax error
```

The node 22 has been used twice in the terminal list. The error message the LVS will report in this case is "Duplicate node names not allowed". You must edit your schematic netlist to avoid the use of duplicate node names in a subcircuit definition.

The net name "0" is translated by the LVS to "GND". See page 222.

The other restriction is the use of global nodes in a subcircuit terminal list. CDL allows this, but it is not supported by the LVS.

Example:

```
.GLOBAL 1001
SUBCKT  AIT  16  14  600  11  23  1001 22   45  46   ! Syntax error
```

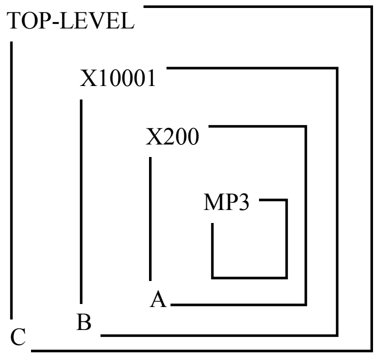
The global node 1001 has been used in the terminal list of subcircuit AIT. The LVS will report the error with the message "Global node not allowed as subcircuit argument". You must remove these nodes from the .SUBCKT statement.

Node Names

The term node refers to both device nodes and net nodes unless stated otherwise.

When the LVS processes a hierarchical schematic netlist, only global net nodes and devices in the top-level subcircuit preserve their original name. All other device and net names are expanded to uniquely identify them. The subcircuit name at the highest level will come first, then other names will be appended with a '.' in between each name as you proceed down the hierarchy. For example, look at the following schematic netlist fragment.

```
.SUBCKT  C
X10001 B
.ENDS C

.SUBCKT  B
X200 A
.ENDS B

.SUBCKT  A
MP3 7 4 12 6 PMOSDEFLT
.ENDS A
```



Device MP3 is inside subcircuit A. This subcircuit is added to subcircuit B with the instance name of X200. Subcircuit B is added to subcircuit C with an instance name of X10001. The device name in the LVS output will be X10001.X200.MP3.

Expanded node names have no upper limit on the number of characters.

The net name "0" is translated by the LVS to "GND". See page 222.

Device names, net names, and parameter names in the original netlist may be up to 132 characters long. Net names may consist solely of numeric characters, but device names and parameter names must begin with a non-numeric character.

Leading 0's are ignored for net names. "00145", "0145" and "145" are all translated to the node name "145".

Inserting a Top-Level Subcircuit

All devices to be verified by the LVS must be enclosed in subcircuits. If your schematic netlist contains devices which are not enclosed in a .SUBCKT/.ENDS pair of statements, the LVS will issue an error message similar to "FATAL ERROR: Device was not declared within SUBCKT" and abort.

If your schematic netlist has this problem, there are two solutions. You can edit the schematic netlist to gather all top-level device statements into one place, then add a .SUBCKT statement before the first device and a .ENDS statement after the last top-level device.

If you are automatically generating your schematic netlist from a schematic capture program, you can generate a dummy top-level schematic. In this case, the top-level devices will be automatically contained in a subcircuit.

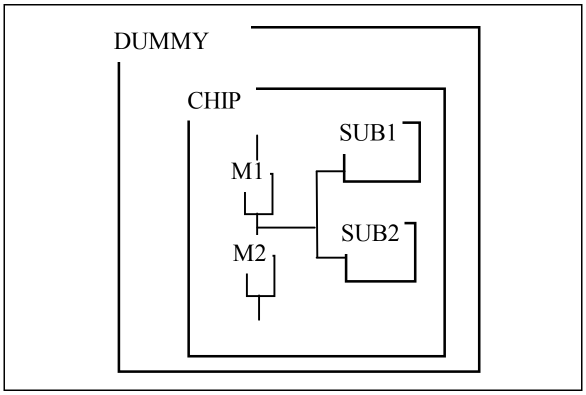


Figure 78: Schematics for netlist.

If your top-level schematic is named "CHIP" as shown in Figure 78, simply add "CHIP" to another schematic page named "DUMMY". Now output your schematic netlist from the "DUMMY" page rather than "CHIP". The devices on page "CHIP" will automatically be contained in a subcircuit with that name. Use that name as the parameter in the control file option TOP_LEVEL_SUBCKT_IN_SCHEMATIC-_FILE. (See page 258.)

Summary of How to Prepare a Schematic Netlist for the LVS

See an example of this process in the **Quick LVS Tutorial** on page 25.

1. Create a separate file which includes all of the *.SCHMODEL statements. You can name this file SCHMODEL.NET

2. Check the original schematic netlist for syntax that could cause problems (e.g. unsupported CDL syntax, see page 225).

   All devices must be contained in subcircuits. You must have a top-level .SUBCKT statement, and an .ENDS statement after the last top-level device. (See previous page.) Use the name of the top level .SUBCKT statement in the control file option TOP_LEVEL_SUBCKT_IN-_SCHEMATIC_FILE.

   For this example, let us assume that the original schematic netlist has the file name SCH.NET.

The LVS command line option /S can be used to override the filename in the *.SCHEMATIC statement.

3. Create a new file, with a name similar to LVS_SCH.NET. This file should include the lines:

```
*Schematic netlist for LVS check
.INCLUDE  SCHMODEL.NET
*.SCHEMATIC  SCH.NET
.END
```

The .END statement is essential at the end of file.

Use the LVS_SCH.NET file for the schematic netlist on the LVS command line.

If you prefer, you can add the *.SCHMODEL statements directly to the original schematic file and use that file name on the LVS command line. However, using the LVS_SCH file method above, you can avoid editing the original schematic netlist.

Layout Netlists

The LVS uses layout netlists generated by the NLE circuit extractor. The NLE uses data generated from an ICED32™ cell. The layout netlists generated by the NLE have a .EXT extension and are binary files.

The LVS also requires a separate ASCII model file for the layout netlist. Creating the model file is explained below.

The *.LAYMODEL Statement

The LVS needs a model for each unique device recognized by the NLE circuit extractor. These statements are very similar to the *.SCHMODEL statements in a schematic netlist. The syntax of the *.LAYMODEL statement, used to define device models in the layout netlist, is shown below.

The *.LAYMODEL statement is the equivalent of the *.EXTMODEL statement in beta versions of the LVS.

```
*.LAYMODEL model_name  device_type ...
     ... [LTLR = l_tolerance] ...                 ⎫
     ... [WTLR = w_tolerance] ...                 ⎬  Tolerance
     ... [AREATLR = a_tolerance] ...              ⎪  parameters
     ... [VALUETLR = v_tolerance] ...             ⎭
     ... [LOFFSET = l_offset] ...                 ⎫  Device
     ... [WOFFSET = w_offset] ...                 ⎬  modifiers
     ... [BENDS_CR = correction_factor] ...       ⎭
     ... [BOX_GM = (YES | NO)] ...                ⎫  Layout
     ... [MANHATTAN_GM= (YES | NO)] ...           ⎬  restrictions
     ... [REST_GM = (YES | NO)] ...               ⎭
     ... [C_PERIMETER = c_perim] ...              ⎫
     ... [C_AREA = c_area] ...                    ⎪
     ... [R_CONTACT = r_contact]...               ⎬  Device
     ... [R2_CONTACT = r2_contact]...             ⎪  characteristics
     ... [R2_WIDTH = r2_width]...                 ⎪
     ... [OHMS_PER_SQUARE = r_value] ...          ⎭
```

(continued on next page)

```
... [ L=length ] ...                          Default
... [ W=width ] ...                           value
... [ M=value ] ...                           parameters
... [ AREA=area ] ...
... [ PERIMETER=perim ] ...
... [ VALUE=value ] ...
... [ ALL=(YES | NO) ] ...
... [ ALLMERGE=(YES | NO) ] ...
... [ ALLCOLLAPSE=(YES|NO)] ...
... [ ALLMATCH=(YES | NO) ] ...
... [ ALLFILTER=(YES | NO) ] ...
... [ SMERGE=(YES | NO) ] ...
... [ PMERGE=(YES | NO) ] ...
... [ DSIZE=(YES | NO) ] ...
... [ CHAIN=(YES | NO) ] ...
... [ DCHAIN=(YES | NO) ] ...
... [ DMODEL=(YES | NO) ] ...
... [ SERIES=(YES | NO) ] ...                 Control
... [ PARALLEL=(YES | NO) ] ...               file
... [ DCOLLAPSE=(YES | NO) ] ...              overrides
... [ MODEL=(YES | NO) ] ...
... [ PARAM=(YES | NO) ] ...
... [ OPEN=(YES | NO) ] ...
... [ ONE_CNCT=(YES | NO) ] ...
... [ TWO_CNCT=(YES | NO) ] ...
... [ SHRT=(YES | NO) ] ...
... [ GATE_NET=(YES | NO) ] ...
... [ SD_NET=(YES | NO) ] ...
... [ BASE_NET=(YES | NO) ] ...
... [ SWAP=(YES | NO) ] ...
... [ TYPE= [P] [S] [G] [C] [I] [O] [B]       Pad type
][19]
```

[19] The TYPE parameter is used only when *device_type* = PAD.

The *model_name* keyword must be exactly the same identifier as the model name used in the device recognition rule in the NLE rules file. If you will be using the MATCH_*device*_MODELS options in the LVS control file, the *model_name* must also be exactly the same as the model name in the device model in the other netlist.

The *device_type* should come from the list supplied in the table in Figure 79.

The only required parameters are the *model_name* and the *device_type*. All other parameters are optional.

Tolerance parameters determine how close the values in this netlist must be to the values in the other netlist for the values to be considered a match.

Device modifier parameters are used to shrink or bloat geometry before dimensions are used to calculate the value of a device.

Layout restrictions will cause the values of devices with unusual layouts to be ignored.

Device characteristics parameters are used to calculate capacitor and resistor values.

Default value parameters define values for devices that cannot be calculated by the NLE circuit extractor.

Control file override parameters allow you to override options in the control file on a per-model basis.

Pad type parameters (used only for pad devices) allow you to define the pad type of a pad device. (See page 337 for more details on these parameters.)

Physical device category	***.LAYMODEL *device_type* keyword**	**Optional tolerance parameters**	**Optional device characteristics and default value parameters**

MOSFET	NMOS
	PMOS
		LTLR = *l_tolerance,*
		WTLR = *w_tolerance*
			L=*length* and W=*width*
			M=*multiplier*

JFET	NJF
	PJF
		AREATLR = *a_tolerance*	AREA=*area*
			M=*multiplier*

GaAsFET	GASFET	AREATLR = *a_tolerance*	AREA=*area*
			M=*multiplier*

BIPOLAR	NPN*n*
	PNP*n*
	LPNP*n*
	LNPN*n*
		AREATLR = *a_tolerance*	AREA=*area*
			M=*multiplier*

DIODE	D	AREATLR = *a_tolerance*	AREA=*area*
			M=*multiplier*

CAPACITOR	CAP	VALUETLR = *v_tolerance*	C_PERIMETER = *c_perim*
			C_AREA = *c_area*
			AREA=*area*
			PERIMETER=*perim*
			VALUE=*value*
			M=*multiplier*

RESISTOR	RES	VALUETLR = *v_tolerance*	R_CONTACT = *r_contact* or
			R2_CONTACT = *r2_contact*
			R2_WIDTH = *r2_width*
			OHMS_PER_SQUARE = *r_value*
			VALUE=*value*
			M=*multiplier*

INDUCTOR	IND	VALUETLR = *v_tolerance*	VALUE= *value*
			(value never calculated from layout)
			M=*multiplier*

TXLINE	TXLN	VALUETLR = *v_tolerance*	VALUE= *value*

PARASITIC-_CAPACITOR	PCAP	None (Never matched to devices in other netlist.)	C_PERIMETER = *c_perim*
			C_AREA = *c_area*
			(No default values)
			M=*multiplier*

PAD	PAD	None	TYPE = [P] [S] [G] [C] [I] [O] [B]
Subcircuit	SUBCKT	N/A	N/A

Figure 79: Valid *.LAYMODEL *device_type* keywords.

(The optional *n* used in the device types for bipolar devices represents the number of emitters or collectors in the device. When *n* is not used, the default is a single emitter or collector device. When *n* is used, it must be a positive integer.)

The PCAP device type exists only in layout netlists. It is used to model parasitic capacitors found by the NLE. Parasitic capacitors are **never** matched to devices in the schematic netlist. However, they can be very useful when you use the GENERATE_SPICE_NETLIST_FROM_THE_EXTRACTOR_OUTPUT = YES control file option. All parasitic capacitors found in the layout can be included for simulation purposes. If you prefer to filter these devices, so that only those above a threshold value are included in the generated spice netlist, use the control file option DELETE_PARASITIC_CAPACITORS_LESS_THAN = *filtval*.

Device models of PAD type devices can be used to verify pad connections. The pad device type is the only device type that uses the TYPE parameter. It is used to identify the type of a pad in the layout. See **Pad Connection Verification** on page 335 for more information.

The SUBCKT device type is used to expand a single layout device into other devices. (See **Multiple Emitter or Collector Devices** on page 331 for some examples.)

Tolerance Parameters

Note that value checking must be enabled in the control file with the option MATCH-_*device*-_PARAM-ETERS = YES.

Tolerance parameters are not intended to allow for process variation of device dimensions. They are used to allow a matched device to have a small difference in value in each netlist without an error message. When parameter checking is enabled, matched devices that have a difference in value outside the tolerance range will result in a parameter error messages.

There are three reasons why you may want to override the default tolerance of .0005 (or .05%) for some device models.

> Devices which have complex formulas for value calculation may have larger than normal floating point calculation deviations which result in a value mismatch.
>
> Devices with non-critical device dimensions may be approximated in the layout because of spacing or grid alignment issues. For example, if your schematic netlist calls for a 127Ω resistor, and in your technology resistors have a resistance of 10Ω/square, you may not be able to create the required resistor on grid. If the exact value of the resistor is not critical, you may want to add a slightly higher tolerance than the default to avoid many parameter error messages that you consider false errors.
>
> When a device is laid out as several devices in series or parallel, small mismatches due to diffusion shrinkage in transistors or contact resistance in resistors may add up to a large mismatch in the merged device. You may need a slightly higher tolerance to allow this type of device to match the value in the schematic.

The tolerance parameters are percentages expressed in decimal form real numbers in the range 0:1. The default value of all tolerance parameters is 0.0005, or .05%. When the tolerance is this small, the values of devices matched from the two netlists must be almost exactly the same. The default is non-zero so that small deviations caused by floating point calculations do not cause false errors.

A tolerance of 0 indicates that the values of matching devices from each netlist must be exactly the same. A tolerance value of 1.0 translates into a 100% tolerance. A 100% tolerance allows a device to match even when the value in the second netlist is twice the value in the first netlist.

Example:

```
*.LAYMODEL  PTRANSISTOR    PMOS    LTLR=0.1      WTLR=0.1
*.LAYMODEL  NTRANSISTOR    NMOS    LTLR=0.05     WTLR=0.05
```

See **LVS Output Files** for more information on the parameter error summary report.

These models define PTRANSISTOR as a PMOS device with a length and width tolerance of 10%. This means that the parameter values of length and width of a device in the second netlist must be within ±10% of the values of the matched device from the first netlist. If the device from the second netlist is outside of this range, the devices will still match, however the parameter error summary report will report a parameter error message for this device. The NTRANSISTOR device is defined as a NMOS device and the length and width tolerances are 5 percent.

Only the tolerances defined in the second netlist will be used. In an LVS comparison, the tolerances must be defined in the *.LAYMODEL statements of the layout netlist.

Device Modifiers

There are two methods for specifying the size of devices in a schematic netlist:

As-drawn dimensions	the dimensions represent the size of the device as it is drawn in the layout.
Effective dimensions	the size of the device has been corrected for diffusion shrinkage or other fabrication effects.

Most schematic netlists are created specifying as-drawn dimensions for the values of devices. The simulation models of the devices take into account that the device size will change as it is fabricated.

The NLE usually creates the layout netlist using as-drawn dimensions. (However the NLE BLOAT and SHRINK rules can be used to modify device layers in the layout netlist to extract effective dimensions.)

When both netlists use as-drawn dimensions, there is no discrepancy, and the comparison will be successful without extra parameters in the device models.

However, if your schematic netlist uses effective dimensions, and your layout netlist uses as-drawn dimensions, you can correct this discrepancy using the LOFFSET and WOFFSET parameters in the layout netlist device models.

JFET[20]
MOSFET
RESISTOR

Figure 80: Valid device categories for the LOFFSET and WOFFSET parameters.

The LOFFSET and WOFFSET parameters are used to adjust the dimensions of devices in the netlist **before** any calculations or comparisons are performed. (If devices are merged, the adjustments to dimensions specified with these parameters are applied before the merge.) These parameters can be used only for devices where the NLE extracts length and width rather than area or perimeter.

If the LOFFSET=*l_offset* parameter is used in a model, the length used in calculations and comparisons will be:

effective_length = as-drawn_length + *l_offset*

The WOFFSET parameter works in exactly the same manner.

effective_width = as-drawn_width + *w_offset*

For example, if MOSFET devices in your technology undergo a .16 micron diffusion shrinkage during fabrication, you account for this when you layout a device for this technology. If you need a device with an effective width of 4.0 microns after fabrication, you will create a device with a width of 4.16 in the layout. If your schematic netlist specifies effective dimensions (this is not typical) and the corresponding device is defined with a width of 4.0, you would have to use a large tolerance parameter to get the devices to pass the parameter value test.

It is much better to adjust the dimensions of devices with accuracy, then use a small tolerance which allows you to find real errors.

Example: ***.LAYMODEL SHRINK_NMOS NMOS WOFFSET= -.16 WTLR=.0001**

[20] The LOFFSET and WOFFSET parameters are valid for JFET device models only when length and width are extracted from the layout rather than area.

See the BENDS_CR option (covered next) to correct dimensions of bent devices.

If the layout device mentioned above with a width of 4.16 uses this device model, the length would be adjusted to:

4.16 - .16 = 4.0

When the LVS compares the value of this device to the schematic device with a length of 4.0, the device values will match, even with such a small tolerance.

Example: ***.LAYMODEL RESISTOR RES LOFFSET= .1 WOFFSET= -.2**
***+ OHMS_PER_SQUARE = 5**

When this model is used for resistors in your layout, .1 will be added to each extracted resistor length, and .2 will be subtracted from each extracted resistor width. This will be done before the resistance of the device is calculated using the value of OHMS_PER_SQUARE.

Be careful to supply the offset values in the same terms as the layout units. Most layouts assume microns as the user unit. If you write the device model with a 'U' suffix:

***.LAYMODEL SHRINK_NMOS NMOS LOFFSET= -.16U**

The LVS will interpret this statement as:

***.LAYMODEL SHRINK_NMOS NMOS LOFFSET= -.16e^{-6}**

Then only .00000016 microns will be subtracted from each layout length dimension.

JFET[21]
MOSFET
RESISTOR

Figure 81: Valid device categories for BENDS_CR.

The optional [BENDS_CR = *correction_factor*] parameter is used to modify the way the LVS will calculate the value of non-rectangular devices for which the NLE extracts a length and width.

When a device is rectangular, the NLE calculates the width as the length of the end sides and the length as area/width. The length will then be equal to the length of the centerline of the rectangle from end-side to end-side.

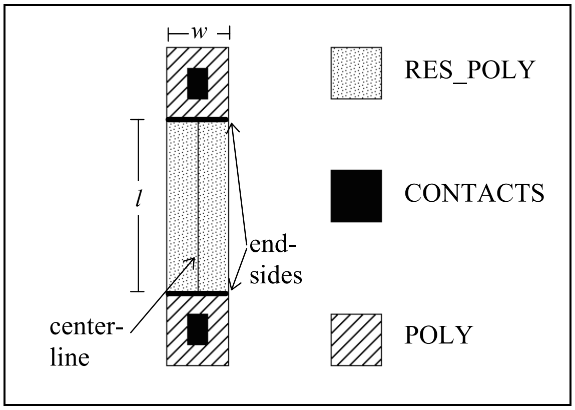


Figure 82: Rectangular resistor.

To see more details on NLE device dimension recognition, see page 117.

When the device is not rectangular, the length is still calculated as area/width, which usually means the length is the centerline length, however, the centerline is not a straight line. (See Figure 84 on the next page for an example.)

If you do not consider the length of the centerline to be the length of a bent device, add the BENDS_CR parameter to the device model for these devices.

The correction is based on the number of bends. For resistors, the correction is made to the length of the resistor. For transistors the correction is made to the width of the device.

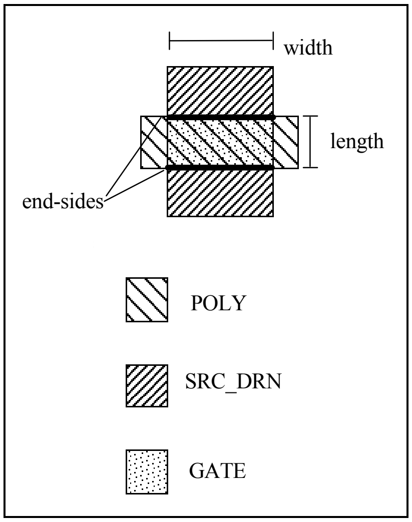


Figure 83: Rectangular transistor.

The dimensions of a device with the

[21] The BENDS_CR parameter is valid for JFET device models only when length and width are extracted from the layout rather than area.

BENDS_CR parameter defined in the device model will be:

For resistors:
***Length* = *Center_line_length* - (*Num_bends* ∗ (*Correction_factor* ∗ *width*))**

For transistors:
***Width* = *Center_line_width* - (*Num_bends* ∗ (*Correction_factor* ∗ *length*))**

The number of bends is calculated from the number of vertices as follows:

$$\frac{\textbf{Number of vertices}}{\textbf{2}} - \textbf{2}$$

A typical value for the *correction_factor* for bent resistors is .5 or 50% of the width. This value is based on manhattan layouts where all bends are 90°. If the bends of your devices are not at 90°, use an appropriate value for *correction_factor.*

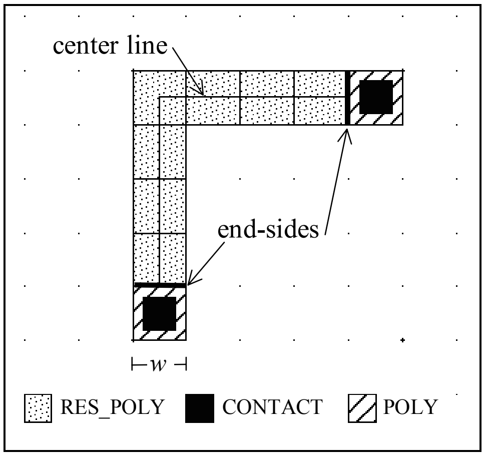


Figure 84: Non-rectangular resistor.

As it is drawn, the resistor shown in Figure 84 contains 7 squares. Most designers would consider this equivalent to a unbent resistor with less than 7 squares. The square at the bend adds less resistance to the device than the other squares. We want to modify the stored value of this device by subtracting a small correction factor. The number of bends is 1, the width is 1, and the length of the centerline is 7. If the following layout device model is used for this device:

Example: ***.LAYMODEL POLY_RES RES BENDS_CR=.5**

the length of the resistor above will be corrected as follows before computing the resistance and comparing it to the value in the other netlist:

Length = 7 - (1 ∗ (.5 ∗ 1)) = 6.5

These corrections are made **after** any corrections indicated by the LOFFSET or WOFFSET device modifier parameters.

If the device model for the POLY_RES resistor in the example above is:

Example:

```
*.LAYMODEL  POLY_RES  RES
*+  LOFFSET=1.2   WOFFSET=0.2   BENDS_CR=0.5
*+  OHMS_PER_SQUARE = 1000     R_CONTACT = 10
```

The LOFFSET and WOFFSET processing is performed first. The new length and width will be:

```
Length =  7 + 1.2      = 8.2
Width =   1 + 0.2      = 1.2
```

Then the bends correction would be made:

Length = 8.2 - (1 ∗ (.5 ∗ 1.2)) = 7.6

Finally the value would be calculated using the model parameters for OHMS-_PER_SQUARE and R_CONTACT:

```
Value = R_CONTACT + OHMS_PER_SQUARE ∗ (Length/Width)
Value = 10 +     1000 ∗  (7.6/1.2)
Value = 10 +     1000 ∗  6.333      =   6343 = 6.343 KΩ
```

Layout Restrictions

The NLE categorizes device layouts into three classes:

Boxes: Rectangles with all sides either vertical or horizontal,

Manhattan: Polygons with horizontal and vertical sides,

and

Rest: Devices with at least one side that is not horizontal or vertical.

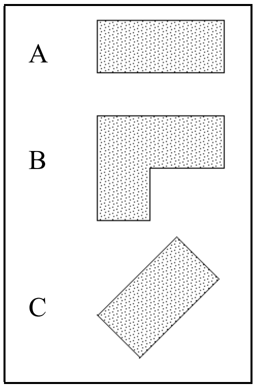


Figure 85: Device layouts: A) Box, B) Manhattan, C) Rest

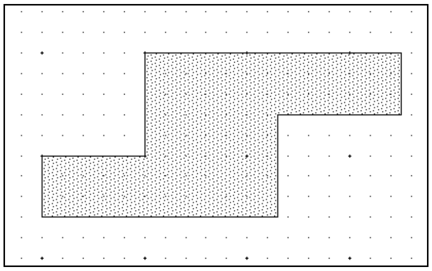


Figure 86: Manhattan device with varying width.

The NLE should be able to correctly recognize the dimensions of boxes and manhattan geometry. However, some types of manhattan geometry may result in the NLE reporting incorrect dimensions. The device with the manhattan layout shown in Figure 86 will probably have incorrect dimensions since the width is not constant.

Devices that are not boxes or manhattan layouts are more likely to have incorrect dimensions in the layout netlist. If you wish to prevent the LVS from comparing the dimensions the NLE has recognized for devices in certain classes, you can add layout restriction parameters to your *.LAYMODEL statement.

You may want to do this to prevent incorrect dimensions from being compared to the values in the other netlist. If the dimensions in the layout netlist for a non-manhattan device are incorrect, but they just happen to be within the error tolerance of the device in the other netlist, you will not get a parameter error message for the device. You may never realize that the device is the incorrect size.

The three layout restriction parameters are [BOX_GM = (YES | NO)], [MANHATTAN_GM= (YES | NO)], and [REST_GM = (YES | NO)]. The defaults for all three parameters are "YES". We do not recommend that you ever use the "BOX_GM=NO" parameter, since the NLE can always correctly recognize the dimensions of this class of devices. The "MANHATTAN_GM = NO" parameter should also not be used unless you have seen a problem with your manhattan devices.

Many of the device layouts in the third class will have their dimensions calculated correctly, however the values of these types of devices should be looked at carefully in your initial LVS runs. If the NLE is not extracting the device dimensions correctly, add the "REST_GM = NO" parameter to the corresponding layout netlist device model. The LVS will then use a value of '0' for the device and it will never pass a parameter error check.

For example, you have many resistors in your layout, and most of them are either boxes or manhattan layouts, but a few use non-manhattan geometry. You see in an initial LVS run that these non-manhattan devices have incorrect dimensions extracted for them by the NLE. You should add the "REST_GM = NO" parameter to your resistor device model. Only the non-manhattan devices will have their dimensions ignored by the LVS. All other resistors will still have their calculated values compared to values in the other netlist. The non-manhattan resistors will have error messages listed in the parameter error summary report.

Default value parameters (page 243) will **not** be used when devices have had their values ignored through the use of the layout restriction parameters. These devices can be assigned values using the label method described on page 346.

Device Characteristics

If you want the LVS to calculate resistance or capacitance values from device measurements made by the NLE, you must supply the appropriate device characteristics parameters. For capacitor device models, you must specify both C_PERIMETER and C_AREA. Similarly, resistor models should specify

OHMS_PER_SQUARE and optionally R_CONTACT or R2_CONTACT and R2_WIDTH to calculate resistance values. See the examples on page 343.

Default Value Parameters

See **Default Value Parameters** in **Schematic Netlists** on page 209 for more details on these parameters.

The default value parameters will not override values calculated by the NLE circuit extractor. They are useful when the NLE cannot calculate the value of a device.

For example, the NLE does not currently calculate a device value for inductors. You can however have the NLE differentiate between different inductors by model names. If these different inductor models have different VALUE parameters on their *.LAYMODEL statements, these values will be used as the values of the devices, and the values can then be verified against the other netlist.

Example:

```
*.LAYMODEL  IND_90  IND  VALUE=90
*.LAYMODEL  IND_40  IND  VALUE=40
```

See page 211 for examples of using an M parameter in a schematic device model.

The M=*multiplier* parameter is used differently in a layout netlist model than in a schematic netlist model. In a schematic netlist model, you would use an M parameter to simulate the number of devices in parallel. In the layout device models, M parameters are used to account for the way area is defined for Bipolar, JFET, GaAsFET, and Diode devices in schematic netlists.

Another way to assign a device value in the layout is to add a label on the device. See page 346.

When you define the value of a Bipolar, JFET GaAsFET or Diode device in a schematic netlist, you are really defining an area multiplier rather than an area. The value supplied on the device statement is multiplied by an area factor supplied in the .MODEL statement for the device. The LVS ignores these .MODEL statements, so the area factor must be accounted for in some manner.

In the layout netlist, the value of one of these devices is the actual area. If you compare the multiplier in the schematic netlist devices to the actual area in the layout netlist without accounting for this discrepancy, you will get parameter value mismatches for every device.

One way to compare the values of these types of devices is to compare the area multipliers rather than the area. This will allow the layout netlist to be consistent with the schematic netlist. This is done by adding the M=*multiplier* parameter to the device model in the layout netlist.

When the M=*multiplier* parameter is used in a *.LAYMODEL device model, the device area used for comparison will be:

device_area_compared = actual_area * *multiplier*

Set *multiplier* equal to:

$$\frac{1}{\text{AREA\_FACTOR\_FROM\_SCHEMATIC\_.MODEL\_STMT}}$$

For example, if you are using JFET devices as shown in the following schematic netlist fragment:

.MODEL JMODEL NJF AREA=20

J1 1 2 3 JMODEL 2

The multiplier you need to use to scale the actual area in the layout netlist is:

$$\frac{1}{20} = 0.05$$

To account for the area scaling in the layout netlist device model, add the M=*multiplier* to the *.LAYMODEL statement as shown below:

Example: ***.LAYMODEL JMODEL NJF M=0.05**

When the above model is used, JFET devices in the layout netlist using the JMODEL model will have their device areas multiplied by 0.05. If the layout contains a device with an actual area of 40, the value of the device for comparison or spice netlist generation will be:

40 * 0.05 = 2

Now the schematic netlist device will match without a parameter error. (Do not add the AREA=*area* parameter to the schematic *.SCHMODEL device model.)

If you generate a spice netlist for simulation from the layout data with the GENERATE_SPICE_NETLIST_FROM_THE_EXTRACTOR_OUTPUT = YES control file option, the value of the J1 device will be stored as 2 which will be consistent with the .MODEL simulation model.

Control File Override Parameters

These parameters are used to allow specific models for certain devices to be treated differently than other models. For instance, for some models of PNP devices, you want to allow series merges, but for others you need to avoid allowing any device transformations.

The use of control file override parameters in *.LAYMODEL statements is exactly the same as their use in *.SCHMODEL statements. Refer to the examples and description starting on page 212.

Preparing the Model File

For an example layout netlist model file, see page 23.

You must create a *.LAYMODEL statement for each unique device type recognized by the NLE circuit extractor. These statements are usually grouped together in an ASCII model file.

Once you have the model file created, you usually combine it with the binary layout netlist created by the NLE circuit extractor in a third file. This third file is the layout netlist file you refer to on the LVS command line. The model file is added to the layout netlist file with an .INCLUDE command. The binary layout netlist is added to the file with the *.LAYOUT command.

The syntax of the *.LAYOUT statement is as follows:

The name of the binary layout netlist can be overridden with the /L option on the LVS command line.

.LAYOUT [*path/*]*file_name

where *file_name* is the name of the binary layout netlist file generated by the NLE circuit extractor. If the binary file is not in the same directory as the rest of the layout netlist, you can supply the directory path. If no file extension is provided in *file_name*, the extension .EXT is assumed. The *.LAYOUT statement is used exactly as you would an .INCLUDE statement.

The model file should be read by the LVS before the binary .EXT file. All models must be defined in the model file before they are referred to in the binary layout netlist. The LVS will report an error if a model is not defined before it is referenced.

Example:

```
*Layout netlist for LVS check
.INCLUDE LAYMODEL.NET
*.LAYOUT CELL.EXT
.END
```

Commands Supported in the Layout Netlist

Command	Use	Refer to page(s)
*.DEVLABEL	Override device labels	361
.END	Mark end of netlist	250
.ENDS	Subcircuit models	331
*.GROUND_NET	Define ground nets	221
.INC or .INCLUDE	Include another netlist file	250
*.FORMAT	Subcircuit models	331
*.LAYMODEL	Device models	229
*.LAYOUT	Include binary layout netlist from NLE	246
*.NETLABEL	Override net labels	361
*.NOCOLLAPSE	Prevent device collapse	222
.PARAM	Allow parameter substitution in models	248
*.PINS	Prevent net disappearance and define pad devices	222 and 335
*.POWER_NET	Define power nets	221
.SUBCKT	Subcircuit models	331
*.VIRTUAL	Virtually connect nets	223

Figure 87: Supported commands in layout netlist

Commands which are valid in the ASCII portion of the layout netlist are listed in Figure 87. Most of these commands were already covered in the schematic netlist section. Their use in the layout netlist is exactly the same as their use in the schematic netlist. The remaining commands are described in advanced subjects later on in the manual. Refer to the table for the page numbers where they are described.

All commands specific to the LVS have a '*' prefix so that these commands will be ignored by other programs which read netlists.

You can use .PARAM commands in the model file of the layout netlist to allow parameter substitution in the device models. This can make the model file easier to write and easier to update for future uses. Remember that the .PARAM command allows substitution of numbers, not strings.

Example:

```
.PARAM  POLY_SHRINK = -0.16

*.LAYMODEL  POLY_RES  RES     WOFFSET =  {POLY_SHRINK}
*.LAYMODEL  NMOS      NMOS    LOFFSET =  {POLY_SHRINK}
```

The parameters must be used according to the syntax of a schematic netlist language (SPICE, PSPICE, HSPICE, or CDL). If you are performing an LVS comparison, the syntax used is indicated by the control file option SECOND-_SCHEMATIC_FILE_FORMAT, or by the *.FORMAT command in the layout netlist file. (See page 333.)

Review of Node Labels

To see how to label nodes in the layout netlist without reprocessing the layout through the NLE circuit extractor, see page 361.

The use of node labels in the layout is described in detail in the NLE section of this manual. See page 138 for complete details. Here, we will review a few points that affect the LVS comparison.

No layout net or device labels are required for the LVS to match the two netlists. However, use of named nodes in the layout can make mismatches easier to diagnose and speed verification of symmetric circuits. Nets and devices in the layout can be labeled through the use of text components in the ICED32™ cell. If the text components are not on design layers, the NLE rules file moves them to the design layers to label the nodes.

We recommend that all node labels be placed in the top level cell in the layout. All node labels in nested cells will be ignored by the NLE unless you specify that they are global nodes using the ':' or '::' suffix on the label. When the control file option RECOGNIZE_GLOBAL_TEXT_IN_SUBCELLS=NO is used, even node labels with a ':' suffix will be ignored if they are located in a nested cell. Nodes using the '::' suffix in subcells will always be processed.

Details on the node correspondence file are provided in Using a Node Correspondence File on page 355.

You can use labels in the top level cell to label nets or devices local to nested cells. Simply place a text component in the top level cell over a component in a nested cell. If you are using a node correspondence file, the name of the local node can be whatever you like. Simply use the correspondence file to associate the label in the layout with the flattened schematic node name. If you are not using a node correspondence file, label the node with the name in the flattened schematic netlist. Details on how these names are generated from a hierarchical netlist are covered in **Node Names** on page 226.

The .POK file which the NLE uses to create the layout netlist cannot contain labels with more than fifteen characters. Create all node labels in the layout using fifteen or less characters. Longer node labels will be ignored.

Node labels can be case sensitive. If the ICED32™ layout editor is configured to allow lower case text (through the TEXT or USE commands), lower case characters in node labels will not be translated to upper case for the LVS unless the control file option FORCE_ALL_LAYOUT_LABELS_TO_UPPER_CASE = YES is used.

Labels on device id layers can be used to assign a value to a device for which the NLE is unable to determine the dimensions. The use of these labels is described on page 346.

Labels can be used to control how the LVS performs the comparison. Node labels can be used to create points of forced correspondence between the two netlists. The relevant control file options for node equivalence are described beginning on page 261.

Node labels can also affect how the layout netlist is modified by logic transformations. The control file defines special characters which will prevent device collapses when used in a node label. Another special character can be used to indicate virtual connections in the layout. See **Advanced Uses of Node Labels** on page 349 for more information.

Node labels can be added or modified without re-executing the NLE circuit extractor through the use of node label overrides. See **Using Node Label Overrides** on page 361 to learn more about this feature.

Summary of How to Prepare a Layout Netlist for the LVS

The entire sequence of events to prepare a layout netlist for LVS verification is:

1. Within ICED32™, use the DRC command to produce the input file for the NLE program.

2. Generate a flat layout netlist using the NLE circuit extractor. The output binary file will have an .EXT extension. For example, you can name the binary layout netlist CELL.EXT.

3. Prepare a model file containing the *.LAYMODEL statements as explained above. This file can be named LAYMODEL.NET.

4. Create a new file, with a name similar to LVS_LAY.NET. This file should look similar to:

The *.LAYOUT statement can be overridden with the /L option on the LVS command line.

```
*Layout netlist for LVS check
.INCLUDE LAYMODEL.NET
*.LAYOUT CELL.EXT
.END
```

Use the LVS_LAY.NET file for the layout netlist on the LVS command line. The .END statement is essential at the end of file.

The LVS Control File

The control file options allow you to direct the netlist-netlist comparison algorithms and manage the generation of various output files. An example control file, Q:\ICED\[22]SAMPLES\LVS\CONTROL.LVS, is provided with the installation. We suggest that you always copy this control file to your working directory and modify the copied file.

You should not delete any lines from the file. Do not change the order of the options in the file. Missing lines will result in errors reported to the screen and the netlist comparison will not be performed. Modify the defaults only where you need to. Blanks can be used to make the file more readable. Quotes can be used around string parameters, such as filenames, but they are not required.

See **Command Line Syntax** in **Running LVS** for details on the LVS command line.

Some options in the control file can be overridden. The LVS command line has parameters to override several options. These are indicated in the margin notes next to the appropriate options. Also, all options in the Individual Device Options section can be overridden for specific device models by using parameters on the *.SCHMODEL statements in a schematic netlist, or by parameters on the *.LAYMODEL statements in a layout netlist.

The LVS control file options are divided into six sections. These sections are indicated with comments in CONTROL.LVS with the titles below in the same order. Details on the options in each section follow on the page indicated.

The control file also has a few options at the end specifically for the LPE utility.

DIRECTORY PATH & FILE NAME EXTENSION	page 256
COMPARISON TYPE & FILE FORMAT	page 257
LVS RUNTIME OPTIONS	page 259
OPTIONAL OUTPUT FILES	page 271
OUTPUT FILES	page 281
INDIVIDUAL DEVICE OPTIONS	page 283

[22] Remember that Q:\ICED represents the drive letter and path where you have installed ICED32™.

To quickly find the page where a specific option is described, you can use the alphabetical list below.

DIRECTORY PATH & FILE NAME EXTENSION

OUTPUT_DIRECTORY_PATH = *dir_path*

The /O command line parameter can override this option.

This option is used to define the default directory for output files containing the results of the netlist-netlist comparison. The LVS can create the last directory in *dir_path*, if it does not already exist. If it does already exist, we recommend that this directory be empty. Old files will be overwritten without warning.

If *dir_path* begins with a '\' (or a drive letter followed by ":\"), the path is assumed to be defined from the root directory. When *dir_path* does not begin with a '\' or a drive letter, the path is defined relative to the current directory. If the LVS cannot find or create the directory, it reports an error and terminates the run.

Example: **OUTPUT_DIRECTORY_PATH = "results"**

The OUTPUT_DIRECTORY_PATH option above will store all files in a subdirectory of the current directory. If the current directory is C:\ICED\MYCHIP, the files will be stored in C:\ICED\MYCHIP\RESULTS.

Example: **OUTPUT_DIRECTORY_PATH = \results**

In this example *dir_path* begins with a '\', so the directory path is relative to the root directory. If the current drive is D:, the reports will be stored in D:\RESULTS, regardless of the current directory.

Note that the quotes are optional.

Example: **OUTPUT_DIRECTORY_PATH = .**

When you set *dir_path* to '.', all files will be stored in the current directory.

OUTPUT_FILE_NAME_EXTENSION_FOR_LVS = *file_ext*

This option allows you to provide a default file name extension for all output files. You do not need to provide an extension with any of the output file names supplied in other options in the control file. The *file_ext* parameter should be a valid DOS file name extension of one, two or three characters.

Example: **OUTPUT_FILE_NAME_EXTENSION_FOR_LVS = "lvs"**

This option cause the LVS to create all output files with the default extension ".lvs". This default extension can be overridden by specifying an extension with the file name parameter in any of the control file options that specify output files.

COMPARISON TYPE & FILE FORMAT

TYPE_OF_COMPARISON = (LVS | SVS | LVL)

The /C command line parameter will override this option.

Use this option to inform the LVS which type of netlists are to be compared. Exactly one of the following choices must be used:

LVS	Layout Vs Schematic
SVS	Schematic Vs Schematic
LVL	Layout Vs Layout

Example: **TYPE_OF_COMPARISON = LVS**

SCHEMATIC_FILE_FORMAT= (CDL | PSPICE | HSPICE | SPICE)

The /F or /F1 command line parameters will override this option

This option informs the LVS which schematic netlist syntax should be used by the input parser. This option is used for netlist1. (See the SECOND-_SCHEMATIC_FILE_FORMAT option to set the syntax of netlist2.)

Example: **SCHEMATIC_FILE_FORMAT= PSPICE**

TOP_LEVEL_SUBCKT_IN_SCHEMATIC_FILE = *subckt*

The /T or /T1 command line parameters will override this option.

Use this option to identify the name of the top level subcircuit in schematic netlist1. (Even though this option will be ignored if you are running an LVL comparison, you must supply some string.) If your schematic netlist has no top level subcircuit, see page 227 to learn how to prepare the schematic netlist.

If you provide a subcircuit name that exists in the schematic netlist, but is NOT the top level subcircuit, the LVS will use that subcircuit as the top-level circuit. In this case, subcircuits not referred to in the specified subcircuit will be ignored. If this subcircuit does not correspond to the circuit in the layout netlist, many false error messages will result. Be sure to specify this parameter carefully. You can use a nested subcircuit name if that is the only circuit you wish to verify and the other netlist is prepared accordingly. This allows you to avoid cutting and pasting a schematic netlist when you need to verify only one subcircuit.

Example: **TOP_LEVEL_SUBCKT_IN_SCHEMATIC_FILE = "OPAMP"**

If you specify a circuit name that does not exist on a .SUBCKT statement in the netlist, the LVS will warn you immediately and abort the run.

SECOND_SCHEMATIC_FILE_FORMAT=(CDL | PSPICE | HSPICE | SPICE)

and

TOP_LEVEL_SUBCKT_IN_SECOND_SCHEMATIC_FILE= *subckt*

The /F2 and /T2 command line parameters will override these options.

If you are executing a SVS (Schematic Vs. Schematic) comparison, these two options inform the LVS of the schematic file format and top level subcircuit name for the second schematic netlist.

If you are performing an LVL or LVS comparison using SUBCKT device models in a layout netlist, the SECOND_SCHEMATIC_FILE_FORMAT option can be used to define the spice syntax used to parse the SUBCKT device models in the second netlist. In a layout netlist, the *.FORMAT statement will override this option in the control file.

The TOP_LEVEL_SUBCKT_IN_SECOND_SCHEMATIC_FILE option will always be ignored for a LVL or LVS comparison.

LVS RUNTIME OPTIONS

TAKE_CARE_OF_LOGIC_EQUIVALENCES_WHILE_MATCHING ... ... = (YES|NO)

This option controls device transformations performed before the netlists are compared. It is intended primarily for CMOS digital design.

Example: **TAKE_CARE_OF_LOGIC_EQUIVALENCES_WHILE_MATCHING = YES**

See Pull-up and Pull-down Pseudo Devices on page 324 for examples.

Selecting YES for this option will allow the LVS to perform an additional series collapse of already collapsed pseudo devices. This will usually result in a single pseudo device for each pull-down or pull-up circuit. Collapsing pull-down and pull-up paths into pseudo devices will allow matching of device structures which are logically equivalent but topologically dissimilar.

Even when you do not want to allow dissimilar topologies in your two netlists, this option can be useful. A logic error in one pull-up or pull-down circuit may cause a large number of unmatched devices in other circuits. This option can be useful in locating logic errors in pull-up or pull-down circuits when the unmatched device list includes many devices which are correctly connected.

These control file options are described beginning on page 292.

To use this feature, the COLLAPSE_SERIES_LOGIC_*device*S and COLLAPSE_PARALLEL_LOGIC_*device*S options **must** also be set to YES (where *device* is the category of device for which you want to enable logic equivalence transformations).

USE_LOCAL_MATCHING = (YES | NO)

The LVS matching algorithm can use either a global matching algorithm or a faster local matching algorithm to find one to one correspondence between the two netlists. Using YES for this option will force the local matching algorithm to be used. This is the recommended algorithm in all cases.

FORCE_ALL_LAYOUT_LABELS_TO_UPPER_CASE = (YES | NO)

The NLE will store all node labels in the binary layout netlist exactly as they are typed in the ICED32™ cell. If the labels are in lower or mixed case they are stored that way in the file.

If you want to have all node labels translated to upper case before the comparison, set this option to YES. If you prefer to use the labels exactly as they are typed, set this option to NO.

RECOGNIZE_GLOBAL_TEXT_IN_SUBCELLS = (YES | NO)

The /G command line option will override this option.

Normally, all text labels in nested cells are ignored by the LVS. Nodes can only be labeled by text components in the top-level cell. You can override this behavior by using global text labels. There are two types of global text labels, those ending with a single colon, ':' and those ending with a double colon, '::'.

If you use single colon global text labels to label nodes in the layout and set this option to YES, these labels will be processed by the LVS even when they are located in a nested cell.

Node labels created with a double colon suffix will always be recognized by the LVS. They are not required to be in the top-level cell, and setting this option to NO will **not** stop them from being recognized.

LAYOUT_TEXT_MODE= (EQUIV | AUTO | SKIP)

The /M command line option will override this option.

This option controls how node names are used to make forced correspondences by the LVS. Use of node name correspondences is recommended for highly symmetric circuits, or for diagnosing large numbers of mismatches.

When EQUIV is used, the LVS will use the equivalences in the node correspondence file defined by the next option. The flags in the file will determine how the equivalences are used. (See page 357 for details.)

The AUTO option will associate all labeled nodes in the layout netlist with schematic nodes with identical names. When the AUTO mode is used, no node correspondence file is required.

The SKIP option will prevent node labels from being use to make forced points of correspondence between the two netlists. Layout labels using special characters, as described in **Advanced Uses of Node Labels** in **Running LVS**, will still be processed by the LVS.

When the AUTO or SKIP options are used, the node correspondence file is ignored.

The /E YES command line parameter can be used to terminate the LVS if errors are found in the correspondence file.

INPUT_FILE_OF_NAME_EQUIVALENCES = *input_equiv_file*

When the LAYOUT_TEXT_MODE option (see previous option) is set to EQUIV, this option provides the name of the node correspondence file to be used. If you have indicated the AUTO or SKIP mode, the file need not exist, however, do not delete this line from the control file.

Example: **LAYOUT_TEXT_MODE = EQUIV**
INPUT_FILE_OF_NAME_EQUIVALENCES = "projxeqv.lvs"

You can generate a preliminary node correspondence file with the GENERATE-_NAME-_EQUIVA-LENCES control file option.

Using these options in the control file will cause the file projxeqv.lvs to be read by the LVS. The node equivalences in the file will be used according to the options in the file and the following control file option.

See **Using a Node Correspondence File** on page 355 to learn more about this feature.

USE_EQUIVALENCES_FOR_INITIAL_MATCHING= (YES | NO)

When the LAYOUT_TEXT_MODE option is set to EQUIV or to AUTO, the node correspondences can be handled two ways. When USE_EQUIV-ALENCES_FOR_INITIAL_MATCHING is set to YES, the node corres-pondences will be made before the LVS proceeds with the rest of the circuit matching.

If you set this option to NO, the correspondences are made only when the LVS fails to find a match without using the correspondences. The LVS may fail to find a match when circuits have a great deal of symmetry.

We recommend that you use YES for this option only when your circuits are highly symmetric, or when you need to diagnose large numbers of mismatches where apparently correctly connected devices are listed as mismatches. When you use YES, and there are errors in the node correspondence file, the LVS may be unable to find any correspondence between the two netlists. However, if you use NO, the program will always be able to proceed with the match. If the circuits are not highly symmetric, there is little or no speed improvement when you use the USE_EQUIVALENCES_FOR_INITIAL_MATCHING=YES option.

SET_NO_PROGRESS_LIMIT = *num_passes*

See Symmetric Circuits for examples and more details about how the LVS breaks symmetry.

This option sets the minimum number of passes the LVS will execute before it begins breaking symmetry. When a circuit has a great deal of symmetry, and node labels in the layout are not used to provide forced points of correspondence between the netlists, the LVS can execute many passes without making any progress.

The *num_passes* parameter must be a positive integer. We do not recommend using a value less than 2, since the net graphs built by the LVS improve with each pass, and matches are less likely when you allow the LVS only one pass. Forcing the LVS to make arbitrary matches before the end of the second pass will probably result in false matches and many false error messages.

Example: **SET_NO_PROGRESS_LIMIT = 2**

Using this option will force the LVS to take one of the courses of action listed below if it cannot find a correspondence between the netlists in 2 passes. This is a typical value for *num_passes*, and the default in the sample control file provided with the LVS.

When the LVS has executed 2 passes without making progress in matching the netlists, it will take one of the following actions:

Nodes forced to be equivalent when these actions are taken will be listed in the forced matches report. See page 384.

1) If there are node equivalences in the node correspondence file that have not yet been used, the LVS will match one of these pairs of nodes. Then the LVS will continue attempting to match the netlists.

2) If no unused node equivalences are left, and the INTERACTIVE-_MODE option (covered below) is set to YES, the LVS will go into this mode and allow you to select one net from each list to make a node correspondence.

3) If the LVS cannot make progress with either of the previous options, then it will begin picking arbitrary nets from lists of potential matches to make a forced correspondence.

SET_NET_SIZE_LIMIT_WHEN_PRINTING_CONNECTIONS=*num_devs*

By default, when the LVS fails to match a net, the unmatched devices report will list all devices connected to the net. If the unmatched net is GND or VDD, the number of devices could be very large. You can control the maximum number of devices to be listed by setting this option.

The *num_devs* parameter must be a positive integer.

Example: **SET_NET_SIZE_LIMIT_WHEN_PRINTING_CONNECTIONS = 10**

INTERACTIVE_MODE = (YES | NO)

When the LVS has difficulty in matching devices and nets because of symmetry in the circuits, it may arbitrarily pick a net or a device from each circuit and match them. An arbitrary match is rarely a good starting point. You can avoid arbitrary matches by using YES for this option, or by using a node correspondence file.

The INTERACTIVE_MODE=YES option is recommended only when you prefer not to use a node correspondence file. See the LAYOUT_TEXT_MODE, INPUT_FILE_OF_NAME_EQUIVALENCES, and USE_LABELS_FOR-_INITIAL_MATCHING options above for details.

In interactive mode, the LVS displays on the screen possible options from each circuit netlist. It displays five options at a time. You can ask for more options by typing a <Y> or <N> (type the single character) to the prompt on the screen.

The interactive mode display is similar to:

```
            NO#    SCHEMATIC   LAYOUT
            0
            1
            .
            .
            .
Some more choices[y] :
```

If you simply type <Enter>, more choices will be displayed. If you type a <N> at the "Some more choices" prompt, you will see the prompt:

```
Schematic choice :
Layout choice:
```

You should type a node number under the "NO#" heading for each netlist to indicate that they represent the same node in each circuit.

TREAT_FIRST_LINE_IN_SPICE_NETLIST_AS_COMMENT_LINE = ... (YES | NO)

The original spice syntax defines the first line of a spice file as a comment whether it was prefixed by a "*" or not. If you follow this syntax, set this option to YES. In this case the first line of the schematic netlist will be ignored.

If you prefer that the first line is not ignored by the LVS, set this option to NO.

ENABLE_VIRTUAL_CONNECTIONS = (YES | NO)

The /V command line option will override this option.

Set this option to YES if you want to allow virtual connections of nets in the layout netlist. Virtual connections can be useful in verifying designs which are not yet completely wired, or when you verify a subcircuit which uses connections in a higher level cell.

When this option is set to NO, the LVS will report errors for nets in the layout which have the same name, but which are not electrically connected. Using virtual connections in the layout allows you to override this behavior and force the LVS to consider separate nets as a single node.

When node numbers for a virtually connected net are listed in the reports, the syntax used will be "*node_number*[V,*n*]", where *n* indicates the number of nets which have been virtually connected.

One way to create virtual connections is to label the nets in the layout using the special character suffix defined with the following option. For other methods, see **Assigning Virtual Connections** on page 358.

Be sure to disable virtual connections in the final LVS runs on your chip or real open circuits may not be found. All virtual connections will be disabled when this control file option is set to NO.

SPECIAL_CHARACTER_FOR_VIRTUAL_CONNECTIONS = *virt_char*

To see a list of different methods for assigning virtual connections, see page 358.

This option is used to specify a character that can be used in layout net labels to indicate nets that are virtually connected. (To allow this feature to be used, the ENABLE_VIRTUAL_CONNECTIONS option must also be set to YES. See the previous option.)

To assign virtual connections, set *virt_char* using this option to some unique character. Then use this character as a suffix in net labels on nets to be virtually connected. The character will not be stripped from the netname. (For example, the LVS will not automatically virtually connect the nodes "VDD" and "VDD:". You can virtually connect these two nodes in the node correspondence file or with the *.VIRTUAL command in the netlist.)

Example: **ENABLE_VIRTUAL_CONNECTIONS = YES**
SPECIAL_CHARACTER_FOR_VIRTUAL_CONNECTIONS = :

When *virt_char* is ':' (the default in the sample control file), you can have several separate nets labeled "VDD:" and the LVS will consider all of them to be electrically connected. All nets labeled "GND:" will also be considered to be virtually connected. However "GND:" and "VDD:" nets will not be considered virtually connected to each other.

If the colon (':') is not a valid character in your schematic netlist language, you can set *virt_char* to a different character. The *virt_char* character (and the characters specified in the next several options) should all be single characters different from each other.

ENABLE_NO_COLLAPSE_OF_DEVICES = (YES | NO)

If the control file options for a specific device type allow devices to be collapsed (e.g. series merge or parallel device collapse), you may wish to identify certain nets in a way that prevents any devices attached to them from being collapsed. Set this option set to YES and follow the instructions provided in the next option. When you add appropriate labels in the layout to these nets, all devices connected to them will be prevented from disappearing due to device collapses. This can be useful in verifying critical nets such as clocks.

When this option is set to NO, devices attached to nets labeled with the character defined by the next option will **not** be prevented from collapsing.

SPECIAL_CHARACTER_FOR_NO_COLLAPSE_OF_DEVICES... ..._CONNECTED_TO_A_NET = *dev_char*

This option is used to specify a character which can be used in layout net labels to prevent the collapse of any devices connected to those nets. The previous option must also be set to YES to enable this feature.

To see examples of how this special character is used, see page 352.

Devices can be collapsed into series, parallel, or pseudo circuits if options enable such collapses (refer to the options starting on page 287). If you want to prevent these collapses for all devices attached to a specific net:

> use the ENABLE_NO_COLLAPSE_OF_DEVICES = YES option,
>
> set *dev_char* to a unique character with this option,
>
> and
>
> label these nets with this character used as a prefix in the highest level cell in the layout.

The *dev_char* parameter should be a single character different from the other special characters.

Example:

ENABLE_NO_COLLAPSE_OF_DEVICES = YES
SPECIAL_CHARACTER_FOR_NO_COLLAPSE_OF_DEVICES...
...CONNECTED_TO_A_NET = $

You can also prevent devices from collapsing by using the *.NOCOLLAPSE command in the netlist.

In this example, *dev_char* is set to '$'. This is the character used in the sample control file supplied with the LVS installation.

ENABLE_NO_COLLAPSE_OF_A_NET = (YES | NO)

To see examples which demonstrate how this feature is different from ENABLE_NO-_COLLAPSE-_OF-_DEVICES, see page 352.

Occasionally, you may wish to prevent certain nets from disappearing from the layout netlist due to device collapses. This is different from preventing all devices attached to a net from collapsing. If you wish to preserve a net while allowing devices attached to it to take part in collapses which do not result in the disappearance of the net, use this feature. This can be useful in insuring that I/O pins which connect to circuitry at a higher level are preserved in the layout netlist.

When this option is set to YES, devices connected to nets labeled with the character defined by the next option can be used in series, parallel, or pseudo circuit collapses if the corresponding device options (see page 287) enable such collapses. However, if a specific collapse will result in the net from disappearing from the netlist, the collapse will be prevented.

When the above option is set to NO, devices attached to nets labeled with the character defined by the next option will **not** be prevented from collapsing.

SPECIAL_CHARACTER_FOR_NO_COLLAPSE_OF_A_NET = *net_char*

A different method of preventing nets from disappearing is using the *.PINS command in the netlist.

The character specified with this option is used to prevent a net from disappearing from the layout netlist due to device collapses. Use this character as the prefix for node labels in the layout.

The *net_char* parameter should be a single character different from the other special characters.

Example:

ENABLE_NO_COLLAPSE_OF_A_NET = YES
SPECIAL_CHARACTER_FOR_NO_COLLAPSE_OF_A_NET = #

These lines set *net_char* to '#', and enable the special processing of nets prefixed with that character.

SPECIAL_CHARACTER_FOR_PRINTING_SERIES_MERGES=*ser_char*

This character will be used by the LVS when naming devices created from series merges. (Series merges are enabled through the use of the MERGE_SERIES-_*device*S options in the control file. See page 287.) The name of a device created by a series merge will be the names of the original devices separated by *ser_char*.

Example: **SPECIAL_CHARACTER_FOR_PRINTING_SERIES_MERGES = @**

In the sample control file supplied with the LVS installation, *ser_char* is set to '@'. When this option is used, and a series device is created by the LVS, the name of the merged device will be similar to DEV1@DEV2@DEV3..., where DEV1, DEV2, and DEV3 are the names of the original devices in the netlist.

SPECIAL_CHARACTER_FOR_PRINTING_PARALLEL_MERGES=*par_char*

This character will be used by the LVS when naming devices created from parallel merges. (Parallel merges are enabled through the use of the MERGE-_PARALLEL_*device*S options in the control file. See page 289.) The name of a device created by a parallel merge will be the names of the original devices separated by *par_char*.

Example: **SPECIAL_CHARACTER_FOR_PRINTING_PARALLEL_MERGES = &**

In the sample control file supplied with the LVS installation, *par_char* is set to '&'. When this option is used, and a parallel device is created by the LVS, the name of the collapsed device will be similar to DEV1&DEV2&DEV3..., where DEV1, DEV2, and DEV3 are the names of the original devices in the netlist.

SPECIAL_CHARACTER_FOR_PRINTING_DEVICES_IN_A_DEVICE...
..._CELL_INSTANCE = *dev_cell_char*

There are some types of physical devices which cannot be represented as a single device in a schematic netlist. Multiple emitter NPN devices are an example. These devices can be expanded in the layout netlist. Expanded devices will be named using this character. See **Multiple Emitter or Collector Devices** in **Device Transformations** for an example.

Example: **SPECIAL_CHARACTER_FOR_PRINTING_DEVICES_IN_A_DEVICE...**
..._CELL_INSTANCE = !

The *dev_cell_char* is set to '!' in the sample control file supplied with the LVS installation. You can set *dev_cell_char* to the character required by your schematic netlist language.

OPTIONAL OUTPUT FILES

See **LVS Output Files** for more details on all reports generated by the LVS.

The LVS generates several different reports. The following options control which reports are generated and what files the data is stored in. All files will be stored in the directory indicated in the OUTPUT_DIRECTORY_PATH option described on page 256. Do not include a directory path with any of the file name parameters of the following options.

GENERATE_NAME_EQUIVALENCES = (YES | NO)

See **Using a Node Correspon-dence File** in **Running LVS**.

The LVS can generate a node correspondence file listing all node labels in the layout netlist and the equivalent node names in the other netlist. The file can be used as an input node correspondence file in subsequent runs of the LVS. The name of the output file should be provided in the next option.

Example: **GENERATE_NAME_EQUIVALENCES = NO**

Using this option in the control file will prevent the creation of a node correspondence file based on the actual netlist data.

When you set this option to YES, nodes which are labeled in the layout, but which have not been matched to a node in the other netlist, will be indicated with an '*'. The line in the generated file will look similar to:

```
*     =     NETABC
```

You should replace the '*' with the appropriate node name in the other netlist before using this file as an input node correspondence file with the control file option INPUT_FILE_OF_NAME_EQUIVALENCES.

OUTPUT_FILE_OF_NAME_EQUIVALENCES = *equiv_file*

When you use the GENERATE_NAME_EQUIVALENCES = YES option, this parameter is used as the file name for the node correspondence file the LVS will generate. If you do not provide a file extension, the default file extension set by the OUTPUT_FILE_NAME_EXTENSION_FOR_LVS option will be used.

Example: **GENERATE_NAME_EQUIVALENCES = YES**
OUTPUT_FILE_OF_NAME_EQUIVALENCES = "EQUIVOUT.TXT"

Using these options in the control file will cause the LVS to create a node correspondence file listing all nodes labeled in the layout netlist and the corresponding node names in the other netlist. This file (like all reports generated by the LVS) will be created in the directory specified by the OUTPUT_DIRECTORY_PATH option. The file name will be EQUIVOUT.TXT.

PRINT_NET_LABELS_IN_A_SEPARATE_FILE = (YES | NO)

Setting this option to YES will cause the LVS to create a file that will list all nodes labeled in the layout netlist. The node number of each labeled net will be indicated. This file can be used as an input file to your next run of the LVS to override node labels found in the layout.

See Using Node Label Overrides on page 361 for more details and an example.

If you need to change a node labeled in error in the layout, or add new labels, you can edit this file instead of editing the layout and reprocessing it through the NLE circuit extractor. This can save time.

You add this file to the layout netlist with an .INCLUDE statement. Changes made with this file are processed as though the changes were made to the layout even though you have not altered your layout in any way.

Node numbers can change when you edit the layout and re-execute the NLE. Therefore, the file generated by this option may not be valid after you recreate the binary layout netlist with the NLE.

The name of the output file is determined by the next option. You can include unlabeled nets in the list with the PRINT_ALL_UNLABELED_NETS-_WHOSE_DEGREE_GREATER_THAN option.

OUTPUT_FILE_OF_NET_LABELS = *label_file*

If you have used the PRINT_NET_LABELS_IN_A_SEPARATE_FILE = YES option (described above), the data will be stored in the file named with this parameter. Use a valid DOS filename. Do not include the path. The file extension is optional. You can surround *label_file* with quotes for readability. They are not required.

Example:

PRINT_NET_LABELS_IN_A_SEPARATE_FILE = YES
OUTPUT_FILE_OF_NET_LABELS = "LABELOVR.NET"

When these options are used in the control file, all nodes with labels in the layout netlist will be listed in the file LABELOVR.NET in the directory specified by the OUTPUT_DIRECTORY_PATH option.

PRINT_ALL_UNLABELED_NETS_WHOSE_DEGREE_GREATER... ... _THAN = *net_degree*

If you want to include unlabeled nets in the list generated by the PRINT_NET-_LABELS_IN_A_SEPARATE_FILE = YES option, set *net_degree* to a number low enough to include the nets you are interested in.

PRINT_NETS_WITH_ZERO_AND_ONE_CONNECTIONS = (YES | NO)

This option enables or suppresses the generation of a report which lists nets not connected to any device (degree zero nets) and nets connected to only one device (i.e. unterminated nets).

To insure that certain nodes in the layout are unconnected, see the UNCON-NECTED NLE rule.

A zero degree net in the schematic netlist arises when nets listed in the I/O list of a subcircuit are never referred to in a device statement. There are never any nets of zero degree in the layout netlist. Even when stray pieces of conductive material are not connected to any devices, they will not be listed as nets of zero degree.

If you set this option to YES, the lists will be stored in the file name supplied in the next option.

OUTPUT_FILE_OF_NETS_WITH_ZERO_AND_ONE_CONNECTIONS ... ... = *unterm_net_file*

Remember that all files are stored in the directory defined by the **OUTPUT-_DIRECTORY_PATH** option.

All nets which are not connected to a device, or which connect to only one device will be listed in a file with the name *unterm_net_file*. This parameter should be a valid DOS filename. Do not include the path. The file extension is optional. If you do not supply a file extension, the default extension set by the OUTPUT_FILE_NAME_EXTENSION_FOR_LVS option will be used.

This file will be created only if the PRINT_NETS_WITH_ZERO_AND_ONE-_CONNECTIONS option (described above) is set to YES.

Example:

PRINT_NETS_WITH_ZERO_AND_ONE_CONNECTIONS = YES
OUTPUT_FILE_OF_NETS_WITH_ZERO_AND_ONE_CONNECTIONS=...
..."UNTERM"

If you have used the option OUTPUT_FILE_NAME_EXTENSION_FOR-_LVS=LVS, all nets in the schematic netlist which do not connect to any devices will be listed in a file with the name UNTERM.LVS. Nets which connect to only one device in either netlist will also be listed.

PRINT_COLLAPSED_DEVICES_IN_A_SEPARATE_FILE = (YES | NO)

Devices formed from a simple merge will not be listed in this report.

See **Device Trans-formations** for more details on logic collapses.

The LVS performs various transformations on the circuit netlists depending on the user options. In the process of transformation, the LVS can collapse a group of devices of the same type and model into a pseudo device. These pseudo devices are intended to allow for gate terminal input swapping and logic equivalence while matching two circuit netlists.

If you use YES for this option, the LVS will create a file with the list of devices grouped into each pseudo device. If you use NO, no report will be generated, however, these devices will still be listed in the matched device report (or the unmatched device report if they do not match with circuits in the other netlist).

OUTPUT_FILE_OF_COLLAPSED_DEVICES = *collapse_file*

If you have used the PRINT_COLLAPSED_DEVICES_IN_A_SEPARATE-_FILE = YES option (described above), the data will be stored in the file named with this parameter. Use a valid DOS filename. Do not include the path. The file extension is optional.

Example:

PRINT_COLLAPSED_DEVICES_IN_A_SEPARATE_FILE = YES
OUTPUT_FILE_OF_COLLAPSED_DEVICES = "COLLAPSE.TXT"

When these options are used in the control file, the list of devices collapsed to create each pseudo device will be listed in the file COLLAPSE.TXT in the directory specified by the OUTPUT_DIRECTORY_PATH option.

PRINT_SYMMETRIC_MATCHES_IN_A_SEPARATE_FILE= (YES | NO)

This option can enable or suppress the report of symmetric (forced) matches.

See page 365 for more details on symmetric circuits.

If your circuit has large numbers of devices matched incorrectly or unmatched incorrectly, it may be that the LVS was forced to match two nodes from a list of nodes with identical node properties, and that it matched a pair of nodes incorrectly. The most frequent cause of this problem is symmetric circuits.

This option enables the report of a list of pairs of nodes which the LVS was forced to match from candidates of possible matches. When you have large numbers of devices matched incorrectly, look at this report to see which pair of nodes was forced to match in error. You can then chose a correct pair of nodes to be matched by the LVS. Label the node in the layout and associate this label to the equivalent node in the schematic netlist using the node correspondence file.

OUTPUT_FILE_OF_SYMMETRIC_MATCHES = *sym_file*

If the PRINT_SYMMETRIC_MATCHES_IN_A_SEPARATE_FILE option is set to YES, the output will be created in the file indicated with this parameter. *sym_file* must be a valid DOS filename. Do not include the path.

GENERATE_SPICE_NETLIST_FROM_THE_EXTRACTOR_OUTPUT ... ... = (YES | NO)

You can generate a schematic netlist from a layout netlist without running the LVS. Use the LPE utility.

This option controls the generation of a flat schematic netlist from the layout netlist. The file is created only if the type of comparison is LVL or LVS. (When the comparison is LVL, the flat schematic netlist is generated from the second layout netlist.)

This schematic netlist output can be very useful for simulation if you have the NLE recognize parasitic capacitors. If you wish to limit the parasitic capacitors listed to those over a threshold value, use the DELETE_PARASITIC-_CAPACITORS_LESS_THAN control file option. (See page 300.)

The following options in the control file (covered below) apply only to the generated schematic netlist. If you do not want to generate a schematic netlist, skip ahead to page 280.

OUTPUT_FILE_OF_SPICE_NETLIST
SPICE_FILE_FORMAT
PRINT_COMMENTS_IN_SPICE_OUTPUT_GENERATED_BY_LVS
PRINT_FILTERED_DEVICES_IN_SPICE_OUTPUT_GENERATED_BY_LVS
SCALE_CHARACTER_FOR_RESISTORS
SCALE_CHARACTER_FOR_CAPACITORS
SCALE_CHARACTER_FOR_INDUCTORS
REPLACE_NLE_NODES_WITH_MATCHED_SCHEMATIC_NODES

OUTPUT_FILE_OF_SPICE_NETLIST = *spice_file*

If the GENERATE_SPICE_NETLIST_FROM_THE_EXTRACTOR_OUTPUT option is set to YES, the spice data will be created in the file indicated with this parameter. *spice_file* must be a valid DOS filename. Do not include the path. The file extension is optional.

The schematic netlist will be created using the language syntax set by the SPICE_FILE_FORMAT option (covered next).

The /P command line parameter will override this option.

SPICE_FILE_FORMAT = (CDL | PSPICE | HSPICE | SPICE)

This option selects the format of the flat schematic netlist generated from the layout.

Example:

GENERATE_SPICE_NETLIST_FROM_THE_EXTRACTOR_OUTPUT=YES
OUTPUT_FILE_OF_SPICE_NETLIST = "SPICEOUT"
SPICE_FILE_FORMAT = PSPICE

These three options will result in the LVS creating a flat PSPICE format schematic netlist from the layout data. If the OUTPUT_FILE_NAME_EXTENSION_FOR_LVS option is set to "LVS", the filename will be SPICEOUT.LVS.

PRINT_COMMENTS_IN_SPICE_OUTPUT_GENERATED_BY_LVS = ...
...(YES | NO)

If you desire that comments be added to the schematic netlist created by the LVS, set this option to YES. These comments include device instance names from the matched devices in the other netlist and details on collapsed devices.

PRINT_FILTERED_DEVICES_IN_SPICE_OUTPUT_GENERATED_ ...
... BY_LVS = (YES | NO)

When you want the LVS to include devices in the generated schematic netlist which are filtered out of the layout netlist before comparison, set this option to YES. Devices may be filtered if you have set any of the control file options which begin with the word "IGNORE" to YES. (See page 297.)

SCALE_CHARACTER_FOR_RESISTORS = ("*r_char*" | " ")

and

SCALE_CHARACTER_FOR_CAPACITORS = ("*c_char*" | " ")

and

SCALE_CHARACTER_FOR_INDUCTORS = ("*i_char*" | " ")

If one of these options is set to a non-blank character, that character will be used to identify the units of the values of devices in the indicated category in the generated spice netlist.

For example, you have defined the layout device model for resistors with the following statement:

***.LAYMODEL RESMODEL RES OHMS_PER_SQUARE = 5**

A resistor in the layout netlist using this model with dimensions L=10 and W=2 will have its value calculated as:

(10 / 2) * 5 = 25

If you want the value of this device to be followed with a "K" in the generated spice netlist (so that it is interpreted as 25 Kohms), use the following option in the control file:

Example: **SCALE_CHARACTER_FOR_RESISTORS = "K"**

The other options work in a similar manner. If you want to have the character 'p' printed after each capacitor value in the generated spice netlist (so that the values are interpreted as picofarads), use the control file option SCALE-_CHARACTER_FOR_CAPACITORS = 'p'. Using the option SCALE-_CHARACTER_FOR_INDUCTORS = 'n' will result in the values for inductors being interpreted as nanohenries.

REPLACE_NLE_NODES_WITH_MATCHED_SCHEMATIC_NODES...
... = (YES | NO)

This option applies only to parasitic capacitor statements in the generated spice netlist when the comparison is LVS. Simulating your circuit with parasitic capacitors extracted from the layout can be very useful.

If you intend to simulate the entire generated spice netlist, rather than using cut and paste methods to use only the parasitic capacitor statements, set this option to "NO".

However, you may not want to simulate the generated spice netlist because your original spice netlist contains probe points or other information not included in the generated spice netlist. If you use cut and paste methods to edit the parasitic capacitor statements into your original schematic netlist, the net names in the statements will not match. Layout netlist node numbers will be used instead of schematic netlist net names.

If you set this option to YES, the LVS will replace the node numbers indicated in parasitic capacitor statements with the netnames used in the original schematic netlist. This requires that the nets have been matched successfully by the LVS. All nets that have not been successfully matched with nets in the schematic netlist will be represented by the string "????".

This is the last control file option that applies only to the format of the generated spice netlist.

PRINT_NETS_AND_THEIR_DEGREES = (YES | NO)
and
PRINT_ALL_NETS_WHOSE_DEGREE_GREATER_THAN = *degree*

See **Device Transfor-mations** in **Running LVS** to learn about the types of transformations performed before degree is calculated.

These options control the generation of a report listing nets and their degrees. (Degree is defined as the number of devices to which a net connects.) The net degrees are calculated after performing transformations on devices in the netlists.

This report can be very useful in diagnosing shorts and opens. The discrepancies in net degrees between the two netlists often allow you to quickly find nets which are shorted together (degrees higher than expected) or open (several nets with a low degree instead of one with a high degree).

See the **Advanced Tutorial** to see how useful this file can be.

The report is enabled by setting PRINT_NETS_AND_THEIR_DEGREES to YES. When the report is enabled, the counts of all nets sorted by degree from each netlist is at the top of the report. A more detailed listing, with net names and numbers, is restricted to those nets with a degree larger than the *degree* parameter in the PRINT_ALL_NETS_WHOSE_DEGREE_GREATER_THAN option. *degree* must be a positive integer.

OUTPUT_FILE_OF_NET_DEGREES = *netdeg_file*

The list of nets and their respective degrees will be created in a file with the name *netdeg_file.* This parameter should be a valid DOS filename. Do not include the path. The file extension is optional.

This file will be created only if the PRINT_NETS_AND_THEIR_DEGREES option (described above) is set to YES.

PRINT_LIST_OF_FILTERED_DEVICES = (YES | NO)

Use this option to generate a list of filtered devices. Devices may be filtered if you have set any of the control file options which begin with the word "IGNORE" to YES. (See page 297.)

If you set this option to NO, you will not be provided with a list of devices which have been ignored by the LVS.

OUTPUT_FILE_OF_FILTERED_DEVICES = *filt_dev_file*

If the PRINT_LIST_OF_FILTERED_DEVICES option is set to YES, the data will be created in the file indicated with this parameter. *filt_dev_file* must be a valid DOS filename. Do not include the path.

OUTPUT FILES

See **LVS Output Files** for more details on all reports generated by the LVS.

These options control the most useful reports generated by the LVS.

PRINT_MATCHED_DEVICES_AND_NETS = (YES | NO)

This option can suppress the generation of the list of devices and nets which match in the two netlists. The devices will be sorted by type. Those devices with parameter mismatches outside the tolerances specified by the device models in the second netlist will be indicated by the string "Parameter Error". The list of devices with parameter mismatches is also provided in the next report.

If disk space is a concern, you can disable this report by setting this option to NO. If you set this option to YES, the file name is set with the next option.

OUTPUT_FILE_OF_MATCHED_DEVICES_INCLUDING_PARAMETER... ..._ERRORS = *match_file*

Remember that all files are stored in the directory defined by the **OUTPUT-_DIRECTORY_PATH** option.

The list of matched devices and nets will be created in a file with the name *match_file*. This parameter should be a valid DOS filename without a path.

This file will be created only if the PRINT_MATCHED_DEVICES_AND_NETS option is set to YES.

OUTPUT_FILE_OF_DEVICES_WITH_PARAMETER_ERRORS ... ... = *parm_err_file*

Device parameter tolerances are usually defined by *.LAYMODEL statements in the layout netlist. See page 233.

Use this parameter to define the name of the file to store the list of matched devices with value or dimension errors. A device will be listed if it matches in the two netlists, but the parameter values indicated in each netlist differ by a value greater than the tolerance specified by the device model.

In other words, use this file to locate devices which are correctly connected, but are the wrong size.

This report cannot be disabled, however, the MATCH_*device*_PARAMETERS options must be set to YES to enable this type of verification for specific device categories. If all MATCH_*device*_PARAMETERS options are set to NO, this file will not list any devices.

OUTPUT_FILE_OF_UNMATCHED_DEVICES_AND_NETS=*unmatch_file*

This option defines the name of the file where the list of unmatched devices and nets will be stored. This report cannot be disabled.

The devices in this report will be sorted by type and degree (the number of terminals). The device terminals for each device are listed and if a matched net is connected to the terminal, the netname is indicated. When an unmatched net is attached to a device, the string "?????" is indicated instead of the net name.

The report of unmatched nets are also sorted by degree (the number of devices attached to the net). The devices attached to each net are listed. When an unmatched device is attached to a net, the string "?????" is listed instead of the device name. The maximum number of devices listed for each net is controlled by the option SET_NET_SIZE_LIMIT_WHEN_PRINTING_CONNECTIONS.

OUTPUT_FILE_OF_FINAL_RESULTS_OF_NETLIST_COMPARISON ...
... = *summary_file*

This option provides the file name for the comparison summary report. It cannot be disabled. You must **ALWAYS** look at this file carefully.

INDIVIDUAL DEVICE OPTIONS

This section of options controls how devices in specific categories are transformed during preprocessing and how model names and parameter values are matched. Ten different device categories are recognized by the LVS. The device categories are listed in Figure 88.

MOSFET
BIPOLAR
GaAsFET
JFET
DIODE
CAPACITOR
RESISTOR
INDUCTOR
TXLINE
PARASITIC-_CAPACITOR

Figure 88: Valid LVS device categories.

Most of the following options can be overridden for specific device models in the netlists with parameters on the *.SCHMODEL or *.LAYMODEL statements. See Figure 73 on page 214 to see which parameter to use to override each option. The control file option provides the default for both netlists unless it is noted otherwise in the description.

When typing the options in the control file, replace the string "*device"* with one of the device category strings from the table given for that specific option. Not all options are valid for each device category. All options for a specific device category are grouped together in the control file.

SCALE_*device*_LENGTH_AND_WIDTH = *scale*

These options will scale the length and width of devices in **a schematic netlist** by the value provided for *scale*.

MOSFET
JFET
CAPACITOR
RESISTOR

Figure 89: Valid device categories for the SCALE...-LENGTH_AND-_WIDTH options.

Example: **SCALE_MOSFET_LENGTH_AND_WIDTH = 1e6**

See an example of using this feature on page 28.

Using this option in the control file will multiply the length and width of each MOSFET device in the schematic netlist by 10^6. This is very useful when the units of device length and width in the schematic netlist are given in meters (e.g. W=6u) and the units in the layout are in microns (e.g. W=6).

If you prefer to scale the final value of a device, rather than the dimensions, you can use the next option.

SCALE_*device*_VALUE = *scale*

These options will scale the value of devices in **a schematic netlist** by the value provided for *scale*. This can avoid false errors when the units used to express the value of a device are different in the two netlists.

CAPACITOR
RESISTOR
INDUCTOR

Figure 90: Valid device categories for the SCALE...VALUE options.

Example: **SCALE_RESISTOR_VALUE = .001**

Using this option in the control file will multiply the value of each resistor in the schematic netlist by .001 or 10^{-3}.

When you prefer to scale the dimensions of the device, use the previous option.

NUMBER_OF_PINS_FOR_*device* = (2 | 3 | 4| *)

This option affects how devices in the **schematic netlist** are preprocessed.

When this option is set to a '3', bipolar transistors, GaAsFETs, and JFETs, the devices will be stored as 3 terminal devices. If a fourth terminal connection is indicated for a device in the schematic netlist, it will be ignored.

When a '4' is used, only 4 terminal devices will be allowed. If only 3 terminals are defined for a these devices in the schematic netlist, the string defined by the BULK=*bulk_node_name* parameter in the corresponding *.SCHMODEL device model will be used for the fourth terminal. If the BULK option is not used in the corresponding device model, it will be flagged as an error and reported in the log file, "LVS.LOG".

Device category	Valid numbers
BIPOLAR	3, 4, or *
GaAsFET	3, 4, or *
JFET	3, 4, or *
DIODE	2, 3 or *
CAPACITOR	2, 3 or *
RESISTOR	2, 3 or *
INDUCTOR	2, 3 or *

Figure 91: Valid device categories for the NUMBER_OF_PINS... options.

Example: **NUMBER_OF_PINS_FOR_MOSFETS=4**

(In schematic netlist:)

```
*.SCHMODEL  NMOS  NMOS   BULK=VSS
*.SCHMODEL  PMOS  PMOS    BULK=VDD

MN5 OUT 19 12  NMOS  W=4U L=1.0U
```

When this control file option and the device models shown are used, the LVS will add the net VSS to the device statement for MN5. The LVS will then proceed with the comparison as though the device statement was written as:

```
MN5 OUT 19 12 VSS  NMOS  W=4U L=1.0U
```

In this case, NMOS devices should be recognized by the NLE as having 4 terminals.

For diodes, capacitors, resistors, and inductors, if the corresponding NUMBER_OF_PINS_FOR_*device* option is set to 2, the corresponding devices will be stored and compared as two terminal devices. When the option is set to 3, and the third terminal is not supplied in the device statement in the schematic netlist, the string after the BULK keyword in the device model will be used as the third terminal.

If this option is set to '*', both 3 terminal and 4 terminal bipolar transistors, GaAsFETs, or JFETs will be allowed. When this option is set to '*' for diodes, capacitors, resistors, or inductors, both 2 and 3 terminal devices will be allowed. However, devices with different numbers of terminals will never match (e.g. a 3 terminal device will never match a 4 terminal device).

SWAP_GAASFET_SOURCE_DRAIN = (YES | NO)

SWAP_JFET_SOURCE_DRAIN = (YES | NO)

SWAP_EMITTER_AND_COLLECTOR_TERMINALS = (YES | NO)

and

SWAP_CAPACITOR_TERMINALS = (YES | NO)

See more details on terminal swapping on page 316.

Use these options to enable swapping of terminals for specific device categories. The source and drain of MOSFET devices are always considered swappable, so are the terminals of resistors and inductors. The terminals of diodes and transmission line (TXLINE) devices can never be swapped. The SWAP-_EMITTER_AND_COLLECTOR_TERMINALS option applies only to LPNP and LNPN types of bipolar devices. The terminals of NPN and PNP type devices are never swapped.

Example: **SWAP_JFET_SOURCE_DRAIN = YES**

Using this option in the control file will allow the source and drain of JFET devices to be swapped if necessary to match devices in the two netlists.

See an example of using overrides for specific device models on page 212.

This option can be overridden for specific models of devices with the SWAP=NO parameter on the device models (*.SCHMODEL or *.LAYMODEL statements) in **both** netlists.

MERGE_SERIES_*device*S = (YES | NO)

See page 315 for more information on the difference between merges and collapses.

This option will allow devices connected in series to be merged. For transistors to be considered as candidates for a merge, the gate terminals of all devices must be connected.

MOSFET
BIPOLAR
GAASFET
JFET
CAPACITOR
RESISTOR
INDUCTOR
PARASITIC-_CAPACITOR

Figure 92: Valid device categories for MERGE-_SERIES options.

Setting this option to YES for specific device categories can result in a match of logically equivalent but physically dissimilar circuits. For example, a single resistor in one netlist will match with several resistors connected in series in the other netlist.

Only the LPNP and LNPN types of bipolar transistors may be merged. To allow series merges of these types of devices, the control file option SWAP_EMITTER-_AND_COLLECTOR_TERMINALS = YES must also be used. Even when you use the control file option MERGE_SERIES_BIPOLARS = YES, PNP and NPN types of transistors will **not** be merged.

Unless the corresponding MERGE_*device*S_OF_DIFFERENT_MODELS = YES option is used (see page 290), only devices with identical models can be merged. When used with transistors, the devices must be connected to the same substrate. The area of a merged device is the sum of the device areas collapsed to create it.

Example: **MERGE_SERIES_RESISTORS=YES**

Using this option in an LVS control file will merge all resistors connected in series into single devices.

```
R1  1  2  RESMODEL  L=2  W=3
R2  2  3  RESMODEL  L=2  W=3
R3  3  4  RESMODEL  L=2  W=3
R4  4  5  RESMODEL  L=2  W=3
R5  5  6  RESMODEL  L=2  W=3
```

The resistors in the schematic netlist fragment above will be merged into a single device. The length of the new device will be the sum of the lengths of the merged devices. The name of the new device will be the old device names concatenated together. The device names will be separated by a special character defined with the SPECIAL_CHARACTER_FOR_PRINTING-_SERIES_MERGES control file option. This character is '@' in the sample control supplied with the LVS installation. The resulting resistor device would be as follows:

```
R1@R2@R3@R4@R5  1  6  RESMODEL  L=10  W=3
```

The SMERGE, ALLMERGE, or ALL options in the device models in the netlists can be used to override series merges for specific device models. If you used the MERGE_SERIES_RESISTORS=YES option but then defined the RESMODEL resistor model in the schematic netlist with the following *.SCHMODEL statement, the series resistors in the schematic netlist shown above would not be merged. However, resistors using other models would still be merged.

```
*.SCHMODEL  RESMODEL  RES  SMERGE=NO
```

Series diodes can never be merged. The MERGE_SERIES_DIODES option must always be set to NO.

MOSFETS are treated somewhat differently than other three terminal devices. For MOSFET devices, this parameter only merges devices with identical length and width. (To allow the merge of devices of different sizes, see the MERGE_DISSIMILAR_SIZED_MOSFETS option below).

MERGE_PARALLEL_*device*S = (YES | NO)

See page 318 for more information and examples.

This option will allow devices connected in parallel to be collapsed. Setting this option to YES for specific device categories can result in a match of logically equivalent but topologically dissimilar circuits.

To allow two terminal devices with different models to be merged, use the appropriate MERGE_*device*S_OF_DIFFERENT_MODELS = YES control file option. Otherwise, only devices with identical models can be merged. The net connections for all terminals of the devices must be identical. When used with transistors, the devices must be connected to the same substrate. The area of a merged device is the sum of the device areas collapsed to create it.

MOSFET
BIPOLAR
GAASFET
JFET
DIODE
CAPACITOR
RESISTOR
INDUCTOR
PARASITIC-_CAPACITOR

Figure 93: Valid device categories for MERGE-_PARALLEL options.

Example: **MERGE_PARALLEL_RESISTORS=YES**

Using this option in an LVS control file will merge all resistors connected in parallel into single devices.

```
R1  1  2  L=2  W=3
R2  1  2  L=2  W=3
R3  1  2  L=2  W=3
R4  1  2  L=2  W=3
R5  1  2  L=2  W=3
```

The resistors in the schematic netlist fragment above will be merged into a single device. The name of the new device will be the old device names concatenated together and separated with a special character defined by the SPECIAL-_CHARACTER_FOR_PRINTING_PARALLEL_MERGES control file option. This character is '&' in the sample control supplied with the LVS installation. The width of the new device will be the sum of the merged devices. The resulting resistor device would be as follows:

```
R1&R2&R3&R4&R5  1  2  L=2  W=15
```

The PMERGE, ALLMERGE, or ALL options can be used to override parallel merges for specific device models in the netlists.

Only LPNP and LNPN types of bipolar transistors may be merged.

MOSFETS are treated somewhat differently than other three terminal devices. For MOSFET devices, this option enables only merges of devices with identical length and width. (To allow the merge of devices with different sizes, see the MERGE_DISSIMILAR_SIZED_MOSFETS option below).

MERGE_DISSIMILAR_SIZED_MOSFETS = (YES | NO)

See page 318 for details on dimension calculation for merges of dissimilar sized devices.

This option applies only to MOSFETS. The previous two options will merge only MOSFET devices with identical length and width. If you want the LVS to merge dissimilar sized devices, set MERGE_DISSIMILAR_SIZED_MOSFETS to YES. The MERGE_PARALLEL_MOSFETS= YES or MERGE_SERIES-_MOSFETS = YES option must also be used.

MERGE_*device*S_OF_DIFFERENT_MODELS = (YES | NO)

This option is valid only for the two-terminal devices listed in Figure 94. For these devices, it is possible to allow series or parallel merges between devices with different models. This option can be overridden for specific device models by the DMODEL, ALLMERGE, or ALL parameters in the netlist device models.

Device Category	**Merged Model Name**
CAPACITOR	CAPACITOR
RESISTOR	RESISTOR
INDUCTOR	INDUCTOR
PARASITIC-_CAPACITOR	PCAPACITOR

Figure 94: Valid device categories for MERGE......DIFFERENT-_MODELS options.

When devices of the same model are merged, the device model name remains unchanged. However, when devices of different models are merged, the model name assigned to the merged device is indicated by the table in Figure 94. Some of the model parameters, including tolerance parameters, from this model will be used instead of those defined for the original devices.

Example: **MERGE_RESISTORS_OF_DIFFERENT_MODELS=YES**

To see the effect of the above control file option, let us say you have two resistor models RESDIFF and RESPOLY. The device models are defined with the following statements in the layout netlist:

```
*.LAYMODEL  RESDIFF  RES  VALUETLR=.3  R_CONTACT =.1
*+ OHMS_PER_SQUARE=1

*.LAYMODEL  RESPOLY  RES  VALUETLR=.1  R_CONTACT =.1
*+ OHMS_PER_SQUARE=2
```

If two resistors in series, one of each model, were merged into one device, the model of the merged device would be RESISTOR.

However, the calculated value of the merged device would be accurate. The value of each resistor is calculated from the device dimensions and values of R_CONTACT and OHMS_PER_SQUARE in the netlist device models before the devices are merged.

When the LVS performs device parameter checking, the value tolerances defined for RESDIFF and RESPOLY will be ignored because the merged device has a model of RESISTOR before it is matched to a device in the other netlist. If a device model for RESISTOR is defined, that value tolerance would be used. If no model for RESISTOR exists, the default tolerance of .0005 will be used.

MERGE_*device*_CHAINS = (YES | NO)

This option controls the parallel merge of transistors which do not have all terminals shorted together. For bipolar transistors (LPNP and LNPN device types only, NPN and PNP devices types are never merged), the bases of the devices must be shorted and the collectors and emitters must be connected in chains. For FET transistors, the gates must be shorted together and the sources and drains connected in chains.

This is best explained with examples. See page 320 for a more complete explanation of this process.

MOSFET
BIPOLAR
GAASFET
JFET

Figure 95: Valid device categories for the MERGE-_*device*_CHAINS options.

MERGE_OUT_OF_ORDER_*device*_CHAINS = (YES | NO)

This option controls the parallel merge of transistors which do not have all terminals shorted together. The previous option will allow only transistors in chains which are connected in the same order in each chain to be merged. This option allows the transistors which make up the chains to be in different order in the connected chains.

When the LVS uses this option, warnings will be listed in the log file and the comparison summary report and also on the screen. All out of order chains will be listed in the unmatched devices report.

See page 321 for examples and more details.

MOSFET
BIPOLAR
GAASFET
JFET

Figure 96: Valid device categories for the MERGE-_*device*_CHAINS options.

COLLAPSE_SERIES_LOGIC_*device*S = (YES | NO)

See page 322 in **Device Transformations** for more information and examples.

This option controls whether 3 or 4 terminal devices of a specific category, which are connected in series but have different gates, will be collapsed into a pseudo device with swappable gate terminals. Enabling this option will result in a match between logically equivalent but physically different structures.

MOSFET
BIPOLAR
GAASFET
JFET

Figure 97: Valid device categories for COLLAPSE-_SERIES-_LOGIC options.

For a group of devices to be collapsed, the devices must use the same model and share the same substrate.

Example: **COLLAPSE_SERIES_LOGIC_MOSFETS=NO**

When logic collapses are disabled by using the control file option above, the LVS will not match circuits that are logically equivalent, but physically dissimilar due to the order of the gate connections.

Setting this option to YES will often prevent cascading errors due to misconnections, making it easier to diagnose circuit mismatches.

The SERIES, ALLCOLLAPSE, or ALL options on individual device models in the netlists can be used to override creation of series pseudo device collapses for specific device models.

Only LPNP and LNPN types of bipolar transistors may be collapsed. To allow series logic collapses of these types of devices, the control file option SWAP_EMITTER_AND_COLLECTOR_TERMINALS = YES must also be used.

Details on the devices collapsed to form a pseudo device will be reported in the file defined by the control file option OUTPUT_FILE_OF_COLLAPSED_DEVICES.

COLLAPSE_PARALLEL_LOGIC_*device*S = (YES | NO)

This option controls whether parallel devices (with 3 or 4 terminals) of a specific category with different gates will be collapsed into a pseudo device. For a group of devices to be collapsed, the devices must use the same model and share the same substrate.

MOSFET
BIPOLAR
GAASFET
JFET

Figure 98: Valid device categories for COLLAPSE-_PARALLEL-_LOGIC options.

Example: **COLLAPSE_PARALLEL_BIPOLARS = YES**

The gates of the pseudo device will be swappable.

See **Device Transformations** on page 323 for more information and examples.

COLLAPSE_DISSIMILAR_SIZED_*device*S = (YES | NO)

The two previous control file options enable the logic collapse of devices. When either of those options is enabled, this option controls whether or not devices with different sizes can be collapsed.

MOSFET
BIPOLAR
GAASFET
JFET

Figure 99: Valid device categories for COLLAPSE-_DISSIMILAR-_SIZED options.

Example:

COLLAPSE_SERIES_LOGIC_MOSFETS = YES
COLLAPSE_PARALLEL_LOGIC_MOSFETS = YES
COLLAPSE_DISSIMILAR_SIZED_MOSFETS = YES

See page 328 for details on how parameter values and gate order are verified when this option is set to YES.

When you use these options in your control file, MOSFET devices connected in series with different sizes will be collapsed so that the gates of the circuit are swappable. This may mean that two circuits that are logically equivalent, but with very different electrical properties, will match without a warning in the unmatched devices list. However, error messages will be indicated in the parameter error report.

MATCH_*device*_MODELS = (YES | NO)

All devices have a model name in each netlist. Set this option to YES if you want the LVS to verify that the model names for specific devices match in the two netlists.

MOSFET
BIPOLAR
GAASFET
JFET
DIODE
CAPACITOR
RESISTOR
INDUCTOR
TXLINE

Figure 100: Valid device categories for MATCH...-MODELS options.

Example: **MATCH_RESISTOR_MODELS=YES**

```
RA  1  2  RESMODEL  L=2  W=3

RB  1  2    L=2  W=3
```

If the two lines above represent statements from each of two netlists you are verifying against each other, and you have enabled verifying model names for resistors using the example option shown above, then the two devices will not match. (Resistor RB will have a model name of RESISTOR by default. See Figure 75 on page 218.) If you instead used MATCH_RESISTOR_MODELS=NO, the two devices would match.

These control file options do not change the fact that device categories and types must always match. Even when MATCH_MOSFET_MODELS=NO is used, a PMOS device will never match an NMOS device.

You must be careful to assign identical model names in both netlists when you enable model name checking. If the model names of a pair of candidates for a match are not exactly the same in each netlist, the device will not be matched.

MATCH_*device*_PARAMETERS = (YES | NO)

See **Parameter Calculation** on page 343 for more details.

Setting this option to YES forces the LVS to check for parameter value errors between matched devices of each device category.

Device category	**Parameters verified**
MOSFET	Length and Width
BIPOLAR	Area
GAASFET	Area
JFET	Area
DIODE	Area
CAPACITOR	Capacitance
RESISTOR	Resistance
INDUCTOR	Inductance
TXLINE	Value

Figure 101: Valid device categories for MATCH...PARAMETERS options.

The schematic netlist parameter values can be specified on the device statement, or defaults provided by the *.SCHMODEL statements can be used. In the layout netlist, the parameter values can be calculated by the NLE, or defaults can be used for specific models using *.LAYMODEL statements. The *.LAYMODEL statements are where you should provide tolerances for value matching.

If you have enabled device merges, the parameter value verified will be the merged device value. If you have enabled logic collapses, the values of the individual devices that were collapsed will be verified.

Unlike the MATCH_*device*_MODELS options above, a mismatch on parameter values will not prevent a device from matching, however, the value mismatch will be reported in the file specified by the OUTPUT_FILE_OF_DEVICES-_WITH_PARAMETER_ERRORS option in the control file.

IGNORE_UNCONNECTED_*device*S = (YES | NO)

Some circuit layouts include many unconnected devices by design (e.g. semi-custom layouts). If you want these devices filtered out by the LVS before matching the two netlists, set this option to YES. This option controls devices that have no net connections to any terminal. (See options below for partially connected devices.)

If you want unconnected devices to be flagged as errors, set this parameter to NO. In this case, all unconnected devices will be listed as unmatched devices. If you prefer to filter them from the matching algorithm, but still list the filtered devices in a separate report, use the PRINT_LIST_OF-_FILTERED_DEVICES = YES option.

MOSFET
BIPOLAR
GAASFET
JFET
DIODE
CAPACITOR
RESISTOR
INDUCTOR

Figure 102: Valid categories for IGNORE_UN-CONNECTED and IGNORE_ONE-_TERMINAL-_CONNECTED options.

IGNORE_ONE_TERMINAL_CONNECTED_*device*S = (YES | NO)

This option controls how devices with only one connected terminal are handled. If you want the LVS to filter out devices with only one terminal attached to a valid net (VDD for instance), set this option to YES.

If you consider devices with only one connected terminal to be in error, set this parameter to NO. In this case, all unmatched devices of this kind will be listed in the unmatched device list.

IGNORE_TWO_TERMINALS_CONNECTED_*device*S = (YES | NO)

This option controls how 3 or 4 terminal devices with only 2 connected terminals are handled. If you want the LVS to filter out these devices for a specific category, set this option to YES.

The devices filtered out by any of the IGNORE_... options can be listed by using the PRINT-_LIST_OF-_FILTERED-_DEVICES = YES option.

If you consider devices with unconnected terminals to be in error, set this parameter to NO. If your circuit uses bipolar transistors with collectors which are unconnected by design, set this parameter to NO.

When this option is set to NO, and there is no corresponding device in the other netlist, a device of this kind will be listed in the unmatched device list.

MOSFET
BIPOLAR
GAASFET
JFET

Figure 103: Valid device categories for IGNORE_TWO_TERMINAL-_CONNECTED options.

IGNORE_SHORTED_*device*S = (YES | NO)

Setting this option to YES will force the LVS to ignore devices which have shorted terminals. Three terminal FET devices with a shorted source and drain are considered shorted devices by the LVS. Bipolar transistors with a short between the collector and the emitter are also considered shorted devices. See Figure 105.

If you prefer that shorted devices are not removed from the netlists, set this parameter to NO. Then, all unmatched shorted devices will be listed in the unmatched device report.

If you have written the NLE rule set to recognize transistors with shorted terminals as capacitors (using the NODES option in the DEVICE rule, see page 135), they will not be filtered out of the layout netlist even if this option is set to YES. This is because the binary layout netlist created by the NLE will list a two terminal capacitor for such a device, not a transistor with shorted terminals.

MOSFET
BIPOLAR
GAASFET
JFET
DIODE
CAPACITOR
RESISTOR
INDUCTOR
TXLINE

Figure 104: Valid device categories for IGNORE-_SHORTED... options.

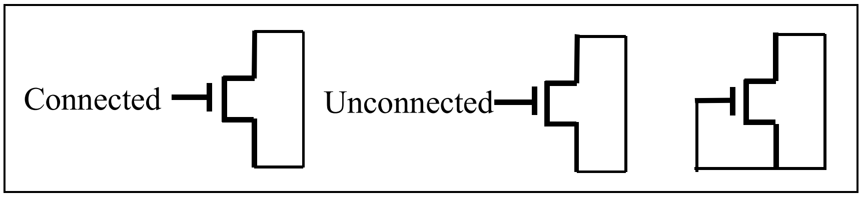


Figure 105: Devices considered shorted by the LVS.

IGNORE_MOSFET_IF_GATE_PIN_IS_TIED_TO_CRITICAL_NET = (YES | NO)

This option, and the following two options, rely on the LVS recognizing the names of the nets which represent ground and power. Ordinarily these nets are treated just like any other nets. However this option will not be able to filter devices unless the LVS knows which nets are power and ground. See page 221 to learn how the LVS determines which nets are power and ground nets.

This option will filter NMOS type devices from either netlist when the gate terminals are connected to a ground net. PMOS devices will be removed from the netlist when the gate terminals are connected to a power net.

IGNORE_MOSFET_IF_SOURCE_AND_DRAIN_PINS-_ARE_TIED_TO_CRITICAL_NET = (YES | NO)

This option will filter NMOS type devices from either netlist when the source and drain terminals are both connected to a ground net. PMOS devices will be removed from the netlist when the source and drain terminals are both connected to a power net. (See note above to learn the importance of defining power and ground nets.)

IGNORE_BIPOLAR_IF_BASE_PIN_IS_TIED_TO_CRITICAL_NET = (YES | NO)

This option will filter NPN and LNPN type devices from either netlist when the base terminals are connected to a ground net. PNP and LPNP devices will be removed from the netlist when the base pins are connected to a power net. (See note above to learn the importance of defining power and ground nets.)

DELETE_*device*S_LESS_THAN = *filtval*

Parasitic devices can be created in a layout netlist by the NLE. These devices are not intended to match devices in a schematic netlist. However you may want to be aware of their presence in your design. If you create a schematic netlist from the layout (by using the GENERATE_SPICE-_NETLIST_FROM_THE_EXTRACTOR_OUTPUT control file option with the LVS, or by using the LPE utility) you can include these devices for simulation purposes.

PARASITIC-_CAPACITOR

Figure 106: Valid device categories for the DELETE-_*device*S_LESS-_THAN options.

There can be thousands of these devices in your layout, and often their value is too small to be significant. If you want to filter these devices, retaining only those over a threshold value, use that value for *filtval*. If you prefer that all parasitic devices be reported, set *filtval* to 0.

Example **DELETE_PARASITIC_CAPACITORS_LESS_THAN = 1p**

This control file option will filter out from the layout netlist all parasitic capacitors with a value less than 1 picofarad.

Summary of How to Prepare the Control File for the LVS

1. Copy the sample control file, CONTROL.LVS, to a new file. (We strongly suggest that you copy the sample control file (Q:\ICED\[23]SAMPLES\LVS\CONTROL.LVS) and edit only the copy. Keep the original control file for reference.)

2. Edit the options to customize the LVS for your particular design. Edit only the fields after the '=' on each line. Do not delete any lines.

3. Look at the filter options (options beginning with the word IGNORE, see page 297). If you do not want any devices recognized by the NLE to be ignored by the LVS, set all filters to NO.

4. If you want to enable device value or dimension checking, set the appropriate MATCH_*device*_PARAMETERS options to YES.

See page 380 to learn more about the optional output files.

5. Enable all the output files that may be useful. You may want to enable all reports on your initial run. For subsequent runs, disable unnecessary reports. Large designs can result in considerable disk space being consumed by reports.

6. Use the modified control file name as the first argument in the LVS command line.

[23] Remember that Q:\ICED represents the drive letter and path where you have installed ICED32™.

Running the LVS Circuit Comparison

Command Line Syntax

```
LVS [path\]control_file_name  [path\]netlist1_file_name  [path\]netlist2_file_name ...
        ... [ @file_name ] ...
        ... [ /c format ] ...
        ... [ /e ( yes | no ) ] ...
        ... [ /f (or /f1) sch_format ] ...          Command
        ... [ /f2 sch2_format ] ...                 line options
        ... [ /g ( yes | no ) ] ...
        ... [ /i in_path ] ...
        ... [ /l (or /l1) ext_file ] ...
        ... [ /l2 ext_file ] ...
        ... [ /m text_mode ] ...
        ... [ /o dir_path ] ...
        ... [ /p sch_out_format ] ...
        ... [ /s sch_file_name ] ...
        ... [ /t (or /t1) sch_subckt ] ...
        ... [ /t2 sch2_subckt ] ...
        ... [ /v ( yes | no ) ]
```

The LVS requires three input files. The details about how to prepare these files is covered in the previous chapters. We suggest that all input files be in the current working directory rather than supplying paths with the file names.

The command line options are not required. They override defaults and options in the control file. The command line options will be covered in more detail below.

You can specify the path for all input files with the /i command line option.

The *control_file_name* parameter defines the name of the runtime options control file. (A sample control file, CONTROL.LVS, is provided with the LVS installation.) If you do not supply a file extension in *control_file_name*, an extension of ".lvs" is assumed. If the control file is in the current directory (or in the directory specified with the /i command line option), the *path* is not required.

The *netlist1_file_name* and *netlist2_file_name* identify the netlists to compare. If the files are in the current directory, the *path* is not required.

The netlists can be either schematic or layout netlists. The order of the two netlists on the command line depends on the comparison to be performed. (See below.)

A schematic netlist must be prepared as described in the **Schematic Netlists** chapter. The netlist must be written in one of the following four circuit description languages: SPICE, HSPICE, PSPICE, or CDL. A file extension of ".net" is appended by default to the schematic netlist file name if you do not supply an extension.

A layout netlist must be prepared as described in the **Layout Netlists** chapter. The most important component is a binary file created by the NLE circuit extractor. No other circuit extractor is currently supported. A file extension of ".lay" is appended by default to the layout netlist file name if you do not supply an extension.

If you supply either netlist file name with a final '.' (e.g. "schtest."), the LVS assumes that the input file has no extension.

The LVS can perform three types of netlist comparisons. The three choices are LVS (Layout Vs. Schematic), LVL (Layout Vs. Layout), and SVS (Schematic Vs. Schematic). The comparison choice is set by the /c command line option or by the TYPE_OF_COMPARISON option in the control file.

For an LVS comparison, the schematic netlist must be referred to by *netlist1_file_name*. *netlist2_file_name* must identify the layout netlist. This order is required.

Example: **LVS CONTROL.LVS SCH.NET TEST.LAY**

This LVS control line will execute the program using the options in the file CONTROL.LVS, the schematic netlist SCH.NET and the layout netlist TEST.LAY. When performing an LVS comparison, the schematic netlist file name is typed **before** the layout netlist file name.

For a SVS comparison, both *netlist1_file_name* and *netlist2_file_name* must identify schematic netlists.

If a LVL comparison is being performed, both *netlist1_file_name* and *netlist2_file_name* must identify layout netlists. If you use the GENERATE_SPICE_NETLIST_FROM_THE_EXTRACTOR_OUTPUT option in the control file, the Spice netlist will be generated from *netlist2_file_name.*

Most of the command line options override options in the control file. Others override program defaults. The case of the option is not important. The order of the options does not need to be the same as that used in the syntax description, but the three file name parameters must come first in the order shown. Blanks can be added for readability, but are not required.

You can write all command line parameters and options in a file and refer to that file using the *@file_name* option in the command line.

Example: **LVS @LVSRUN.TXT**

```
control.lvs
sch.net
lay.net
```

Figure 107: Contents of LVSRUN.TXT.

If the contents of LVSRUN.TXT look like Figure 107, the LVS will use the parameters as though they were all typed at the command line. You can have other *@file_name* parameters in the file, nested up to ten deep.

You can use more than one *@file_name* option in the command line. You can combine file name parameters and options typed on the command line with *@file_name* options. Options are read from left to right, and if an option is used more than once, the option on the right will override a previous option.

Example: **LVS @LVSRUN.TXT @LVSOPT.TXT /O NEWRSLT**

```
/o results
/i mypath
```

Figure 108: Contents of LVSOPT.TXT.

This example shows how you can use one options file to specify the file names and another to specify the command line options. The /O option specifies the output directory path. (This option is described below.) The "/O NEWRSLT" command line option will override the "/o results" option in the file LVSOPT.TXT. Note that case is irrelevant.

The remaining command line options are described below in alphabetical order.

[/c *format*] When this option is used, you can override the TYPE-_OF_COMPARISON option in the control file. The *format* parameter must be one of the following:

LVS
SVS
LVL

[/e (yes | no)] Use the "/e yes" option if you want the LVS to terminate when errors are found in the node correspondence file. Otherwise, when errors are present, the LVS will continue and the errors will be reported in the LVS.LOG file.

[/f *sch_format*]
or
[/f1 *sch_format*] This option allows you to override the SCHEMATIC-_FILE_FORMAT option in the control file. The *file_format* parameter must be one of the following strings:

CDL
PSPICE
HSPICE
SPICE

Both options, /f and /f1, perform the same function. You may want to use the /f1 form for readability when you also use the /f2 below.

The *.FORMAT statement in a layout netlist will override the /f2 option.

[/f2 *sch2_format*] This option allows you to override the SECOND-_SCHEMATIC_FILE_FORMAT option in the control file. The *sch2_format* parameter must come from the list given above for the /f option.

[/g (yes | no)] Use this option to override the control file option RECOGNIZE_GLOBAL_TEXT_IN_SUBCELLS. This will enable or disable the use of node labels which use a colon (':') suffix which are nested in subcells.

[/i *in_path*] The *in_path* parameter defines the default directory for all input files. The LVS will search in this directory for the control file and netlists as well as any optional input files. *in_path* can be a fully qualified directory path beginning with a drive letter or '\', or it can be relative to the current directory.

When the /i option is not used, the current directory will be used as the default directory.

When a directory path is supplied with a file name, the LVS will search for the file in that directory rather than the default directory.

[/l *ext_file*]
or
[/l1 *ext_file*] When the comparison type is LVS, either of these options allow you to override the name of the binary layout netlist file for comparison. The name of the file given by *ext_file* will override the file name provided in the *.LAYOUT statement in the layout netlist file.

If the comparison type is LVL this option will override the *.LAYOUT statement in the first layout netlist file.

[/l2 *ext_file*] If the comparison type is LVL, this option will override the *.LAYOUT statement in the second layout netlist file. Do not use it if you are performing a LVS or SVS comparison.

[/m *text_mode*] Use this option to override the control file option LAYOUT_TEXT_MODE. This option controls how node labels in the layout netlist are used for forced node correspondence. The *text_mode* parameter must be one of the following:

EQUIV
AUTO
SKIP

[/o *dir_path*] This command line option overrides the OUTPUT_DIRECTORY_PATH option in the control file. The *dir_path* parameter defines the default path for all output files the LVS creates.

[/p *sch_out_format*] This option will override the SPICE_FILE-_FORMAT option in the control file. This will be the format of the spice file generated from the layout. Valid strings for the *sch_out_format* parameter are the same as for the /f command line option above.

[/s *sch_file_name*] When this option is used, the file indicated on the *.SCHEMATIC statement in the first schematic netlist is ignored, and the file *sch_file_name* is used instead. You can specify a directory path with the file name if the file is not in the current directory. Do not use this option if the comparison type is LVL.

[/t *sch_subckt*] or [/t1 *sch_subckt*]	Use either option to override the TOP_LEVEL-_SUBCKT_IN_SCHEMATIC_FILE option in the control file.
[/t2 *sch2_subckt*]	Use this option to override the TOP_LEVEL-_SUBCKT_IN_SECOND_SCHEMATIC_FILE option in the control file. Use this option only when performing an SVS comparison.
[/v (yes | no)]	Use this option to override the control file option ENABLE_VIRTUAL_CONNECTIONS. This will enable or disable the LVS from making connections between nets which are not physically connected. **You should use the "/v=no" option on the final run of a chip for which virtual connections were defined, so that you do not prevent the LVS from finding real open circuits.**

Example: **LVS CONTROL.LVS SCH.NET TEST.LAY /t MUX /l MUX.EXT**

This example shows you how to perform a comparison of a subcircuit rather than an entire chip without editing any of the input files.

The LVS command line above uses two options. The /t option will cause the LVS to override the TOP_LEVEL_SUBCKT_IN_SCHEMATIC_FILE option in the control file and use "MUX" as the top level circuit in the schematic netlist. The rest of the schematic netlist (except for subcircuits referenced in the MUX subcircuit) will be ignored.

The /l option will cause the file name on the *.LAYOUT statement in the TEST.LAY file to be ignored. The data in the file MUX.EXT will be used instead as the binary layout netlist.

Runtime Errors

If there are syntax errors in either netlist, or in the control file, this is reported immediately on your screen and in the LVS.LOG file in the following form:

> LINE : *line_number* IN FILE : *file_name*
> ERROR # *err_number* : *err_description*
> *file_line*
>
> (or)
>
> LINE : *line_number* IN FILE : *file_name*
> WARNING # *warning_number* : *warning_description*
> *file_line*
>
> (or)
>
> FATAL ERROR : *error_message*

The *file_line* is the actual line of the netlist file or control file that contains the error. The parameter that caused the warning or error will be surrounded by chevrons ("<< >>").

The LVS aborts if it finds syntax errors in any input file. Warnings will not result in an abort, but should be noted and resolved before the next run. The LVS will not report more than 15 errors before aborting.

Large numbers of mysterious errors in the control file are often the result of a missing line. Never delete lines from a control file. (We recommend that you backup the original control file supplied with the installation and never edit it so that you can always start with a fresh copy.)

If syntax errors in the schematic netlist are listed because the LVS is ignoring the first line, try the control file option TREAT_FIRST_LINE_IN_SPICE-_NETLIST_AS_COMMENT_LINE = NO.

If the LVS aborts with a message similar to "FATAL ERROR: Device was not declared within SUBCKT", see page 227 to learn how to prepare the schematic netlist for the LVS.

If you receive the message, "Insufficient Memory", the LVS was unable to run to completion with the amount of RAM memory available on your computer. Since the LVS at this time does not create virtual memory through a swap file, there are no settings you can change to avoid the problem. You must free more memory for the LVS through system settings or by terminating memory resident programs.

The LVS run can be interrupted by using the <CTRL><BREAK> or <CTRL><C> keyboard combinations.

Overview of Matching Algorithm

The LVS does not require any forced correspondence between the two netlists. The use of node names is not required to match nets and devices. The LVS uses a powerful graph coloring algorithm in combination with a local matching algorithm to find one-to-one correspondence between the device nodes and net nodes of the two netlists[24]. Node properties in the circuits are calculated and sorted to find one-to-one correspondence.

Nodes in the layout are labeled with text components. See page 248.

The NLE assigns unique integer values to all devices and nets in a layout netlist. These node numbers will be used when reporting unmatched devices and nets. You can use commands in the ICED32™ layout editor to highlight specific nodes in the layout using the node numbers listed in the reports generated by the LVS. See page 389. If the devices and nets are named through the use of text component labels, the LVS will list the node names as well as the node numbers.

Device categories and types are listed in the table on page 210.

The LVS matches devices of same device category and type only. The model names of the devices must also match if the MATCH_*device*_MODELS option for that device category is set to YES in the control file.

An initial set of node equivalences can sometimes speed the matching algorithm, but is not required. If the circuits are highly symmetric, a few forced points of correspondence between the two netlists may be necessary. Symmetric circuits have large numbers of virtually identical circuits. See Symmetric Circuits on page 365 for important details on performing a comparison between highly symmetric circuits.

[24] These algorithms are based on those described in "GeminiII: A Second Generation Layout Validation Program" by Carl Ebling, Department of Computer Science, University of Washington.

Filters can be used to remove unconnected or shorted devices from the netlist comparison. The control file options which begin with the string "IGNORE" will remove from the netlist any devices in a specific category which meet different criteria. This can be useful when performing verification on layouts which use a semi-custom template.

Read about the filter options on page 297. A quick summary of these options appears on page 387.

Various merges and collapses can be performed on the netlists before the LVS attempts to match them. The next section has details on these transformations.

Device Transformations

See page 214 for a table describing the device model parameters which override control file options.

The LVS can transform the circuitry of specific categories of devices in the input netlists. These transformations are enabled or disabled for both netlists through the options in the control file. These options can be overridden for specific models of a device in a netlist through the use of control file override parameters in the device model statements (*.SCHMODEL or *.LAYMODEL). The transformations take place before the two netlists are compared.

There are four different transformations where devices will be combined before circuit comparison is performed. (These transformations are described in more detail on the following pages.)

The total number of devices, after each type of transformation is processed, is reported in the comparison summary report. (See page 376.)

Merges: A device merge can be performed when devices in the same category are connected in series or parallel and can be transformed into a single device in the same category. This type of transformation allows the LVS to calculate the value of the merged device.

Chain Merges: Devices which are prevented from a parallel merge because they are in separate series chains can still be merged with this type of transformation. (See an example on page 320.) The values of devices merged with this transformation will be calculated and compared.

Series and Parallel Collapses: This type of transformation takes three or four terminal devices connected in series or parallel, with different gates or base terminals, to be collapsed into a circuit with several swappable terminals. It is this circuit, or pseudo device, rather than the individual devices, which will be compared.

Pull-up and Pull-down Circuits: Entire circuits can be collapsed into pseudo devices with groups of swappable terminals.

These transformations allow the successful comparison of designs that are logically equivalent but very different physically.

These types of transformations can also be used to help diagnose circuit errors. Even when you want to verify that your designs are physically equivalent, if you have circuit misconnections that you are having trouble locating, you may want to perform a few LVS runs with these transformations enabled to allow you to isolate the logic problem. See page 421 for an example of how this process works.

Another transformation allows certain terminals of a device to be swapped to allow it to match an equivalent device in the other netlist. See the tables on the following pages to see how to enable or disable terminal swapping for specific categories of devices.

One last circuit transformation, which can help you to overcome limitations in the schematic netlist languages, is the ability to define subcircuits in the layout netlist. Devices like multiple emitter bipolar transistors must be simulated in the schematic netlist as collections of single emitter transistors, even though they are single devices in the layout. This presents a real problem for circuit comparison that the LVS solves by letting you expand single devices in the layout netlist into collections of devices.

Terminal Swapping

The terminals of some categories of two terminal devices can always be swapped by the LVS when attempting to match devices in the two netlists. Others can never be swapped. See Figure 109.

Device category	**Terminal swapping allowed?**
DIODE	Never
CAPACITOR	Enabled by control file option SWAP_CAPACITOR_TERMINALS or by the SWAP parameter in device models.
RESISTOR	Always
INDUCTOR	Always

Figure 109: Two terminal device terminal swapping.

Capacitors are the special case. When the control file option SWAP_CAPACITOR_TERMINALS = YES is used, the terminals of capacitors can be swapped to allow devices in the two netlists to match. When it is set to NO, the terminals cannot be swapped. However, this behavior can be overridden for specific device models by using the SWAP parameter on the *.SCHMODEL model (for schematic netlists) and the *.LAYMODEL model (for layout netlists).

The source and drain for some three or four terminal devices can be swapped. See Figure 110. LPNP and LNPN type bipolar devices can have their emitter and collector swapped as an option in the control file. The ability of the SWAP device model parameter to override the control file options for these device categories is the same as that for capacitors described above.

Device category	**Terminal swapping allowed?**
MOSFET	Source and drain terminal swapping is always enabled
BIPOLAR	LPNP and LNPN: Enabled by control file option SWAP_EMITTER_AND-_COLLECTOR_TERMINALS or by the SWAP parameter on device models. PNP and NPN: Never swapped
GaAsFET	Enabled by control file option SWAP_GAASFET_SOURCE_DRAIN or by the SWAP parameter on device models.
JFET	Enabled by control file option SWAP_JFET_SOURCE_DRAIN or by the SWAP parameter on device models.
TXLINE	Never

Figure 110: Multi-terminal device terminal swapping.

When you use the SWAP override parameter on a device model, be sure that the device model in the other netlist uses the same SWAP parameter. A device that has swapping enabled will never match a device that has swapping disabled.

Device Merges

Transistors connected in parallel or in series with different gates can be collapsed as well. See page 322.

The LVS can merge devices that are connected in series or in parallel into a single device. For transistors to be considered as candidates for a merge, the gate terminals of all devices must be connected and they must use the same substrate. For example: a MOSFET transistor that is created as a fingered device in the layout can be merged into the equivalent single device before comparison.

MOSFET
BIPOLAR
GAASFET
JFET
DIODES[25]
CAPACITOR
RESISTOR
INDUCTOR

Figure 111: Valid device categories for merges.

The name of a merged device in the schematic netlist will be the old device names concatenated together. Special characters defined in the control file will separate the device names. The control file options are SPECIAL-_CHARACTER_FOR_PRINTING_SERIES_MERGES and SPECIAL_CHARACTER_FOR_PRINTING_PARALLEL_MERGES. In the sample control file supplied with the LVS installation, the series merge character is '@' and the parallel merge character is '&'.

In the layout netlist, merged devices will be named by a unique node number followed by the string "[M,*n*]", where *n* represents the number of merged devices.

See page 287 for another example of a series merge of resistors.

These merges are enabled or disabled using one of two methods. The default for all models of a given device category is defined in the control file with the MERGE_SERIES_*device*S and MERGE-_PARALLEL_*device*S options, where *device* is one of the strings in the table shown in Figure 111. For specific device models, the PMERGE, SMERGE, ALLMERGE, or ALL options in the device model definition can be used to override the default. (See page 212 for an example of how to use these

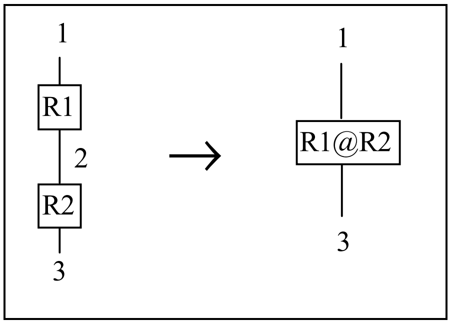


Figure 112: Series merge

[25] Diodes can be merged when connected in parallel only. Diodes connected in series are never merged.

overrides.) Only devices using the same model name will be merged, unless the control file option MERGE-_*device*S_OF_DIFFERENT-_MODELS = YES (or the DMODEL=YES override in the device model definition) is used.

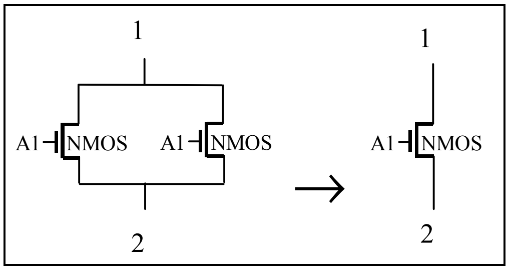


Figure 113: Parallel merge

The parameter value of the merged device is computed automatically. Merged bipolar, GaAsFET, or JFET devices will all be assigned the area of the sum of the merged devices.

Adjustments to device sizes defined with the LOFFSET, WOFFSET, or BENDS_CR keywords in the device models will be performed on the individual devices before the value of a merged device is calculated.

The MERGE_SERIES_MOSFETS and MERGE_PARALLEL_MOSFETS options control the merge of devices with the same dimensions. If the devices are in series, they must all have the same width. The length of the resulting series device is sum of the lengths of the original devices. If the devices are in parallel, they must all have the same length. The width of the resulting parallel device is sum of the widths of the original devices.

If you want the LVS to merge MOSFET devices with different dimensions, the MERGE_DISSIMILAR_SIZED_MOSFETS option must be set to YES in the control file (or the DSIZE=YES parameter must be used in the device models of the netlists.)

For MOSFET category devices connected in series with different widths, the length and width are calculated as follows (where L_1 through L_n are the lengths of the original devices, and W_1 through W_n are the original widths):

Series Merge

$$\text{Length} = \sqrt{(L_1*W_1 + L_2*W_2 + ... + L_n*W_n) * (L_1/W_1 + L_2/W_2 + ... + L_n/W_n)}$$

$$\text{Width} = \sqrt{(L_1*W_1 + L_2*W_2 + ... + L_n*W_n) / (L_1/W_1 + L_2/W_2 + ... + L_n/W_n)}$$

MOSFET devices merged in parallel will have their length and width calculated as follows:

Parallel Merge

$$\text{Length} = \sqrt{(L_1*W_1 + L_2*W_2 + ... + L_n*W_n) / (W_1/L_1 + W_2/L_2 + ... + L_n/W_n)}$$

$$\text{Width} = \sqrt{(L_1*W_1 + L_2*W_2 + ... + L_n*W_n) * (W_1/L_1 + W_2/L_2 + ... + L_n/W_n)}$$

Merges of Devices in Chains

Transistors that are connected in series chains connected in parallel can be merged with this transformation. Look at Figure 114. The two transistors attached to net A1 could take part in a parallel merge if nodes 3 and 4 were connected. You may prefer that these devices are merged even though these nodes are not connected.

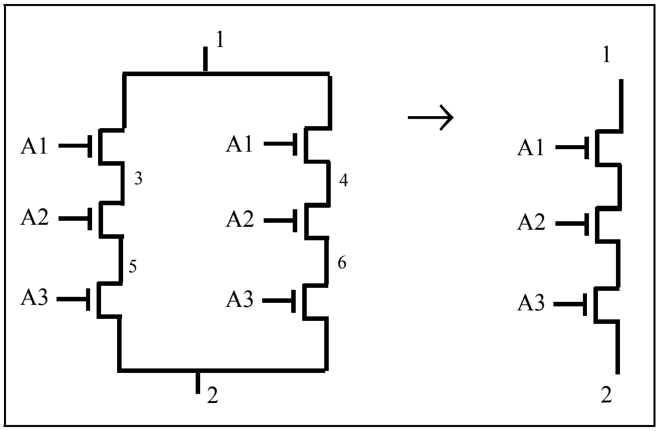


Figure 114: Merge of device chains.

When the corresponding MERGE_*device*_CHAINS = YES control file option is used, devices in that category will be merged as shown in Figure 114. Nets 3 and 4 will be virtually connected, as will nets 5 and 6.

The default provided in the control file option MERGE_*device*_CHAINS can be overridden for specific models of a device with the device model keywords CHAIN, ALLMERGE, or ALL. When any of these parameters are used to allow chain merges, circuits that are physically different, but logically equivalent, in each netlist can be matched successfully.

The values of the merged devices will be calculated in the same manner as simple merges. The value of each merged device will be verified against the value of the matched device in the other netlist.

The LVS has no limit on the number of devices connected in series in each chain, or on the number of chains connected in parallel. However the devices must be connected in the same order in each device chain unless out-of-order chain merges are enabled.

Out-of-order chain merges are enabled with the control file option MERGE-_OUT_OF_ORDER_*device*_CHAINS = YES. This default for all devices in the specified category can be overridden with the device model keywords DCHAIN, ALLMERGE, or ALL.

When out-of-order chain merges are enabled, the devices in the chains shown in Figure 115 will be merged. Note that nets A2 and A3 are swapped in the second chain. You may consider such misconnections acceptable. If not, do not enable this type of transformation.

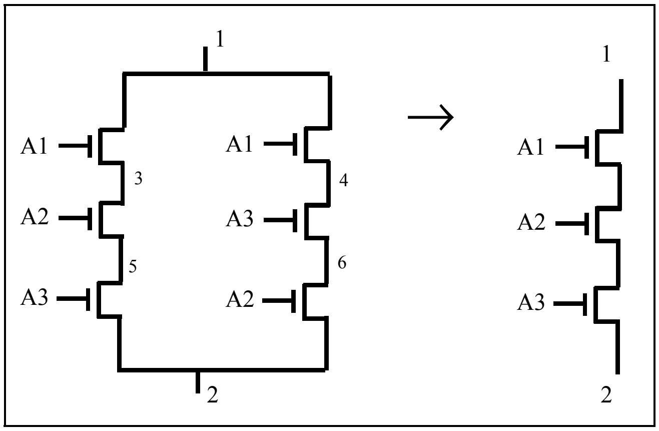


Figure 115: Merge of out-of-order device chains.

Even when out-of-order chains are enabled, a warning will be printed for each out-of-order chain merged. These warnings will be displayed on your screen and printed in the LVS log file and the comparison summary report. Details on each out-of-order chain will be listed in the unmatched devices report.

Series Logic Collapses

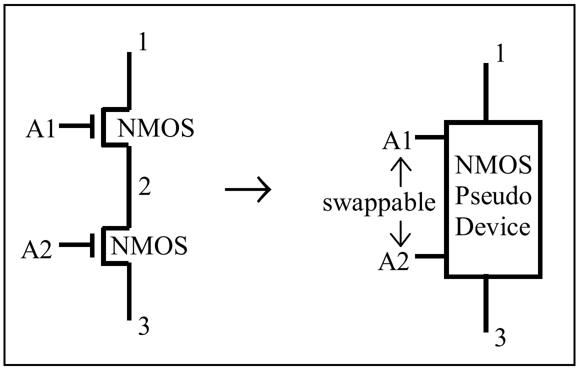


Figure 116: Series logic collapse.

The LVS can create a pseudo device from series transistors with different gates. This type of collapse is controlled by the COLLAPSE_SERIES-_LOGIC_*device*S options in the control file. This type of collapse can be enabled for any type of three or four terminal device. The collapse is performed only on devices of the same category, type, and model. The devices must be connected to the same substrate.

The gates of the pseudo device will be treated as swappable when matching an equivalent pseudo device in the other netlist. However, when devices with different sizes are collapsed (because you have used the corresponding control file option COLLAPSE_DISSIMILAR_SIZED_*device*S=YES), and the order of the gate terminals of the matched pseudo device is different in each netlist, an error message will be generated in the parameter error report. You will also be warned when the parameter values of the original devices do not agree. More details on this process are described on page 328.

In other words, a discrepancy in the order of the signal connections will not prevent a pseudo device from matching. However, when the collapsed devices are different sizes, you will be warned about the discrepancy in the parameter error report. Most designers do not consider discrepancies of this type to be logic errors, but the timing properties of each circuit may be completely different.

The form of each pseudo device name is "*node_number*[P,*n*]", where *node_number* is the node number of one of the original devices, and *n* is the number of original devices.

Devices formed from a simple merge will not be listed as pseudo devices.

All pseudo devices can be listed by using the PRINT_COLLAPSED_DEVICES-_IN_A_SEPARATE_FILE=YES option in the control file. The report will be stored in the file defined by the control file option OUTPUT_FILE_OF-_COLLAPSED_DEVICES. The names and parameter values of each device collapsed to create the pseudo device will be reported here.

Series logic collapses take place before parallel logic collapses (covered next). Once the parallel logic collapses are performed, if you want the LVS to perform additional series logic collapses, you must use the TAKE_CARE_OF_LOGIC-_EQUIVALENCES_WHILE_MATCHING = YES option. See Pull-Up and Pull-Down Pseudo Devices on page 324.

Parallel Logic Collapses

Parallel transistors with different gates can be collapsed into a pseudo device. The gates of the pseudo device will be treated as swappable. You enable or disable this type of transformation by using the COLLAPSE_PARALLEL-_LOGIC_*device*S options in the control file. This type of collapse can be

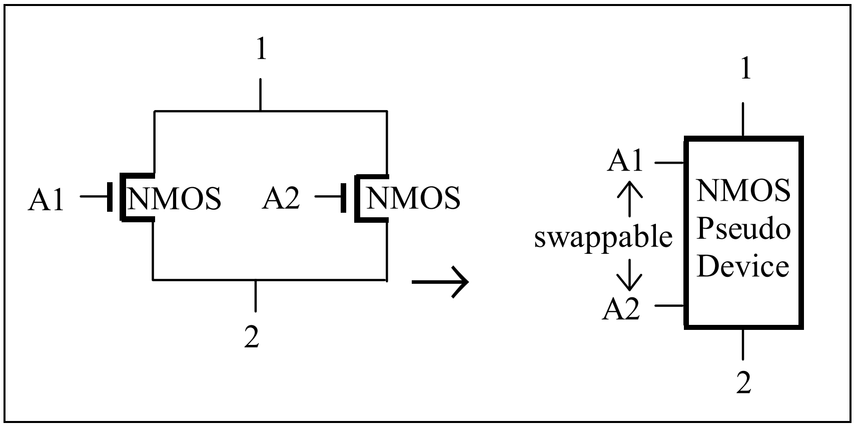


Figure 117: Parallel logic collapse.

enabled for any type of three or four terminal device. The collapse is performed only on devices of the same category, type, and model. The devices must be connected to the same substrate.

The criteria for matching and verifying this type of pseudo device is the same as that used for series logic collapses.

Pull-Up and Pull-Down Pseudo Devices

This optional transformation allows circuits which are logically equivalent to be matched from the two netlists even when the topology is very different. The first transformation on the circuit in Figure 118 is caused by the COLLAPSE-_SERIES_LOGIC_MOSFETS = YES and COLLAPSE_PARALLEL_LOGIC-_MOSFETS = YES control file options. The series collapse of the two pseudo devices in the center of Figure 118 to create the pseudo device on the right is enabled by the TAKE_CARE_OF_LOGIC_EQUIVALENCES_WHILE-_MATCHING = YES control file option. This creates a pseudo device with groups of swappable gates. The circuit in Figure 118 will then match successfully to the circuit in Figure 119.

While this type of transform is most useful for CMOS digital designs, it can be used for any three or four terminal devices.

The collapse is done only for devices using the same model. The devices must all be connected to the same substrate.

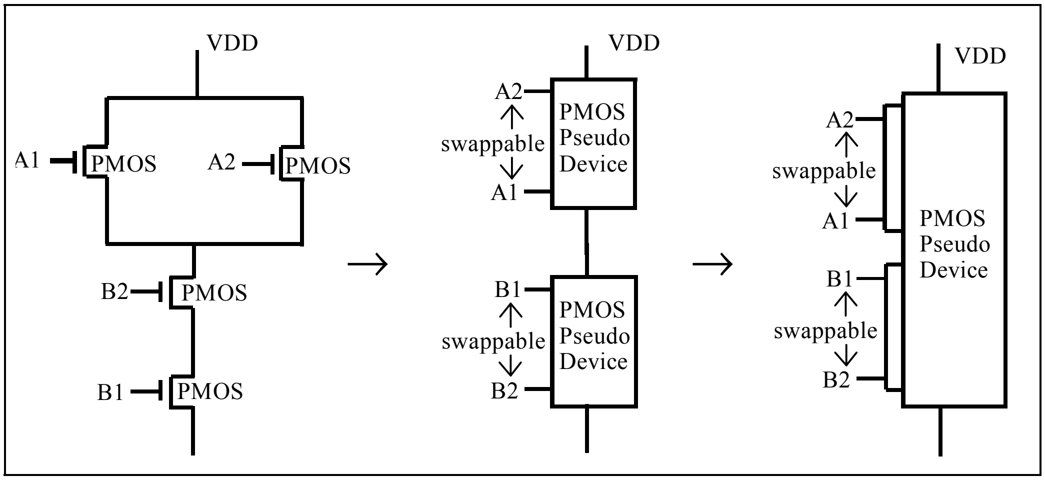


Figure 118: Circuit A Pull-up pseudo device collapse.

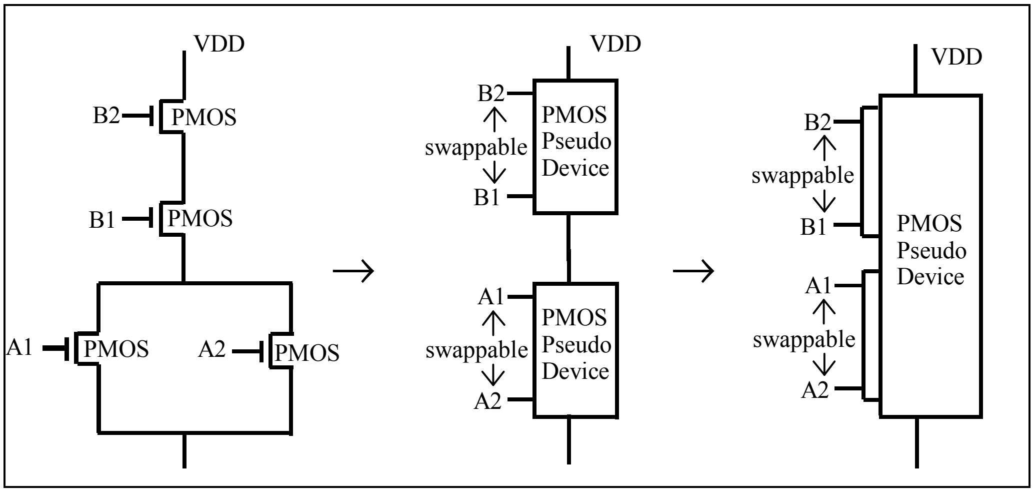


Figure 119: Circuit B Pull-up pseudo device collapse.

True logic errors, like the one shown in Figure 120, will be caught by the LVS. If the TAKE_CARE_OF_LOGIC_EQUIVALENCES_WHILE_MATCHING = YES control file option is used, the LVS collapses circuit A and circuit B to pseudo devices. The pseudo devices will not be matched because nodes A2 and A3 are misconnected.

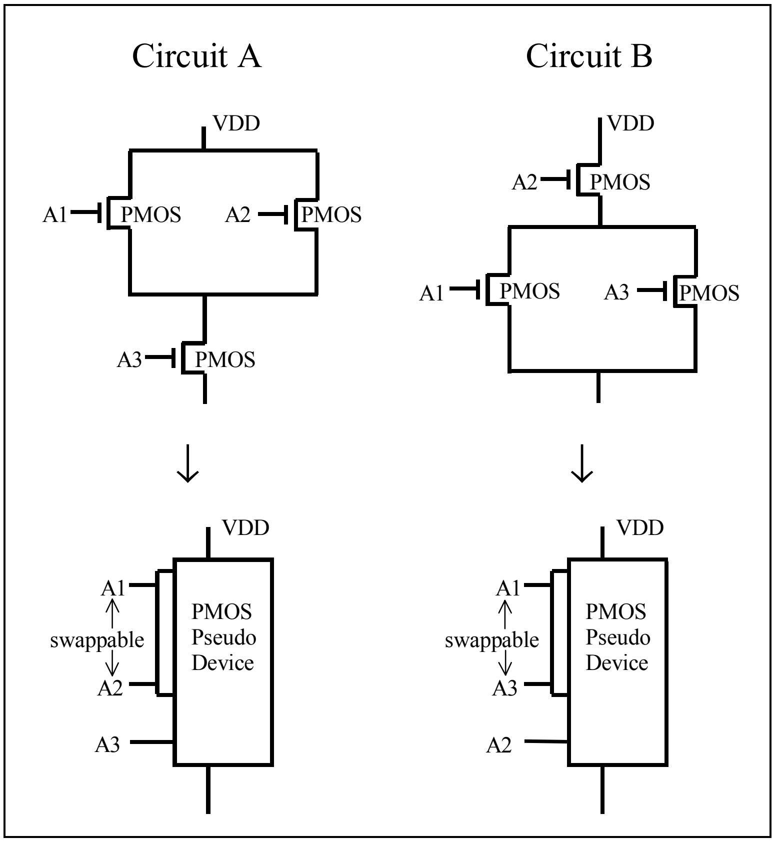


Figure 120: Pseudo devices which will not match.

When a pseudo device fails to match, the entire pseudo device is listed in the unmatched devices report. (The control file option OUTPUT_FILE_OF-_UNMATCHED_DEVICES_AND_NETS defines the name of the unmatched devices report.) The original devices included in each pseudo device will also be listed here.

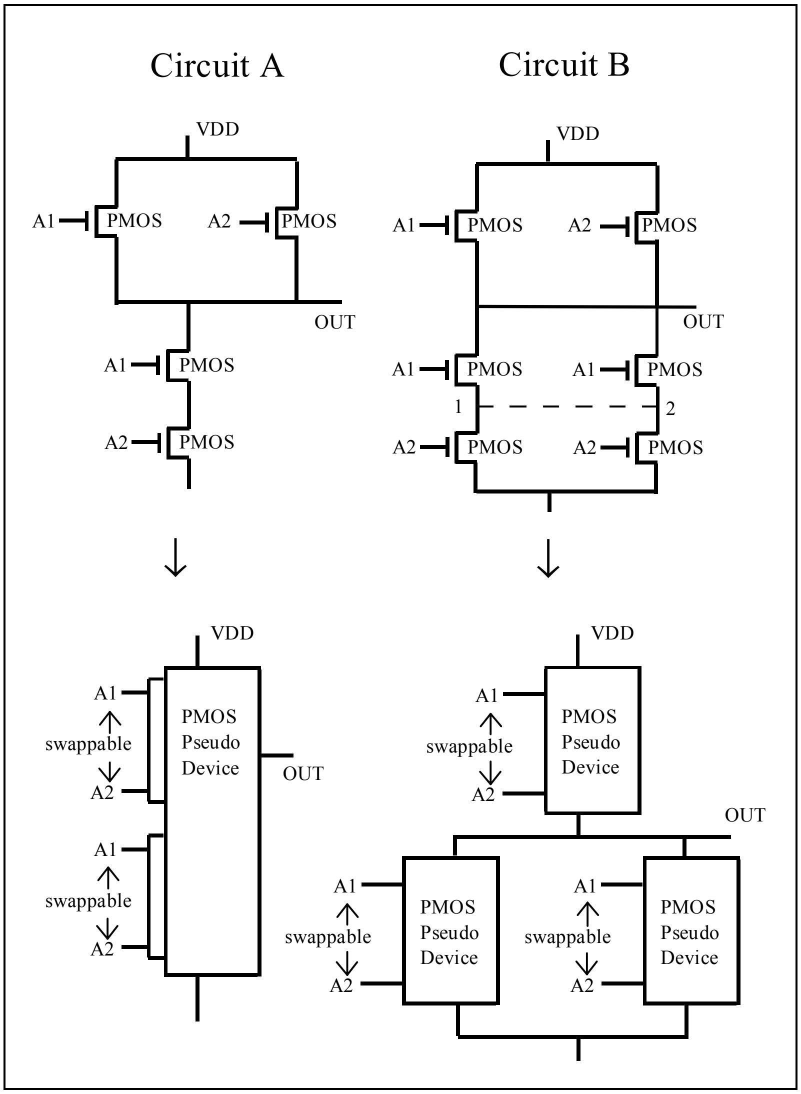


Figure 121: Open will prevent creation of pseudo device.

For another example, see Figure 121. Even when TAKE_CARE_OF_LOGIC-_EQUIVALENCES_WHILE_MATCHING is set to YES, the LVS would fail to match circuit A and B.

You can overcome this problem in three different ways:

allow chain merges as described on page 320,

or

connect nodes 1 and 2 as shown by the dotted line,

or

virtually connect nodes 1 and 2. (See **Assigning Virtual Connections** on page 358 for details on how to virtually connect nodes.)

Any of these three methods will allow the pairs of devices connected to A1 and A2 to form parallel merges and then form the single pseudo device.

If you have disabled all device collapses by setting the COLLAPSE_SERIES-_*device*S and COLLAPSE_PARALLEL_*device*S options to NO, and the only difference between the two netlists is that nodes 1 and 2 are unconnected in one netlist, the discrepancy may be difficult to determine from the error reports. The LVS might fail to match other circuits because no logical equivalence has been attempted.

Parameter Value and Signal Order Verification of Collapsed Circuits

When devices of the same size are collapsed to form a pseudo device, the LVS assumes that the gates can be connected in any order. In this case, the LVS will not issue error messages if gates are connected in different orders in the matched pseudo devices in either netlists. The LVS will verify that the size of the devices is the same in both netlists.

These collapses must be enabled with the COLLAPSE-_DISSIMILAR-_SIZED-_*device*S=YES control file option in addition to the other logic collapse options discussed earlier.

When the LVS performs a series collapse of devices with different sizes, the gates are not truly swappable. Look at Figure 122. The width of the device attached to signal B is 2. The width of the device attached to signal C is 4. The device attached to signal C may be wider because this signal arrives a bit later than signal B. Or maybe the circuit is designed this way so that the device attached to GND is larger. In any case, if the signals or device sizes are swapped in the circuit in the other netlist, the circuits will not function in an identical manner.

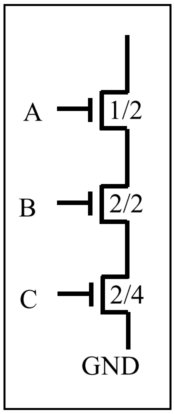


Figure 122: Series logic collapse of dissimilar sized devices.

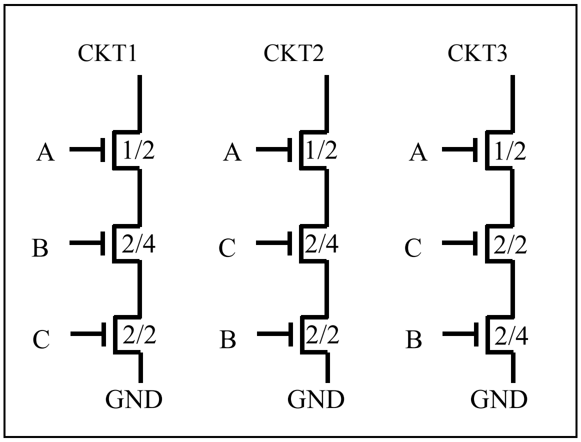


Figure 123: Three circuits which match the circuit in Figure 122.

Now look at the three circuits in Figure 123. If you enable the collapse of dissimilar sized devices, the LVS will match any of these circuits to the circuit in Figure 122 since the gates of collapsed devices are swappable. The circuits will not be listed in the unmatched devices report since they have been matched successfully. However, the LVS will report errors for each of these circuits in the parameter error report and the matched devices report.

The file names of these reports are determined by parameters in the control file. See pages 371 and 382.

Circuit 1 will result in an error message for the devices attached to signal B and signal C. These errors are indicated with the phrase "PARAMETER ERROR". The order of the gates is the same as the circuit in Figure 122. However, the size of the device attached to GND is different in each circuit.

In circuit 2, the size of the device attached to signal B is the same size as the device attached to signal B in Figure 122. The sizes of the devices attached to signal C are also the same in both circuits. However, the order of the gates is different in the two circuits. The LVS will issue a "GATE SIGNAL SWITCHED" error message for each device in the parameter error file. You may consider this a false error for your circuit, but if the width of the device attached to GND is supposed to be larger than the other devices, this is a real error.

When circuit 3 is matched, the LVS will issue both a "PARAMETER ERROR" and a "GATE SIGNAL SWITCHED" error message for both devices.

When pseudo devices are formed from series collapses of collapsed circuits, it may not be obvious which device in the pseudo circuit caused an error message. The LVS will report the gate signal position of each device in the pseudo device. To understand how the gate signal position is determined, look at Figure 124. The gate signal position is determined relative to the starting node of the circuit. (The starting node of a circuit may be the reverse of what you would expect from looking at a schematic. There is no way for the LVS to determine "which way is up".)

The first number of the gate signal position represents the order of the gates at the highest-level series collapse. As you move away from the starting node, the gate signal position is incremented. If a collapsed device at this level is formed from another series collapse, the gate position of this second level collapse is indicated to the right of the first position. The positions are delimited with a '.'.

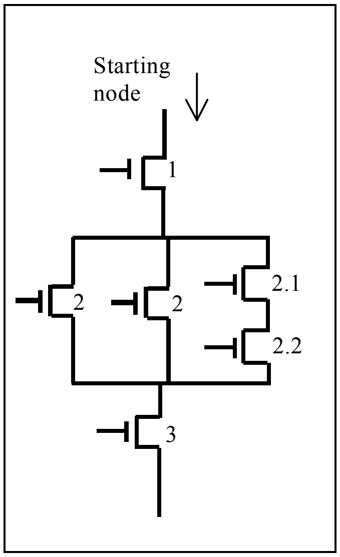


Figure 124: Collapsed circuit with gate signal positions indicated.

Multiple Emitter or Collector Devices

None of the schematic netlist languages valid for the LVS support a device primitive for a multiple emitter or collector bipolar transistor. You must always simulate such devices as collections of single emitter or collector devices. This causes a real problem when attempting to match a layout netlist with such a device. However, the LVS program provides two simple solutions.

The easiest method is to allow the LVS preprocessor to automatically split multiple emitter or collector bipolar devices in a layout netlist into single bipolar devices before comparison. You must create a *.LAYMODEL statement which uses a device type with a number suffix indicating the number of emitters or collectors.

Example: ***.LAYMODEL MULT_EMITR_NPN NPN3**

When this device model exists in the layout netlist, and you have used an NLE rules file which creates MULT_EMITR_NPN devices, the LVS will be able to split such devices into several single emitter NPN devices. This will allow the circuit to match with the schematic netlist representation.

If the layout netlist contains a device such as:

```
Q1  1  2  3  4  5  MULT_EMITR_NPN
```

this device will be expanded, before comparison, into 3 NPN devices as seen below

```
Q1!1  1  2  3  NPN
Q1!2  1  2  4  NPN
Q1!3  1  2  5  NPN
```

The character used to separate the original device name from the number suffix ('!' in this example), is determined by the control file option SPECIAL-_CHARACTER_FOR_PRINTING_DEVICES_IN_A_DEVICE_CELL-_INSTANCE.

If this method does not meet your needs, you can use SUBCKT statements in the layout netlist device models. SUBCKT device models expand a single layout device into several other devices.

Example:

```
*.format  pspice

*.laymodel  S2_LPNP   SUBCKT MULT_COLL_PNP2
*.laymodel  S3_LPNP   SUBCKT MULT_COLL_PNP3
*.laymodel  S4_LPNP   SUBCKT MULT_COLL_PNP4
*
.subckt  MULT_COLL_PNP2  1  2  3  4
q1 1 3 4 PNP
q2 2 3 4 PNP
.ends

.subckt  MULT_COLL_PNP3  1  2  3  4  5
q1 1 4 5 PNP
q2 2 4 5 PNP
q3 3 4 5 PNP
.ends

.subckt  MULT_COLL_PNP4  1  2  3  4  5  6
q1 1 5 6 PNP
q2 2 5 6 PNP
q3 3 5 6 PNP
q4 4 5 6 PNP
.ends
```

The LVS will expand devices which use SUBCKT device models in the layout netlist in the same manner as .SUBCKT statements in a schematic netlist. They are written using SPICE, PSPICE, HSPICE, or CDL syntax. If you use subcircuits in your layout netlist, you must tell the LVS which syntax to use when parsing the SUBCKT models.

In an LVS comparison, where the layout netlist is the second netlist on the LVS command line, the spice language syntax is specified by the control file option SECOND_SCHEMATIC_FILE_FORMAT. In the layout netlist, the control file option can be overridden by using the *.FORMAT statement. The *.FORMAT statement overrides the option in the control file and the /f2 option on the LVS command line. The *.FORMAT statement must occur before the SUBCKT models.

The syntax of the *.FORMAT statement is:

***.FORMAT (CDL | PSPICE | HSPICE | SPICE)**

SUBCKT models can be nested using X statements. All subcircuits must expand into devices which have already been defined. Parameter passing is allowed according to the spice syntax selected.

Example:

```
*.format pspice
*.laymodel  npn  subckt  mult_npn
*.laymodel  npntype  npn
.subckt mult_npn 1 2 3 4 5 6 7 PARAMS: area=4
q1 3 1 2 npntype {area}
q2 3 6 5 npntype {area}
q3 3 6 7 npntype {area}
.ends
```

Pad Connection Verification

The UNCONNECTED rule in the NLE can verify that pads which are unconnected by design are not accidentally shorted to an unlabeled net.

You should take special care, when preparing your chip for LVS verification, that you have a method in place to verify pad connections. Unconnected pads, or pads which short to nets by accident, can turn a groundbreaking design into wasted silicon. Many designers have learned that simply assuming that the pads are connected correctly is very dangerous.

This problem is often overlooked because pad connections are not reflected in the schematic netlist. **Even if your chip passes an LVS check, if your schematic netlist does not include pad connections, the pad connections have not been checked.**

You may be tempted to verify that nets attach to the correct pads by simply adding node labels in the layout where the wires attach to the pads. However, **the LVS does not automatically match nets by name**. The LVS uses the node properties of a net (i.e. what devices the net connects to) to match nets in the two netlists. The node names are usually ignored.

The best way to verify pad connections is to treat each pad in the layout as a device. When each pad is a device, the pad connections will be verified as carefully as the other devices in your design. In addition, the LVS will automatically verify that the names of matched nets are identical in both netlists when those nets are connected to pad devices.

When we define pads in the layout as devices, we can also identify different types of pads as different types of devices and verify that a net connects to the right type of pad. For instance, if you have a clock net that must connect to a pad with the appropriate speed constraints, the LVS will insure that this is the case just as it verifies that a net connects to an NMOS device rather than a PMOS device.

Defining Pads in Schematic Netlist

You do not have to add pad devices to the schematic netlist. However, you must identify which nets connect to pads. This is accomplished very easily with the *.PINS statement. For each net that connects to a pad, list the name of the net followed by a colon (':') then the appropriate pad type character.

The valid pad types are shown in Figure 125. (These characters are the same as those used by the DRACULA program.) The LVS will verify that the pad types of matched nets are identical in each netlist.

Type character	**Pad type**
P or **S**	Power pad
G	Ground pad
C	Clock pad
I	Input pad
O	Output pad
B	Bi-directional pad

Figure 125: Valid pad types.

Example: ***.PINS VDD:P SIGNAL1:I**

When this line is included in the schematic netlist, net VDD is connected to a power pad and SIGNAL1 is connected to an input pad.

You can have multiple *.PINS statements in a netlist. They should not be located inside of a subcircuit definition.

Example:

***.PINS VDD:P**
***.PINS TESTPOINT1:B**
***.PINS CLOCK:C**

When these statements are used in a schematic netlist, net "VDD" is connected to a power pad. The net "TestPoint1" is connected to a bi-directional pad. "CLOCK" is connected to a clock pad.

All nets which occur in a *.PINS statement will be prevented from disappearing from the netlist due to device collapses.

Defining Pad Devices in Layout Netlist

See Node Labels beginning on page 138 to learn more about LVS node labels.

There are three steps to recognizing pad devices in the layout netlist. First you must label each pad with a text component identical to the net name in the schematic netlist. Next, you use layout layers and NLE rules to extract pad devices from the layout. Finally, you define pad types for each unique type of pad device with *.LAYMODEL device models.

Adding labels to each pad insures that each net is not only connected to a pad, but is connected to the right pad. The LVS will warn you when you have unlabeled pads in your layout. A pad is labeled in the layout by adding a text component.

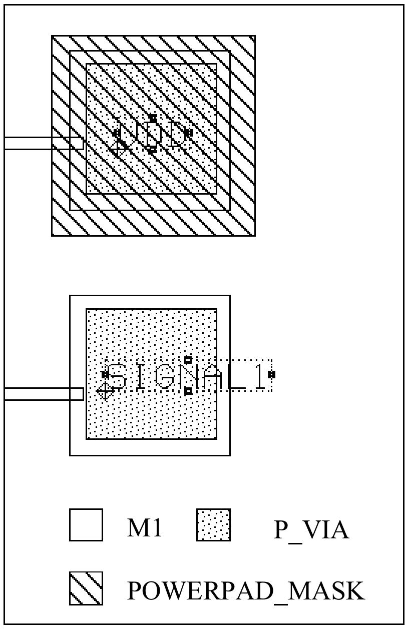


Figure 126: Two pads. Note that text component labels are selected to show you that origins are covered by p_via shapes.

- The text component must be in the top-level cell.
- It must be added on the same layer used to recognize the device, or be copied to that layer by the NLE.
- The origin of the text component must be covered by the shape that is used to recognize the pad device.

For the NLE to extract pad devices from the layout, you must identify an id-layer that the NLE rules file will use to recognize which shapes represent pad devices. If you want to recognize more than one type of pad, you must process the layers so that each unique type of pad has a unique id-layer to identify it. You specify each unique pad id-layer in the device rule for that type of pad. You must then identify which layer to use as the terminal or pin of the pad. The pin layer is usually the wire layer that is used to connect the pad.

The following NLE rules file uses a via layer used only by pads (p_via for passivation via) to identify pad devices. A dummy layer called powerpad_mask is used to separate the via layer into two device id layers to separate power pads from signal pads. The pad text component labels are on the pad_text_in layer. These labels are moved to the device id layers with the ATTACH TEXT rule.

Example:

```
input layer        {
        10 m1
        12 m2
        13 via1
        20 p_via                 !via layer used only on pads
        110 powerpad_mask   !dummy layer used to identify power pads
        120 pad_text_in         !layer with text components used to label pads
}
scratch layer  {
          iopad_id
          powerpad_id
          pad_text
}
connect  m1  m2  by  via1

!Separate power pads from signal pads with dummy layer powerpad_mask
powerpad_id = p_via and powerpad_mask
iopad_id = p_via and not powerpad_mask

!Text components on pad_text_in layer are used to label both types of pads
!Text must be copied to scratch layer for attach text rule
pad_text = pad_text_in
attach text  pad_text  powerpad_id  iopad_id

device  io_pad  id=iopad_id  {
          M1       1/node
          pins = M1
}
device  pwr_pad  id=powerpad_id  {
          M1       1/node
          pins = M1
}
```

Learn more about the DEVICE rule beginning on page 113.

The device rules above are written so that a shape on the pad device id layer must touch a single node on the M1 layer to be considered a pad. The node number of the touching shape on M1 will be stored as the net connection of each pad device.

Now you must add a device model to the layout netlist model file for each unique type of pad device. These device models define the devices as pad devices and assign the appropriate pad types with the TYPE keyword.

Example:

```
*.laymodel  io_pad      pad  type=b
*.laymodel  pwr_pad    pad  type=p
```

Any net that connects to an io_pad device will be connected to a bi-directional pad. Any net that connects to a pwr_pad device will be connected to a power pad.

The schematic netlist does not need to contain devices that correspond to pad devices. However, each net in the schematic netlist that connects to a pad should be included in a *.PINS statement which defines the pad type for the net.

Example:

```
*.PINS  VDD:P
*.PINS  SIGNAL1:B
```

Diagnosing Pad Misconnections

The name of the comparison summary is listed on the console by the LVS. The comparison summary lists the file names of all other reports.

The comparison summary (usually "RESULTS.LVS") will list the number of pad devices found in the layout across from the caption "NUMBER OF PAD DEVICES". The number of pad devices in a schematic netlist is always zero. The total number of unmatched pads is also listed. The total number of pins defined in a schematic netlist is listed a little further down in the same report across from the caption " NUMBER OF I/O PINS ".

Details on each matched pad is provided near the bottom of the matched device report. Details on each pad that is unmatched, or is matched with a discrepancy in pad types, is listed in the unmatched devices report. Search for the phrase "PAD CONNECTIONS" to locate these summaries.

The LVS will list unlabeled pads (when they connect to a valid net) in the unmatched devices report. You should label all pads, or prevent them from being recognized as pads. (See Ignoring Testpoint Pads on page 342.)

Variable Pad Types

If a layout pad device can match several different types of pads, you can use more than one pad type character in the *.LAYMODEL statement.

Example: ***.laymodel in_pad pad type=cib**

This device model will allow nets attached to in_pad devices in the layout to match with nets in the schematic netlist which are defined as pad types C, I, or B. All of the following nets in the schematic netlist will match correctly if they are attached to in_pad devices in the layout:

Example: ***.PINS NET1:C NET2:I NET3:B**

You cannot specify multiple pad types in a *.PINS statement. The schematic netlist must be unambiguous about the pad type for each net.

Pad Protection Circuitry

If your layout contains pad protection circuitry, but your schematic netlist layout does not, you should mask out the protection devices in the layout with a dummy layer so they are not included in the layout netlist and then wind up in the unmatched devices report.

If your protection devices are not recognized by your NLE rule set (e.g. fingered static protection devices, or poly wires without dummy shapes to classify them as resistors), you need take no special steps to insure that they are ignored by the NLE.

You must use caution to insure that the unrecognized protection devices do not cause unintentional opens or shorts. Look at Figure 127. The gate layer must be removed from the diffusion layer as it is for the rest of the chip, or the MOS devices will result in shorts from the signal to VSS and VDD. If the resistor is removed from the signal wire layer, the signal will not be connected to the pad, and the LVS may report an open.

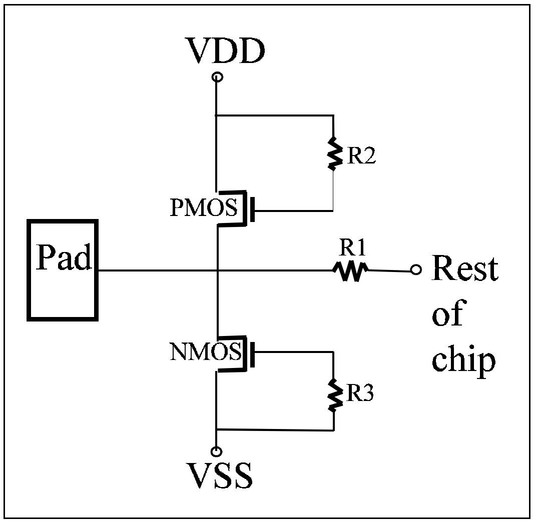


Figure 127: Schematic of pad with protection circuitry.

Example:

```
ngate_all  = ndiff and poly
n_src_drc = ndiff and not poly

!remove protection devices from id-layer
ngate = ngate_all and not pad_mask

transistor nmos id = ngate {
          gate = poly
          s$d = n_src_drn
          bulk = p_well
}
```

You must also insure that the layer you use for the pad device id-layer touches the wire layer of the nets connected to the devices in the rest of the layout. Be careful that real opens will not be prevented from being found. It is the connection of the net to the pad device id-layer that will be verified by the LVS.

Ignoring Testpoint Pads

If your layout contains testpoint pads that you do not want to label and verify, you should prevent these pads from being included in the layout netlist. Unlabeled pads that connect to other devices are considered errors by the LVS and they will be listed in the unmatched devices report.

You can prevent the NLE from recognizing testpoint pad devices in different ways. If you recognize your pads using a dummy layer as the pad device id layer, simply insure that the testpoint pads are not covered by the dummy layer.

If your testpoint pads have different dimensions than your real pads, you can remove the testpoint pads from the pad device id layer with the IS_BOX rule.

Example: **pad_id = isbox (passivation_via, (50,50))**

The above rule will copy only pads that are exactly 50 units square to the pad_id layer. Other shapes on the passivation_via layer are ignored.

If your pads cannot be differentiated by size, and you use a design layer (e.g. passivation_via) as the pad device id layer, you can mask these pads with a new dummy layer with a NLE rule similar to:

Example: **pad_id = passivation_via and not testpoint_mask**

Parameter Calculation

See Device Merges on page 318 for the calculation of dimensions of merged MOSFET devices with dissimilar sizes.

The LVS compares the parameter values of matched devices if the appropriate option in the control file for that device category is set to YES. (See Figure 128 below.) This can be overridden for specific device models by using the PARAM=NO, ALLMATCH=NO or ALL=NO override parameters in the device models of **both** netlists.

All parameter errors are listed in the file specified by the OUTPUT_FILE_OF-_DEVICES_WITH_PARAMETER_ERRORS option in the control file.

Device category	**Parameters checked**	**Control file option**
MOSFET	Length and Width	MATCH_MOSFET_PARAMETERS
JFET	Area	MATCH_JFET_PARAMETERS
BIPOLAR	Area	MATCH_BIPOLAR_PARAMETERS
GaAsFET	Area	MATCH_GAASFET_PARAMETERS
DIODE	Area	MATCH_DIODE_PARAMETERS
RESISTOR	Value= R_CONTACT + (OHMS_PER_SQUARE ∗ (Length/Width)) or R2_CONTACT /(Width +R2_WIDTH) + (OHMS_PER_SQUARE ∗ (Length/Width))	MATCH_RESISTOR_PARAMETERS
CAPACITOR	Value= (C_PERIMETER * Perimeter) + (C_AREA * Area)	MATCH_CAPACITOR_PARAMETERS
INDUCTOR	Value	MATCH_INDUCTOR_PARAMETERS
TXLINE	Value	MATCH_TXLINE_PARAMETERS

Figure 128: Parameters verified by the LVS.

See page 240 for an example of how LOFFSET, WOFFSET and BENDS_CR affect the value of a device.

In the layout netlist, the Length, Width, Area, and Perimeter values are all calculated by the NLE. The R_CONTACT, R2_CONTACT, R2_WIDTH, OHMS_PER_SQUARE, C_PERIMETER, C_AREA, and tolerance values are set by *.LAYMODEL device models. If any device has a *.LAYMODEL device model which uses the LOFFSET, WOFFSET, or BENDS_CR parameters, these dimension corrections are applied **before** device calculation and comparison.

Resistors

Contact resistance for resistors can be accounted for in two different ways. If you provide R_CONTACT in the *.LAYMODEL device model, that will be the value used for contact resistance, and the total resistance will be:

$$\textbf{R\_CONTACT} + (\,\textbf{OHMS\_PER\_SQUARE} * (\textbf{Length/Width})\,)$$

If you do not supply R_CONTACT in the device model, but you do supply R2_CONTACT, then the contact resistance used is inversely proportional to the width of the resistor. In this case, the contact resistance is:

$$\frac{\textbf{R2\_CONTACT}}{\textbf{Width}}$$

where Width is the width of the resistor. The total resistance will be:

$$\frac{\textbf{R2\_CONTACT}}{\textbf{Width}} + (\,\textbf{OHMS\_PER\_SQUARE} * (\textbf{Length/Width})\,)$$

Example:

```
*.LAYMODEL  RES1  RES   R2_CONTACT=10
*+                      OHMS_PER_SQUARE=100
```

If the length of a resistor is 20 and the width is 4, the resistance would be:

$$\frac{10}{4} + 100 * \frac{20}{4} = 502.5$$

If you require the contact resistance to be more accurate, you can use the R2_WIDTH parameter to account for contacts that are less wide than the resistor. When R2_WIDTH is not supplied in the device model, it defaults to 0. You should set R2_WIDTH to a **negative** value to subtract a width offset from the width of the resistor so that an accurate contact width can be used to calculate resistance.

When you do supply R2_WIDTH, you must also supply R2_CONTACT. When both of these parameters are specified, the total resistance will be:

$$\frac{\textbf{R2\_CONTACT}}{\textbf{Width +R2\_WIDTH}} \;\textbf{+ (OHMS\_PER\_SQUARE} * \textbf{(Length/Width))}$$

Look at Figure 129. Most technologies require a gap between the side of the resistor and the contact via hole. Here we have shown the gaps as w1 and w2. (w1 is always equal to w2.) To accurately calculate the contact resistance, the sum of w1 and w2 is subtracted from the width of the resistor to calculate the contact width.

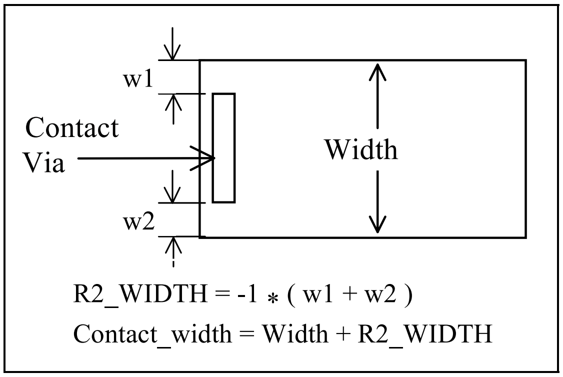


Figure 129: Resistor with contact width different than resistor width.

The contact does not need to look exactly like the one shown in Figure 129. You may use an array of via holes. You should set R2_WIDTH to a **negative** value that adjusts the width of the resistor to provide an accurate contact width.

Capacitors

Capacitance for a capacitor device is calculated by the LVS using the perimeter and area recognized by the NLE. The formula for capacitance shown in Figure 128 is:

(C_PERIMETER * Perimeter) + (C_AREA * Area)

If your layout contains a CAP1 device with dimensions 2 x 4, the NLE will store a value of 12 for its perimeter and 8 for its area. If the model in the layout netlist is defined with the following statement:

Example: ***.LAYMODEL CAP1 CAP C_PERIMETER=1p C_AREA=2p**

the value of the device will be

$$\mathbf{(1e^{-12} * 12) + (2e^{-12} * 8) = 26e^{-12} = 26p}$$

Inductors and Other Devices for Which the NLE Cannot Calculate a Value.

The NLE does not calculate the value of inductor devices from the geometry in the layout. You can define the value of an inductor in the layout in two ways:

add a label of the form "VALUE=*value*" on the device id layer,

or

define a default value on the *.LAYMODEL statement for a specific model of an inductor device.

You can use either of these methods with other categories of devices. The default value on the *.LAYMODEL card will be used only when the NLE does not calculate a value and the label method is not used. The label method will override any other parameter value measurement or default.

See Node Labels beginning on page 138 to learn more about the placement and processing of labels.

To use a label to define a parameter value for a device, add a text component on a layer used to assign node labels to the device id layer. (This can be the device id layer itself, or a layer used in the ATTACH TEXT NLE rule.) The format of the label must be:

val_string=value

Where *val_string* is one of the following keywords:

AREA
VALUE
L
W

Scaling and Tolerances

Use the SCALE_*device*_LENGTH_AND_WIDTH and SCALE_*device*_VALUE command file options to avoid false errors due to the use of different units in the two netlists.

If the capacitor in the example on page 346 were matched with the following schematic netlist device:

C1 1 2 CAP1 26

The device would fail the parameter value check since $26e^{-12}$ does not equal 26. You do not have to edit the schematic netlist to correct this problem. Simply use the control file option:

SCALE_CAPACITOR_VALUE = 1e-12

See page 233 to learn more about changing the default tolerance for specific device models.

The parameter values of devices do not need to match exactly. The default tolerance is .0005, or .05%, of the device values in the **first netlist**. You can override this default tolerance on the *.LAYMODEL (or *.SCHMODEL) model statements in the **second netlist** typed on the LVS command line (usually the layout netlist).

If the capacitor mentioned above with the value of 26 used the device model:

Example:

```
*.LAYMODEL  CAP1  CAP  C_PERIMETER=1  C_AREA=2
*+VALUETLR = .01
```

and the device in the schematic netlist was

```
C1  1  2  CAP1  26.2
```

The device would still match since

26.2 - 26 = .2 = device_error < error tolerance = .01 * 26.2 = .262

Advanced Uses of Node Labels

See **Node Labels** in **NLE Circuit Recognition** and **Review of Node Labels** in **Layout Netlists** for the basics of adding labels to the layout.

Node labels in the layout can be used to name nets and devices so the LVS reports are somewhat easier for you to follow. In addition these labels can direct the LVS comparison in several ways. Labels in the layout can be used when assigning forced points of correspondence between the layout and the other netlist. In addition, the control file assigns special characters for use in node labels which control how the layout is interpreted. (The special characters are not stripped from the name given to the net. If you use the node name in the node correspondence file you should include the special character in the name.)

Text components in the layout must be in the top level cell unless they are global nets. Global nets are indicated with a colon ':' as the suffix on the node label. If node labels ending in a single colon ':' are contained in nested cells, they will be processed only if the control file option RECOGNIZE_GLOBAL_TEXT_IN-_SUBCELLS=YES is used. Global nets ending with a double colon '::' will always be processed.

Forced Points of Correspondence

Labels in the layout can be used to force points of correspondence between nodes in the layout and nodes in the other netlist. These node equivalences can be made before the LVS proceeds with its matching algorithms (initial equivalences), or later on if the LVS comes to a point during the comparison where it can no longer make progress (optional equivalences). When initial equivalences are made incorrectly, they can prevent the LVS from matching anything. When optional equivalences are used, a mistake is less likely to prevent any progress in the matching algorithms.

To assign node equivalences, the control file option LAYOUT_TEXT_MODE must be set to EQUIV or AUTO. When EQUIV is used, you must prepare a node correspondence file as described on page 355. The AUTO mode will take each labeled node and attempt to match it to an identically named node in the other netlist.

In either case, the node equivalences can be initial points of correspondence, or held in reserve until after the matching algorithm has made no progress for a certain number of passes.

+/- flags in the node correspondence file can override this control file option.

If the control file option USE_EQUIVALENCES_FOR_INITIAL_MATCHING = YES is used, all equivalences are used as initial points of correspondence. When this option is set to NO, the node equivalences are held in reserve to be used when the LVS can make no progress. We recommend setting this option to NO unless you have a highly symmetric circuit. There is little or no speed improvement when using initial points of correspondence, and errors in the node equivalences can hinder rather than help matching.

The number of passes the LVS will make before using the node equivalences to make progress is set by the control file option SET_NO_PROGRESS_LIMIT.

Virtual Connections in the Layout

Virtual connections in the schematic netlist between nets with different names can be made with the *.VIRTUAL command.

Virtual connections in a layout netlist indicate that two or more nets that are not electrically connected should be considered the same node. This is especially useful when you are verifying subcircuits that use material in a higher level cell to connect nodes. For example, if your subcell has two separate ground nodes which both connect to a ground bus in a higher level cell, you do not have to add metal to connect them for circuit comparison. You need only virtually connect them for the LVS to match the circuit.

These virtual connections can be made using labels in the layout with a special character defined in the control file as the suffix. (See page 358 for other methods of assigning virtual connections that do not require adding labels.)

The SPECIAL_CHARACTER_FOR_VIRTUAL_CONNECTIONS option (':' in the sample control file) assigns a character which is used as a suffix to identify nets which the LVS should assume are virtually connected. For instance, you use several separate VDD nets in your design that will be connected later, or at a higher level. If you label them "VDD:" in the layout, the LVS will act as though they are electrically connected.

When nets which have been virtually connected are listed in the reports, their node numbers will all be listed with the syntax *node_number*[V,*n*] , where *n* indicates the number of nets virtually connected.

If not used correctly, this can lead to opens not being found in the layout. **We suggest that you be very careful to remove all use of this feature before a final test on a chip.** The easiest ways to insure that no virtual connections are made is to use the control file option ENABLE_VIRTUAL_CONNECTIONS = NO, or the LVS command line option "/V=NO".

Node Labels Which Prevent Device Collapses

The control file option ENABLE_NO-_COLLAPSE-_OF_DEVICES enables this special processing.

Two special characters are defined in the control file to prevent nets from disappearing when devices are merged or collapsed. The control file option SPECIAL_CHARACTER_FOR_NO_COLLAPSE_OF_DEVICES_CONNEC-TED_TO_A_NET defines a character used to preserve all devices attached to a net. The value of this parameter in the sample control file provided with the LVS installation is '$'. When a net is labeled with a string using a prefix of '$', no devices connected to that net will be collapsed into series, parallel, or pseudo devices. This can be useful for critical nets, such as clocks, where you want to verify that every device attached to that net is exactly the same in the two netlists.

The control file option ENABLE_NO-_COLLAPSE-_OF_A_NET enables this special processing.

The control file option SPECIAL_CHARACTER_FOR_NO_COLLAPSE_OF-_A_NET is used to prevent a net from disappearing due to device collapses. The sample control file assigns '#' for this character. Devices attached to a net labeled with this character may be collapsed as long as the collapse does not remove the net from the netlist. For example, if you add a text component with the string "#PAD1" to a pad on a chip, no device collapses or merges will be performed that make that net disappear from the netlist.

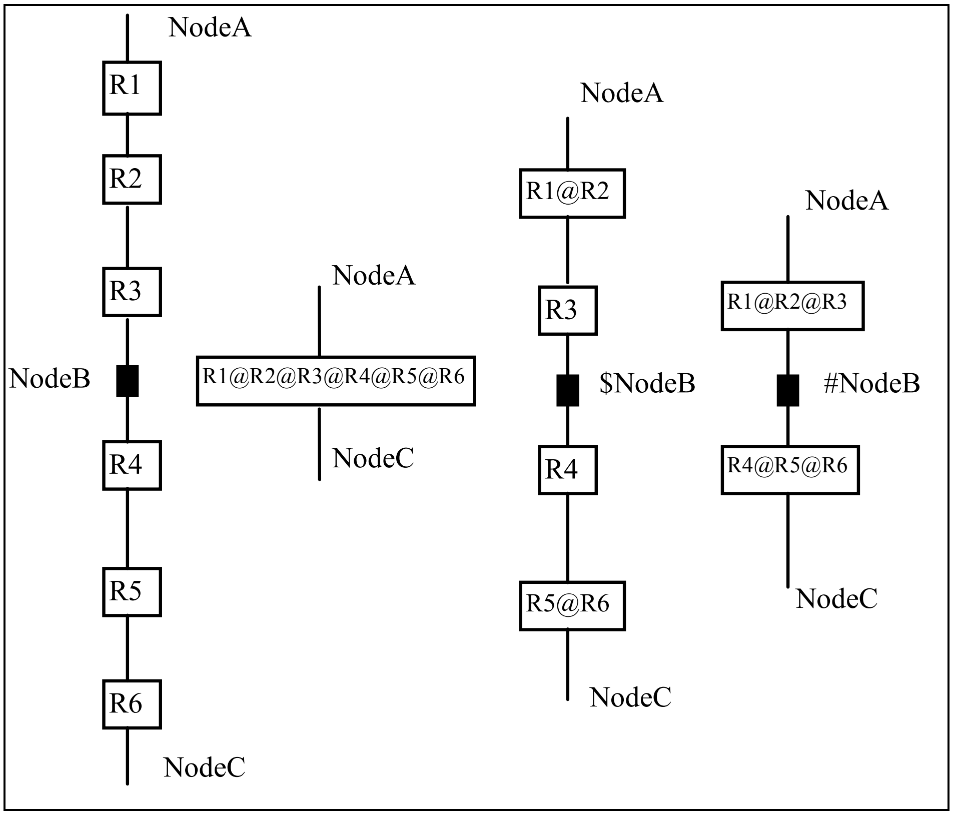


Figure 130: Effect of special characters in net labels.

To see the effect of these two characters in node labels, look at Figure 130. If you label a node using no special character prefix, all 6 resistors will be collapsed into one resistor as long as the MERGE_SERIES_RESISTORS=YES option is used in the control file. In this case, NodeB vanishes completely from the netlist. When the '$' prefix is used in the net label, none of the devices attached to the net are collapsed in the series device. Resistors R1 and R2 are collapsed, but not R3. When the '#' prefix is used, NodeB is preserved, but the devices on either side are collapsed.

The *.NOCOLLAPSE statement in a netlist has the same effect as a node label using the special character SPECIAL_CHARACTER_FOR_NO_COLLAPSE-_OF_DEVICES_CONNECTED_TO_A_NET. This is the only way to prevent collapses of any devices connected to a specific net in the schematic netlist.

Example: ***.NOCOLLAPSE CLOCK**

Using this statement in a schematic netlist will prevent device collapses on all devices which connect to the net with the name "CLOCK".

The *.PINS statement in a netlist has the same effect as a node label using the special character SPECIAL_CHARACTER_FOR_NO_COLLAPSE_OF_A-_NET. This will prevent the disappearance of a specific net in the schematic netlist due to device collapses or merges. In addition, all nets in the i/o list of the top-level subcircuit will be treated as though they are included in a *.PINS statement.

Example: ***.PINS NETA NETB**

.SUBCKT TOP NETC NETD

If TOP is the top-level subcircuit in a schematic netlist, these statements will prevent the disappearance of the nets NETA, NETB, NETC, and NETD from device collapses. The ASCII part of the layout netlist should include a *.PINS statement for all four nets or have them labeled with node labels which begin with the prefix defined by SPECIAL_CHARACTER_FOR_NO_COLLAPSE-_OF_A_NET. Otherwise, different device collapses in each netlist may prevent the two netlists from matching.

Using a Node Correspondence File

Purpose of the File

The LVS command line option /E will cause the LVS to halt when syntax errors are found in the node correspondence file. When this option is not used, the LVS will still print error messages in the log file, but it will continue despite the errors.

The node correspondence file is used to define pairs of equivalent nodes. Each pair contains one node from each netlist. These equivalences are used to force a match between the nodes. These equivalences can be defined as **initial equivalences** (i.e. initial points of correspondence before the LVS begins its matching algorithm) or **optional forced equivalences**. Optional equivalences are used as forced points of correspondence only when the matching algorithm fails to make progress matching the circuit after a set number of passes. Highly symmetric circuits will frequently fail to match unless you provide some points of correspondence.

The node correspondence file can also be used to define virtual connections in a layout netlist without requiring you to edit the layout. This will be covered below.

The node correspondence file can be created in any ASCII text editor, or you can allow the LVS to create an initial node correspondence file from a preliminary LVS run using the GENERATE_NAME_EQUIVALENCES=YES control file option. Edit this file and use it as an input file in your next run.

File Syntax

The format each line of the node equivalence file is:

Netlist1_net **=** ***Netlist2_net***

where *Netlist1_net* is the name of a net in the first netlist, usually the schematic netlist. *Netlist2_net* is the name of a net in the second netlist, usually a label in the layout netlist.

For example, if you have a net in the schematic netlist with the name "CLK1", and the equivalent net in the layout is labeled with the text "CLOCK", write the line in the node equivalence file as follows:

Example: **CLK1 = CLOCK**

Comments are also allowed in the file. To write a comment, simply use an '*' as the first character on the line.

Node labels are case sensitive. If you have used the control file option FORCE_ALL_LAYOUT_LABELS_TO_UPPER_CASE=NO, be sure to use the same case as the node name in each netlist. If you typed the statement above as:

Example: **CLK1 = clock**

and the label in the layout netlist is "CLOCK", the LVS will not be able to make the node correspondence.

Setting the Control File Options

To use a node correspondence file, specify the following control file options:

LAYOUT_TEXT_MODE=EQUIV
INPUT_FILE_OF_NAME_EQUIVALENCES=*input_equiv_file*

where *input_equiv_file* is the name of your file.

To use the equivalences only when the LVS cannot continue matching the circuit without making a random forced match from lists of nodes with identical properties, specify the control file option:

USE_EQUIVALENCES_FOR_INITIAL_MATCHING=NO

This will default all equivalences to be optional equivalences. (You can override this default for certain nets with the use of flags in the file. See below.) We recommend that you set the above option to NO so that errors in the node correspondence file will not prevent any matches from being found.

You set the number of passes the LVS will perform before resorting to the optional equivalences with the control file option:

SET_NO_PROGRESS_LIMIT=*num_passes*

Using +/- Flags

To override the default definition for each equivalence set by the USE_EQUIVALENCES_FOR_INITIAL_MATCHING control file option, you can use the '+' and '-' flags for certain nets.

If the line in the node correspondence file begins with a '+', it will be used as an initial equivalence. This means that the nets will be matched before the LVS begins its matching algorithm.

Using a '-' as the first character will assign the equivalence as a optional equivalence. The nets will be forced to match only if the LVS fails to match the circuits after a number of passes defined by the SET_NO_PROGRESS_LIMIT control file option.

Example:

```
*Schematic    Layout
+CLOCK1   =   CLOCKA
-X2.7     =   SIGNALX
NET1      =   NETA
```

CLOCK1 will be matched to CLOCKA before attempting to match the circuits even if the control file option USE_EQUIVALENCES_FOR_INITIAL-_MATCHING=NO is used. If CLOCK1 does not really correspond with the net labeled CLOCKA in the layout, the LVS may fail to match any nets or devices. Net 7 in subcircuit X2 will not be matched to SIGNALX unless the LVS fails to match the circuits after several passes. The NET1/NETA equivalence will be an initial equivalence only when the control file option USE_EQUIVALENCES-_FOR_INITIAL_MATCHING=YES is used.

Assigning Virtual Connections

Virtual connections are defined as connections of nets that are not electrically connected in the layout. When two nets are virtually connected, the LVS will treat them as though they are the same net. **If you use virtual connections, you should remove them before the final LVS run on your chip by using the control file option ENABLE_VIRTUAL_CONNECTIONS=NO, (or the LVS command line option "/V=NO") or real opens may not be found.**

There are four ways to virtually connect nets in the layout netlist.

See page 350 for details on assigning virtual connections in the layout.

1) Place identical text labels in the layout on each net. The labels should end with the character defined by the control file option SPECIAL_CHARACTER_FOR_VIRTUAL_CONNECTIONS. In this case, the NLE circuit extractor must be executed again for changes made in the layout to be used by the LVS.

2) Use the node correspondence file to virtually connect two or more labeled nets. (See below.)

3) Add *.VIRTUAL statements to the ASCII part of the layout netlist.

4) Use node label overrides in the ASCII part of the layout netlist to virtually connect two nets by their node numbers assigned by the last pass of the NLE circuit extractor. In this case, the nets do not need to be labeled in the layout netlist at all, and you do not need to re-execute the NLE. (See **Using Node Label Overrides** on page 361 for more details.)

To virtually connect nets with different labels in the layout using the node correspondence file, list all net labels on the right side of the equivalence statement.

Example:

```
*Schematic      Layout
+VDD  =         VDD VDD: POWER
```

In this example, the nets labeled "VDD", "VDD:", and "POWER" will all be virtually connected and matched to the schematic net VDD before the LVS matching algorithm begins.

This method can be used to virtually connect separate nets with the same label even when you have not used the suffix defined by the SPECIAL_CHARACTER_FOR_VIRTUAL_CONNECTIONS control file option (usually ':'). When two or more separate nets are identically labeled **without** this character, they are indicated as errors by the LVS. However, you can virtually connect them in the node correspondence file with an asterisk '*'.

Example:

```
*Schematic      Layout
+VDD  =         VDD* VDD:
```

If you have several separate nets labeled "VDD" in the layout, they will all be virtually connected by the above statement. All nets labeled "VDD:" will also be virtually connected to the same net.

Using Node Label Overrides

When you run an LVS comparison on your circuit for the first time, you may wish that you had added more node labels to the layout before you ran the NLE to extract the layout netlist. Running the NLE circuit extractor can be time consuming. If all you need to do to make your next LVS run more useful is to add (or modify) node labels, you do not need to run the NLE again. You can add node labels to the layout netlist with node label override statements.

The syntax of the node label override statements is:

.NETLABEL** ***node_number label
and
.DEVLABEL** ***node_number label

To determine the node number of a specific net in the layout while in the ICED32™ layout editor, you can use the node outliner commands. See page 389.

where *.NETLABEL statements add or modify labels on nets, and *.DEVLABEL is used to label devices. The *node_number* parameter must be the node number generated by the NLE as reported in the LVS output files. The *label* parameter is the string which will be used to label the node.

Once these statements in the layout netlist are processed, the nodes are labeled as though the labels were added to the layout itself and processed again with the NLE. You do not need to modify the layout at all.

However, each time you do modify the layout and re-execute the NLE to create a new binary layout netlist, the node numbers may change. Node override statements that were correct for one binary layout netlist, may not be correct with a new binary netlist. When you edit the layout for a new NLE extraction, you should add the labels to the actual layout and delete the old node label overrides.

The control file option ENABLE-_VIRTUAL-_CON-NECTIONS = YES must be used to allow virtual connections.

For example, let us say that you discover on your first LVS run that the net VDD is open. Many devices are unmatched because they connect to one of two different nets instead of a single net. You could solve this problem by going back to the layout and adding net labels that virtually connect the net. You could add the label "VDD:" to each net to virtually connect them.

If you solve the problem this way, you will need to run the NLE program again to extract the binary layout netlist. If you are processing a large circuit, this may take a long time.

You can avoid running the NLE again to solve such a simple problem with two node label override statements. Get the node numbers of both nets by using the node outliner commands in the layout editor, or from the LVS reports. These numbers are used in the unmatched device report and the net degree report. If the node numbers are 145 and 328, write the statements:

Example:

```
*.NETLABEL  145  VDD:
*.NETLABEL  328  VDD:
```

Remember that node labels may be case sensitive. If you want your node labels in upper case, type them in upper case, unless you have used to control file option FORCE_ALL_LAYOUT_LABELS_TO_UPPER_CASE=YES.

You must add these statements to the layout netlist. Since you cannot edit the binary layout netlist, you must add these statements to the ASCII file you use to combine the layout netlist device models to the binary layout netlist. We recommend that you create these statements in a separate file and use an .INCLUDE statement to add them to the ASCII layout netlist file.

```
*Layout netlist for LVS check
.INCLUDE LAYMODEL.NET
*.LAYOUT CELL.EXT
.INCLUDE LABELS.NET
.END
```

The node labels processed last take precedence over node labels already processed. When the node label overrides are processed before the binary netlist, labels present in the layout will override them. Unless you want the binary layout netlist to override your overrides, place them in the layout netlist after the *.LAYOUT statement.

If you want to use many node label overrides, (especially if you need to modify labels already in the layout) you can run the LVS with the control file option PRINT_NET_LABELS_IN_A_SEPARATE_FILE = YES. This will create a file with node label override statements for all nodes with labels in the layout and all unlabeled nets over a certain degree. You can then use this file in subsequent runs of the LVS. Refer to the control file option PRINT_NET-_LABELS_IN_A_SEPARATE_FILE on page 272 for more details.

This file can be edited with any ASCII editor. You can change the lines in the file, which will have the same effect as editing the labels in the layout and re-executing the NLE. You can also add lines to this file which has the same effect as adding node labels to the layout.

Symmetric Circuits

Forced matches will be listed in the optional List of Forced Matches report.

The LVS performs its circuit comparison based on the relationship of each device to other devices around it. When many devices have identical relationships, the LVS may have to force a random match between devices from lists of identical devices to continue its matching algorithm. A random match is rarely a good starting point for circuit comparison.

A circuit that has many devices with identical relationships to other devices is called a symmetric circuit. Look at Figure 131. This design consists solely of four identical opamps. The devices in each opamp are identical. Unless you instruct the LVS to make forced correspondences between the two netlists, NetA may be matched to Net1, Net3 Net5, or Net7. Any of these matches would be completely valid.

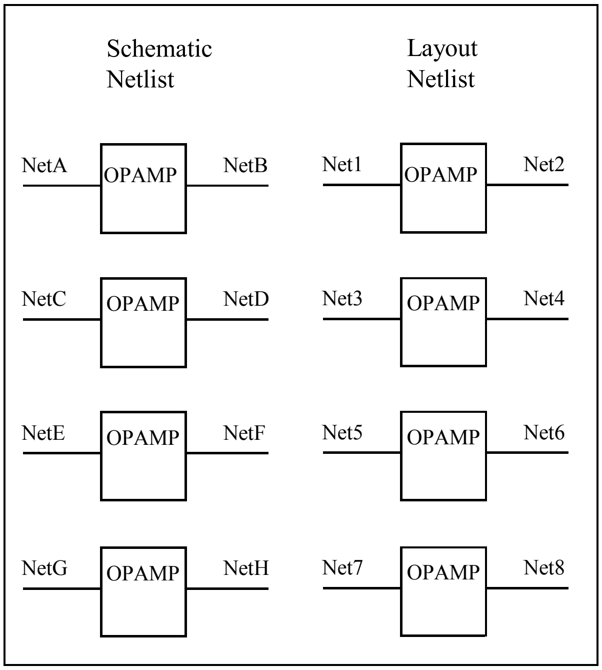


Figure 131: Symmetric circuit

When your circuit is as completely redundant as this one, you probably do not care if NetA is matched to Net3. It will be relatively easy to decipher the reports even though the nets are not matched as you would match them by looking at the circuits.

However, now let us assume that the opamps are not identical. The resistor values in each opamp are the only difference. Now the fact that NetA is matched to Net3 means that the resistors will be listed as having parameter value mismatches. This is not a real error, but it is an unavoidable result of a random match to begin the matching algorithm.

The LVS will use parameter values as criteria for performing the forced match. However, if the LVS chooses one of the identical transistors for the forced match, instead of one of the resistors, the parameter values are identical in each opamp, and the different value of the resistor elsewhere in the circuit will not prevent the LVS from making an incorrect forced match.

It may not be too difficult to figure out what went wrong when the symptom is a few parameter value mismatches. However, if one of these opamps does have a real error in the layout, it can be difficult to determine which opamp has the problem when the LVS is matching a different opamp than the one you thought it was. Large numbers of cascading errors can make the reports difficult to use.

All of these problems are easily avoided by forcing the LVS to perform a few forced correspondences when it is unable to continue its matching algorithm. This will prevent the LVS from forcing a random match between nodes from each netlist.

Forced correspondences can be defined with one of three methods:

1) The control file option LAYOUT_TEXT_MODE=AUTO will match all nodes pairs with identical names in each netlist. (The control file option USE_EQUIVALENCES_FOR_INITIAL_MATCHING=YES must also be used.)

2) The node correspondence file can be used to supply the LVS with corresponding nets from each netlist. (See page 355.) Use the control file option LAYOUT_TEXT_MODE=EQUIV in this case.

3) The INTERACTIVE_MODE = YES option in the control file can be used to select a forced point of correspondence during the LVS run. In this mode, you interactively select a node number from each netlist. You do not need any node labels in your circuit for this option.

We do not recommend option 3 because you will need to know the node numbers of the nodes in each netlist while the LVS is running. If you forget to write these numbers down based on a previous run of the LVS, or mistype one node number, the results of the LVS comparison may be much worse that if you did not attempt the forced correspondence at all.

If you want to use option 2 (the recommended choice), but you do not want to re-execute the NLE to add the labels to the layout, there is a shortcut you can use. Create a node label override file that adds the required node labels to the layout netlist without re-executing the NLE. (See page 361.)

Use care when assigning forced correspondences. If you specify one in error, the result can often be a mismatch of every other net and device. Keep initial correspondences in the node correspondence file to a minimum, and make additional correspondences optional so that they will be used only if the LVS cannot continue the matching algorithm without making a forced match. You indicate that a correspondence is optional with the '-' prefix in the node correspondence file.

LVS OUTPUT FILES

Reports Generated by the LVS

A log file and three other reports that are always generated by a successful LVS run. In addition, there are several optional reports which can be useful in diagnosing discrepancies between the two netlists being compared.

LVS.LOG

The OUTPUT-_DIRECTORY-_PATH control file option determines the location of all reports other than LVS.LOG.

One additional output file is the *cell_name*.P8K file generated by the LVS for the node outliner commands. See page 389.

The log file will always have the name "LVS.LOG". Unlike the other reports, it is always created in the **current directory**. This file is simply a log of all console output generated by the LVS. When the run is successful, there is little information here. There will be a message for each label found in a layout netlist, and a summary of activity done by each pass of the program. The summary for the last pass will tell you how many devices and nets were matched. The final line of the log tells you where the comparison summary report, often "RESULTS.LVS", can be found. It is very important to look at this file carefully, even when no errors are mentioned in the log. All other reports will always be found in the same directory as the comparison summary report.

If there are syntax errors in one of the netlists or the control file, the error messages are stored in the log file to help you diagnose the problem. The format of error messages is described on page 311. Only 15 error messages will be printed before the LVS aborts. If additional errors exist, they will not be reported until you re-execute the program.

You should always look at the end of this report for references to report files that list problems with the netlists. Look at each of the indicated files. You may wish to refer to other reports for additional information that will help you diagnose problems. Details on these other reports are found on the following pages.

Non-Optional Output Files

The three reports generated by every LVS run will be created in the directory defined by the control file option OUTPUT_DIRECTORY_PATH. Most of these reports include a header which lists the input file names, the time and date of the run, the elapsed time of the run, the memory allocated, and the LVS version number. The reports are summarized in Figure 132.

	Control file option which defines file name	**Information found in file**
Comparison summary	OUTPUT_FILE_OF_FINAL-_RESULTS_OF_NETLIST-_COMPARISON	Number of devices and nets before and after preprocessing Number of devices and nets matched and unmatched Number of devices with parameter errors Number of I/O pins Number of filtered, collapsed, and forced match devices Number of floating nets A list of device counts sorted by device type A summary of most control file options, and individual device model options A summary of what other reports were generated and their file names
Parameter error summary	OUTPUT_FILE_OF_DEVICES-_WITH_PARAMETER_ERRORS	List of matched devices with size or value mismatches
Unmatched devices report	OUTPUT_FILE_OF-_UNMATCHED_DEVICES-_AND_NETS	Details on each unmatched device in each netlist Details on each unmatched net in each netlist Lists of out-of-order device chains Lists of mismatched pad devices

Figure 132: Non-optional output files.

Comparison Summary

It is important to inspect this file carefully after every LVS run. The console messages do not indicate all errors.

The file begins with a header that includes the input file names, the time and date of the run, the elapsed time of the run, the memory allocated, and the LVS version number.

The other information contained in this file includes:

SUBCKT / CELL NAME The top level subcircuit of schematic netlists and/or the top level cell of layout netlists will be indicated here.

NUMBER OF DEVICES BEFORE PREPROCESSING The total number of devices found in each netlist is given. This is not the total number of devices compared. The preprocessing is done before the comparison and can change these numbers substantially. (Pad devices are not included in this total or in the totals for the next 5 lines.)

NUMBER OF DEVICES AFTER PREPROCESSING This is the total number of devices from each netlist which were compared. The preprocessing can filter out devices based on the control file options. The preprocessing also involves device transformations that merge or collapse several devices into a single device. These collapses are controlled by the control file and/or parameters in the device models in each netlist. The number of filtered devices and collapsed devices is provided below.

The optional matched devices report will list details on each matched device.

NUMBER OF DEVICES MATCHED This is the total number of devices which matched in the two netlists. Some or all of these matched devices may have parameter errors.

NUMBER OF DEVICES NOT MATCHED Any non-zero numbers in these fields mean that there is a discrepancy between the two netlists. See the unmatched devices report for a list of all unmatched devices.

NUMBER OF DEVICE PARAMETERS NOT COMPUTED If the value of a device in the schematic netlist is missing, or if a device in the layout netlist cannot be computed because the dimensions were not recognized by the NLE (and there is no default value in the device model), the device will be included in this total. This total may include devices with unusual layouts (e.g. non-manhattan layouts) for which the NLE cannot recognize dimensions.

Default value parameters can be supplied in the *.LAYMODEL device models.

A non-zero number in either column means that devices in the netlist have not had their values verified against the other netlist. Unmatched devices will be included in this total. Devices counted in this total will be listed in the parameter error file with the string "***" indicated for the value.

Parameter checking is enabled in the control file by the MATCH_*device*-_PARAMETERS options. This default can be overridden in the netlist device models by the PARAM parameter.

NUMBER OF PARAMETER ERRORS Each matched device can be verified for value or dimension discrepancies between the two netlists. This verification can be disabled for specific device types or device models. You should verify that parameter checking is enabled for all of the device types and models you need verified. **If parameter checking is disabled for some of your devices, this number may not be an accurate account of the parameter errors present in your circuit.**

The parameter error summary report will list all devices with parameter errors.

NUMBER OF NETS BEFORE PREPROCESSING The total number of nets in each input netlist is reported. These totals are calculated before virtual connections are made and before devices are filtered, merged, or collapsed.

NUMBER OF NETS AFTER PREPROCESSING Transformations of devices can remove nets from the netlist. This number is the total number of nets which were compared from each netlist.

NUMBER OF NETS MATCHED This is the total number of nets which matched from each netlist.

NUMBER OF NETS NOT MATCHED A non-zero number in either column indicates a discrepancy between the two netlists. See the unmatched devices report for details on each unmatched net.

NUMBER OF PAD DEVICES The number of pad devices recognized by the NLE in a layout netlist will be listed here. Since pad devices are not valid in a schematic netlist, there will always be a zero in a schematic netlist column. Pads indicated through the use of the *.PINS statement will not be listed here.

See Pad Connection Verification on page 335 for important information on how to verify pad connections.

NUMBER OF PAD DEVICES MATCHED If pad devices are matched to nets in the other netlist defined with the same pad type character (usually through the use of the *.PINS statement) they will be counted here.

NUMBER OF PAD DEVICES NOT MATCHED If you have a non-zero number in the layout netlist column for this line, this means you have pad devices which are not being matched to appropriate nets in the other netlist. This indicates an error in your pad connections.

See page 320 for examples of out-of-order device chains.

NUMBER OF OUT_OF_ORDER TRANSISTOR CHAINS If the control file option MERGE_OUT_OF_ORDER_*device*_CHAINS (or the device model overrides DCHAIN, ALLMERGE, or ALL) has caused out-of-order transistor chains to be merged, this total will be non-zero. See the unmatched device report for a list of all out-of-order device chains.

NUMBER OF ERRORS IN LAYOUT LABELS The errors referred to on this line include single nets labeled with two different labels or multiple nets with the same label (which have not been virtually connected). The LVS log file and the unmatched device report will both contain detailed error messages about these nets.

NUMBER OF I/O PINS The number of nets which are considered pins from each netlist is listed here. In a schematic netlist, a net is considered a pin when it is in the argument list of the top-level subcircuit. In a layout netlist, the net must be labeled with the prefix defined by the control file option SPECIAL-_CHARACTER_FOR_NO_COLLAPSE_OF_A_NET, or the net must connect to a pad device. Nets which are listed in a *.PINS statement in either netlist are also considered i/o pins.

NUMBER OF UNCONNECTED NETS The total number of nets which are not connected to any device after preprocessing, are provided here. The list of nets is contained in the file set by the OUTPUT_FILE_OF_NETS_WITH-_ZERO_AND_ONE_CONNECTIONS control file option. Nets in a schematic netlist which occur in a subcircuit argument list, but which do not connect to any devices, will be included in the total.

NUMBER OF FLOATING/DEGREE-ONE NETS The total number of nets which connect to only one device are listed here. These nets will be listed in the same report as the unconnected nets. All i/o pins that connect to only one device will be included in this total.

Disable virtual connections with the "/v=no" LVS command line option.

NUMBER OF VIRTUAL CONNECTIONS If nets have been virtually connected through any of the methods shown on page 358, the total will be reported here. **If this is your final run on your chip, and if the number in the layout column is non-zero, you have opens in your layout which are being hidden by virtual connections.**

For details on device filters, see page 387.

NUMBER OF FILTERED DEVICES This line reports the number of devices removed from each netlist due to options in the control file and device models. Devices can be filtered because they are unconnected, hanging, shorted, or because they are parasitic devices below a certain tolerance.

NUMBER OF FILTERED NETS When devices are filtered out of the netlist, the nets connected to them may disappear from the netlist. When this is the case, the total number of nets removed from each netlist is indicated here.

See pages 318 - 325 for details on device collapses.

NUMBER OF COLLAPSED DEVICES When devices are collapsed due to options in the control file or in the device models, the total number of new devices created which replace the collapsed devices in each netlist is provided here. These collapses may be series or parallel merges, series or parallel logic collapses, or pseudo device collapses.

NUMBER OF LABELED NET MATCHES When labels in the layout netlist are used to make forced correspondences between the two netlists, the count will be listed here.

Forced net and device matches are listed in the optional forced matches report.

NUMBER OF FORCED NET MATCHES If the LVS matching algorithms make no progress after the number of passes defined by the control file option SET_NO_PROGRESS_LIMIT, the LVS will force equivalences between pairs of nets from each netlist. The count of forced net matches will be listed here.

NUMBER OF FORCED DEVICE MATCHES This is the total number of forced matches of devices rather than nets.

Details on the device transformations are found on page 315.

Next in this report is a list of device counts sorted by device type. The number of terminals follows the device type separated by a '/' (e.g. "NPN/4"). The total number of devices after each device transformation is listed on a line after the device category name. The first line for each device category represents the totals in the first netlist (usually the schematic netlist) the next line lists the totals for the second netlist (usually the layout netlist).

The next table contains counts of devices sorted by device model. The total before and after preprocessing for each netlist is given here.

The device count information is followed by a summary of the individual device options for each device category. The defaults in the control file come first. This is followed by a table of the overrides for each device model.

The column headings and the control file device options they represent are:

SWAP	SWAP_*device*_SOURCE_DRAIN, SWAP_EMITTER_AND_COLLECTOR_TERMINALS, or SWAP_CAPACITOR_TERMINALS
SMERGE	MERGE_SERIES_*device*S
PMERGE	MERGE_PARALLEL_*device*S
CHAIN	MERGE_*device*_CHAINS
DCHAIN	MERGE_OUT_OF_ORDER_*device*_CHAINS
DSIZE	MERGE_DISSIMILAR_SIZED_MOSFETS
DMODEL	MERGE_d*evice*S_OF_DIFFERENT_MODELS

SERIES	COLLAPSE_SERIES_LOGIC_*device*S
PARALLEL	COLLAPSE_PARALLEL_LOGIC_*device*S
MODEL	MATCH_*device*_MODELS
PARAM	MATCH_*device*_PARAMETERS
OPEN	IGNORE_UNCONNECTED_*device*S
ONE-CNCT	IGNORE_ONE_TERMINAL_CONNECTED_*device*S
TWO-CNCT	IGNORE_TWO_TERMINALS_CONNECTED_*device*S
SHRT	IGNORE_SHORTED_*device*S
GATE-NET	IGNORE_MOSFET_IF_GATE_PIN_IS_TIED_TO-_CRITICAL_NET
SD-NET	IGNORE_MOSFET_IF_SOURCE_AND_DRAIN_PINS-_ARE_TIED_TO_CRITICAL_NET

Other control file options follow. The file names for each report are given here, as well as the values for most other control file options.

Parameter Error Summary

See **Parameter Calculation** on page 343 for details.

This report lists each device which has been matched but has different parameter values in the two netlists. The parameter values compared depend on the type of device and on options in the control file and device models for each netlist. The parameters compared can be device area, length and width, or value.

If parameter checking has been turned off for a device type with the control file option MATCH_*device*_PARAMETERS = NO, or in the device models of both netlists using the override parameters PARAM=NO, ALLMATCH=NO, or ALL=NO, the presence of real errors in the netlists will **not** be listed in this report.

If devices without values exist in either netlist, their values cannot be compared to devices in the other netlist. Devices without values will have the string "***" listed as the value. In this case, the comparison summary report will have a non-zero number listed for NUMBER OF DEVICE PARAMETERS NOT COMPUTED.

The default tolerance for parameter checking is .0005, or .05%. This default can be overridden in the device models for netlist2. In an LVS comparison, the tolerances must be provided in the device models of the **layout netlist**. A device will be listed as an error only if the difference between the device parameters is greater than the tolerance.

The count of devices in error is listed is provided near the top of the report.

Several details on each device are listed, including the label or unique identifier that represents the name of each device. The coordinates for devices in the layout netlist will be indicated.

Device names from a schematic netlist are represented hierarchically. The highest-level subcircuit instance name is provided on the left. If a device in the schematic netlist represents a merged device, the device name will be each original device name concatenated together and separated by special characters. These characters are defined in the control file. Series merges will be separated by the character defined in the option SPECIAL_CHARACTER_FOR-_PRINTING_SERIES_MERGES (usually '@'). Parallel merges will be separated by the character defined in the option SPECIAL_CHARACTER_FOR-_PRINTING_PARALLEL_MERGES (usually '&').

Nets in the layout are represented by their node numbers. These node numbers can be used with the node outliner commands in the ICED32™ layout editor to highlight nodes in the layout. (See page 389.) When a net is formed from a virtual connection of nets, it will be followed by the character 'V' and the number of original nets virtually connected in square brackets (e.g. 81[V,2]).

A device in the layout which has been formed from a device merge will be represented by a unique node number followed by the character 'M' and the number of devices merged in square brackets (e.g. 296[M,3]). Devices formed from a logic collapse will be followed by the character 'P'. The coordinates listed for one of these devices are the coordinates of one of the original devices.

When you get a large number of false errors due to the device values being represented in different units, this can be easily corrected in your next run, without re-executing the NLE. The control file options SCALE_*device*-_LENGTH_AND_WIDTH and SCALE_*device*_VALUE allow for unit discrepancies between the two netlists. See those control file options for details.

Unmatched Devices Report

If large numbers of devices which are correctly connected appear in this report, look at the list of forced matches report to see if the LVS made a poor choice for a forced point of correspondence. See page 384.

This report lists details on each unmatched device and net from each netlist. The count of unmatched devices and nets is provided near the top of the report. (The reference to out-of-order device chains refers to transistor chains which have been merged even though the devices were in different orders in connected chains. See below.)

The list of unmatched devices from each netlist comes first. The first line assigns an incremented error number to the device. Then the device instance name or number is provided on the next line. The format of the device instance name is described above. The coordinates of the device and then its value are provided next. The device model name, device type, and the number of nets connected are provided on the next line.

A detailed report of each net connected to the device comes last. The net name or number is listed, then the terminal number, and terminal name. Finally the net name or number from the other netlist which is a potential match is listed under the column heading "MATCH ?". If no potential match was found, the string "?????" is listed instead.

The list of unmatched nets from each netlist comes next. Each net is provided with an error number, then the net name is reported. The number of devices to which the net connects is listed, followed by details for each device. The device name, terminal number, terminal name, and potential match device name from the other netlist are provided.

The SET_NET_SIZE_LIMIT_WHEN_PRINTING_CONNECTIONS control file option can limit the number of devices listed for unmatched nets.

Note that if you have a large number of false errors from terminal swapping which is allowed in your technology, you can avoid these false errors with the control file options: SWAP_*device*_SOURCE_DRAIN, SWAP_EMITTER-_AND_COLLECTOR_TERMINALS, or SWAP_CAPACITOR_TERMINALS.

If you have defined pad devices in either netlist (as described on page 335), and mismatches exist between the pad connections in the two netlists, details will be provided near the end of the report, separate from the other unmatched devices.

See page 320 for examples of out-of-order device chains.

If out-of-order device chains exist in either netlist, they will be listed here. An out-of-order device chain is a transistor chain which has been merged to another connected chain even though the devices were in different orders in the different chains.

Optional Output Files

Several optional reports can be generated according to options in the control file. Most of the control file options are located in the OPTIONAL OUTPUT FILE section. The exception is the matched devices report which is in the OUTPUT FILES section. These reports will be created in the same directory as the non-optional reports. Most of these reports include a header which include the input file names, the time and date of the run, the elapsed time of the run, the memory allocated, and the LVS version number. The reports are summarized in the table on the next page.

	Control file option which enables report	Control file option which defines file name	Information found in file
Matched devices report	PRINT_MATCHED-_DEVICES_AND-_NETS	OUTPUT_FILE_OF-_MATCHED_DEVICES-_INCLUDING-_PARAMETER_ERRORS	List of all matched devices, including instance name, model name, device type, and parameters from each netlist. Each device with a parameter mismatch is indicated with the error message "PARAMETER ERROR". List of all matched nets.
List of unconnected and floating nets	PRINT_NETS-_WITH_ZERO_AND-_ONE-_CONNECTIONS	OUTPUT_FILE_OF-_NETS_WITH_ZERO-_AND_ONE-_CONNECTIONS	List of nets which are not connected to any device List of nets which connect to only one device
List of collapsed devices	PRINT-_COLLAPSED-_DEVICES_IN_A-_SEPARATE_FILE	OUTPUT_FILE_OF-_COLLAPSED_DEVICES	List of pseudo devices formed from logic collapses. (Merges are **not** included.)
List of forced matches	PRINT-_SYMMETRIC-_MATCHES_IN_A-_SEPARATE_FILE	OUTPUT_FILE_OF-_SYMMETRIC-_MATCHES	Lists of pairs of nodes forced to correspond.
Spice output	GENERATE_SPICE-_NETLIST_FROM-_THE_EXTRACTOR-_OUTPUT	OUTPUT_FILE_OF-_SPICE_NETLIST	Schematic netlist generated from the layout (including parasitic devices).
Net degree report	PRINT_NETS_AND-_THEIR_DEGREES	OUTPUT_FILE_OF-_NET_DEGREES	Counts of nets sorted by degree. List of nets over a certain degree.

Continued on next page

Filtered device list	PRINT_LIST_OF-_FILTERED-_DEVICES	OUTPUT_FILE_OF-_FILTERED_DEVICES	List of devices removed from each netlist.
Node equivalence file	GENERATE_NAME-_EQUIVALENCES	OUTPUT_FILE_OF-_NAME-_EQUIVALENCES	Net equivalences found between the labels in the layout and the schematic netlist.
Node labels file	PRINT_NET-_LABELS_IN_A-_SEPARATE_FILE	OUTPUT_FILE_OF-_NET_LABELS	List of nodes labeled in the layout with the corresponding node numbers.

Figure 133: Optional LVS reports.

Matched Devices Report

In this report, every device that matched in each netlist is listed. Each netlist has a column.

The first line is simply a counter. This number is prefixed by "# :". The next line has the device instance name. Device names from a schematic netlist are represented hierarchically. The highest level subcircuit instance name is provided on the left. Device instance names or node numbers in the layout netlist can be used to locate the device in the ICED32™ layout editor. (See page 389.)

The coordinates of the device are printed next. (The coordinates of a device in a schematic netlist are always 0.) The next line contains the device model name and type. The parameter values for the device are on the last line. The list of terminals is not included as it is in the unmatched device report.

Any device that has a parameter mismatch between the two netlists is indicated with the error message "PARAMETER ERROR". This information is also reported in the non-optional parameter error summary. (See page 377.)

List of Unconnected and Floating Nets

This report lists all nets of degrees zero and one. The degree of a net is defined as the number of devices it connects to.

To insure that certain nodes in the layout are unconnected, see the UNCONNECTED NLE rule.

A zero degree net in the schematic netlist arises when nets listed in the I/O list of a subcircuit are never referred to in a device statement. There are never any nets of zero degree in the layout netlist. Even when stray pieces of conductive material are not connected to any devices, they will not be listed as nets of zero degree.

The format of the report is similar to the unmatched device report. The counter (indicated by the string "# :") is followed on the next line by the net instance name. If the net is in the floating net list, the next line lists the device instance name, the terminal number, and the terminal name. Finally the net name or number from the other netlist which is a potential match is listed under the column heading "MATCH ?". If no potential match was found, the string "?????" is listed instead.

This information is generated after the device filters have been processed. Nets that disappear from the netlist due to the device filters will not be included. If you do not filter unconnected devices, you may see a floating net indicated for each terminal of a unconnected device.

List of Collapsed Devices

This report lists pseudo devices. While the format is similar to the matched device report, the order of the devices in each column is arbitrary. The pseudo device in the netlist column on the left is probably not matched to the device in the other column. See the matched device report (page 382) for a report where the device on the left is matched to the one on the right.

See page 322 for details on logic collapses.

Devices formed by a simple merge will not be listed here. The devices listed are generated when the control file options COLLAPSE_SERIES_LOGIC_*device*S = YES, COLLAPSE_PARALLEL_LOGIC_*device*S = YES, or TAKE_CARE-_OF_LOGIC_EQUIVALENCES_WHILE_MATCHING = YES or the netlist device model options SERIES = YES, PARALLEL = YES, ALLCOLLAPSE = YES, or ALL = YES are used.

For each pseudo device, the first line is an incremented counter. The next line is the pseudo device node number or name. The coordinates of one of the devices is listed, then the device model and type. (Remember that the LVS will only form pseudo devices from collections of devices of the same type and model.)

Details on the devices that were collapsed to form the pseudo device follow next. The device instance name, coordinates, and parameter values are listed for each device.

List of Forced Matches

Read Symmetric Circuits on page 365 for details on forced matches.

When the matching algorithms make no progress for the number of passes set by the control file option SET_NO_PROGRESS_LIMIT, the LVS is forced to choose a point of correspondence between the two netlists. The pairs of nodes that were forced to match will be listed here. These matches may be made in error, especially when your circuit is highly symmetric.

These matches may be arbitrary matches, matches you selected interactively when you have used the INTERACTIVE_MODE=YES control file option, or labeled nets in the layout which were not defined as initial equivalences in the node correspondence file.

When many false mismatches are listed in the unmatched device report, look at this file to see if an incorrect forced match started the algorithms off on the wrong path.

See Using a Node Correspondence File on page 355.

To avoid incorrect forced matches in your next run, select several correct pairs of nodes for forced correspondence. Label them with text components in the layout and indicate that they are initial equivalences in the node correspondence file. If you want to avoid running the NLE again to process node labels in the layout, you can add node labels using node label overrides. (See page 361.)

Spice Output

This report is a schematic netlist generated from the layout data. If you are comparing two layout netlists, this netlist will be generated from layout netlist2. This netlist can be used for simulation, or as an auxiliary device report for diagnosing mismatches.

Several control file options determine the format of the generated schematic netlist. SPICE_FILE_FORMAT controls the syntax of the schematic netlist. This option and the other relevant control file options are described beginning on page 276.

The DELETE_PARASITIC_CAPACITORS_LESS_THAN control file option can be used to filter out devices less than a threshold value from this Spice file. This allows you to simulate your design with all significant parasitic capacitors present in your layout.

Net Degree Report

See the Advanced Tutorial for examples of how useful this file can be when diagnosing circuit discrepancies.

This report is a quick summary of nets sorted by degree. (Remember that degree is defined as the number of devices to which a net connects.) This report can be very helpful in diagnosing opens and shorts. You can easily see major discrepancies in nets of high degree that indicate shorts or opens.

The number of nets of each degree is listed for each netlist and the difference in the counts is reported on the right. If your netlists match, all numbers in the right hand column should be zero.

If you have an unmatched net of high degree in one netlist and two or more unmatched nets in the other netlist whose degrees add up to the high degree net in the first netlist, it is likely that you have a short in the first netlist, or an open in the second netlist.

Any non-zero number next to DIFFERENCE IN TOTAL NUMBER OF PINS indicates a circuit mismatch. Pins in this case refers to all device terminals. The total number of terminals of all devices (after preprocessing) in the second netlist is subtracted from the first netlist. See the net degree report and the unmatched device report for details on the problem.

You can optionally list the names or numbers of nets over a certain degree with the PRINT_ALL_NETS_WHOSE_DEGREE_GREATER_THAN control file option. These nets will be listed under the LISTING OF NETS headings. The names or numbers of nets in a layout netlist can be used to highlight the nets in the layout using the node outliner commands. (See page 389.)

Filtered device list

This report lists all devices filtered by preprocessing. Several classes of devices can be filtered according to options in the control file. See Figure 134.

Control file option	**Devices filtered**
IGNORE_UNCONNECTED_*device*S	devices where no terminal is connected to another device.
IGNORE_ONE_TERMINAL_CONNECTED_*device*S	devices with only one terminal connected to another device.
IGNORE_TWO_TERMINALS_CONNECTED_*device*S	3 or 4 terminal devices with more than one unconnected terminal.
IGNORE_SHORTED_*device*S	devices with shorted terminals (Shorts between the substrate and source or drain of 4 terminal devices are not considered shorted.)
IGNORE_MOSFET_IF_GATE_PIN_IS_TIED-_TO_CRITICAL_NET	NMOS devices with gate terminals tied a ground net and PMOS devices with gate terminals tied to a power net.
IGNORE_MOSFET_IF_SOURCE_AND_DRAIN_PINS-_ARE_TIED_TO_CRITICAL_NET	NMOS devices with source and drain terminals both tied a ground net. PMOS devices with source and drain terminals both tied to a power net.
IGNORE_BIPOLAR_IF_BASE_PIN_IS_TIED-_TO_CRITICAL_NET	NPN and LNPN devices with base terminals tied a ground net. PNP and LPNP devices with base terminals tied to a power net.
DELETE_*device*S_LESS_THAN	parasitic devices whose value is less than the specified tolerance.

Figure 134: LVS filters.

See page 221 to specify which nets are power or ground nets.

If you have enabled these filters, but consider such devices to be errors, you should look carefully at this report.

Node Equivalence File

This report will list each labeled net in the layout netlist with the corresponding net name from the other netlist. This file can be edited and used as an input node correspondence file in future LVS runs.

See Using a Node Correspondence File on page 355 for more details.

Nets which are labeled in the layout, but which were not matched to nets in the other netlist will be listed as comments under the heading "Unmatched layout labels". If you want to force the LVS to match a net in this list to a certain net in the other netlist, type that net name on the left of the '=' and remove the '*' that makes the line a comment. Then use this file as an input node correspondence file in your next run.

Node Labels File

This file will list all nodes which are labeled in the layout. Each node will be listed with the node number assigned to this node by the NLE circuit extractor. These node numbers are the same ones used in other reports (e.g. the node numbers of nets attached to terminals in the unmatched devices report.)

See Using Node Label Overrides on page 361 for more details.

You can edit this file and use it as in input file in the layout netlist to override labels in the layout.

If you want to include unlabeled nets over a certain degree in this file, use the PRINT_ALL_UNLABELED_NETS_WHOSE_DEGREE_GREATER_THAN control file option.

Remember that the node number in the layout netlist may change when you re-execute the NLE. If you use this file as an input file for subsequent LVS runs, it will be valid only until you regenerate the layout netlist using the NLE.

Using the Node Outliner Commands

The node outliner commands are executed in the ICED32™ layout editor to highlight the geometry that forms a specific net or device. You can use these commands to pinpoint a net or device in the layout using the node number listed in the LVS reports. There is also a command to tell you the name or number of a net or device when you click on a specific piece of geometry in the layout.

The node outliner commands will make the selected net or device blink for several seconds. The ICED32™ layout editor must be launched with COLORS=16 (the default) for these commands to use the BLINK command. If you launch ICED32.EXE with the COLORS=8 command line parameter, you should remove it before launching the editor to use the node outliner commands.

The IC32.BAT file is usually used to launch ICED32.EXE. Edit the command line in this file.

The following commands launch the program OUTLINER.EXE to process geometry. If the ICED32™ layout editor is not reserving some memory for other applications, you may see the error message "Insufficient memory" when you use any of the node outliner commands. To reserve memory, you will need to quit, alter the ICED32.EXE command line, and relaunch the editor.

The command line parameter you should use to reserve memory depends on the operating system. If you are running DOS in a shell of a multitasking operating system (e.g. WINDOWS or OS/2), add an appropriate "USE" parameter at the end of the ICED32.EXE command line. If you are using native DOS, add the parameter "RESERVE=2000".

See the ICED32™ layout editor reference manual for more details on the USE and RESERVE command line parameters.

The node outliner commands need to access files created by the NLE and LVS programs for information on the electrical connections and for the node identifiers. One file is created automatically when you execute the NLE circuit extractor to create the binary layout netlist. The name of the file is *cell_name*.P9K, where *cell_name* is the same string as that in the file name of the binary layout data file, *cell_name*.POK. The other file is the *cell_name*.P8K file created by the LVS.

The existence of the .P9K file is enough to execute the node outliner commands, but you will not be able to use a node name to locate a net. Only node numbers may be used to highlight nets when the .P8K file does not exist. This means that you can use the node outliner commands before you execute the LVS, however the commands are easier to use once the LVS has been run.

All layers used in connection and device recognition rules will be included in the node outliner file unless you add the SAVE or NO_SAVE rules to your NLE rule set.

Do not delete the .P9K file (generated by the NLE) or the .P8K file (generated by the LVS). They must be located in the directory of the cell file.

To execute the node outliner commands:

Launch the ICED32™ layout editor to edit the cell used to create the binary layout netlist.

You can include the @NODES command in your startup command file run on each new cell. See the Layout Editor Reference Manual.

The command **@NODES** must be executed once in this cell to define the other commands. This will create the new commands **N0** (the number 0, not the letter 0)**, N1, ND,** and **NN**. Once the @NODES command has been executed, the command definitions will be stored with the cell. When you try to execute the N0, ND, or NN commands, and the @NODES command has not been executed in the current cell, the editor will respond with error messages.

Type **N0** to highlight a net or device by its node number (or node name if the LVS has already generated the .P8K file). You will be prompted to type the net or device number. (You can type the node name at this prompt if the .P8K file exists.) The geometry that comprises the indicated node will be copied temporarily to layer 250. This geometry will then blink for several seconds.

Type **ND** to clear all temporary geometry from layer 250.

If several devices have been merged, you need to enter only one of the node numbers to highlight all devices merged with it.

Type **N1** to use the cursor to select a component and highlight all of the geometry electrically connected to it. Place the cursor inside the boundary of the component, not on its edge. The geometry will be copied temporarily to layer 250, then it will blink for several seconds. The node number you have selected will be displayed at the bottom of the screen.

If you prefer to select a node with the cursor to report the attached node number, without highlighting it, use the **NN** command.

You can change the width of the outlines created by the N0 or N1 commands by changing the default width of layer 250 with the layout editor LAYER command.

Each time you use N0 or N1, you are adding geometry from the .P9K file to layer 250 using color 15. All geometry using color 15 will blink when you execute N0 or N1. We recommend that you use ND before using N0 or N1 for a new node.

If the cursor is over more than one node when you execute the N1 or NN commands, more than one node will be selected. Try to place the cursor in a different place where geometry for only the node of interest is located with no other shapes overlapping it.

If you are unable to select only the single node of interest, it is probably because you cannot avoid selecting other nodes such as a substrate or well. Move the cursor slightly so that it is not on top of the net of interest or any other obvious node. Execute the NN command and note the "background" node numbers reported on the history line on your screen. Now when the N1 or NN commands report several node numbers, you can subtract the background node numbers from the list to determine the node number of interest. Then execute the ND command to clear layer 250, and finally, use the N0 command on the node number of interest to highlight only that one net.

Let us examine this fragment from an unmatched devices report:

```
# :1
LAYOUT :291
X :474          Y :-395
LENGTH :1       WIDTH :6
MODEL :NMOS TYPE :NMOS     CONNECTED_TO : 4 nets.
TERMINAL         NET_NAME                  MATCHED_NET
DRAIN            1                         ?????

GATE             3                         10058

SOURCE           72                        OT1

SUBSTRATE        1                         ?????
```

If the editor is unable to recognize the N0 command, type @NODES at the prompt to define the node outliner commands.

You can highlight the unmatched device with the command

N0 291

To highlight only the entire net attached to the source, use the commands

```
ND
N0  72
```

Once you have used N0 or N1 to create the geometry, you can make it blink continuously with the command:

```
BLINK T=0
```

Press any key to make it stop blinking.

Now look at this fragment of a parameter error summary:

```
          SCHEMATIC                            |          LAYOUT
                                               |
                                               |
# :1                                           | # :1
XIN1.MN1                                       | 295
X : 0            Y :0                          | X : 305                Y :180
MODEL :MN   TYPE :NMOS                         | MODEL :NMOS         TYPE :NMOS
LENGTH :4      WIDTH :2                        | LENGTH :1    WIDTH :2
                                               |
                                               |
```

If we edit this cell and execute the **@NODES** command, we could highlight the device with the incorrect length by typing **N0**, then at the prompt, typing **295**. Let us say that the view window is currently showing only part of your cell, and nothing lights up on your screen. This could mean that the device is somewhere beyond the borders of your screen.

Now execute the commands:

```
VIEW ALL
BLINK T=0
```

The first command fits the entire cell in the view window. The second command makes the geometry created by the N0 command blink again so you can locate the device. Now you can zoom in and execute the blink command again if you desire.

Once you are done outlining nodes, execute the **ND** command to clear all geometry from layer 250 before saving the cell.

The LPE Utility

The LPE utility is used to translate a binary layout netlist (created from ICED32™ cell data by the NLE utility) into an ASCII schematic netlist. This netlist can be used as an input to a circuit simulator. If the NLE rules file is designed to extract parasitic capacitors, they will be included in the netlist.

The LPE can optionally perform the same device transformations and filters the LVS performs before creating the netlist.

No comparison is made between the transformed layout netlist and another netlist.

The netlist created by the LPE will be flat, with no hierarchical subcircuits.

LPE Command Line Syntax

```
LPE  [path\]control_file_name  [path\]layout_netlist_name  ...
       ... [ @file_name ] ...              ⎫
       ... [ /g ( yes | no ) ] ...         ⎪
       ... [ /i in_path ] ...              ⎪
       ... [ /l ext_file ] ...             ⎬  Command
       ... [ /o dir_path ] ...             ⎪  line options
       ... [ /p sch_out_format ] ...       ⎪
       ... [ /v ( yes | no ) ]             ⎭
```

The LPE should be run at the DOS prompt, outside of the ICED32™ layout editor.

See page 251 for a complete description of the control file.

The LPE requires two input files. The first is the control file. The LPE uses the same control file as the LVS utility. Many of the options applicable only to the LVS will be ignored by the LPE. However, you should not delete any lines from the control file. The LPE control file parser expects a complete control file. A few control file options are used only by the LPE. These are described below.

The second input file is the layout netlist which will be translated into a schematic netlist. The layout netlist should be prepared in exactly the same manner as one prepared for LVS comparison. See the "Layout Netlists" chapter beginning on page 229 for complete details on preparing this file.

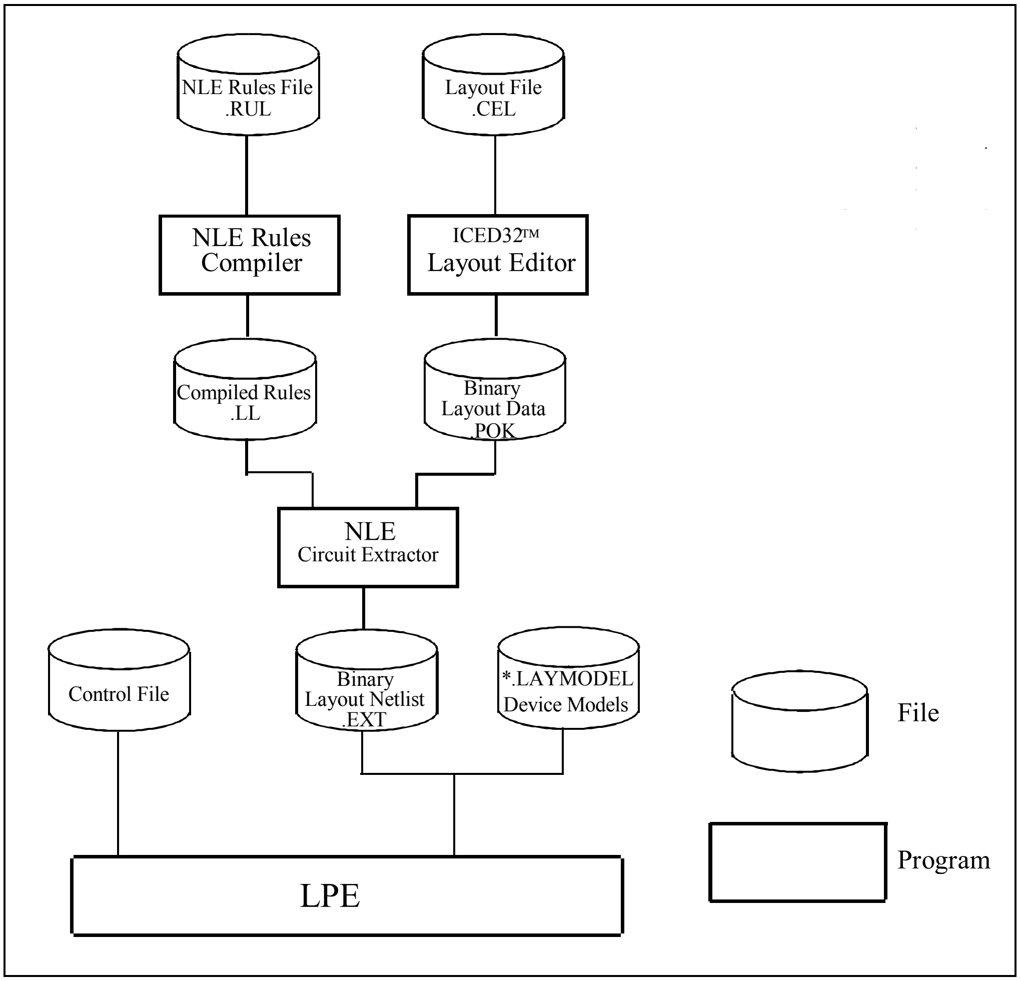


Figure 22: Flow of data for the LPE utility.

The NLE utility is described beginning on page 37.

The most important component of the layout netlist is the binary file generated by the NLE utility. The NLE utility generates this file from a data file created by the DRC command in the ICED32™ layout editor. The NLE uses a rules file to control circuit extraction from the layout data.

The layout netlist can contain node label overrides which allow you to modify node labels in the layout without re-executing the NLE. See page 361.

The layout netlist must also contain device models for each unique device in the layout netlist. These models define how to calculate device values from the device dimensions stored by the NLE. The device models can also contain overrides for options in the LPE control file.

The layout data can contain node labels. These labels will be listed in the output schematic netlist in *.NETLABEL and *.DEVLABEL statements near the beginning of the file which relate the label to the node number. The labels will not be used as the names of the nodes in the device statements. The labels can be used to control device transformations such as preventing the collapse of devices on certain nets. (See "Advanced Uses of Node Labels" on page 349 for details.)

All of the options in the LPE command line perform identical functions to those used in the LVS command line. See page 307 for more complete descriptions of these options.

Command line option	**Purpose**
@file_name	Allow indirection on command line
/g (yes | no)	Enable/disable global node labels
/i *in_path*	Specify input file directory path
/l *ext_file*	Override the *.LAYOUT statement in the layout netlist input file
/o *dir_path*	Specify output file directory path
/p *sch_out_format*	Specify format of output schematic netlist
/v (yes | no)	Enable/disable virtual connections

Figure 23: LPE command line options

Example: **LPE CONTROL.LVS LVS_LAY.NET /O LPERESLT**

This command line, typed at the DOS prompt in the directory created to run the quick tutorial (described on page 19), will run the LPE on the ONEBIT circuit. The /O option overrides the OUTPUT_DIRECTORY_PATH option in the CONTROL.LVS control file. The output schematic netlist (SPICE.LPE) will be created in the LPERESLT subdirectory of the current directory. However, the log file (LPE.LOG) will still be created in the current directory.

The LPE Control File

The LPE control file is usually the same file you use for the LVS program. Some of the lines will be ignored by the LPE, but you should never delete lines from a control file.

You should be familiar with most of the control file options before executing the LPE. The control file is described completely in "The LVS Control File" earlier in the manual. The control file options perform identical functions in the LPE and the LVS. See the table below for the page numbers where several of the most relevant control file options are described.

Control file option	**Purpose**	**Refer to page**
DELETE_PARASITIC_CAPACITORS_LESS-_THAN	Remove parasitic capacitors below a threshold value	300
ENABLE_VIRTUAL_CONNECTIONS	Allow nets to be virtually connected	265
FORCE_ALL_LAYOUT_LABELS_TO_UPPER-_CASE	Enable or disable lower case characters in node names	260
OUTPUT_DIRECTORY_PATH	Set name of directory for most output files	256
PRINT_NET_LABELS_IN_A_SEPARATE_FILE	Enable creation of file used to store net labels found in layout	272
OUTPUT_FILE_OF_SPICE_NETLIST	Set file name of generated netlist	276
SPICE_FILE_FORMAT	Set format of generated netlist	277
SCALE_CHARACTER_FOR_CAPACITORS	Add scale character to each capacitor value	278
SCALE_CHARACTER_FOR_INDUCTORS	Add scale character to each inductor value	278
SCALE_CHARACTER_FOR_RESISTORS	Add scale character to each resistor value	278

Figure 135: Important LPE control file options.

Some important control file options are found under the heading "INDIVIDUAL DEVICE OPTIONS". (See page 283.) These options control how devices are transformed before the output netlist is generated. These options set defaults that can be overridden by parameters in the device models. (See page 245.)

The LPE uses a few extra control file options that we have not covered elsewhere in this manual. You can find these options at the bottom of the control file, under the heading "LPE RUNTIME OPTIONS". These options override similar options in the LVS section of the control file so that you can edit them once for your LPE uses, but leave the remainder of the control file intact for LVS uses.

OUTPUT_FILE_NAME_EXTENSION_FOR_LPE = *file_ext*

This control file option overrides the LVS option OUTPUT_FILE_NAME_EXTENSION_FOR_LVS. (See page 257.) The string *file_ext* will be used as the default file extension for all output files generated by the LPE. Setting *file_ext* to a different string than the LVS extension will cause the LPE to create all files with a different extension than your LVS generated files, so that none of the LVS generated files will be overwritten.

The only files generated by the LPE which do not use this string as the file extension are the LPE.LOG file (which contains a log of all console output generated by the LPE) and the *cell_name*.P8K file for use in the node outliner utilities. (See page 389.)

PRINT_COMMENTS_IN_SPICE_OUTPUT_GENERATED_BY_LPE = ... (YES|NO)

This control file option overrides the LVS option PRINT_COMMENTS_IN-_SPICE_OUTPUT_GENERATED_BY_LVS. (See page 277.) The comments which will be added to the netlist generated by LPE include device labels found in the layout. Each device in the output netlist which was created by collapsing devices in the input netlist will have some details printed about it in the comments. These comments will aid you in reading the output netlist, but will have no effect on the simulatable lines of the file.

PRINT_FILTERED_DEVICES_IN_SPICE_OUTPUT_GENERATED...
... _BY_LPE = (YES|NO)

This control file option overrides the LVS option PRINT_FILTERED-_DEVICES_IN_SPICE_OUTPUT_GENERATED_BY_LVS. (See page 277.) When this option is set to yes, the generated schematic netlist will have lines added to it which represent commented out device statements for each device filtered out of the netlist by the LVS filters.

Devices may be filtered out of the netlist according to options in the control file. The primary use of this feature is to allow only the connected devices in a semi-custom layout to be included in the output netlist. (See page 297.) When these filters are enabled, devices that are unconnected to other devices, or devices which are shorted to specific power or ground nets, will be removed from the netlist.

Example: **OUTPUT_FILE_NAME_EXTENSION_FOR_LPE =LPE**
PRINT_COMMENTS_IN_SPICE_OUTPUT_GENERATED_BY_LPE=YES
PRINT_FILTERED_DEVICES_IN_SPICE_OUTPUT_GENERATED...
... _BY_LPE = YES

These three options use the values supplied with the sample control file included with the installation. The first line sets the default file extension the LPE will use for all output files to "LPE". This will be the file extension used for the generated schematic netlist. If the control file contains the line "OUTPUT_FILE-_OF_SPICE_NETLIST=spice", the name of the generated schematic netlist will be SPICE.LPE.

The next line of the example will result in comments added to the file which will make it easier to read. The last line will cause devices which have been filtered out of the netlist to be included as comments in the file.

Advanced Tutorial

Schematics for this example can be found on page 434.

This tutorial uses files on the installation diskettes to demonstrate how to find circuit errors in an LVS (layout vs. schematic) comparison using the NLE and LVS programs. The NLE rule file and the schematic netlist file are the same files as those used in the quick tutorial at the beginning of this manual. However, for this tutorial, we will run the LVS on the entire TOP181 circuit, rather than the ONEBIT subcircuit.

This tutorial contains excerpts from reports generated by the LVS. As you run the LVS and browse these files, your reports may look slightly different due to your version of the program. However, the basic information should be easy to see even if your reports are not identical.

It is likely that the node numbers we talk about in this tutorial will be different in your reports. Write the node numbers in your reports next to the numbers shown in the examples and use those numbers when typing node numbers in the node outliner commands or in the input files for the LVS.

Copy the following files to a new, empty working directory.

From Q:\ICED\[26]SAMPLES\74181\LVS

CONTROL.LVS	Sample control file
TOP181.CIR	Original schematic netlist
SCHMODEL.NET	Schematic netlist models
LVS_SCH.NET	Schematic netlist including models
S181.RUL	NLE rules file for layout circuit extraction
LAYMODEL.NET	Layout netlist device models
LVS_LAY.NET	ASCII layout netlist including models

From Q:\ICED\SAMPLES\74181\BAD

*.CEL	Cell files for layout

[26] Remember that Q:\ICED represents the drive letter and path where you have installed ICED32™.

Note that the cell files should be copied from the "BAD" subdirectory, not the "CLEAN" subdirectory we used for the quick tutorial. These cell files contain errors that we will find with the LVS. Make the new directory the current DOS directory.

A listing of the rules file, S181.RUL, can be found in the Quick Tutorial.

Our first step is to compile the NLE rules file, S181.RUL. At the DOS prompt, type:

RULESNLE S181

The compiler will create the compiled rules file S181.LL. We will use this file as an input file when we run the NLE in a later step. If you want to browse the log file from the compiler, the file name is S181.RLO.

The next step is to create the layout data file for the NLE. We do this in the ICED32™ layout editor with the DRC command. At the DOS prompt, type:

IC32 TOP181

This will launch the ICED32™ layout editor to edit the top level cell of our sample design. Now type the following command on the prompt line:

DRC

This command will create the layout data file TOP181.POK. This will be the one of the input files for the NLE. Once the file is created, exit the editor without saving by typing the command:

QUIT

Now we are back in DOS and ready to run the NLE. Type at the DOS prompt:

NLE S181 TOP181 TOP181

The NLE will execute using the input files S181.LL and TOP181.POK. Since the third argument on the NLE command line is the same as the cell file name, TOP181, most of the output files created by the NLE (including the binary layout netlist) will use this string as the base file name. This is not required, but

If you are running the NLE in a DOS shell of a multitasking operating system, like Microsoft Windows, and you see a long delay (more than 20 seconds) before you are prompted to press <Enter> to continue, you should add the USE=20000 parameter to the end of the command line.

it may help you find files later. The binary layout netlist will have the file name TOP181.EXT.

The binary layout netlist must be combined with the device models in the file LAYMODEL.NET. This has already been done for you in the file LVS_LAY.NET. Browse these files if you want to be more familiar with how to build the layout netlist.

The layout netlist is now ready for the LVS comparison. The schematic netlist has already been prepared for you. The original schematic netlist is stored in CDL syntax in the file TOP181.CIR. This file must be combined with the device models found in the file SCHMODEL.NET. These two files are combined into a complete LVS schematic netlist in the file LVS_SCH.NET. No changes are required to any of these files.

We will not make any changes to the sample control file, CONTROL.LVS, at this time.

To execute the LVS, type at the DOS prompt:

```
LVS CONTROL LVS_SCH.NET LVS_LAY.NET
```

This command line will run the LVS with the control file CONTROL.LVS, the schematic netlist file LVS_SCH.NET, and the layout netlist file LVS_LAY.NET. The order of the last two files on the command line is important. In an LVS comparison, the schematic netlist **must** be listed before the layout netlist.

The final console messages from the LVS should look similar to this:

```
Check <<results\unmatch.lvs>> file.  There are [4+35] unmatched devices and nets.
Check <<results\param.lvs>> file.  There are 362 parameter errors.
Check <<results\results.lvs>> for summary of netlist comparison.
```

Note that the final console messages from the LVS report that there are 4 unmatched devices and 35 unmatched nets. There are also 362 parameter errors. The final message directs you to look at the file RESULTS.LVS in the RESULTS subdirectory for a summary of the comparison. Browse this file now with your favorite ASCII text editor or file viewer.

Note that each total across from the line "NUMBER OF PARAMETER ERRORS", 362 in each column, is the same number as "NUMBER OF DEVICES MATCHED". This means that every matched device had a parameter error. To see a listing of these errors, look at the parameter error summary report, PARAM.LVS.

```
SCHEMATIC                              |             LAYOUT
                                       |
# :1                                   | # :1
XINT1.MN1                              | 2494
X :0              Y :0                 | X :16          Y :144
MODEL :MN         TYPE :NMOS           | MODEL :NMOS    TYPE :NMOS
LENGTH :1e-06     WIDTH :6e-06         | LENGTH :1      WIDTH :6
```

Figure 136: Fragment of parameter error summary

The first device in the report should look similar to Figure 136. Note that the values of the device indicated after the words "LENGTH" and "WIDTH" in each column have a discrepancy of exactly $1e^{-06}$. You can see that this is true of every device in the report.

This is due to the fact that the values of the devices in the schematic netlist use units of meters. This is the corresponding device statement in the schematic netlist TOP181.CIR:

MN1 OUT IN VSS VSS MN W=6U L=1.0U

Note that the parameter values in this statement are followed by the units indicator 'U'. When the LVS reads this statement, it multiplies every value by $1.0e^{-6}$, or 0.000001.

The values in the layout are in units of microns. This units discrepancy is the cause of each parameter value mismatch. We can correct this problem, without editing the original schematic netlist, by changing a setting in the control file.

Edit the control file, CONTROL.LVS, and search for the following line:

SCALE_MOSFET_LENGTH_AND_WIDTH = 1

Change the parameter after the '=' to "1e6" so that the line reads:

SCALE_MOSFET_LENGTH_AND_WIDTH = 1e6

Never alter the fields to the left of the '=' in the control file. Never delete any lines.

This will multiply each MOSFET device dimension in the schematic netlist by $1e^6$ or 1,000,000. This will resolve the units discrepancy between the two netlists.

Save the control file and run the LVS program again with the same command line at the DOS prompt.

```
Check <<results\unmatch.lvs>> file.  There are [4+35] unmatched devices and nets.
Check <<results\param.lvs>> file.  There are 7 parameter errors.
```

Now you can see that only seven devices have parameter mismatches. Now that we have solved the major discrepancy involving all devices, we will wait until the circuits have matched to worry about these remaining device value discrepancies. If all nets and devices are not yet matched, false parameter errors can be generated since devices may be matched in error. It is not usually worth trying to resolve parameter errors until all nets and devices are matched.

Now browse the new RESULTS.LVS file. Our next clue to a large number of errors is the fact that the layout netlist has 20 unmatched nets, while the schematic netlist has only 15. Any time that the two netlists have different numbers of unmatched nets is a good indication of a short or open. The easiest way to diagnose this type of problem is the net degree report.

The degree of a net is defined as the number of device terminals to which the net connects. If several unmatched nets in the layout netlist have a degree which adds up to the degree of an unmatched net in the schematic netlist, this implies that opens in the layout are resulting in several unconnected net fragments which should be connected into one net.

Let us browse the net degree report. The name of this file, along with the names of all other LVS output files, is reported in the RESULTS.LVS file. Near the end of this file, you will see the list of file names under the heading:

```
##            OTHER OPTIONS AS SET IN            ##
##               **CONTROL FILE**                ##
```

Search for the string "degree" and you will see that the name of the net degree report is NETDEG.LVS. Browse this file now.

```
NET DEGREE   #SCHEMATIC   #LAYOUT    DIFFERENCE
----------   ------------  ------------  ----------
1                0            1            -1
2                92           94           -2
3                1            1            0
4                2            3            -1
5                29           29           0
6                47           46           1
7                6            6            0
8                7            7            0
9                0            1            -1
10               1            1            0
11               1            0            1
12               1            1            0
15               2            2            0
18               1            0            1
19               1            0            1
20               0            1            -1
21               0            1            -1
22               1            1            0
62               0            1            -1
104              0            1            -1
108              0            1            -1
274              1            0            1
335              0            1            -1
340              1            0            1
```

Figure 137: Fragment of net degree report

The net degree report should contain a block of lines similar to Figure 137. The discrepancy between the nets of higher degree implies that opens exist in the layout netlist. We can see that the schematic netlist contains 1 net of degree 340, but the layout netlist has no net with the same degree. However, the layout netlist contains an unmatched net with a degree of 335. Also, the schematic netlist contains an unmatched net with degree 274, while the layout contains unmatched nets with degrees 108, 104, and 62. Since 108+104+62 = 274, it appears that two opens exist in this net in the layout.

You can set the minimum degree for nets in the detailed listing in the second half of the net degree report with the PRINT_ALL-_NETS-_WHOSE-_DEGREE-_GREATER-_THAN control file option.

To determine exactly which nets have the problem, look at the details on nets of high degree in the second half of the net degree report. This should look like the listing in Figure 138. The name of the net in the schematic netlist with degree 274 is VSS. The node numbers of the three net fragments in the layout netlist whose degrees add up to 274 are nodes 1, 4, and 6. We can use these node numbers to locate the nets in the layout.

Let us launch the ICED32™ layout editor again and use the node outliner commands to look at these nets in the layout. In the editor, type the command:

@NODES

This initializes the other node outliner commands.

You may want to print the net degree report rather than browsing it so that you do not have to quit the layout editor later to browse it again.

To make the node with the node number 1 blink on the display, type:

N0 1

```
###### LISTING OF NETS : SCHEMATIC ######

NET DEGREE : 274
VSS
NET DEGREE : 340
VDD


###### LISTING OF NETS : LAYOUT ######

NET DEGREE : 62
1
NET DEGREE : 104
4
NET DEGREE : 108
6
NET DEGREE : 335
7
```

Figure 138: Fragment of net degree report

If you get the message "Insufficient memory" when you try to execute these node outliner commands, see "Using the Node Outliner Commands" in this manual for details on reserving memory for the commands.

The N0 command will add geometry which represents the net to layer 250, then it makes this geometry blink with the BLINK command. All geometry on layer 250 will blink when you use any of the node outliner commands. You usually want to remove the previous contents of layer 250, before outlining the next node, by using the command ND.

To make node number 4 blink, type the commands:

ND

N0 4

To make node number 6 light up, type:

ND
N0 6

Feel free to change the view of the circuit to get a closer look at the geometry that comprises these nets. It becomes obvious that these three layout nets all represent the same net, VSS. The nets were meant to be connected together in a higher level circuit. You do not have to connect them in the layout to get around this problem, you can virtually connect the nets without physical connections in the layout.

You could add labels in the layout to virtually connect the nets, but this would require re-executing the NLE. It is easier to virtually connect the nets with node label overrides. The nets will be virtually connected by the LVS without re-executing the NLE. We will add these node label overrides in a moment. First, let us look at the problem with net VDD.

The degree of VDD in the schematic netlist is 340. The closest match in the layout is net 7 with a degree of 355. This indicates that some unattached net fragments of VDD exist in the layout. However, since these fragments have a low degree, they do not appear in the detailed listing shown in Figure 138. To find them, we will have to look in the unmatched devices report.

You may want to print the unmatched net report rather than browsing it so that you do not have to quit the layout editor later to browse it again.

The name of the unmatched devices report is UNMATCH.LVS. Quit the ICED32™ layout editor and browse this file now. This file also contains details on unmatched nets. To locate this information, search for the string "LAYOUT NETS".

```
#######################################################
##The following LAYOUT NETS were not matched.      ##
##      **NO POTENTIAL MATCHES**                   ##
#######################################################
# :1
LAYOUT :1650
CONNECTED TO :1 devices.
TERMINAL  DEVICE_NAME        MATCHED_DEVICE ?
DRAIN     2340               XXB1STF.XNA1_6.MP1
```

Figure 139: Fragment of unmatched devices report

The report on the first unmatched net should look similar to Figure 139. This indicates that node 1650 connects to the drain of device 2340 which has been matched to schematic netlist device XXB1STF.XNA1_6.MP1. When we look up this device in the schematic netlist (TOP181.CIR), the device statement is:

MP1 OUT IN1 VDD VDD MP W=4.0U L=1.0U

in subcircuit NAND4 of subcircuit B1STF. Note that the drain of this device should be VDD. This is a good hint that node 1650 in the layout should connect to VDD, but is open in error. We should look at node 1650 in the layout.

Launch the ICED32™ layout editor again and enter the commands:

Once the @NODES command is executed, the other node outliner command definitions are stored in the cell. If you save the cell, you do not have to execute @NODES again.

@NODES
N0 1650

If you do not see any geometry blinking on your screen, type the commands:

VIEW ALL
BLINK

You should now see something blink. Zoom in on this part of the circuit with the command:

VIEW BOX

Select the new corners of the view window with the mouse.

You can make the geometry blink again at any time by typing the command BLINK. Keep zooming in until you can see the circuitry around this node. Once you are at an appropriate scale, it is easy to see that this node should be connected to net VDD.

You can fix this error by adding geometry to the cell, or by virtual connections. In this tutorial we will fix the error with virtual connections because it is faster. However, when you perform this kind of quick fix for your real circuits, you must remember to fix the errors in the layout later.

The method to find the other VDD net fragments is the same one we used to find net 1650.

```
# :2
LAYOUT :1723
CONNECTED TO :2 devices.
TERMINAL  DEVICE_NAME        MATCHED_DEVICE ?
SOURCE    2404               XXB3STF.XNA3_6.MP1
DRAIN     2405               XXB3STF.XNA3_6.MP3
```

Figure 140: Fragment of unmatched devices report

The report on the second unmatched net should look similar to Figure 140. This indicates that node 1723 connects to the drain of device 2404 which has been matched to schematic netlist device XXB3STF.XNA3_6.MP1. When we look up this device in the schematic netlist, the device statement is:

MP1 OUT IN1 VDD VDD MP W=4.0U L=1.0U

in subcircuit NAND4 of subcircuit B3STF. Note that the drain of this device should be VDD. This is a good hint that node 1723 in the layout should connect to VDD, but is open in error. We should look at node 1723 in the layout.

Launch the ICED32™ layout editor again and enter the commands:

@NODES
N0 1723

Follow the instructions above for zooming in on the node in the display. Once again, it is obvious that node 1723 should be connected to VDD and is open in error.

```
# :3
LAYOUT :23
CONNECTED TO :2 devices.
TERMINAL  DEVICE_NAME        MATCHED_DEVICE ?
DRAIN     2362               XXB1STF.XXR1_2.MP1
SOURCE    2367               XXB1STF.XXR1_2.MP2
```

Figure 141: Fragment of unmatched devices report

The report on the third unmatched net should look similar to Figure 141. This indicates that node 23 is another candidate for an unattached net fragment of net VDD. Looking at the layout confirms this.

To add virtual connections to get around the false errors on net VSS, and the real errors on net VDD, we will use node label overrides. This allows us to virtually connect the net fragments without re-executing the NLE. The best way to add node label overrides is to create a new ASCII text file with the name LBLOVR.TXT. Type the lines shown in Figure 142 replacing the node numbers with the node numbers found in your reports if your numbers are different.

```
*.Netlabel 1 VSS:
*.Netlabel 4 VSS:
*.Netlabel 6 VSS:

*.Netlabel 7    VDD:
*.Netlabel 1650 VDD:
*.Netlabel 1723 VDD:
*.Netlabel 23   VDD:
```

Figure 142: Node label override file LBLOVR.TXT

Be sure to type the colons (':') after the node names as shown. This will allow the net fragments to have the same label without error messages about identical labels on separate nets.

We now need to add these virtual connections to the layout netlist. Edit the file LVS_LAY.NET. Add the following line to the file:

.include lblovr.txt

Be sure to add the new line **after** the *.layout statement as shown in Figure 143. Save this file and re-execute the LVS with the same command line.

```
*
.include laymodel.net
*.layout top181.ext
.include lblovr.txt
.end
```

Figure 143: Modified LVS_LAY.NET

```
Check <<results\unmatch.lvs>> file.  There are [4+16] unmatched devices and nets.
Check <<results\unmatch.lvs>> file.  There are [0+2] virtual connections using labels.
Check <<results\param.lvs>> file.  There are 2 parameter errors.
```

```
NET DEGREE    #SCHEMATIC    #LAYOUT      DIFFERENCE
----------    ------------  ------------  ----------
                          .
                          .
                          .
18                 1            0             1
19                 1            0             1
20                 0            1            -1
21                 0            1            -1
22                 1            1             0
274                1            1             0
340                1            1             0
```

Figure 144: Fragment of new net degree report

The new net degree report will look similar to Figure 144. Note that the report indicates nets of relatively high degree are still unmatched in the two netlists. The layout contains a net of degree 21 and no such net exists in the schematic. Let us take a look at this net in the unmatched device report. Look for the net connected to 21 devices under the "Layout Nets" heading. The listing will look similar to Figure 146.

We can determine that net 271 represents net "C" in the schematic netlist by looking at the first two devices: XXB1STF.XXR1_1.MN3, and XXB1STF.XXR1_1.MN4. Now we compare this net to the listing for net "C" under the schematic nets heading of the unmatched devices report. See Figure 145.

```
# :8
SCHEMATIC :C
CONNECTED TO :19 devices.
TERMINAL  DEVICE_NAME         MATCHED ?
GATE      XXB1STF.XXR1_1.MN3  2563
GATE      XXB1STF.XXR1_1.MN4  2561
GATE      XXB1STF.XXR1_1.MP2  2366
GATE      XXB2STF.XNA2_3.MN4  2601
GATE      XXB2STF.XNA2_3.MP4  2387
GATE      XXB2STF.XNA2_4.MN2  2572
GATE      XXB2STF.XNA2_4.MP2  2391
GATE      XXB2STF.XNA2_5.MN1  2580
GATE      XXB2STF.XNA2_5.MP1  2400
GATE      XXB3STF.XNA3_3.MN2  2594
GATE      XXB3STF.XNA3_3.MP2  2419
GATE      XXB3STF.XNA3_4.MN2  2590
GATE      XXB3STF.XNA3_4.MP2  2415
GATE      XXB3STF.XNA3_5.MN2  2604
GATE      XXB3STF.XNA3_5.MP2  2409
GATE      XXB3STF.XNA3_6.MN2  2586
GATE      XXB3STF.XNA3_6.MP2  2405
DRAIN     XXBIT2.XIN5.MN1     2631
DRAIN     XXBIT2.XIN5.MP1     2424
```

Figure 145: Fragment of new unmatched devices report

```
# :8
LAYOUT :271
CONNECTED TO :21 devices.
TERMINAL  DEVICE   MATCHED_DEVICE ?
GATE      2563     XXB1STF.XXR1_1.MN3
GATE      2561     XXB1STF.XXR1_1.MN4
GATE      2366     XXB1STF.XXR1_1.MP2
GATE      2601     XXB2STF.XNA2_3.MN4
GATE      2388     XXB2STF.XNA2_3.MP3
GATE      2572     XXB2STF.XNA2_4.MN2
GATE      2391     XXB2STF.XNA2_4.MP2
GATE      2580     XXB2STF.XNA2_5.MN1
GATE      2399     XXB2STF.XNA2_5.MP2
GATE      2583     XXB2STF.XNA2_6.MN1   ⇐
GATE      2402     XXB2STF.XNA2_6.MP1   ⇐
GATE      2594     XXB3STF.XNA3_3.MN2
GATE      2418     XXB3STF.XNA3_3.MP3
GATE      2590     XXB3STF.XNA3_4.MN2
GATE      2414     XXB3STF.XNA3_4.MP3
GATE      2604     XXB3STF.XNA3_5.MN2
GATE      2409     XXB3STF.XNA3_5.MP2
GATE      2586     XXB3STF.XNA3_6.MN2
GATE      2405     XXB3STF.XNA3_6.MP2
SOURCE    2649     XXBIT1.XIN5.MN1
SOURCE    2426     XXBIT1.XIN5.MP1
```

Figure 146: Fragment of new unmatched devices report

What we are looking for is the cause of the discrepancy in the net degrees for this net. Look at the connections of the XXB2STF circuit. Note the two devices which are marked with arrows ('⇐') in the listing of the layout net 271. According to the schematic listing for net "C", this net should not connect to the XNA2_6 subcircuit of XXB2STF at all. This is the cause of the discrepancy.

By looking in the schematic netlist file (TOP181.CIR) at the B2STF subcircuit, we can see that subcircuit XNA2_6 should connect to net "D" not net "C".

XNA2_6 VDD VSS D I 10123 NAND2

We can locate this error in the layout by launching the ICED32™ layout editor and outlining the first device listed in error in the listing for net 271, node number 2583. In the editor, type the commands:

@NODES
N0 2583

This will cause the first device connected in error to net "C" to blink. Resize the window and zoom in on this area of the circuit so your window clearly shows the error as shown in Figure 147. Use a combination of SELECT commands and the MOVE command to relocate the VIA2PLY cell (circled in Figure 147) and the connecting wire to the wire labeled D. Be careful to use MOVE SIDE when moving the vertical wire so that you move only the bottom edge of the wire and do not leave an open in the wire.

Once the error is fixed in the layout, type the commands:

DRC
EXIT

Now execute the NLE again by typing at the DOS prompt:

NLE S181 TOP181 TOP181

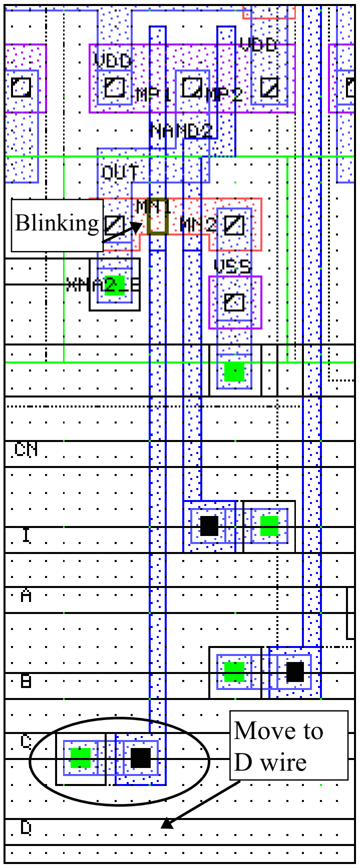


Figure 147: Portion of TOP181 cell

When the utility has completed run the LVS again with the same command:

LVS CONTROL LVS_SCH.NET LVS_LAY.NET

```
Check <<results\unmatch.lvs>> file.  There are [0+8] unmatched devices and nets.
```

Browse the net degree report or the unmatched devices report and you will see that there are still some discrepancies between the two netlists. Let us try a slightly different method to diagnose the next problem.

We can allow the LVS to collapse the logic of both netlists to form higher level pseudo devices. This can often simplify finding misconnections by minimizing the number of cascading errors caused by the misconnections.

To force the LVS to collapse devices into higher-level pseudo devices in both netlists, we should edit the control file. Edit the file CONTROL.LVS. Search for "COLLAPSE_SERIES_LOGIC_MOSFETS”. Change the values for this line and the next two lines to YES as shown below.

COLLAPSE_SERIES_LOGIC_MOSFETS = YES
COLLAPSE_PARALLEL_LOGIC_MOSFETS = YES
COLLAPSE_DISSIMILAR_SIZED_MOSFETS = YES

Save the file and re-execute the LVS with the command line below:

LVS CONTROL LVS_SCH.NET LVS_LAY.NET /O COLLAPSE

The /O option will result in all reports being created in the "COLLAPSE" subdirectory rather than in the "RESULTS" subdirectory. This will allow you to compare the two different methods if you desire.

```
Check <<collapse\unmatch.lvs>> file.  There are [8+4] unmatched devices and nets.
```

This reduces the number of unmatched nets to two in each netlist, for a total of four. Look at the unmatched nets in the unmatched devices report COLLAPSE/UNMATCH.LVS. (See Figure 148 and Figure 149.)

From this report we can see that net "F" in the schematic corresponds to net 274 in the layout. However, net "F" connects to 11 devices while net 274 connects to 9 devices. To find the 2 missing devices, notice that net 273 in the layout has 2 extra devices. By comparing the lists, we can see that devices 2754[P,2] and 2788[P,2] connect to net "E" by mistake and should connect to net "F" instead.

```
# :1
SCHEMATIC :F
CONNECTED TO :11 devices.
TERMINAL  DEVICE_NAME        MATCHED ?
GATE    100039               2739[P,3]
GATE    100040               2754[P,2]  ⇐
GATE    100057               2755[P,3]
GATE    100116               2788[P,2]  ⇐
GATE    100153               2830[P,3]
GATE    100164               2838[P,3]
GATE    XXB0STF.XXR0_1.MN3   2536
GATE    XXB0STF.XXR0_1.MN    42534
GATE    XXB0STF.XXR0_1.MP    22364
DRAIN   XXBIT1.XIN9.MN1      2638
DRAIN   XXBIT1.XIN9.MP1      2425

# :2
SCHEMATIC :E
CONNECTED TO :18 devices.
TERMINAL  DEVICE_NAME       MATCHED ?
GATE    100009               2757[P,5]
GATE    100055               2756[P,4]
GATE    100069               2759[P,5]
GATE    100075               2766[P,3]
GATE    100079               2752[P,4]
GATE    100081               2764[P,4]
GATE    100085               2748[P,4]
GATE    100145               2839[P,4]
GATE    100152               2829[P,4]
GATE    100156               2833[P,3]
GATE    100157               2831[P,4]
GATE    100162               2836[P,4]
GATE    100171               2852[P,5]
GATE    100172               2853[P,5]
GATE    XXB0STF.XXR0_1.MN1   2535
GATE    XXB0STF.XXR0_1.MP1   2357
DRAIN   XXBIT1.XIN5.MN1      2631
DRAIN   XXBIT1.XIN5.MP1      2424
```

Figure 148: Unmatched devices report

```
# :1
LAYOUT :274
CONNECTED TO :9 devices.
TERMINAL  NAME       MATCHED ?
GATE      2739[P,3]  100039
GATE      2755[P,3]  100057
GATE      2830[P,3]  100153
GATE      2838[P,3]  100164
GATE      2536       XXB0STF.XXR0_1.MN3
GATE      2534       XXB0STF.XXR0_1.MN4
GATE      2364       XXB0STF.XXR0_1.MP2
DRAIN     2638       XXBIT1.XIN9.MN1
DRAIN     2425       XXBIT1.XIN9.MP1

# :2
LAYOUT :273
CONNECTED TO :20 devices.
TERMINAL NAME        MATCHED ?
GATE      2757[P,5]  100009
GATE      2754[P,2]  100040                  ⇐
GATE      2756[P,4]  100055
GATE      2759[P,5]  100069
GATE      2766[P,3]  100075
GATE      2752[P,4]  100079
GATE      2764[P,4]  100081
GATE      2748[P,4]  100085
GATE      2788[P,2]  100116                  ⇐
GATE      2839[P,4]  100145
GATE      2829[P,4]  100152
GATE      2833[P,3]  100156
GATE      2831[P,4]  100157
GATE      2836[P,4]  100162
GATE      2852[P,5]  100171
GATE      2853[P,5]  100172
GATE      2535       XXB0STF.XXR0_1.MN1
GATE      2357       XXB0STF.XXR0_1.MP1
SOURCE    2631       XXBIT1.XIN5.MN1
SOURCE    2424       XXBIT1.XIN5.MP1
```

Figure 149: Unmatched devices report

The syntax of the device name in the layout netlist, "2754[P,2]", indicates that device 2754 is a pseudo device formed from 2 devices. You can use the simple node number, "2754" to locate both of the devices in the layout. To locate the problem in the layout, launch the ICED32™ layout editor and type the commands:

ND
N0 2754

Resize the widow so that you can see the blinking device clearly. Move the net connection shown in Figure 150 to the wire labeled 'F'.

Now type the commands:

DRC
EXIT

Extract a new netlist by using the same **NLE** command line we have used before.

There are still other errors in the circuit. It will be easier to see the rest of the problems if the LVS does not collapse devices. **Edit the control file to set all three of the collapse options you changed above back to NO.**

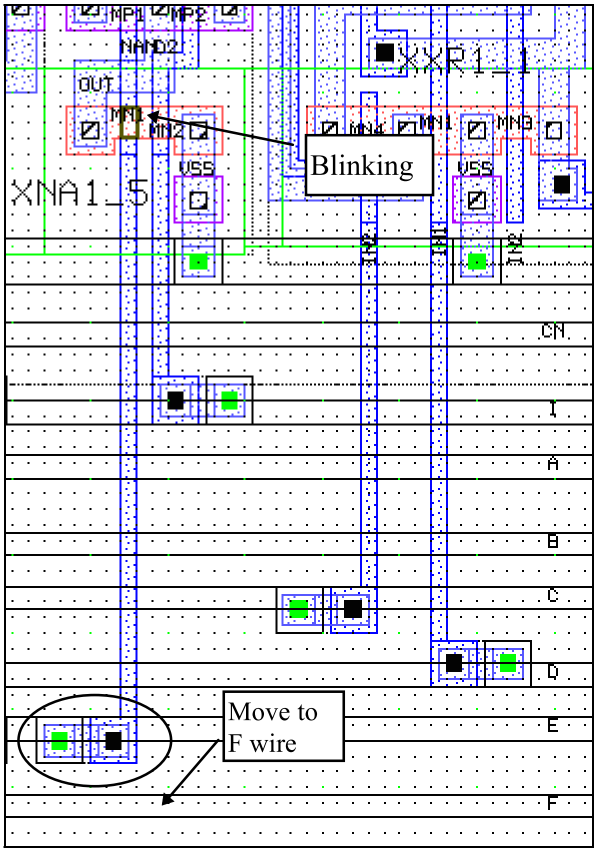


Figure 150: Portion of TOP181 cell

Run the LVS again, changing the output directory to a new string if you desire.

Check <<no_coll\unmatch.lvs>> file. There are [4+0] unmatched devices and nets.

The console messages tell you that all nets now match. However, there are still 4 unmatched devices, 2 in each netlist.

This problem is less obvious to find than it would appear. The problem is that the LVS reports 2 devices in the schematic netlist circuit XNAT2 as unmatched, when in fact these devices appear to be connected correctly. The problem is that these devices are supposed to connect to net F0B, however this net is incorrectly connected in another circuit.

```
# :1
SCHEMATIC :XNAT2.MN4
X :0             Y :0
LENGTH :1        WIDTH :4
MODEL :MN        TYPE :NMOS
                 CONNECTED_TO :4 nets.
TERMINAL   NET_NAME        MATCHED_NET ?
DRAIN        XNAT2.10113   1920
GATE         F0B           28
SOURCE       VSS           2681[V,3,VSS:]
SUBSTRATE    VSS           2681[V,3,VSS:]

# :2
SCHEMATIC :XNAT2.MP4
X :0             Y :0
LENGTH :1        WIDTH :4
MODEL :MP        TYPE :PMOS
                 CONNECTED_TO :4 nets.
TERMINAL   NET_NAME        MATCHED_NET ?
DRAIN        10059         147
GATE         F0B           28
SOURCE       VDD           2680[V,4,VDD:]
SUBSTRATE    VDD           2680[V,4,VDD:]
```

Figure 151: Fragment of unmatched devices report

```
# :1
LAYOUT :2501
X :268           Y :143
LENGTH :1        WIDTH :4
MODEL :NMOS      TYPE :NMOS
                              CONNECTED_TO : 4 nets.
TERMINAL   NET_NAME    MATCHED_NET ?
DRAIN        1920                 XNAT2.10113
GATE         29             XXB0STF.XXR0_8.10148
SOURCE       2681[V,3,VSS:]       VSS
SUBSTRATE    2681[V,3,VSS:]       VSS

# :2
LAYOUT :2322
X :273           Y :134
LENGTH :1        WIDTH :4
MODEL :PMOS      TYPE :PMOS
                              CONNECTED_TO : 4 nets.
TERMINAL   NET_NAME    MATCHED_NET ?
DRAIN          2680[V,4,VDD:]     VDD
GATE           29           XXB0STF.XXR0_8.10148
SOURCE         147                10059
SUBSTRATE      2680[V,4,VDD:]     VDD
```

Figure 152: Fragment of unmatched devices report

The net in the layout which connects to devices 2501 and 2322, is node number 29. In the layout editor type the following commands to locate the unmatched device 2501.

```
VIEW ALL
ND
N0 2501
```

Now make net 29 blink in the layout with the layout editor command:

```
ND
N0 29
```

We can see that this net connects to other devices on the left side of the circuit. By zooming in on the area around device 2501, we can see that the text components in the layout label this net as F0B. This is consistent with what we see in the schematic representation of this device in Figure 151. However, note that the LVS has matched net 28 to the schematic net F0B, not net 29. This match has been made in error, causing other devices to mismatch later. To see why the LVS has made this error, let us take a closer look at net 28.

To make net 28 blink on your screen, type the commands:

```
ND
N0 28
```

When you zoom in on this area, you can see that node 28 is an internal net to the XXR0_8 XOR circuit.

The clue to the real problem is the net listed as the gate terminal for layout devices 2501 and 2322, XXB0STF.XXR0_8.10148. This net is supposed to be a net internal to the XOR circuit XXR0_8 in the XXB0STF circuit. If we look at this circuit in the schematic netlist, we can see that net F0B is an output of this circuit.

```
XXR0_8 VDD VSS 10083 10085 F0B XOR
:
.SUBCKT XOR VDD VSS IN1 IN2 OUT
MN1 10148 IN1 VSS VSS MN W=3U L=1.0U
MN2 10152 10148 VSS VSS MN W=2U L=1.0U
MN4 OUT IN2 10148 VSS MN W=2U L=1.0U
MN5 OUT 10149 10152 VSS MN W=4U L=1.0U
MN3 10149 IN2 VSS VSS MN W=2U L=1.0U
MP2 10149 IN2 VDD VDD MP W=4U L=1.0U
MP1 10148 IN1 VDD VDD MP W=5U L=1.0U
MP3 OUT 10149 10148 VDD MP W=5U L=1.0U
MP4 OUT 10148 10149 VDD MP W=6U L=1.0U
.ENDS
```

By looking at the connections for this circuit in the schematic netlist (see bolded lines above), we can see that they agree with the connections for net 28 in the layout (see Figure 153.) This net should connect to the net labeled F0B and then to devices 2501 and 2322 in the XNAT2 circuit.

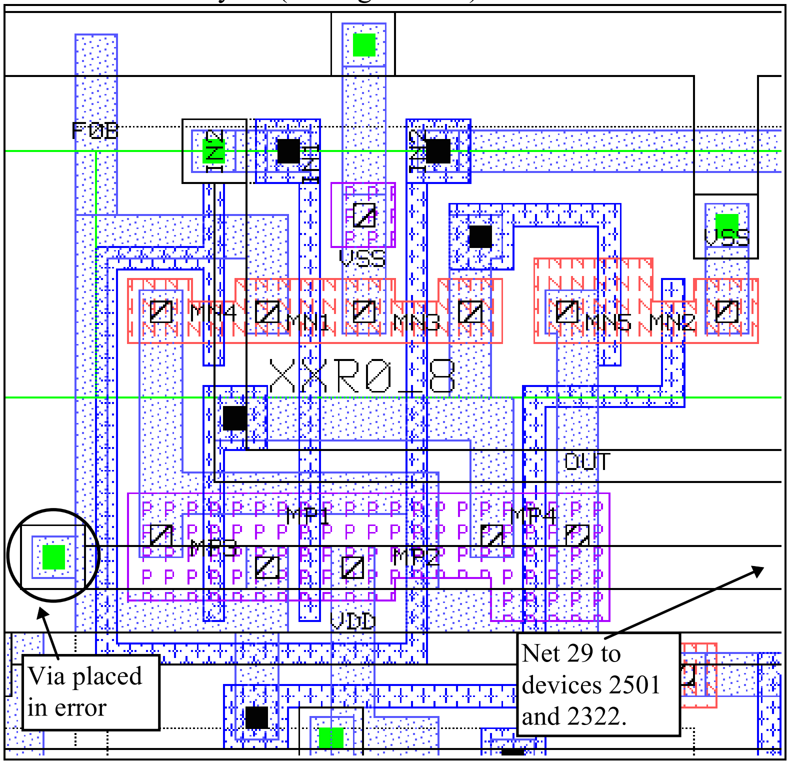


Figure 153: XXR0_8 XOR circuit.

The net in the layout that is labeled F0B connects to the wrong net in the XOR circuit. The cause of this appears to be that this XOR_1 cell was mirrored rather then rotated when it was added to the diagram. The output is now on the right, rather than on the left as it is in the other XOR circuits. However, since the other connections appear to be correct, we will not fix the problem by rotating the cell. We will just do a quick fix that will leave the MT1 wire with the F0B label in the upper left still connected to the wrong net, but we will connect the output of this XOR to the other devices to which it should connect.

We need to move the circled via cell to the MT1 wire labeled "OUT". However, this via cell is not placed in the TOP181 cell we are currently editing. It is nested in the B0STF cell. We could QUIT the editor, then edit cell B0STF, but it is much easier to use a nested edit command to edit the nested cell.

The TEDIT SELECT command is used to traverse one level of cell nesting at a time to edit a nested cell. It is the best way to edit a nested cell if you are not familiar with how the geometry is nested.

Type at the prompt:

TEDIT SELECT

Then place your cursor on the edge of the circled via cell shown in Figure 153. Click the left mouse button. Now the prompt line indicates that you are editing the B0STF cell.

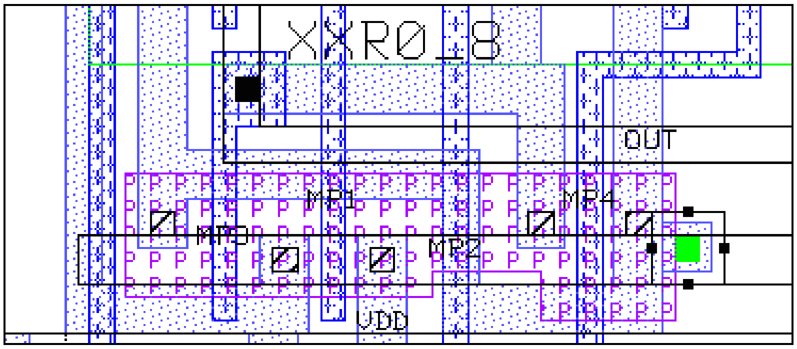


Figure 154: Portion of B0STF cell with selected via moved to correct location.

Select the via cell and use the MOVE X command to move it to the location shown in Figure 154. Be sure that the MT1 box in the via cell touches the MT1 wire labeled "OUT".

Type EXIT and return to the TOP181 cell.

Now type the DRC command to export the modified layout to the NLE program. Exit the editor, then execute the NLE and LVS programs again as shown on page 408. The console messages now indicate that all devices and nets have been successfully matched. However, there are still two parameter value errors.

```
Check <<results\param.lvs>> file.  There are 2 parameter errors.
```

```
         SCHEMATIC                          |              LAYOUT
                                            |
# :1                                        | # :1
XXB3STF.XIN3_9.MN1                          | 2529
X :0                  Y :0                  | X :427             Y :142
MODEL :MN             TYPE :NMOS            | MODEL :NMOS        TYPE :NMOS
LENGTH :1             WIDTH :6              | LENGTH :1          WIDTH :2
                                            |
# :2                                        | # :2
XXB3STF.XIN3_9.MP1                          | 2327
X :0                  Y :0                  | X :427             Y :134
MODEL :MP             TYPE :PMOS            | MODEL :PMOS        TYPE :PMOS
LENGTH :1             WIDTH :12             | LENGTH :1          WIDTH :4
```

Figure 155: PARAM.LVS parameter error summary

You can locate one of these devices in the layout by editing the TOP181 cell and then executing the commands:

```
ND
N0 2529
```

Once you zoom in the display, the circuit will look like Figure 156. Unfortunately, this INV_1X cell is mislabeled XNA3_8. It really is the XIN3_9 subcircuit of the XXB3STF circuit as indicated by the matched device in the parameter summary. You can follow the connections to prove this to yourself. (Note that the output node, labeled "GB" in the layout is one of the inputs to the XNA3_10 NAND2 circuit, which is labeled correctly.)

When we look at these circuits in the schematic netlist (see Figure 157), we can see that XIN3_9 should be an INV3 circuit, not a INV1 circuit. This is the cause of the device size mismatches.

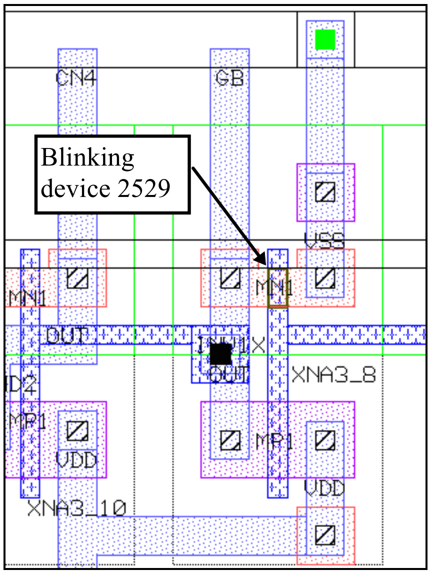


Figure 156: Portion of TOP181 cell with node 2529 outlined.

```
.SUBCKT B3STF A B C CN CN4 D E F G GB H PB VDD VSS
XIN3_1 VDD VSS B 10128 INV1
XNA3_2 VDD VSS A D 10131 NAND2
XNA3_3 VDD VSS A C F 10135 NAND3
XNA3_4 VDD VSS A C E H 10136 NAND4
XNA3_6 VDD VSS A C E G PB NAND4
XNA3_8 VDD VSS 10128 10131 10135 10136 10133 NAND4
XNA3_10 VDD VSS GB 10142 CN4 NAND2
XIN3_9 VDD VSS 10133 GB INV3
XNA3_5 VDD VSS A C E G CN 10142 NAND5
.ENDS
.SUBCKT INV1 VDD VSS IN OUT
MN1 OUT IN VSS VSS MN W=2U L=1.0U
MP1 OUT IN VDD VDD MP W=4U L=1.0U
.ENDS
.SUBCKT INV3 VDD VSS IN OUT
MN1 OUT IN VSS VSS MN W=6U L=1.0U
MP1 OUT IN VDD VDD MP W=12U L=1.0U
.ENDS
```

Figure 157: B3STF, INV1 and INV3 subcircuits in TOP181.CIR

We must replace the INV_1X cell in the B3STF cell with a INV_3X cell to correct this problem. To do this, we will use the TEDIT command again to edit the cell B3STF.

Type the TEDIT command as follows:

TEDIT SELECT

Place your cursor over one of the components in the inverter (any spot over the outlined device 2529 will do). Now press the left mouse button. You are now editing the B3STF cell.

Now select the INV_1X cell with the command:

SELECT CELL INV_1X IN

Then select two points with the cursor so that the select rectangle overlaps the inverter. Select marks will appear on the cell boundary and the prompt line will show you that one item is selected. Delete this cell with the command:

```
DELETE
```

We can correct the text at this point as well. Select the text component "XNA3_8". We can edit this text without deleting the component by using the @ED command. Type at the prompt:

```
@ED
```

You can use the @ED command to edit other attributes of a component, such as the width of wires.

At this point, an ASCII text editor will be launched with the ADD command for the text component shown on the screen. Change the string "XNA3_8" to "XIN3_9". Then exit and save the file. Now the ICED32™ layout editor resumes with the modified component shown on the screen.

We must add the INV_3X cell next. We will have to mirror it about the y-axis to add it in the correct orientation. Type the command:

```
ADD  CELL  INV_3X  MY  AT 422, 153
```

We must fix one short the old VDD wiring now causes. (See Figure 158.) To select the wire, type the commands:

```
UNSELECT ALL
SELECT  LAYER  MT1  SIDE   IN  428, 128  429, 130
```

This will select only the sides we need to shift. Now use the MOVE SIDE X command to shift these sides to the left so that the MT1 polygon no longer overlaps the inverter cell.

Now type the EXIT command so that you are editing the TOP181 cell again. Type the DRC and EXIT commands. Then run the NLE and LVS again.

Unfortunately, this last change caused the node numbers applied by the NLE to change. The node label overrides we added back on page 417 now refer to the wrong node numbers. This is one drawback of using node label overrides.

You can see the symptom of this problem in the console message:

```
Check <<results\unmatch.lvs>> file. There are [0+4] unmatched devices and nets.
```

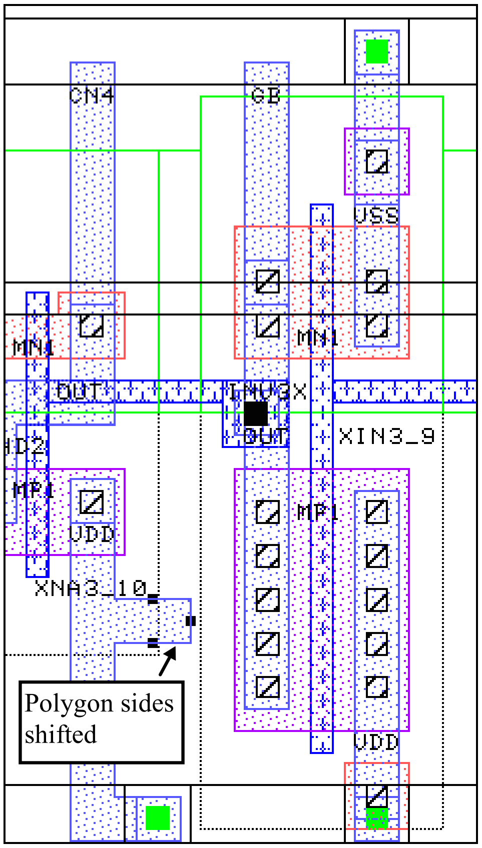


Figure 158: B3STF cell with INV_3X cell placed and short fixed.

To see more details on why this happened, look at the unmatched devices report UNMATCH.LVS. (See Figure 160.) Note that this is virtually identical to the unmatched net listing shown in Figure 139 on page 415 and Figure 140 on page 416. These are the new node numbers we must virtually connect.

At this point, we can edit the overrides to correct the node numbers, or add labels to the layout and re-execute the NLE to pass these labels to the LVS to virtually connect the nodes.

We will edit the overrides in this tutorial, but you may want to try to solve the problem by virtually connecting the nodes with labels in the layout as an exercise. (Add text components of "VDD:" on design layers for each unconnected fragment of VDD. Then re-execute the NLE.)

Edit the node label override file, LBLOVR.TXT. Change the node number 1723 to 1731 and node 1650 to 1658. The file should now look like Figure 159. Save the file and re-execute the LVS.

```
#####################################################
##  The following LAYOUT NETS were not matched.    ##
##          **NO POTENTIAL MATCHES**               ##
#####################################################
# :1
LAYOUT :1658
CONNECTED TO :1 devices.
TERMINAL  DEVICE_NAME     MATCHED_DEVICE ?
DRAIN         2350            XXB1STF.XNA1_6.MP1

# :2
LAYOUT :1731
CONNECTED TO :2 devices.
TERMINAL  DEVICE_NAME     MATCHED_DEVICE ?
SOURCE        2414            XXB3STF.XNA3_6.MP1
DRAIN         2415            XXB3STF.XNA3_6.MP2

# :3
LAYOUT : 2690[V,2,VDD:]
CONNECTED TO :337 devices.
TERMINAL  DEVICE_NAME     MATCHED_DEVICE ?
SOURCE        2326            XINT1.MP1
SUBSTRATE  2326            XINT1.MP1
:
```

Figure 160: Fragment of unmatched devices report.

The circuits in the two netlists now match exactly. However, remember that since we have fixed the opens in the VDD net with virtual connections, the opens still exist in the layout. If you do these kind of quick fixes when you are running the LVS on your circuits, you must remember to fix the real errors at some point. You should always turn off virtual connections for your final LVS runs on your chip. The easiest way to do this is to add the parameter "/V=NO" to the LVS command line. **If you fail to do this, your quick fixes involving virtual connections will prevent the real opens from being reported.**

```
*.Netlabel 1 VSS:
*.Netlabel 4 VSS:
*.Netlabel 6 VSS:

*.Netlabel 7      VDD:
*.Netlabel 1658 VDD:
*.Netlabel 1731 VDD:
*.Netlabel 23     VDD:
```

Figure 159: New LBLOVR.TXT file.

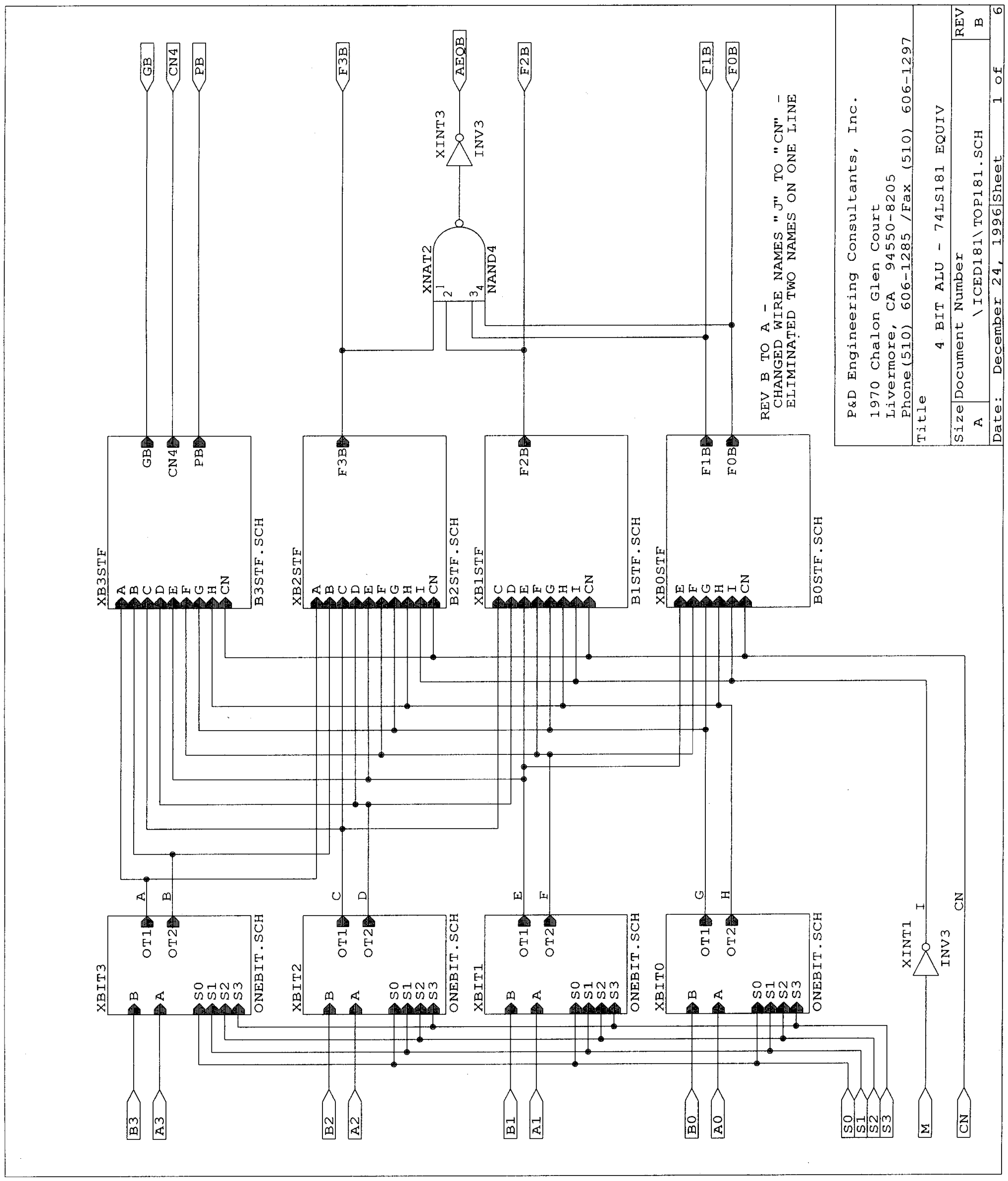
This concludes the tutorial. You can continue to experiment with this circuit. Try changing control file options and comparing the results. If you want to compare results, remember to change the output directory with the "/O" command line option to avoid overwriting your last report directory.

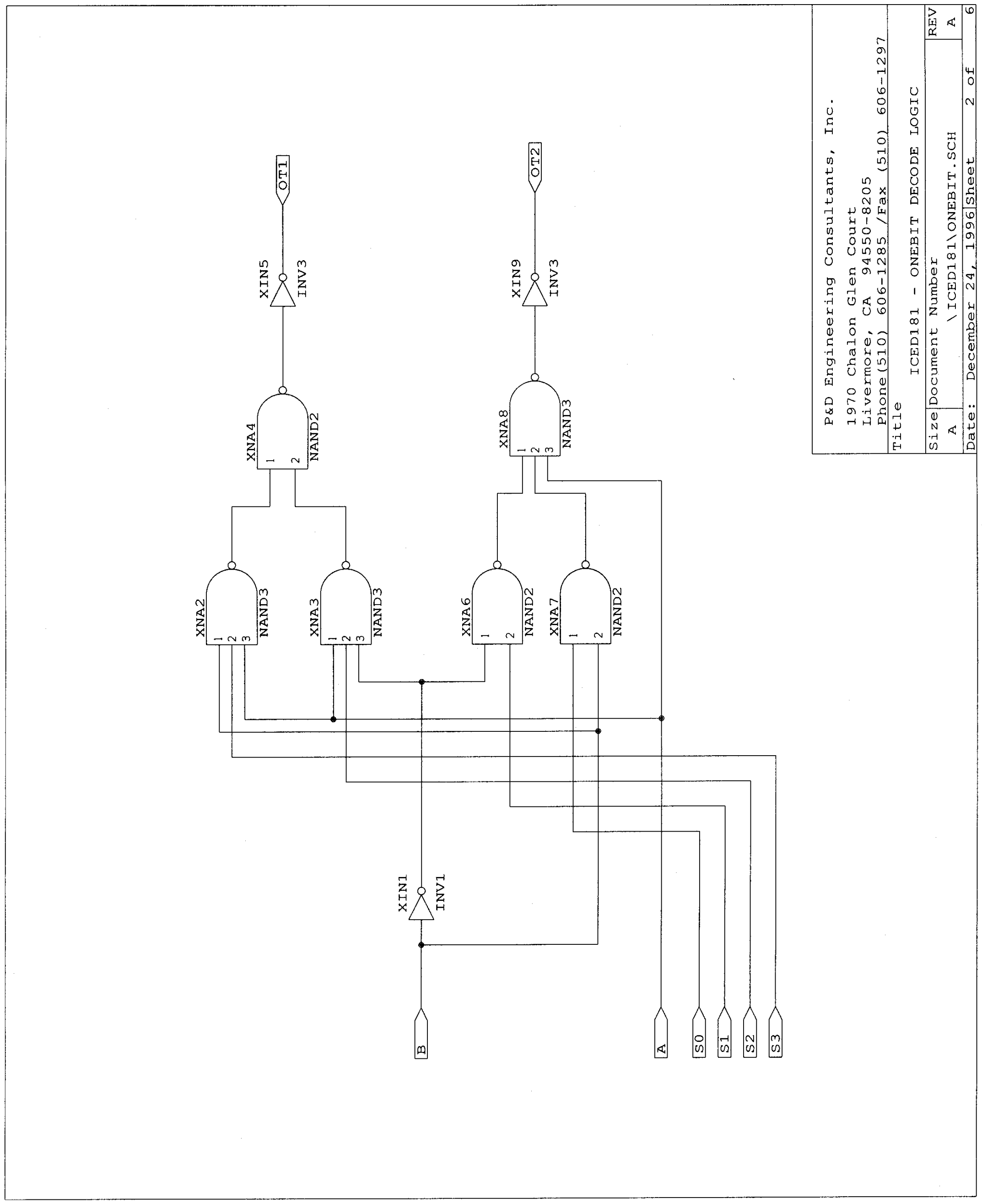
Appendix: Tutorial Schematics

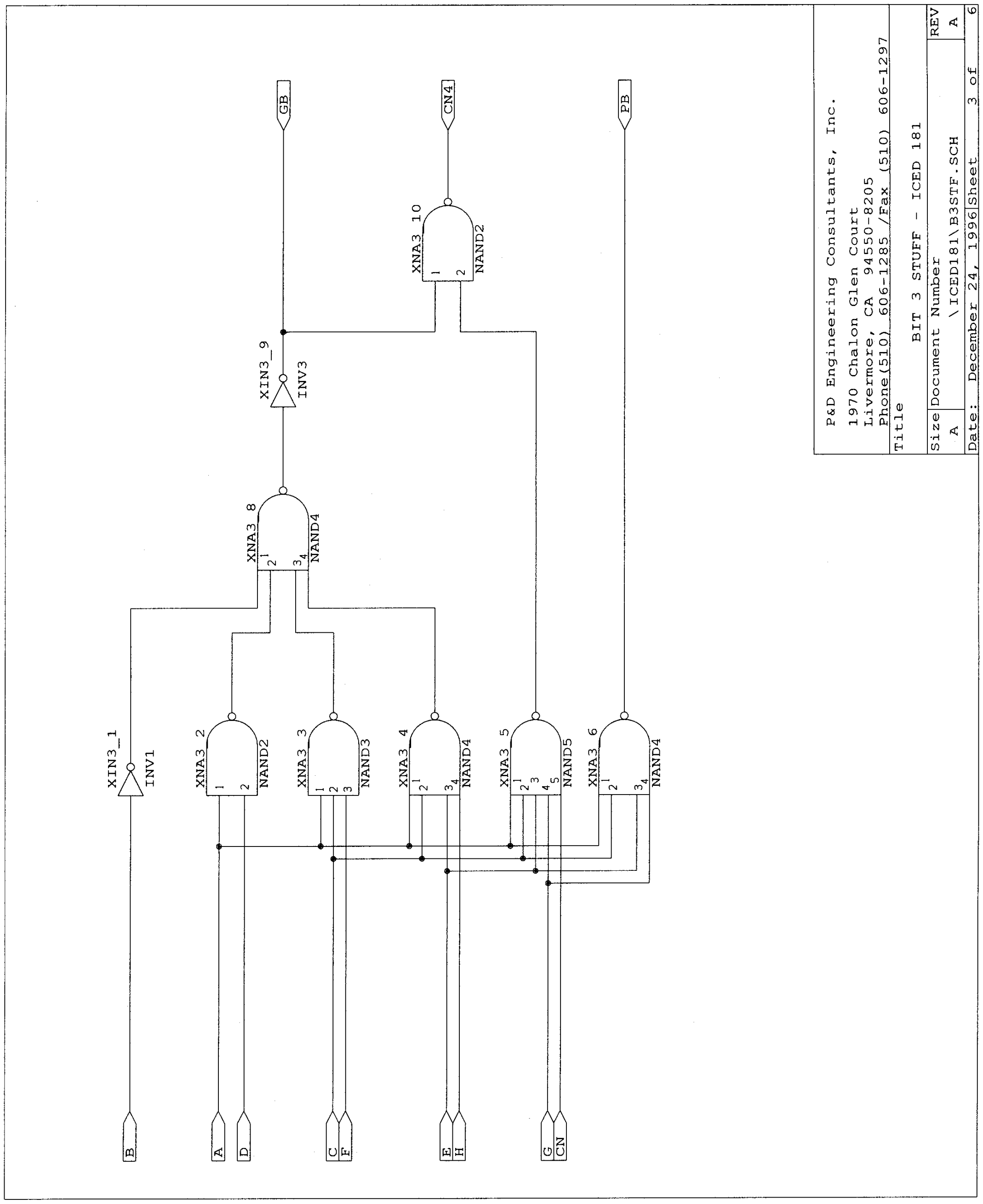
The following pages represent the schematics used to generate the schematic netlist for the Quick and Advanced Tutorials. This schematic netlist, TOP181.CIR, can be found in the Q:\ICED\SAMPLES\74181\LVS directory.

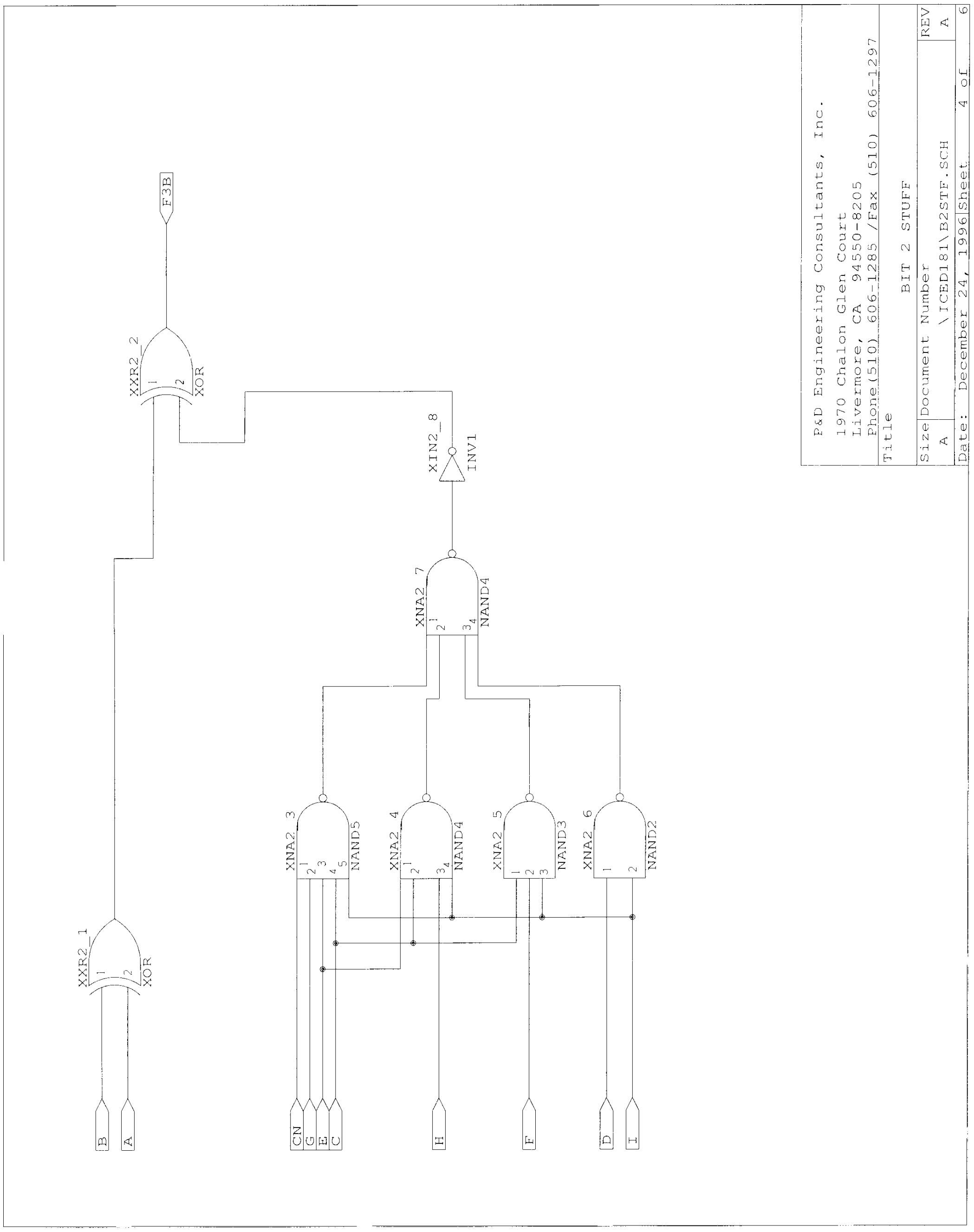
This circuit is a CMOS simulation of a 74181 4-bit ALU. The schematics and layout were provided by Michael Gentry of MGC, Inc.

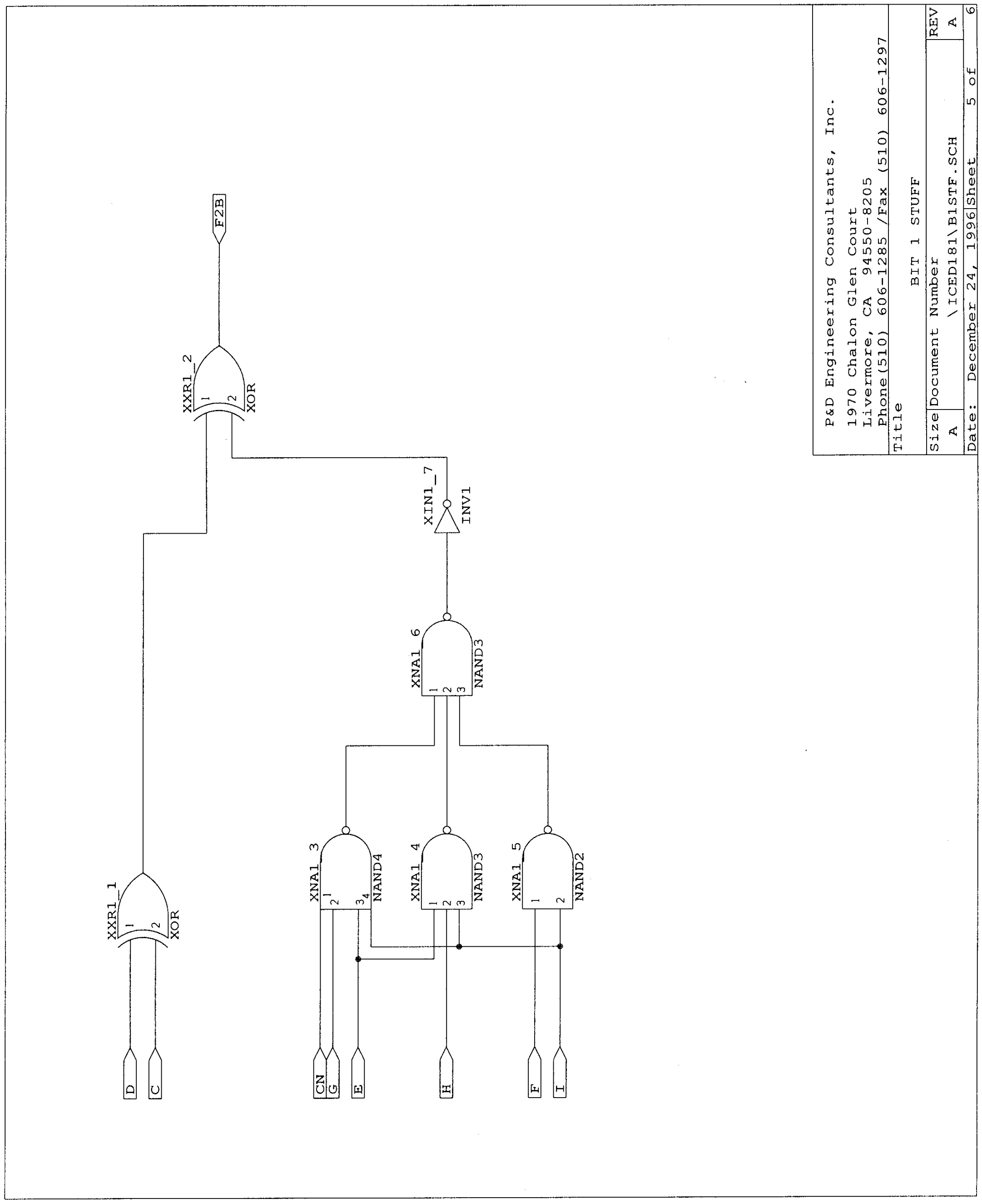
You can download copies of these schematics from the internet and print larger copies on your own printer. See the ICED website at ICEDITORS.COM.

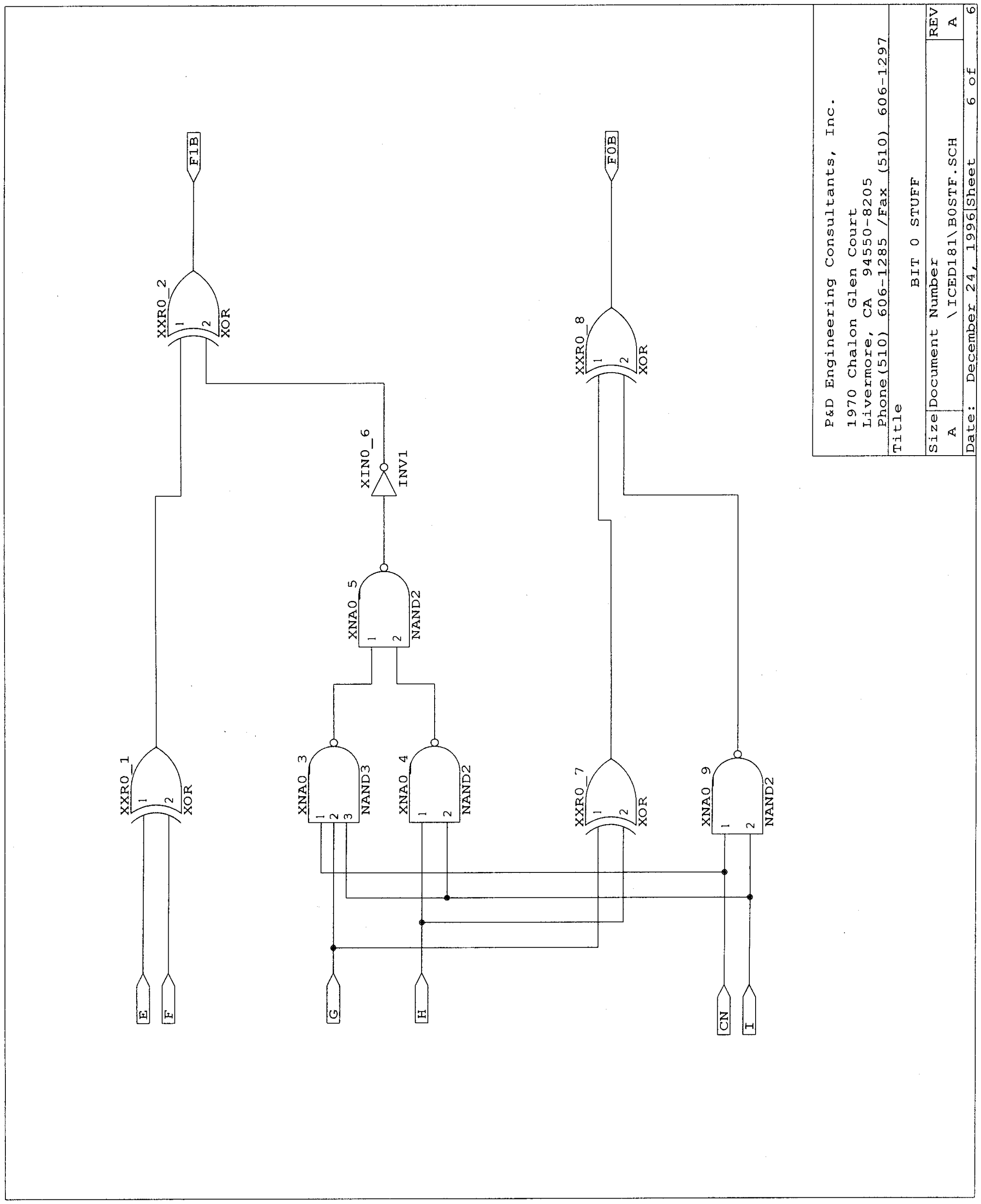

P&D Engineering Consultants, Inc.
1970 Chalon Glen Court
Livermore, CA 94550-8205
Phone(510) 606-1285 /Fax (510) 606-1297
Title
4 BIT ALU - 74LS181 EQUIV
Size A
Document Number \ICED181\TOP181.SCH
REV B
Date: December 24, 1996 Sheet 1 of 6
REV B TO A -
CHANGED WIRE NAMES "J" TO "CN" -
ELIMINATED TWO NAMES ON ONE LINE
XB3STF
B3STF.SCH
XB2STF
B2STF.SCH
XB1STF
B1STF.SCH
XB0STF
B0STF.SCH
XBIT3
XBIT2
XBIT1
XBIT0
ONEBIT.SCH
XNAT2
NAND4
XINT3
INV3
XINT1
INV3
GB
CN4
PB
F3B
AEQB
F2B
F1B
F0B
OT1
OT2
S0
S1
S2
S3
B3
A3
B2
A2
B1
A1
B0
A0
M
CN

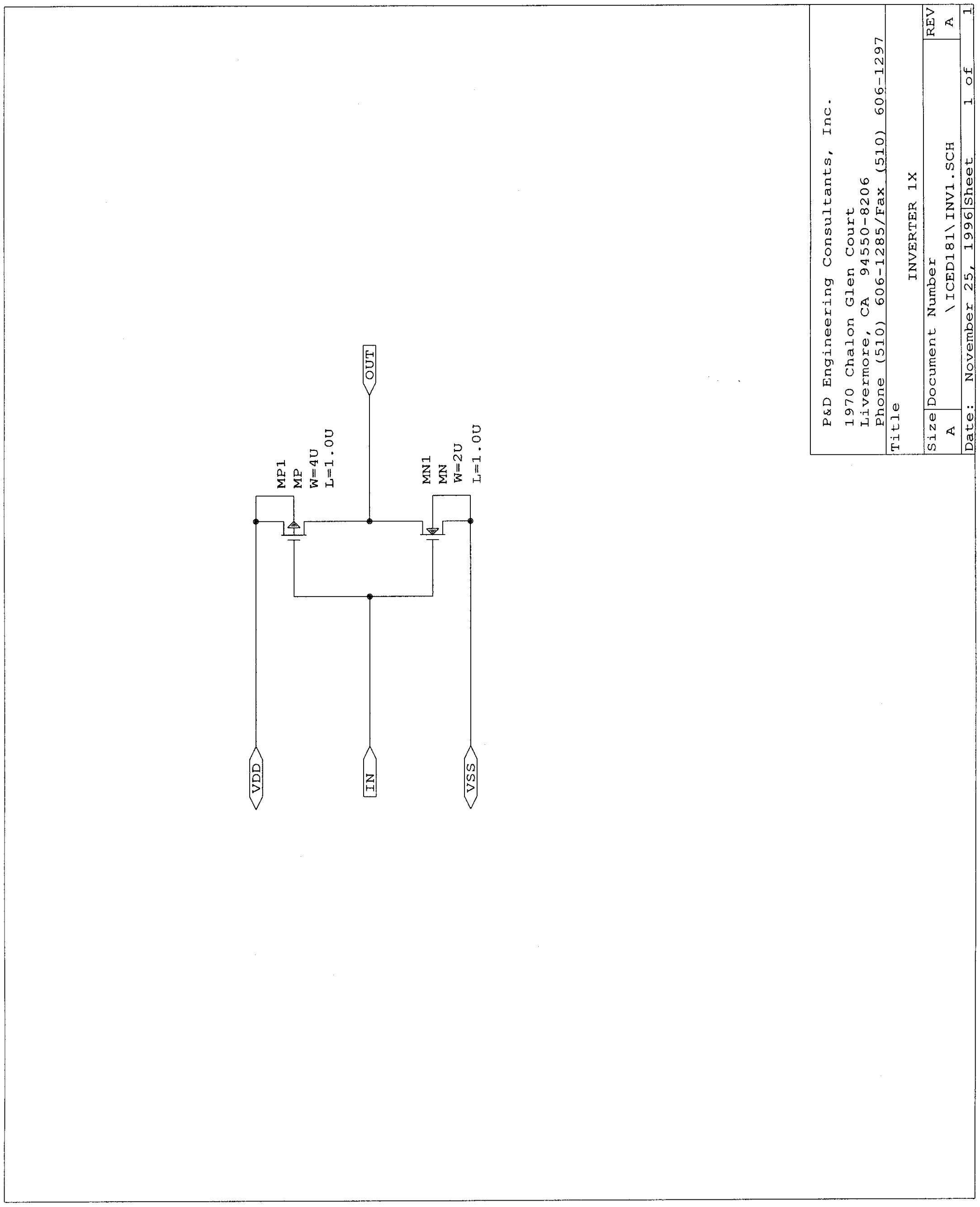

OT1
XIN5
INV3
OT2
XIN9
INV3
XNA4
NAND2
1
2
XNA8
NAND3
1
2
3
XNA2
NAND3
1
2
3
XNA3
NAND3
1
2
3
XNA6
NAND2
1
2
XNA7
NAND2
1
2
XIN1
INV1
B
A
S0
S1
S2
S3
P&D Engineering Consultants, Inc.
1970 Chalon Glen Court
Livermore, CA 94550-8205
Phone(510) 606-1285 /Fax (510) 606-1297
Title
ICED181 - ONEBIT DECODE LOGIC
Size
A
Document Number
\ICED181\ONEBIT.SCH
REV
A
Date: December 24, 1996
Sheet 2 of 6

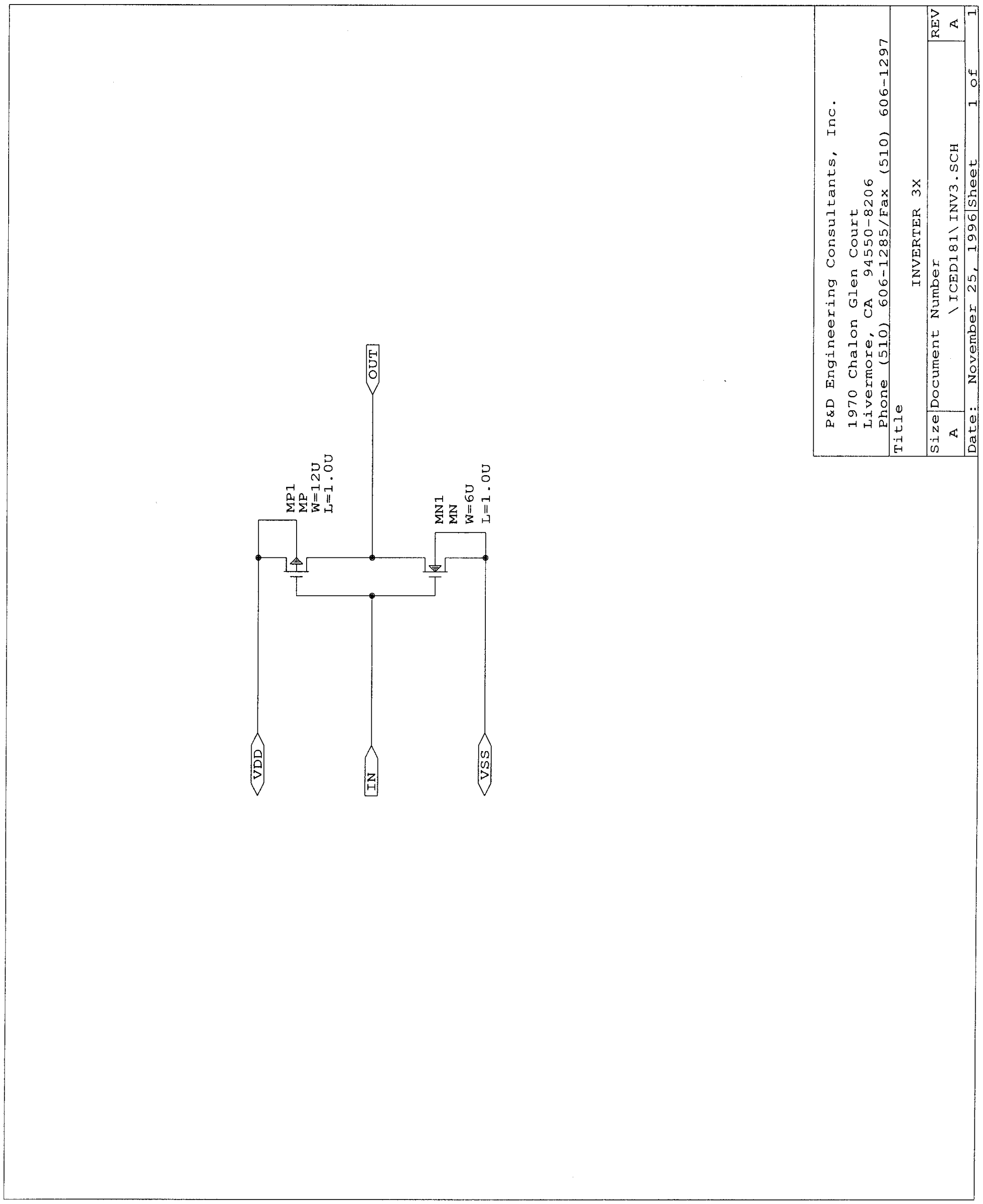

GB
CN4
PB
XNA3_10
NAND2
XIN3_9
INV3
XNA3_8
NAND4
XIN3_1
INV1
XNA3_2
NAND2
XNA3_3
NAND3
XNA3_4
NAND4
XNA3_5
NAND5
XNA3_6
NAND4
B
A
D
C
F
E
H
G
CN
P&D Engineering Consultants, Inc.
1970 Chalon Glen Court
Livermore, CA 94550-8205
Phone(510) 606-1285 /Fax (510) 606-1297
Title
BIT 3 STUFF - ICED 181
Size A
Document Number \ICED181\B3STF.SCH
REV A
Date: December 24, 1996
Sheet 3 of 6

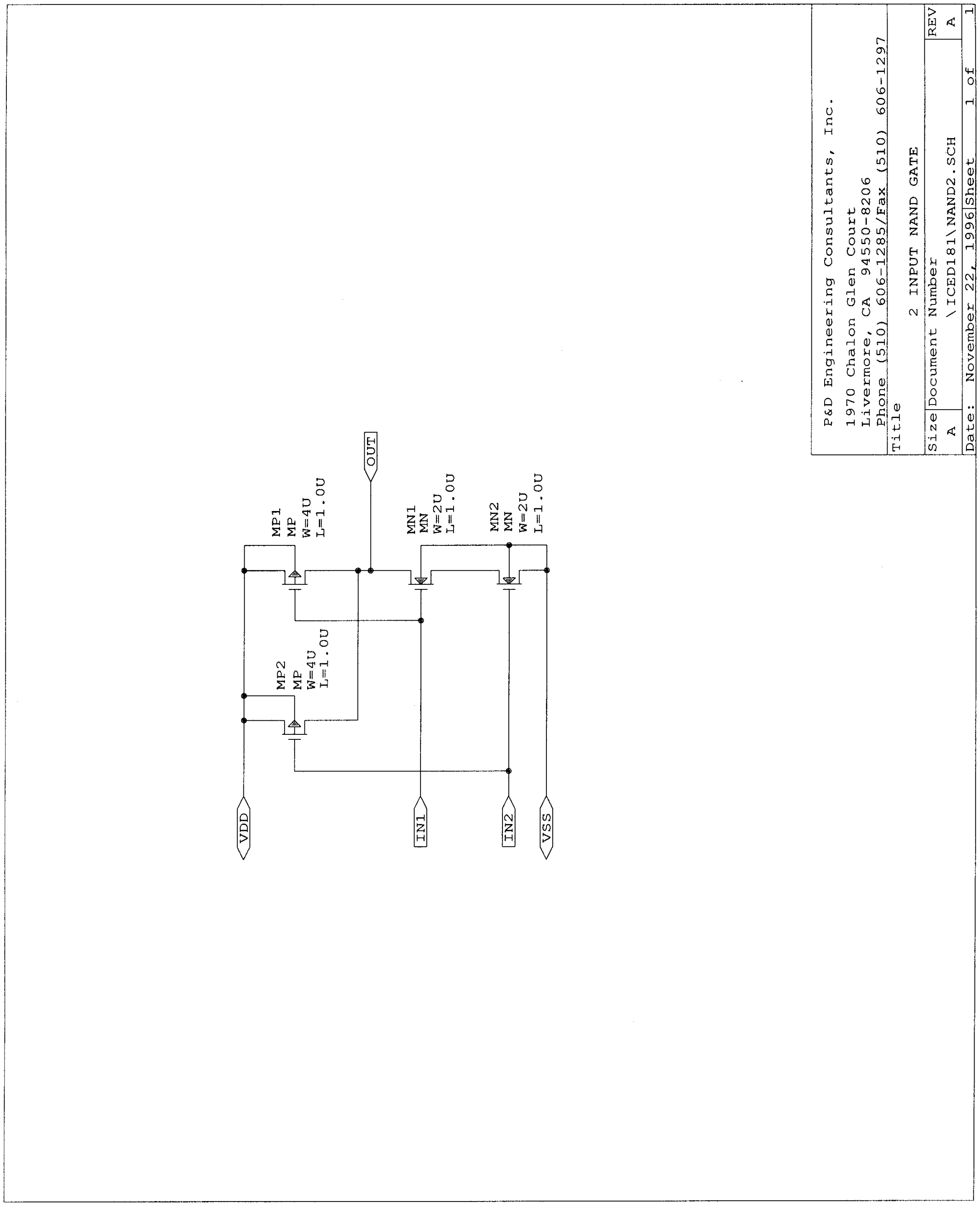

P&D Engineering Consultants, Inc.
1970 Chalon Glen Court
Livermore, CA 94550-8205
Phone(510) 606-1285 /Fax (510) 606-1297
Title
BIT 2 STUFF
Size
A
Document Number
\ICED181\B2STF.SCH
REV
A
Date: December 24, 1996
Sheet 4 of 6
F3B
XXR2_2
XOR
XIN2_8
INV1
XNA2_7
NAND4
XNA2_3
NAND5
XNA2_4
NAND4
XNA2_5
NAND3
XNA2_6
NAND2
XXR2_1
XOR
B
A
CN
G
E
C
H
F
D
I

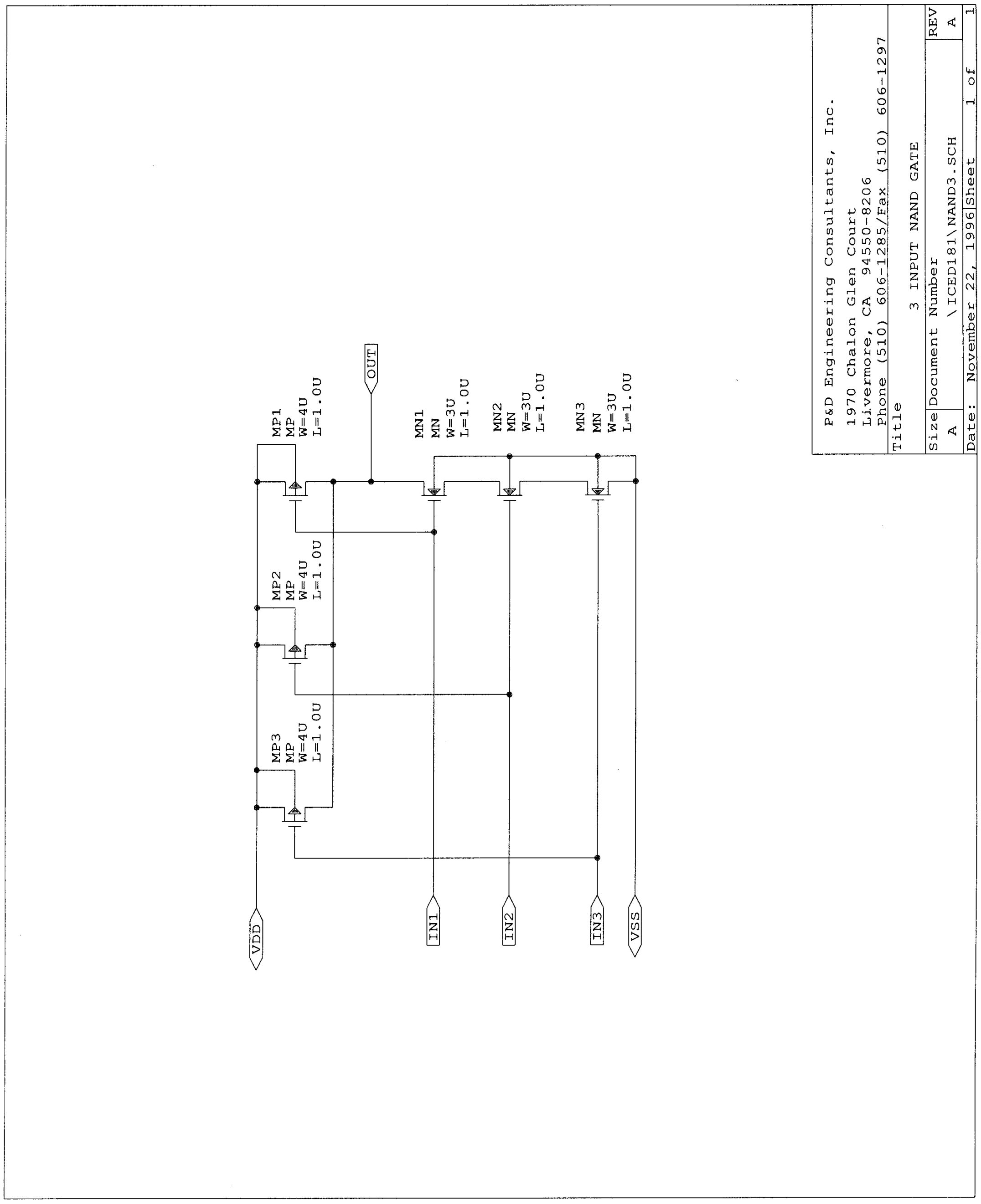

P&D Engineering Consultants, Inc.
1970 Chalon Glen Court
Livermore, CA 94550-8205
Phone(510) 606-1285 /Fax (510) 606-1297
Title
BIT 1 STUFF
Size A
Document Number \ICED181\B1STF.SCH
REV A
Date: December 24, 1996 Sheet 5 of 6
F2B
XXR1_2
XOR
XIN1_7
INV1
XNA1_6
NAND3
XXR1_1
XOR
XNA1_3
NAND4
XNA1_4
NAND3
XNA1_5
NAND2
D
C
CN
G
E
H
F
I

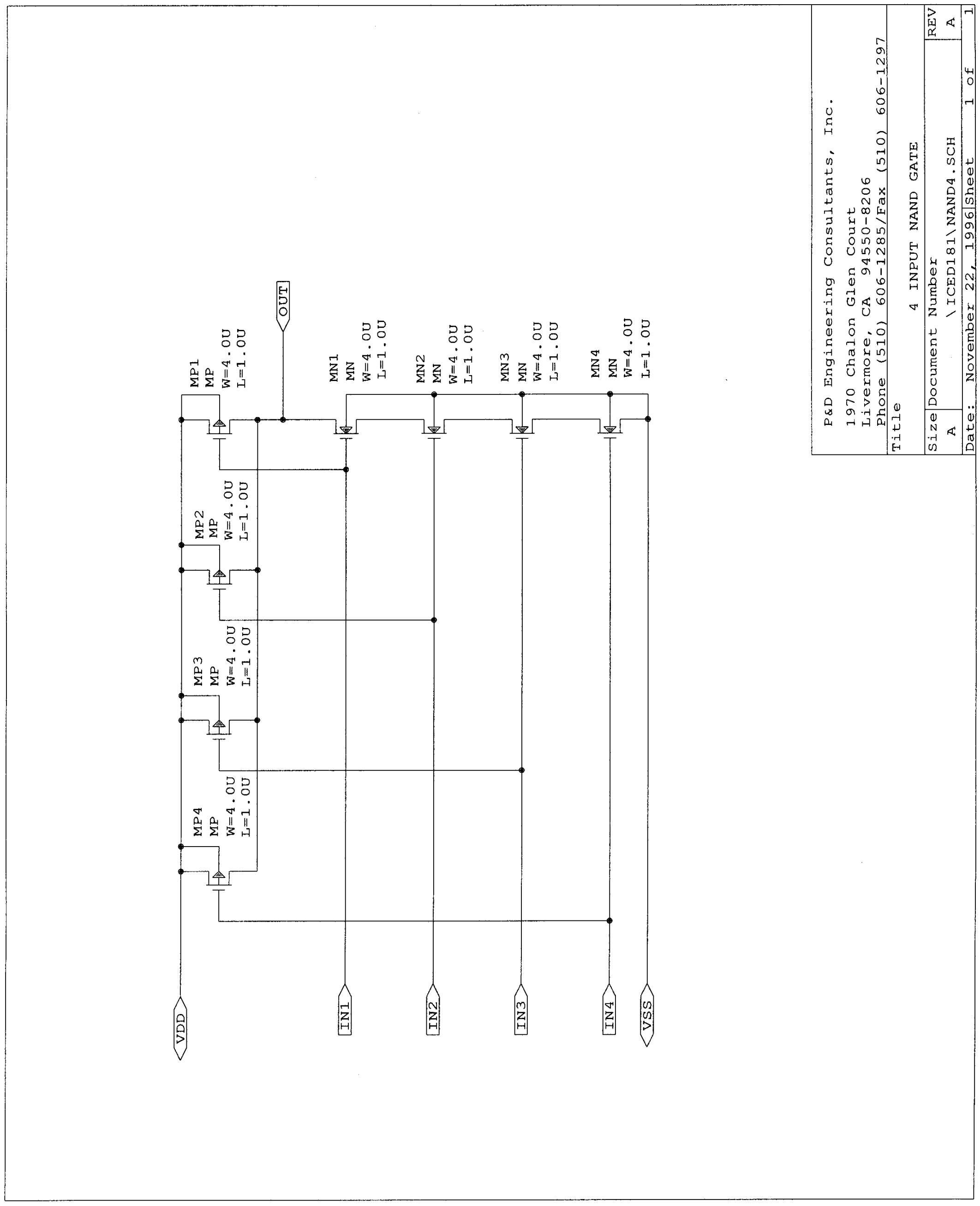

P&D Engineering Consultants, Inc.
1970 Chalon Glen Court
Livermore, CA 94550-8205
Phone(510) 606-1285 /Fax (510) 606-1297
Title
BIT 0 STUFF
Size
A
Document Number
\ICED181\B0STF.SCH
REV
A
Date: December 24, 1996
Sheet 6 of 6
F1B
F0B
XXR0_2
XOR
XXR0_8
XOR
XIN0_6
INV1
XNA0_5
NAND2
XXR0_1
XOR
XNA0_3
NAND3
XNA0_4
NAND2
XXR0_7
XOR
XNA0_9
NAND2
E
F
G
H
CN
I

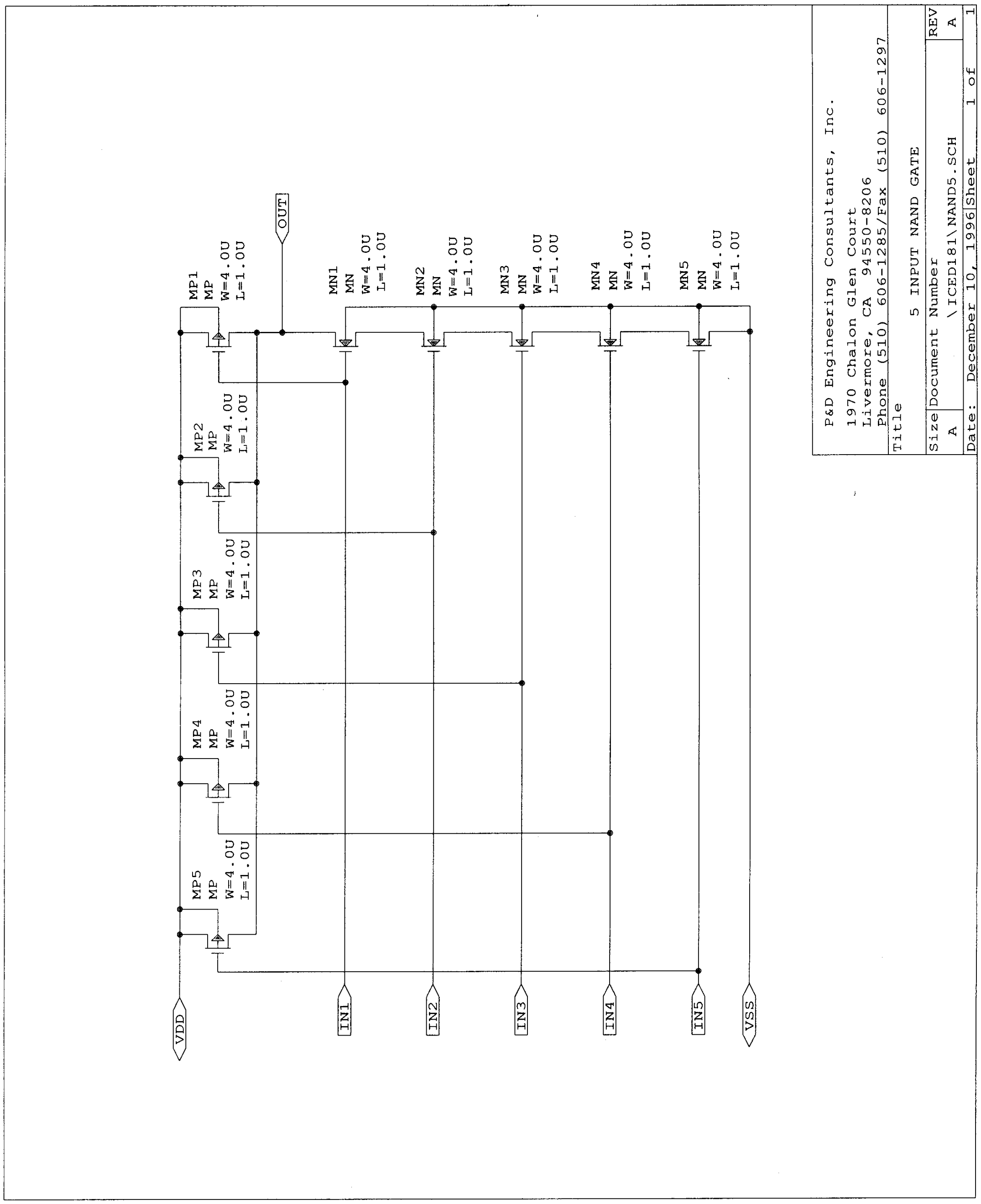

P&D Engineering Consultants, Inc.
1970 Chalon Glen Court
Livermore, CA 94550-8206
Phone (510) 606-1285/Fax (510) 606-1297
Title
INVERTER 1X
Size
A
Document Number
\ICED181\INV1.SCH
REV
A
Date: November 25, 1996
Sheet 1 of 1
MP1
MP
W=4U
L=1.0U
MN1
MN
W=2U
L=1.0U
OUT
VDD
IN
VSS

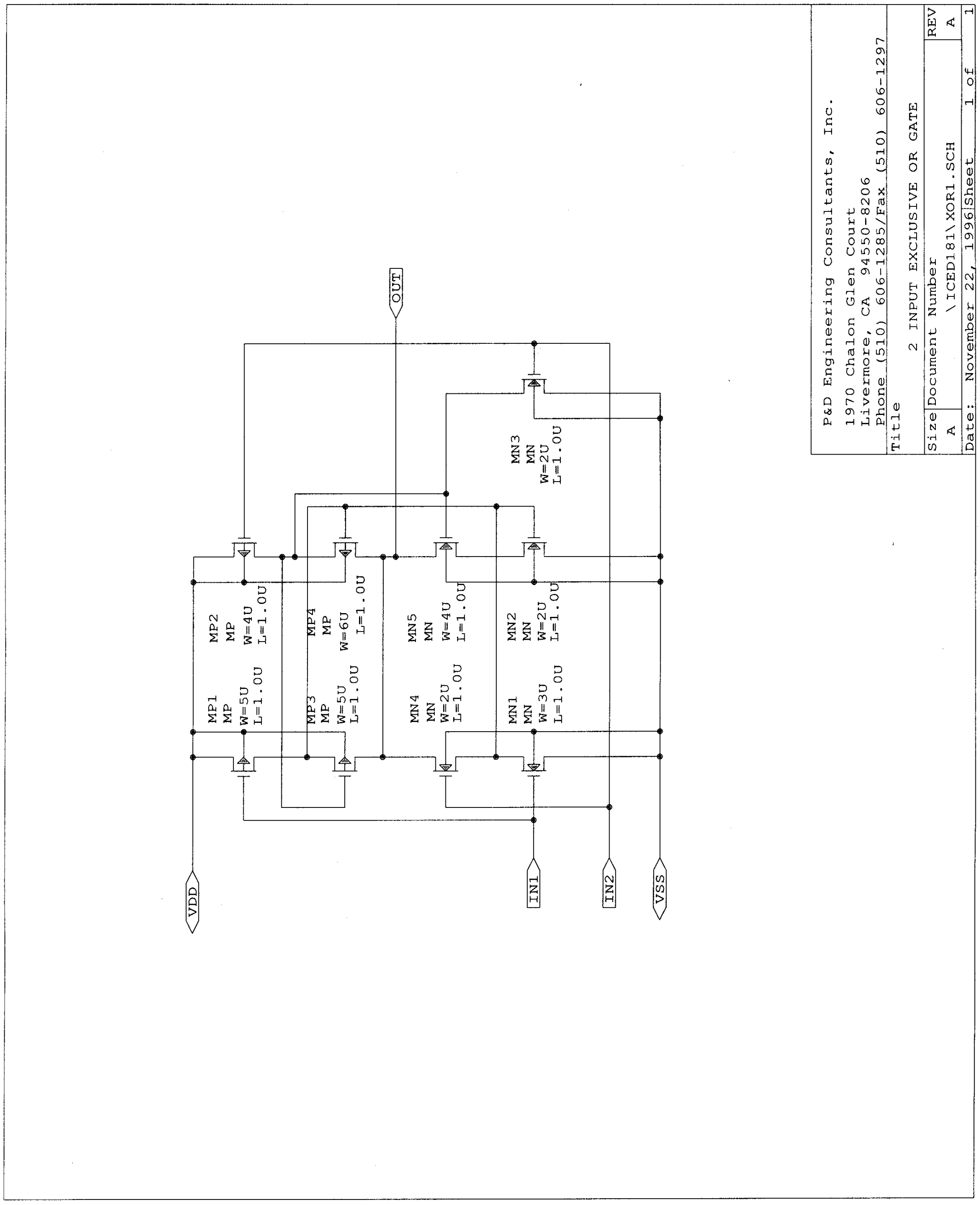

P&D Engineering Consultants, Inc.
1970 Chalon Glen Court
Livermore, CA 94550-8206
Phone (510) 606-1285/Fax (510) 606-1297
Title
INVERTER 3X
Size
A
Document Number
\ICED181\INV3.SCH
REV
A
Date: November 25, 1996
Sheet 1 of 1
OUT
MP1
MP
W=12U
L=1.0U
MN1
MN
W=6U
L=1.0U
VDD
IN
VSS


MP1
MP
W=4U
L=1.0U
MP2
MP
W=4U
L=1.0U
MN1
MN
W=2U
L=1.0U
MN2
MN
W=2U
L=1.0U
OUT
VDD
IN1
IN2
VSS
P&D Engineering Consultants, Inc.
1970 Chalon Glen Court
Livermore, CA 94550-8206
Phone (510) 606-1285/Fax (510) 606-1297
Title
2 INPUT NAND GATE
Size
A
Document Number
\ICED181\NAND2.SCH
REV
A
Date: November 22, 1996
Sheet 1 of 1


MP1
MP
W=4U
L=1.0U
MP2
MP
W=4U
L=1.0U
MP3
MP
W=4U
L=1.0U
OUT
MN1
MN
W=3U
L=1.0U
MN2
MN
W=3U
L=1.0U
MN3
MN
W=3U
L=1.0U
VDD
IN1
IN2
IN3
VSS
P&D Engineering Consultants, Inc.
1970 Chalon Glen Court
Livermore, CA 94550-8206
Phone (510) 606-1285/Fax (510) 606-1297
Title
3 INPUT NAND GATE
Size
A
Document Number
\ICED181\NAND3.SCH
REV
A
Date: November 22, 1996
Sheet 1 of 1


MP1
MP
W=4.0U
L=1.0U
MP2
MP
W=4.0U
L=1.0U
MP3
MP
W=4.0U
L=1.0U
MP4
MP
W=4.0U
L=1.0U
MN1
MN
W=4.0U
L=1.0U
MN2
MN
W=4.0U
L=1.0U
MN3
MN
W=4.0U
L=1.0U
MN4
MN
W=4.0U
L=1.0U
OUT
VDD
IN1
IN2
IN3
IN4
VSS
P&D Engineering Consultants, Inc.
1970 Chalon Glen Court
Livermore, CA 94550-8206
Phone (510) 606-1285/Fax (510) 606-1297
Title
4 INPUT NAND GATE
Size
A
Document Number
\ICED181\NAND4.SCH
REV
A
Date: November 22, 1996
Sheet 1 of 1


MP1
MP
W=4.0U
L=1.0U
MP2
MP
W=4.0U
L=1.0U
MP3
MP
W=4.0U
L=1.0U
MP4
MP
W=4.0U
L=1.0U
MP5
MP
W=4.0U
L=1.0U
OUT
MN1
MN
W=4.0U
L=1.0U
MN2
MN
W=4.0U
L=1.0U
MN3
MN
W=4.0U
L=1.0U
MN4
MN
W=4.0U
L=1.0U
MN5
MN
W=4.0U
L=1.0U
VDD
IN1
IN2
IN3
IN4
IN5
VSS
P&D Engineering Consultants, Inc.
1970 Chalon Glen Court
Livermore, CA 94550-8206
Phone (510) 606-1285/Fax (510) 606-1297
Title
5 INPUT NAND GATE
Size
A
Document Number
\ICED181\NAND5.SCH
REV
A
Date: December 10, 1996
Sheet 1 of 1


P&D Engineering Consultants, Inc.
1970 Chalon Glen Court
Livermore, CA 94550-8206
Phone (510) 606-1285/Fax (510) 606-1297
Title 2 INPUT EXCLUSIVE OR GATE
Size A
Document Number \ICED181\XOR1.SCH
REV A
Date: November 22, 1996
Sheet 1 of 1
OUT
VDD
IN1
IN2
VSS
MP1 MP W=5U L=1.0U
MP2 MP W=4U L=1.0U
MP3 MP W=5U L=1.0U
MP4 MP W=6U L=1.0U
MN4 MN W=2U L=1.0U
MN5 MN W=4U L=1.0U
MN1 MN W=3U L=1.0U
MN2 MN W=2U L=1.0U
MN3 MN W=2U L=1.0U

Index