6 LVS Output Tutorial

Introduction

LVS generates certain output upon execution, depending on which options are checked on the **Setup Window—Verbosity Level** (page 3-169) tab in the setup window. In the following examples, the **.vdb** files are opened in LVS and run with a single invocation. If run in batch mode, user interactions, defaults, and generated output may differ.

All files used in this chapter are available in the ***install_dir*\Samples\LVS** directory.

Parsing Information

LVS first reports parsing information about both of the netlists.

Following is an example of the output **ex1.out**, generated by LVS for **ex1.vdb**. The options for **Show Fragmented Classes**, **Show Automorph Classes**, and **Show Detailed Processing Information** are checked. The option for **Show information on Screen** is also checked, so all information written to the output file is also written to the verification window.

```
Exclude Bulk Nodes.............................OFF
Use Non-polarized Resistors....................OFF
Use Non-polarized Capacitors...................OFF
Replace Series MOSFETs.........................OFF
Optimize shorted & parallel R, C, MOSFETs
             series R and C.....................OFF
Fast Iteration.................................OFF

Parsing file ex1_1.spc...
Including file MODELS.SPC
Flattening network...
Parsing file ex1_2.spc...
Including file MODELS.SPC
Flattening network...

Device        ex1_1.spc     ex1_2.spc
-------       -----------   -----------
```

```
C              4             4
M_NSS          20            20
M_PSS          26            24 MISMATCH (2)
Total element  50            48 MISMATCH (2)
Total nodes    20            20
```

LVS begins the parsing information by stating which file was read and listing any files included in this file. After the network is flattened, the program reports the kinds of elements found and the number of nodes found. The program concatenates the device type with the model name to make up the element type, as in **M_NSS**. This helps distinguish between elements of the same device type that use different models.

If the program detects a numerical difference between the number of different elements and the number of nodes between the two files, it poses a question identifying the problem and requiring instructions to continue or exit. If the **Continue on Element/Node Count mismatch** option on the **Setup Window—Performance** (page 3-167) tab in the setup window is checked, or if LVS is invoked from batch mode, the program does not stop at this point and continues on to the iteration.

Parameter Matching Example

Following is an example of the output **ex2.out**, generated by LVS for **ex2.vdb**. In this example, LVS is instructed to use parameter matching (**Resistance, Capacitance and Inductance values** for **R, C, and L Elements** and **Lengths and Widths** for **MOSFET Elements** on the **Setup Window—Advanced Parameters** (page 3-164) tab), in addition to the topological characteristics. (See Parameter Matching on page 3-205 for more information on this method.)

```
Iterating...
5% done
10% done
...
25% done

Warning: Parametric mismatch between elements
ex1_2.spc: M346:L=2 W=5   (Not all decimals shown)
ex2_1.spc: M6(XSUBA/X34):L=2 W=6 (Not all decimals shown)

Warning: Parametric mismatch between elements
ex1_2.spc: M6:L=2 W=5   (Not all decimals shown)
ex2_1.spc: M6(XSUBA/X33):L=2 W=6   (Not all decimals shown)

50% done
...
```

In this example, during the iteration matching process LVS encountered elements that are topologically matched but are parametrically different. LVS reports this as a warning, but it does not affect the iteration process.

Automorph Class Example

When LVS finishes iteration, it reports any automorphism or fragmentation that occurs. Automorph classes are listed first.

Following is an example of the output **ex3.out**, generated by LVS for **ex3_1.vdb**.

```
*************** REPORTING AUTOMORPHISM **************

Report of elements:
*******************
Automorph class of elements
Ex3_1.spc M1(XSUBA/X34)_2fanout: BULK = 41D/S ( 3, 7) G = 7
Ex3_1.spc M1(XSUBA/X34)_1fanout: BULK = 41D/S ( 3, 7) G = 7
Ex3_2.spc M350      fanout: BULK = 41   D/S ( 3, 7)   G = 7
Ex3_2.spc M341      fanout: BULK = 41   D/S ( 3, 7)   G = 7
-------------------------------------------------
Automorph class of elements
...
...
...
-------------------------------------------------
Automorph class of elements
Ex3_1.spc: 4 element(s)
Ex3_2.spc: 4 element(s)
Ex3_1.spcC1(XSUBA/X34)_2   fanout: POS/NEG ( 35,  41)
Ex3_1.spcC1(XSUBA/X34)_1   fanout: POS/NEG ( 35,  41)
```

```
Ex3_1.spcC1(XSUBA/X33)_2   fanout: POS/NEG ( 35,  41)
Ex3_1.spcC1(XSUBA/X33)_1   fanout: POS/NEG ( 35,  41)
Ex3_2.spcC1_4              fanout: POS/NEG ( 35,  41)
Ex3_2.spcC1_3              fanout: POS/NEG ( 35,  41)
Ex3_2.spcC1_2              fanout: POS/NEG ( 35,  41)
Ex3_2.spcC1_1              fanout: POS/NEG ( 35,  41)
--------------------------------------------------
    32 perfectly matched element class(es)
     7 automorph element class(es)
    20 perfectly matched node class(es)
```

Each element is listed on a separate line. For each element, the netlist file of origin is listed first (**Ex3_1.spc**) followed by the unique name of the element (**M1(XSUBA/X34)_2**). Hierarchical information is provided within the parentheses as well as with the appended number at the end. Whenever LVS encounters a line such as **M1 N39 N23 N25 VDD PSS L=2 W=5 M=2** (line 14 of file **Ex3_1.spc**), the multiplier **M=2** causes LVS to create several identical copies of an element. In this case, two copies are made. To distinguish between them, LVS appends a number at the end of each constructed element name.

The fanout tells what the “electrical” fanout of each terminal is. This describes the number of elements each terminal of the element is connected to, not counting the one obvious connection to itself. For example, element **M1(XSUBA/X34)_2** has 41 elements connected to its bulk terminal, three elements to either its drain or source terminal, seven elements to the other of drain and source, and seven elements to its gate terminal. (LVS cannot distinguish between source and drain for FET transistors since there is no topological distinction between them.) All elements within a single automorph class have exactly the same fanout.

Since the **Detailed Trial Matching to resolve Automorph Classes** option was not checked in this example (see Detailed Trial Matching on page 3-204), an alert box appears asking if LVS should use detailed trial matching to try and match the elements and nodes of the automorph classes. Following are the results of this process:

```
Doing detailed trial matching...
90% done
95% done
100% done

Circuits are equal!
```

During detailed trial matching, LVS attempts to pair up the members of automorph classes. In most cases, this resolves the automorphism and the circuits are found equal. Occasionally the trial matching results in fragmentation; however, in the case of attempting to resolve an automorph class, it does not mean the two circuits are different. It only means that the particular trial matches did not resolve the particular automorph class.

Resolving Fragmentation of an Automorph Class

When detailed trial matching causes fragmentation of an automorph class, there are usually two reasons for this result:

- The trial-matching algorithm is not able to solve the problem. In this case you should provide a prematch file for sets of nodes and elements, giving

LVS a “head start” on matching elements. (See Preiteration Matching on page 3-203 for more information on prematch files.)

- There is, in fact, a difference between the two files being compared.

In either case, the way to move towards a solution at this point is to use a prematch file. Rerun LVS with the **Show Detailed Processing Information** option on the **Setup Window—Verbosity Level** (page 3-169) tab in the setup window checked. This option provides details about the various nodes and elements the detailed trial matching procedure matches. In most cases the last matched pair of elements or nodes causes the fragmentation. You will need to look back in your output and find a different match in the automorph class from which this last pair comes.

For example, if the last matched pair before fragmentation was:

```
Matched Elements C1(XSUBA/X34)_1 and C1_1
```

it means that LVS attempted to match element **C1(XSUBA/X34)_1** with element **C1_1**, and the match failed. Therefore, you should attempt to match a different element with **C1(XSUBA/X34)_1**, and a possible line in your prematch file would be:

```
C1(XSUBA/X34)_1 C1_4
```

The order in which elements and nodes are specified in a prematch file has to reflect the order in which the respective netlist files are specified on the command line.

With this line in the prematch file, rerun LVS again. Include the prematch file on the **Setup Window—File** (page 3-158) tab in the setup window.

If the problem of fragmentation recurs, then repeat the process above and add more lines of information to the prematch file.

If the two files being compared are equal, a few lines in the prematch file are all that is needed. If there is a discrepancy between the two input files, you will need to continue adding information to the prematch file until you are able to determine what the discrepancy is, using the methods of reasoning outlined above.

For the file **ex3_1.vdb**, detailed trial matching results in the following output:

```
....

************* POST ITERATION MATCHING *************
Doing detailed trial matching...
Matched Elements M1(XSUBA/X34)_2 and M350
Matched Elements M19(XSUBA/X34)_1 and M3419
Matched Elements M1(XSUBA/X33)_2 and M3
90% done
Matched Elements M19(XSUBA/X33)_1 and M19
Matched Elements M10(XSUBA/X33)_1 and M10
```

```
Matched Elements M10(XSUBA/X34)_1 and M3410
95% done
Matched Elements C1(XSUBA/X34)_2 and C1_4
Matched Elements C1(XSUBA/X33)_1 and C1_1
Matched Elements C1(XSUBA/X34)_1 and C1_3
100% done
*************** FINAL RESULT ******************
Circuits are equal.
```

Besides providing the information that the circuits are equal, the output also gives information about what elements were matched during the trial matching process. This information can be turned into a prematch file to speed up the iteration process next time LVS is run. The above output is listed in the prematch file as:

```
M1(XSUBA/X34)_2   M350
M19(XSUBA/X34)_1 M3419
M1(XSUBA/X33)_2   M3

M19(XSUBA/X33)_1 M19
M10(XSUBA/X33)_1 M10
M10(XSUBA/X34)_1 M3410

C1(XSUBA/X33)_1   C1_1
C1(XSUBA/X34)_1   C1_3
C1(XSUBA/X34)_2   C1_4
```

The name of the prematch file is **ex3_1.pre**, and it is entered in the **Prematch File** field on the **Setup Window—File** (page 3-158) tab in the setup window in **ex3_2.vdb**. Following is the output generated by LVS for **ex3_2.vdb**.

```
....
....
Processing file C:\lvs_product\Windows\Ex3_1.pre of matched elements/nodes
Matched Elements M1(XSUBA/X34)_2 and M350
Matched Elements M19(XSUBA/X34)_1 and M3419
Matched Elements M1(XSUBA/X33)_2 and M3
Matched Elements M19(XSUBA/X33)_1 and M19
Matched Elements M10(XSUBA/X33)_1 and M10
Matched Elements M10(XSUBA/X34)_1 and M3410
Matched Elements C1(XSUBA/X33)_1 and C1_1
Matched Elements C1(XSUBA/X34)_1 and C1_3
Matched Elements C1(XSUBA/X34)_2 and C1_4

********************************* ITERATING *********************************
Iterating...
....
....

************************POST ITERATION MATCHING ************************
Doing detailed trial matching...
Matched Elements C1(XSUBA/X33)_2 and C1_2
```

LVS matched nine out of ten of the original automorph elements before the iteration process began. In the file **ex3_3.vdb**, the last element is added to the prematch file **ex3_2.pre**. When LVS is run, all automorph classes are matched.

Automorphism occurs when LVS cannot fully distinguish between elements. In the example above, the reason is there are elements in parallel. Since the two files compared represent fully digital designs and since L and W are available for all transistors, as well as the capacitance values for the capacitors, LVS can be rerun with the **Optimize Network** option on the **Setup Window—Options** (page 3-161) tab in the setup window checked. This option optimizes the networks before the iteration begins. In this instance, all cases of automorphism are eliminated and the following output results:

```
Iterating...

Warning: Parametric mismatch between elements
C:\lvs_product\EXAMPLES\Ex1_1.spc: M11(XSUBA/X33)_1:L=2 W=10 (Not all decimals shown)
C:\lvs_product\EXAMPLES\Ex1_2.spc: M11:L=2 W=5   (Not all decimals shown)

Warning: Parametric mismatch between elements
C:\lvs_product\EXAMPLES\Ex1_1.spc: M11(XSUBA/X34)_2:L=2 W=10 (Not all decimals shown)
C:\lvs_product\EXAMPLES\Ex1_2.spc: M3411:L=2 W=5   (Not all decimals shown)

Warning: Parametric mismatch between elements
C:\lvs_product\EXAMPLES\Ex1_1.spc: M6(XSUBA/X33):L=2 W=6   (Not all decimals shown)
C:\lvs_product\EXAMPLES\Ex1_2.spc: M6:L=2 W=5   (Not all decimals shown)

Warning: Parametric mismatch between elements
C:\lvs_product\EXAMPLES\Ex1_2.spc: M346:L=2 W=5   (Not all decimals shown)
C:\lvs_product\EXAMPLES\Ex1_1.spc: M6(XSUBA/X34):L=2 W=6   (Not all decimals shown)


********************** ITERATION SUMMARY ***************************
```

```
38 perfectly matched element class(es)
20 perfectly matched node class(es)

0 error(s), 4 warning(s)

Circuits are equal.
```

In this example only automorph classes of elements have been shown. Automorph classes of nodes are addressed in a similar fashion.

Fragmented Class Example

Fragmentation can occur either during the regular iterative process or during the detailed trial matching procedure in the case of previous automorphism. Following is an example of the output (**ex4.out**) generated by LVS for **ex4.vdb**, including fragmented classes.

```
...
...
...
45% done
50% done
55% done
*************** REPORTING FRAGMENTATION **************

Report of elements:
*******************
Fragmented class of elements
ex1_1.spc: 6 element(s)
ex1_2.spc: 4 element(s)
...
...
...
---------------------------------------------------

Report of nodes:
****************
Fragmented class of nodes
ex1_2.spc 16  connected to  1 M_NSS_D/S     2 M_NSS_G     4 M_PSS_D/S     4 M_PSS_G
```

```
-------------------------------------------------
Fragmented class of nodes
ex1_2.spc 3   connected to 3 M_NSS_D/S1    M_NSS_G      3 M_PSS_D/S      1 M_PSS_G
ex1_2.spc 15  connected to 3 M_NSS_D/S1    M_NSS_G      3 M_PSS_D/S      1 M_PSS_G
-------------------------------------------------
...
...
...


-------------------------------------------------
Fragmented class of nodes
ex1_1.spc  VDD  connected to  4 C_POS/NEG    26 M_PSS_BULK    14 M_PSS_D/S
-------------------------------------------------
     0 perfectly matched element class(es)
     1 fragmented element class(es)
     4 perfectly matched node class(es)
    10 fragmented node class(es)
```

Fragmented element classes are listed first. In this example there are no fragmented element classes. Node classes are listed thereafter. The first class shows node **16** from file **ex1_2.spc**. The node is connected to one drain/source terminal on MOS transistors of type (model) **NSS**: two gate terminals of the same kind of transistor and four drain/source terminals on MOS transistors of type (model) **PSS**, and, finally, to four gate terminals of the same kind.

Resolving a Fragmented Class

If your design uses capacitors, resistors, or inductors, make sure that you treat them consistently either as polarized or nonpolarized devices. If the polarization of resistors is important, make sure you check the **Consider Resistors as Polarized Elements** option on the **Setup Window—Options** (page 3-161) tab in the setup window. If this option is not enabled, some devices may have the positive and negative terminals switched, causing fragmentation.

If your design is fully digital and you have not avoided permuted inputs to gates during layout and schematic design, fragmentation will result. LVS compares elements at a transistor level, and cannot interpret the design at the gate level. Permuted inputs to gates are a common cause of unnecessary fragmentation in digital designs. The **Replace Series MOSFETs** option on the **Setup Window—Options** (page 3-161) tab in the setup window triggers the replacement of series-chain transistors, such as the series of three *n*-transistors in a three-input NAND gate. In this case, the three *n*-transistors are replaced during the iteration process with an imaginary part with three permutable input terminals, representing the A, B, and C terminals of the gate. Since the transistors are now replaced, any fragmentation caused by permuted inputs to gates is eliminated. If your design contains both digital and analog parts, the digital parts need to be separated and compared separately in order to use this function. Otherwise, the replacement routine also transforms series chain transistors in the analog part of your design.

If your circuit files represent a typical ASIC design situation, you may have represented many parallel MOSFETs in your layout with a single MOSFET in

your schematic, with an equivalent width equal to the sum of the widths of your layout MOSFETs. You may also have represented single capacitors and resistors in your schematic with multiple devices in your layout. If either of these cases are applicable, it is worth enabling the **Optimize Network** option on the **Setup Window—Options** (page 3-161) tab in the setup window to optimize the networks before the iteration process begins.

Following is an example of the output generated by LVS for **ex4.vdb**. The output demonstrates the report of nodes in the Fragmentation section.

```
Report of nodes:
****************
Fragmented class of nodes
ex1_2.spc16   connected to  1 M_NSS_D/S   2 M_NSS_G
              4 M_PSS_D/S       4 M_PSS_G
-------------------------------------------------
Fragmented class of nodes
ex1_2.spc3    connected to  3 M_NSS_D/S   1 M_NSS_G
              3 M_PSS_D/S       1 M_PSS_G
ex1_2.spc15   connected to  3 M_NSS_D/S   1 M_NSS_G
              3 M_PSS_D/S       1 M_PSS_G
-------------------------------------------------
Fragmented class of nodes
ex1_2.spc7    connected to  6 M_NSS_G     6 M_PSS_G
-------------------------------------------------
Fragmented class of nodes
ex1_2.spc5    connected to  1 M_NSS_D/S   2 M_NSS_G
              2 M_PSS_D/S       4 M_PSS_G
```

```
-------------------------------------------------
Fragmented class of nodes
ex1_2.spc4    connected to  4 C_POS/NEG 24 M_PSS_BULK
              14 M_PSS_D/S
-------------------------------------------------
Fragmented class of nodes
ex1_1.spc5    connected to  1 M_NSS_D/S   2 M_NSS_G
              5 M_PSS_D/S       4 M_PSS_G
-------------------------------------------------
Fragmented class of nodes
ex1_1.spcN25(XSUBA/X33/)   connected to  3 M_NSS_D/S
              1 M_NSS_G     4 M_PSS_D/S   1 M_PSS_G
ex1_1.spcN25(XSUBA/X34/)   connected to  3 M_NSS_D/S
              1 M_NSS_G     4 M_PSS_D/S   1 M_PSS_G
-------------------------------------------------
Fragmented class of nodes
ex1_1.spc1    connected to  6 M_NSS_G      8 M_PSS_G
-------------------------------------------------
Fragmented class of nodes
ex1_1.spc8    connected to  1 M_NSS_D/S   2 M_NSS_G
              3 M_PSS_D/S4 M_PSS_G
-------------------------------------------------
Fragmented class of nodes
ex1_1.spcVDD  connected to  4 C_POS/NEG   26 M_PSS_BULK
              14 M_PSS_D/S
-------------------------------------------------
     0 perfectly matched element class(es)
     1 fragmented element class(es)
     4 perfectly matched node class(es)
```

```
    10 fragmented node class(es)
```

From the output we know that there is a discrepancy in the number of elements between the two files. File **ex1_1.spc** has two more **M_PSS** elements than **ex1_2.spc**. This fact is reflected in the following two classes:

```
-----------------------------------------------
Fragmented class of nodes
ex1_1.spc     VDD    connected to        4 C_POS/NEG
                     26 M_PSS_BULK       14 M_PSS_D/S
-----------------------------------------------
-----------------------------------------------
Fragmented class of nodes
ex1_2.spc     4      connected to        4 C_POS/NEG
                     24 M_PSS_BULK       14 M_PSS_D/S
-----------------------------------------------
```

It is reasonable at this point to assume that node **4** and node **VDD** are supposed to be matched.

The key to finding the difference between the two files lies in concentrating on the differences in the connectivity between the nodes in the fragmented classes. Concentrating on two particular classes, one from each file, will quickly lead to the root of the problem:

```
-----------------------------------------------
Fragmented class of nodes
ex1_2.spc     15     connected to        3 M_NSS_D/S
```

```
       1 M_NSS_G     3 M_PSS_D/S          1 M_PSS_G
ex1_2.spc     3      connected to         3 M_NSS_D/S
       1 M_NSS_G     3 M_PSS_D/S          1 M_PSS_G
-----------------------------------------------
```

and

```
-----------------------------------------------
Fragmented class of nodes
ex1_1.spc     N25(XSUBA/X34)       connected to
       3 M_NSS_D/S          1 M_NSS_G            4 M_PSS_D/S
       1 M_PSS_G
ex1_1.spc     N25(XSUBA/X33)       connected to
       3 M_NSS_D/S          1 M_NSS_G            4 M_PSS_D/S
       1 M_PSS_G
-----------------------------------------------
```

Node **N25** in subcircuit *XSUBA* in file **ex1_1.spc** in two cases has one more connection to a drain-source terminal of a *p*-transistor than the potential match in the second file, **ex1_2.spc**. Looking at the nodes in their respective netlist files (skipping all but drain-source connections to *p*-transistors) gives:

```
ex1_1.spc:
M1        N39 N23 N25 VDD PSS L=2 W=5 M=2
M11       2   CLK N25 VDD PSS L=2 W=5 M=2
M4        N39 N25 N23 VDD PSS L=2 W=5 M=1
```

```
ex1_2.spc:
M1        1    2    3    4 PSS L=2 W=5
M3        1    2    3    4 PSS L=2 W=5
M11       5    7    3    4 PSS L=2 W=5
M4        1    3    2    4 PSS L=2 W=5
```

We are looking at only half the cases. From the second file, **ex1_2.spc**, we are only concentrating on node **3**. Similar output is generated for node **15**. Looking back at the fragmented classes, we see that node **7** of **M11**, a potential candidate for an error, has six connections to *p*-transistors. The potential equivalent node in file **ex1_1.spc**, node **1** (node **1** gets mapped into the **CLK** node through the hierarchy), has eight connections to *p*-transistors. Therefore, it is safe to assume that either **M11** of file **ex1_2.spc** needs to be duplicated along with the equivalent transistor for the node **15** case, **M3411**, or that the multiplier **M=2** of **M11** in file **ex1_1.spc** has to be replaced with **M=1**. Doing one or the other and rerunning LVS will get through the iteration process to the point of automorphism. (See Automorph Class Example on page 3-220.)

Often there is more than one problem in a fragmented class. Since fragmentation problems tend to hide each other, the fastest approach is to fix the identified problem and then rerun LVS. In the previous example, only one problem caused the fragmentation.

Element Description File Example

This example illustrates the output generated when an element description file is used during the run. An *element description file* is a text file that describes specialized elements used in the design not identified by LVS. The element description file in this example describes the specialized elements (combs, springs, and plates) used in a resonator for a Micro Electro Mechanical System (MEMS) design. LVS compares the elements down to the level of information provided in the element description file. (For information on the format of this file, see Element Description File Format on page 3-261.)

Following are the contents of the element description file **resonator.elm**.

```
|| d comb 1 2 3 4;
|| d plate4 1 2 3 4 5 6 7 8;
|| d fspring 1 2 3 4;
```

Following is an example of the output file **resonator.out**, generated by LVS for **resonator.vdb**. Notice that LVS reports many of these elements as connected to only one pin. Catching nodes with only one connection (floating pins) is a very important check to perform, because not connecting a pin (for example, a bulk node to the substrate) can be fatal in a design. LVS identifies such problems before they become costly.

```
....
Reading element definitions from file reson.elm
```

```
Parsing file lreson.spc...
Flattening network...
Not resolving subckt fspring
Not resolving subckt comb
Not resolving subckt fspring
Not resolving subckt comb
Not resolving subckt plate4
Parsing file sreson.sp...
Flattening network...
....

Device                                                    lreson.spc
sreson.sp
--------------                                            ------------
------------

fspring                                                   2
2
plate4                                                    1
1
comb                                                      2
2

Total elements                                            5
5
Total nodes                                               16
16
Single-pin nodes                                          8
8

Nodes in file lreson.spc connected to only one pin
**********************
```

```
4_e
....
20_m
*********************

Nodes in file sreson.sp connected to only one pin
*********************
N1
....
N5
*********************

********************************** ITERATING **********************************
Iterating...
5% done
10% done
....
....
100% done
************************ REPORTING AUTOMORPHISM************************

Report of elements:
*******************
Automorph class of elements
lreson.spc XPlateInst_plate4 fanout: 1 = 1   2 = 1   3 = 1   4 = 1   5 = 1   6 = 1   7 = 1   8 = 1
sreson.sp XMass5_plate4      fanout: 1 = 1   2 =1   3 = 1   4 = 1   5 = 1   6 = 1   7 = 1   8 = 1
---------------------------------------------------

*************************** ITERATION SUMMARY ***************************
4 perfectly matched element class(es)
1 automorphed element class(es)
16 perfectly matched node class(es)
```

```
************************ POST ITERATION MATCHING ************************
Doing detailed trial matching...
Matched Elements XPlateInst_plate4 and XMass5_plate4

****************************** FINAL RESULT ******************************
Circuits are equal.
Note: Networks only compared as far down as specified in the special element file.
```