

**Department of Electrical and Computer Engineering
University of Wisconsin–Madison**

**ECE 553: Testing and Testable Design of Digital Systems
Fall 2011**

TESTCAD Tool Set

1 Overview

This document contains the manuals for ECE 553 test generation and fault simulation tool set.

Most of the software tools are for combination circuits, and they are compatible at the file level. The software tools for sequential circuits may not have consistent output file formats, and hence, are not compatible. The files through which the tools communicate are represented in this document by generic file names: **netlist**, **faultlist**, and **vectorset**. Figure 1 shows the important components in the tool set and how they interact. A brief summary of each component in the tool set is given next with detailed description given on subsequent manual pages.

This tool set is accessible on HP-RISC workstations at CAE. These workstations can be accessed directly or by remote login or telnet from other workstations. You may either use softlink (i.e, UNIX **ln** command) or set your UNIX environment **PATH** accordingly to use them.

Location of the tool set files and executables:

- Documents: `~ece553/TESTCAD/doc`
- Executables: `~ece553/TESTCAD/bin`
- Netlists: `~ece553/TESTCAD/nets`

ATPG Tools:

Fault List Generator:	listfaults, faultgen
Random Vector Generator:	randvec
Combinational Test Pattern Generators:	podem, catpg
Sequential Test Pattern Generators:	fastest, satpg
Logic Simulator:	lsim
Fulat Simulators:	sfsp, ppcpt, ppsfp
Testability Evaluator:	scoap
String Converter:	symuw2uw
Vector Manipulating Utilities:	reverse, vcompact, accumulate, sortcover

If you have any problems, questions or suggestions, please E-mail the current TA: smillican@wisc.edu

The ATPG software tools block diagram is shown in Figure 1. Each rectangle represents a collection of software tools for a specific functionality. The line interconnections represent how the tools interact with each other using **netlist**, **faultlist**, and **vectorset** files. The directions of the arrows indicate the inputs and outputs of the tools. A brief summary of each tool is given on the next page. The detailed descriptions of the tools are given on the subsequent manual pages.

netlist Circuit description file - This file contains a net list representation of the circuit in `net_list_file` format. All tools described here that process circuit use this format.

faultlist Fault list file - This file contains a list of faults in `fault_list_file` format. The default fault file is **netlist.f**. For example, if the **netlist** file is **n432** then the default fault file is **n432.f**. A fault file is required for fault simulation, but faults could also be supplied interactively for test pattern generation.

vectorset Test vector file - This file is composed of a list or sequence of test vectors in vector set file format. It can be generated manually, by the random test pattern generator or by the test pattern generator

and used by both fault simulators or the logic simulator. The default vector file is **netlist.v**. Thus if the **netlist** file is **n432** then the default vector file is **n432.v**.

listfaults Fault list generator for combinational circuits – Can be used for generating all possible stuck-at faults for a circuit. It has the options to list the faults remaining after reduction by equivalence, by dominance, by use of checkpoints, and meaningful combinations thereof.

faultgen Fault list generator for both combinational and sequential circuits - Can be used for generating all possible stuck-at faults for a circuit. It has the options to list the faults remaining after reduction by equivalence or by dominance. In addition, a **-k** option can be used to disallow gates without fanout.

randvec Random vector generator - Has two purposes. First, it generates random vectors for a given circuit. Second, it fills unspecified (X) entries with 0s and 1s for a given list of test vectors. The proportion of 1s and 0s produced in the bit positions can be specified.

podem Test pattern generator - Generates test vectors for a given circuit and a given fault or fault list. This test pattern generator can be run in batch mode as well as in interactive mode.

lsim Logic Simulator - Produces a set of output vectors (circuit outputs) for a given set of test vectors (circuit inputs) and a given circuit. It can also be used as an event-driven simulator.

sfsfp Single fault, single pattern propagation fault simulator - Simulates a given circuit for a given list of faults and list of test vectors. As a result of this process, it reports undetected faults and fault coverage statistics. The undetected fault outputs are in the fault list file format in order to be used in subsequent test generation or fault simulation. Since it employs a single fault, single pattern propagation method, it is slower than **ppcpt**, but it is useful for simulating partially-specified (three-valued) test vectors. To avoid excessive simulation time, it should not be used to simulate large number of vectors or large number of faults.

ppcpt Parallel pattern critical path tracing fault simulator - Simulates a given circuit for a given list of faults and a given list of test vectors or for internally-generated random vectors. As a result of the simulation, it reports undetected faults and fault coverage statistics. Since it is a parallel pattern simulator, it is best suited for 32 or more test vectors.

ppsfsp Parallel pattern single fault propagation fault simulator - Simulates a given circuit for a given list of faults and a given list of test vectors or for internally-generated random vectors. As a result of the simulation, it reports undetected faults and fault coverage statistics. Since it is a parallel pattern simulator, it is best suited for vector set file containing 32 or more test vectors.

scoap Testability evaluator - Can be used for computing a number of different testability values for lines within a circuit and compute statistics for the entire circuit. The values computed are controllability, observability and testability.

symuw2uw Symbolic UW netlist to UW netlist converter - The formats being used across the tool set only accept integers as names for nets. **symuw2uw** allows the user to write net lists using a meaningful name (string) for each net by mapping the string into a corresponding integer. As a result, a symbol file which contains the mapping will be generated. Given the symbol file and any of the following files: netlist, vector or fault, **symuw2uw** can reconstruct an output file replacing the integers with the original mapped name (string).

reverse,vcompact, accumulate, sortcover Tools for manipulating test vectors - These tools can be used for reducing the test vectors after their generation, to detect redundancy among the test vectors and to allow reduction of the number of test vectors in the final test set.

fastest A sequential test pattern generator - Generates sequence of test vectors for a given sequential circuit and a given fault list. The format used in the output test vectors is not compatible with the vector set file format mentioned above.

catpg A combinational test pattern generator - Generates test vectors for a given combinational circuit and a given fault list. The format used in the output test vectors is not compatible with the vector set file format mentioned above.

satpg A sequential test pattern generator - Generates sequence of test vectors for a given sequential circuit and a given fault list. The format used in the output test vectors is not compatible with the vector set file format mentioned above.

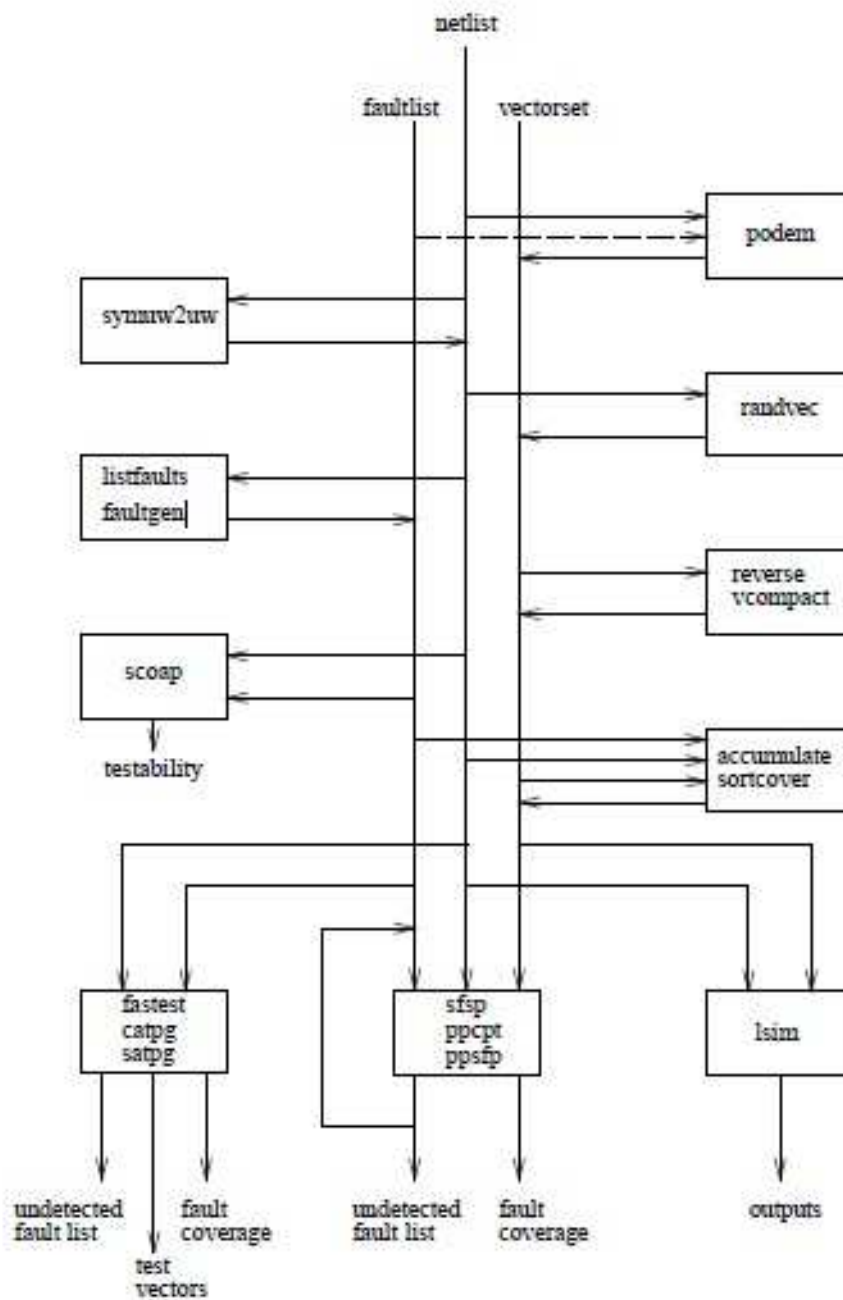


Figure 1: ATPG software tools overview.

2 hints

Hints – several hints for use of this tool set

2.1 DESCRIPTION

Since the tool set is located in `testcad/public/bin`, users may set up their UNIX PATH as follows:

```
% cd ~  
% /fileSpace/people/e/ece553/TESTCAD/testcad.setup  
% source .cshrc
```

If you run out of disk space allocation, use **compress** or **gzip** to compress your files. Whenever you need the original files, use **uncompress** or **gunzip** to uncompress them. Another alternative is to use a floppy disk to do backup as instructed by CAE. For those tools which provide output to standard output, you may use output redirection to store the output into a file as follows:

```
% randvec -n n931 -i 40 > V1  
// generate 40 random vectors and store the results in V1.
```

A useful UNIX command is **diff**. This command can be used to detect any differences between two files. For example, **diff** can be used to compare two output vector files from `lsim` when you wish to compare the response of a given circuit to that of a known good circuit.

3 netlist

netlist - generic circuit description file using the net list file format

3.1 DESCRIPTION

The circuits described in **netlist** can be either combinational or sequential circuits. Each **net** in the net list has a **positive** integer net number (name), a gate type, and a list of fanout nets. The `net_list_file` format used in **netlist** to describe a circuit is defined as follows:

```
circuit      :: { net }
net         :: net_number gate_type [ { fanout } ] ;
net_number  :: integer
gate_type   :: AND | OR | NAND | NOR | NOT | BUF | PI |
             PO | LATCH | SCANL
fanout      :: net_number [ { net_number } ] ... [ { net_number } ]
```

A net with PI as its gate type is a primary input. There are two definitions for a primary output: 1) a net without fanout or 2) a net with PO as its gate type. All tools which require a circuit will accept both definitions for defining primary outputs. However, an option **-k** can be used for blocking the first definition so that only nets with PO as the gate type will be treated as primary outputs; this permits portions of the circuit that reach only primary outputs not labeled as PO to be ignored. An example circuit and its representation in **netlist** are given below to demonstrate the net list file format and its use.

Note that **XOR** (**exclusive_OR**) is not supported in UW format. If there are **XOR** gates, they must be transformed into the equivalent logic representations. In addition, on some tools, the number of fanin to a gate is limited to a maximum of 6. The number of fanout is not limited, however.

In sequential circuit specification, **LATCH** is used to represent D-type memory elements such as D-type flip-flop. **SCANL** is the D-type scannable latch used in scan design. Note however that most of the tools in ATPG tool set can only process combinational elements, i.e., no **LATCH** or **SCANL**. The clock connections for sequential circuits are explicitly declared as follows:

```
CLK PI <LATCH and SCANL net number list>;
```

This line must be the first non-comment line of a sequential circuit netlist description.

3.2 EXAMPLES

Netlist in UW Format

```
1  PI      10  ;
2  PI      16  ;
3  PI      10 11 ;
6  PI      11  ;
7  PI      19  ;
10 NAND    22  ;
11 NAND    16 19 ;
16 NAND    22 23 ;
19 NAND    23  ;
22 NAND    24  ; # May also use default PO as : 22 NAND ;
23 NAND    25  ; # May also use default PO as : 23 NAND ;
24 PO      ;
25 PO      ;
```

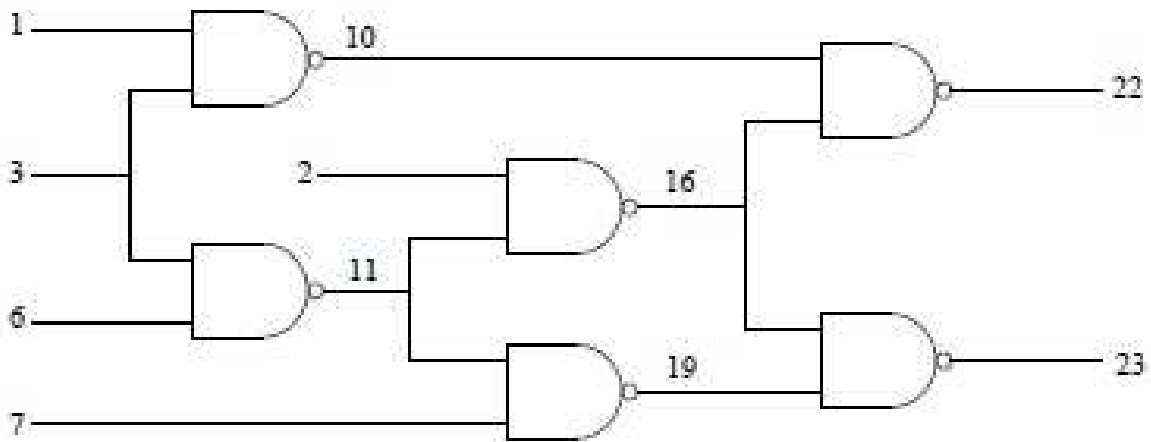


Figure 2: Example circuit diagram.

4 faultlist

faultlist - generic fault list file using the fault list file format

4.1 DESCRIPTION

In a **faultlist** file, each line represents one fault in the circuit described in a **netlist** file. A fault is represented by three fields, namely, net number of the faulty gate output, input net number of the faulty gate, and fault type. Input net number indicates which input line of the gate is faulty. A **0** on input net number field indicates that the gate output line is faulty. Note that this implies that **0** is not a valid net number. Fault type is either **0** for stuck-at-0, or **1** for stuck-at-1. The following examples illustrate the fault list file format for three faults.

4.2 EXAMPLES

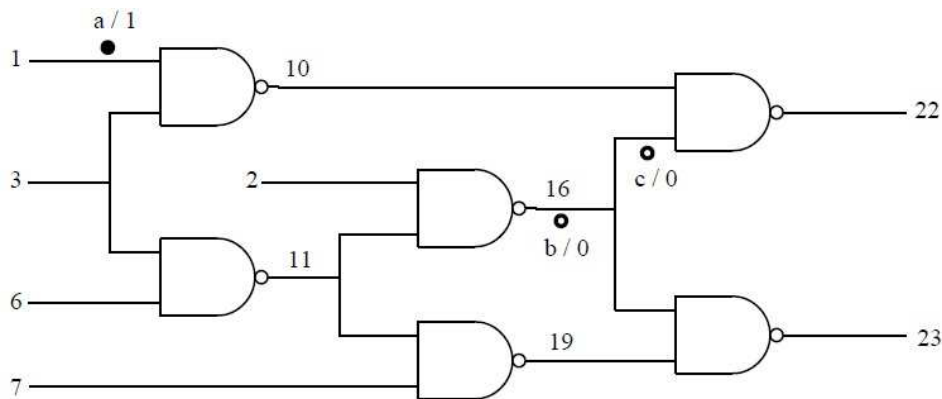


Figure 3: Example circuit diagram.

Fault List Format

```

1  0  1  # Fault a/1: output of gate 1 (PI) stuck at 1
16 0  0  # Fault b/0: output of gate 16 stuck at 0
22 16 0  # Fault c/0: input line 16 of gate 22 stuck at 0

```

5 vectorset

vectorset - generic test vector set file using the vector set file format

5.1 DESCRIPTION

A **vectorset** file may contain one or more test vectors. The vector set file format is defined as follows. Each test vector starts with the key word **Test:** followed by a list of objects each of which corresponds to a primary input and its value. The primary inputs which do not appear in the list are assumed to have unspecified (**X**) values. If the object list is too long and has to use more than one line, another key word **Cont:** is used at the beginning of the following line to represent continuation of the vector. Note that no blank line may appear between lines of a given test vector.

A vector file may also contain other key words. These words are used or generated by some programs. These words are **Fault:**, **NoTest:**, **PO**, and **Weight0:**. The word **Weight0:** is treated as a command by randvec, all other key words are ignored while reading a vector file.

5.2 EXAMPLES

An example **vector** file for the circuit given for the definition of **fault** is as follows:

```

Test:    1/1   2/1   3/1   6/1 7/0
Test:    1/1   3/1   6/1   7/1      // input 2 which does not appear
                                           // in the list is assumed to be X
Test:    1/1   2/0   6/1   7/1      // the same here for input 3

Fault:   19/1                                     // generate by podem
PO:      23/D                                     // output where fault detected
Test:    1/1   2/0   3/1                         // continue to next line
Cont:    6/0   7/1

Weight0: 20                                       // used by randvec, changes the
Test:    1/0   3/0                                 // weight to 20% 0's, 80% 1's
                                           // see randvec for details

```

6 listfaults

listfaults - a fault list generator

6.1 SYNOPSIS

```
listfaults [-n netlist] [-af] [-ae] [-ad] [-cf] [-ce]
```

6.2 DESCRIPTION

listfaults¹ is a fault list generator capable of listing all possible single stuck-at faults for a circuit. **listfaults** expects a **netlist** file in net list file format as input. The default net list file is **CIRCUIT**. The generated fault list will be printed on the standard output in the fault list file format. The output faults can be various reduced fault lists depending upon the option specified. All fault reduction is based on structural analysis. Note that **listfaults** only accepts netlists for combinational circuits.

-n netlist

This option specifies the circuit for which listfaults generates a fault list. The default netlist is CIRCUIT.

-af

List all faults. -af is the default option.

-ae

List all faults after reduction by structural equivalence.

-ad

List all faults after reduction by equivalence, and then by dominance.

-cf

List all checkpoint faults.

-ce

¹listfaults was written by Dominic Go and updated by Byung So of the ECE Department at the University of Wisconsin - Madison.

List all checkpoint faults after reduction by equivalence.

6.3 EXAMPLES

```
listfaults -n n17 -af  
// generates all possible faults for n17
```

```
listfaults -cp  
// lists the checkpoint faults of CIRCUIT
```

7 faultgen

faultgen - a fault list generator

7.1 SYNOPSIS

```
faultgen netlist faultlist [-e] [-a] [-k]
```

7.2 DESCRIPTION

faultgen² is a fault generator capable of listing all possible single stuck-at faults for a circuit. **faultgen** expects a netlist file in net list file format as input. The generated fault list will be printed in **faultlist** file in the fault_list_file format. Both **netlist** and **faultlist** files must be specified. No default files are assumed. **faultgen** accepts either combinational or sequential circuit netlists. The following options are available in **faultgen**. If no option is specified, **faultgen** will list all faults after reduction by structural equivalence, and then by dominance.

-e

List all faults after reduction by structural equivalence.

-a

List all faults. No reduction is performed.

-k

Disallow the gates with no fanout to be treated as PO. With -k option, faultgen will report error and exit if such gates are found in the netlist description.

7.3 EXAMPLES

```
faultgen n17 n17.f
// generates all faults after reduction by structural
// equivalence, and then by dominance for n17,
// store the reduced fault list in n17.f
```

```
faultgen n17 n17.f -e
// generates all faults after reduction by structural
// equivalence only. no dominance reduction is performed.
// store the reduced fault list in n17.f
```

²faultgen was written by Richard Chou of the ECE Department at the University of Wisconsin - Madison.

8 podem

podem - an interactive and batch mode test pattern generator

8.1 SYNOPSIS

```
podem [-n netlist] [-o [vectorset]] [-f faultlist] [-i] [-h] [-k]
```

8.2 DESCRIPTION

podem³ is an automatic/interactive test pattern generator. **podem** generates a test vector for every fault in **faultlist**. The default backtrack limit is 10,000. The options for running **podem** are as follows.

-n netlist

This option tells **podem** to use **netlist** as the input file containing the circuit description. If this option is not specified, the default network is assumed to be **CIRCUIT**.

-o [vectorset]

This option specifies the file where the generated test vectors are to be stored. If not specified, test vectors are output to the standard output. The default vectorset file is **netlist.p**. The vectorset file uses the vector set file format. The logic values D and U are used to represent D and D, respectively, on the PO lines.

-f faultlist

This option tells **podem** to find the fault list in **faultlist** file. The default fault list file is **netlist.f**. If not specified, user must provide a fault list interactively using the **-i** option.

-i

With the **-i** option, users can run **podem** interactively. If **-i** option is given, **podem** reads faults from the terminal and ignores **-f** option.

-h

This option will display a help message to provide information about running **podem**.

-k

³**podem** was implemented by Kyuchull Kim of the ECE Department at the University of Wisconsin - Madison. The algorithm is based on "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," by P. Goel in IEEE Trans. on Computers, vol. C-30, Mar. 1981, pp. 215-222. The backtrack mechanism was modified to improve the performance of the test generator.

This option will prevent nets with no fanout being treated as primary outputs.

8.3 EXAMPLES

```
podem -n n432 -f F
// generates test vectors for n432 for all faults in fault file F.
// The output appear on the standard output(screen).

podem -i -o G
// generates test vectors for CIRCUIT interactively with
// output stored in file G

podem
// generates test vectors for CIRCUIT for all faults in the
// fault list file CIRCUIT.f and puts the results in CIRCUIT.p
```

9 lsim

lsim - a logic simulator

9.1 SYNOPSIS

```
lsim [-n netlist] [-v vectorset] [-a] [-k]
```

9.2 DESCRIPTION

lsim⁴ is logic simulator which simulates the behavior of the circuit in netlist and prints out the output for each input vector to the standard output. **lsim** is a three-valued logic simulator. The three values are **0**, **1**, and **X**. The input vectors are specified in **vectorset**. To execute **lsim**, a user may specify both the netlist and the vectorset files through **-n** and **-v** options, respectively. The default netlist file is **CIRCUIT** and the default vector file is **netlist.v**.

lsim can also be used as an event-driven logic simulator by specifying the option **-a** in the command line. If this option is chosen the simulator will accumulate the values in the previous vectors in the vectorset file when doing the simulations. This means that any input for which a 0 or 1 value is specified in a vector retains that value until it is explicitly changed. Note also that an input which is specified as a 0 or 1 at some stage of the simulation must never be changed to an unspecified (X) value for the remainder of the simulation. The simulator will list each input test vector and its corresponding output to the standard output.

Option **-k** can be used to prevent nets with no fanout being treated as primary outputs.

9.3 EXAMPLES

```
lsim -n n432
    // simulate n432 with n432.v as vector input

lsim -n n432 -v V1 ? V2
    // simulate n432 with V1 as vector input and
    // V2 as the simulator output

lsim
    // simulate CIRCUIT with CIRCUIT.v as vector input
```

⁴**lsim** was implemented by Kyuchull Kim of the ECE Department at the University of Wisconsin - Madison. **lsim** is not related to any commercial product having the same name.

10 sfsp

sfsp - a single fault single pattern, propagation fault simulator

10.1 SYNOPSIS

```
sfsp [-n netlist] [-f faultlist] [-v vectorset] [-u [udf file]]
```

10.2 DESCRIPTION

sfsp⁵ is a single fault, single pattern propagation fault simulator. In running **sfsp**, one may specify the following three input files: a netlist file, a faultlist file, and a vectorset file. Default **sfsp** results consist of fault coverage statistics. In addition, undetected faults can be listed on the standard output by the **-u** option. Also, by specifying a file udf file after **-u**, the undetected faults can be listed in a file rather than in standard output. This file uses the fault list file format so that it can be used as a faultlist file for further processing.

The default netlist file is **CIRCUIT**. The default fault_list file is **netlist.f** and the default vector_set is **netlist.v**. If none of the files are specified, the default files are **CIRCUIT**, **CIRCUIT.f** and **CIRCUIT.v**, respectively.

10.3 EXAMPLES

```
sfsp
// use the default netlist CIRCUIT and find the fault coverage
// based on faults in CIRCUIT.f and the vectors in CIRCUIT.v
```

```
sfsp -n n432
// determine fault coverage for netlist n432 based on the
// faults in file n432.f and the vectors in file n432.v
```

```
sfsp -n n432 -f F1 -u UF1
// determine fault coverage for netlist n432 based on the
// faults in file F1 and the vectors in the file n432.v and
// list the undetected faults in the file UF1
```

⁵**sfsp** was implemented by Byung So of the ECE Department at the University of Wisconsin - Madison.

11 ppcpt

ppcpt - fault simulator (parallel pattern critical path tracing)

11.1 SYNOPSIS

```
ppcpt [-n netlist] [-f faultlist] [-v vectorset] [-r] [-u [udf file]] [-N #] [-C #] [-S]
```

11.2 DESCRIPTION

ppcpt⁶ is a parallel pattern fault simulator using critical path tracing. It is based on two-valued logic simulation. Therefore, all primary inputs must be specified as 0 or 1. If the user does not specify primary input values, ppcpt will fill those values with randomly generated 0's or 1's.

This fault simulator requires three input files: **netlist**, **faultlist**, and **vectorset** files. The default net list file is **CIRCUIT**. The default fault list file is **netlist.f**, and the default vector set file is **netlist.v**. If none of the **net_list**, **fault_list**, and **vector_set** files are specified, the default files are **CIRCUIT**, **CIRCUIT.f** and **CIRCUIT.v**, respectively. Default **ppcpt** results consist of fault coverage statistics. Undetected faults can be listed on the standard output by the -u option. Also, by specifying a file name after -u, the undetected faults can be listed in the given file rather than on the standard output.

-n netlist

Use the circuit description file **netlist** for simulation. The circuit description must be in the net list file format. A line starting with '#' will be viewed as a comment line and will be ignored.

-f faultlist

Use the fault list file faultlist. The format must be the **fault_list** file format. The comment line is the same as for **netlist**.

-v vectorset

Use the test vector file vectorset having the vector set file format. If a vector does not specify all primary input values, **ppcpt** will generate random values for those primary inputs.

-r

Use internally-generated random numbers as test vectors. The total number of vectors must be a multiple of 32. Use -N max passes to specify the multiple of 32 to be used. The default is off.

-u [udf file]

⁶**ppcpt** was written by Byung So of the ECE Department at the University of Wisconsin - Madison. A paper on this simulator appears in JETTA - Journal of Electronic Test: Theory and Applications, August 1993.

Write the undetected fault list in file udf file. If no file is specified, the result will be printed on the standard output.

-N max passes

Use max passes number of passes for fault simulation. Each pass means 32 test vectors. The default is 100 passes (3200 vectors). This option is applicable only with the -r option.

-C max fault coverage

This option is applicable only with the -r option. Confine the fault coverage to max fault coverage in percent without the % sign. The default is 100.

-S

Suppress the default message. The default is off. The default is 100 passes (3200 vectors).

-k

Prevents a net with no fanout from being treated as a primary output.

11.3 EXAMPLES

```
ppcpt -k
// simulates the netlist file CIRCUIT with fault list file
// CIRCUIT.f and vector file CIRCUIT.v and suppresses the
// nets with no fanout as primary outputs. The results are
// displayed on the screen.

ppcpt -n net
// simulates the netlist file 'net' with fault list file
// 'net.f' and vector file 'net.v' with the results displayed
// on the screen.

ppcpt -n net -f my.fault -r -u my.fault.udf -N 10
// netlist file 'net' is simulated with fault list file
// 'my.fault' and 10 passes (320 random patterns) and produce
// the undetected fault list file 'my.fault.udf'.
```

12 ppsfp

ppsfp - fault simulator (parallel pattern singal fault propagation)

12.1 SYNOPSIS

```
ppsfp [-n netlist] [-f faultlist] [-v vectorset] [-r] [-u [udf file]] [-N #] [-C #] [-S]
```

12.2 DESCRIPTION

ppsfp⁷ is a parallel pattern fault simulator which considers one fault at a time. It is based on two-valued logic simulation. Therefore, all primary inputs must be specified as 0 or 1. If the user does not specify primary input values, **ppsfp** will fill those values with randomly generated 0's or 1's.

This fault simulator requires three input files: **netlist**, **faultlist**, and **vectorset** files. The default net list file is **CIRCUIT**. The default fault list file is **netlist.f**, and the default vector set file is **netlist.v**. If none of the netlist, fault list, and vector set files are specified, the default files are **CIRCUIT**, **CIRCUIT.f** and **CIRCUIT.v**, respectively. Default **ppsfp** results consist of fault coverage statistics. Undetected faults can be listed on the standard output by the -u option. Also, by specifying a file name after -u, the undetected faults can be listed in the given file rather than on the standard output.

-n netlist

Use the circuit description file **netlist** for simulation. The circuit description must be in the **net_list** file format. A line starting with '#' will be viewed as a comment line and will be ignored.

-f faultlist

Use the fault list file **faultlist**. The format must be the fault list file format. The comment line is the same as for **netlist**.

-v vectorset

Use the test vector file **vectorset** having the **vector_set** file format. If a vector does not specify all primary input values, ppsfp will generate random values for those primary inputs.

-r

Use internally-generated random numbers as test vectors. The total number of vectors must be a multiple of 32. Use -N max passes to specify the multiple of 32 to be used. The default is off.

-u [udf_file]

⁷**ppsfp** was implemented by Byung So of the ECE Department at the University of Wisconsin - Madison. It is based on the following paper: "Fault Simulation for Structured VLSI" by J. A. Waicukauski, E. B. Eichelberger, D. O. Forlenza, E. Lindbloom, and T. McCarthy. VLSI Systems Design, Vol. 6, No. 12, December, 1985, pp. 20-32.

Write the undetected fault list in file **udf_file**. If no file is specified, the result will be printed on the standard output.

-N max_passes

Use max_passes number of passes for fault simulation. Each pass means 32 test vectors. The default is 100 passes (3200 vectors). This option is applicable only with the -r option.

-C max fault coverage

This option is applicable only with the -r option. Confine the fault coverage to max fault coverage in percent without the % sign. The default is 100.

-S

Suppress the default message. The default is off.

-k

Prevents a net with no fanout from being treated as a primary output.

12.3 EXAMPLES

```
ppsfp -k
// simulates the netlist file CIRCUIT with fault list file
// CIRCUIT.f and vector file CIRCUIT.v and suppresses the
// nets with no fanout as primary outputs. The results are
// displayed on the screen.
```

```
ppsfp -n net
// simulates the netlist file 'net' with fault list file
// 'net.f' and vector file 'net.v' with the results displayed
// on the screen.
```

```
ppsfp -n net -f my.fault -r -u my.fault.udf -N 10
// netlist file 'net' is simulated with fault list file
// 'my.fault' and 10 passes (320 random patterns) and produce
// the undetected fault list file 'my.fault.udf'.
```

13 scoap

scoap - a testability evaluator

13.1 SYNOPSIS

```
scoap [-n netlist] [-o outfile] [-C0C1T0T1O #] [-f faultlist [-F #]] [-h]
```

13.2 DESCRIPTION

scoap⁸ is based on two-valued logic. To run **scoap**, you must provide a **netlist** file. The default netlist file is **CIRCUIT**. **scoap** is able to generate any of the following testability measures for the lines in netlist: controllability, observability and testability. The testability is defined as a sum of the controllability and the observability.

-n netlist

This option specifies the circuit for which **scoap** generates testability measures. The default netlist file is **CIRCUIT**.

-o outfile

This specifies the output file name. The default is **netlist.s**.

-f faultlist

This tells **scoap** to read the faults from the faultlist file and generate testability measures for the faults to the output file.

-F #

If this option is specified after a **fault_list** file name, the number of faults specified by **#** having the highest values for testability will be displayed in descending order.

-h

This option displays a help message to provide information about running **scoap**.

-T0 #

⁸**scoap** was implemented by Dominic Go and updated by Byung So of the ECE Department at the University of Wisconsin - Madison. SCOAP is based on : Goldstein, "Controllability/Observability Analysis of Digital Circuits," IEEE Trans. Circuits and Systems, vol. CAS-26, pp. 685-693, 1979.

This option displays in descending order the top # lines having the highest values for 0-testability. (0-testability of a line is defined as the sum of the 1-controllability and the observability of that line.)

-T1 #

This option displays in descending order the top # lines having the highest value for 1-testability. (1-testability of a line is defined as the sum of the 0-controllability and the observability of that line.)

-C0 #

This option displays in descending order the top # lines having the highest value for 0-controllability.

-C1 #

This option displays in descending order the top # lines having the highest value for 1-controllability.

-O # hfill

This option displays in descending order the top # lines having the highest value for observability.

13.3 EXAMPLES

```
scoap -n n17 -o my.s
// generate the measures for every lines in the circuit n17
// in output file my.s

scoap -n n17 -f n17.f -o my.s
// generate the measures for every faults in the
// fault file n17.f in output file my.s

scoap -n n17 -f n17.f -F 10
// find 10 faults which have the highest testability measures
// among the faults in the fault file n17.f, and print out in
// defaults output file n17.s

scoap -n n17 -o my.s -C1 10
// generate 10 lines which have the highest 1-controllability
// measures in output file my.s
```

14 symuw2uw

symuw2uw - a string converter

14.1 SYNOPSIS

```
symuw2uw -c -t sym netlist [-n netlist] -s symbol tab symuw2uw -u [-n netlist] [-v vectorset] [-f faultlist] -s symbol tab -o outfile
```

14.2 DESCRIPTION

The UW format used across the tool set accept only integers as names for nets. **symuw2uw**⁹ allows the user to write net lists with meaningful names for the nets by mapping the string name used into a corresponding integer in the files to be used by the tools. Also, a symbol tab file which contains the mapping will be generated. Given the symbol tab file and any of the following files: **netlist**, **vectorset** or **faultlist** in their respective formats, **symuw2uw** can construct a corresponding output file in which integers are replaced with the original mapped string name. Thus, it can be used to convert the cryptic files used by the tools into files which are more readable by the user.

-t sym netlist

This option specifies the UW symbolic netlist file to be converted into the UW net list file format.

-n netlist

This specifies the netlist file in UW net list file format; the default is **CIRCUIT**.

-v vectorset

This specifies the vector file in UW vector set file format.

-f faultlist

This specifies the fault list file in UW fault list file format.

-s symbol tab

This specifies the file which stores the mapping (symbol table) of the name strings to integers.

-c

⁹**symuw2uw** was originally called `convert`. Due to the conflict with UNIX utility program `convert`, it was renamed `symuw2uw`. This program was implemented by Dominic Go of the ECE Department at the University of Wisconsin - Madison.

This specifies use of the sym netlist file as input and generates the symbol tab file and the corresponding netlist file.

-u

This specifies use of the symbol tab file and any of the following files in (UW) tool format: **netfile**, **vectorset**, or **faultlist** as input and generation of the outfile file as specified by the -o option with the integers replaced by the mapped string.

-o outfile

This specifies the output file for the option -u.

14.3 EXAMPLES

```
symuw2uw -n n17 -s symbol -c -t original
// use the circuit original which has the net names as strings
// and generate a network in UW tool format n17 and store the
// mapping between the strings and the integers in the file
// symbol
```

```
symuw2uw -v n17.v -u -s symbol -o newn17.v
// use the symbol file symbol and replace the net integer in
// vector file n17.v with the mapped string in the output file
// newn17.v
```

15 reverse

reverse - reverse the order of the test vectors in the vector file

15.1 SYNOPSIS

reverse [-v vectorset]

15.2 DESCRIPTION

reverse¹⁰ reverses the order of the test vectors in the **vectorset** file.

-v vectorset

This option specifies the vector set file; the default is **CIRCUIT.v**.

15.3 EXAMPLES

¹⁰**reverse** was implemented by Dominic Go of the ECE Department at the University of Wisconsin - Madison.

16 vcompact

vcompact - compact a set of test vectors into a smaller set

16.1 SYNOPSIS

```
vcompact [-v vectorset]
```

16.2 DESCRIPTION

vcompact¹¹ compacts the test vectors in vectorset such that if the test set contains vectors (1 X X), (1 1 0), (1 1 X), only the second one will be displayed on standard output. Due to the computational complexity, the current implementation does not guarantee the optimal compaction on the output.

Compaction is needed for generating a reduced set of test vectors. In the example shown above, any faults detected by (1 1 X) and (1 X X) will also be detected by vector (1 1 0). As a result, the former two vectors are not included in the reduced test set after the compaction.

-v vectorset

This option specifies the vector set file; the default is **CIRCUIT.v**.

16.3 EXAMPLES

```
vcompact  
// does the compaction process for the vectors in CIRCUIT.v and  
// output the final set of vectors on the standard output
```

¹¹**vcompact** was originally called **compact**. Due to the conflict with UNIX utility program **compact**, it was renamed **vcompact**. This program was implemented by Dominic Go of the ECE Department at the University of Wisconsin - Madison.

17 accumulate

accumulate - a vector accumulator

17.1 SYNOPSIS

```
accumulate [-n netlist] [-v vectorset] [-f faultlist] [-k] [-o outfile]
```

17.2 DESCRIPTION

accumulate¹² will accept **netlist**, **faultlist** and **vectorset** files as input. It will pick out each vector successively from **vectorset** and simulate it with **sfsp**. If the current vector detects an additional fault from **faultlist**, then it is stored in the output file **outfile** together with previous such vectors; otherwise, it will be discarded.

-n netlist

This option specifies the circuit; the default is **CIRCUIT**.

-v vectorset

This specifies the file containing the initial vectors; the default is **netlist.v**.

-f faultlist

This specifies the fault list file. These faults are used for determining the fault coverage for each vector; the default is **netlist.f**.

-k

This suppresses the use of gates with no fanout as primary outputs. Only those specified as POs will be used.

-o outfile

This specifies the file which stores the final set of vectors after the processing; the default is **netlist.a**.

17.3 EXAMPLES

```
accumulate -n n17
// Do the accumulation process for the vector in n17.v
```

¹²**accumulate** was written by Dominic Go and updated by Byung So of the ECE Department at the University of Wisconsin - Madison.

// and output the final set of vector in n17.a.

18 sortcover

sortcover - sort vectors in term of their fault coverage

18.1 SYNOPSIS

```
sortcover [-n netlist] [-v vectorset] [-f faultlist] [-k] [-o outfile]
```

18.2 DESCRIPTION

sortcover¹³ simulates the circuit in netlist file and the fault list in faultlist file with the test vectors in vectorset file using sfsp. It then sorts the vectors in vectorset file in term of their fault coverage percentages for the faults in faultlist and lists them in descending order.

-n netlist

This option specifies the circuit; the default is **CIRCUIT**.

-v vectorset

This specifies the file containing the initial vectors; the default is **netlist.v**.

-f faultlist

This specifies the fault list file. These faults are used for determining the fault coverage for each vector; the default is netlist.f.

-k

This suppresses the use of gates with no fanout as primary outputs. Only those specified as POs will be used.

-o outfile

This specifies the file which stores the final set of vectors after the processing; the default is netlist.a.

18.3 EXAMPLES

```
sortcover
// sort the vector in CIRCUIT.v in term of their fault coverage
// and output the resultant ordered vectors in file CIRCUIT.a.
```

¹³**sortcover** was written by Dominic Go and updated by Byung So of the ECE Department at the University of Wisconsin - Madison.

19 fastest

fastest - gate level automatic test pattern generator for synchronous sequential circuit

19.1 SYNOPSIS

```
fastest [-p] -c netlist -f faultlist [-o outfile] [-v testvecs] [-u udf file] [-s init state] [-h] [-r]
```

19.2 DESCRIPTION

fastest¹⁴ is a gate level automatic test pattern generator. **fastest** can generate test for a synchronous sequential circuit as well as combinational circuit. However, since **fastest** has been designed only for synchronous sequential circuit, combinational circuit test generation with **fastest** is not economical in terms of test generation time.

Note that **fastest** command arguments except **argv[0]** are not position sensitive; each option can appear in any order. The usage of **fastest** can be displayed by calling **fastest** without any argument.

-p

The **-p** option enables all **fastest** activity on the output, which can be either stdout or a file defined by **-o** option. If **-p** is omitted (disabled), the output by **fastest** contains total statistics only, no activity.

-c netlist

Specifies the circuit file in UW net list file format to be used by **fastest**. If there is an asynchronous signal path in the input circuit, **fastest** will quit with error messages.

-f faultlist

Specifies the **fault_list** for the input circuit file, in UW fault list file format. If there are duplicated faults, it will send WARNINGS and count as a single fault presented. However, if there is an illegal (wrong) fault description, **fastest** will quit after proper error message. If the input circuit is not presented, **fastest** will quit with error message.

-o fstdout j outfileg

¹⁴**fastest** was written by Soo Young Lee of the ECE Department at the University of Wisconsin - Madison. It is based on the following paper: "An Efficient Algorithm for Sequential Circuit Test Generation" by T. P. Kelsey and K. K. Saluja and S. Y. Lee, IEEE Transactions on Computers, November, 1993, pp. 1361-1371.

Defines the output file name which will contain the process by **fastest**. If `-o` option is omitted, the **fastest** output is directed to standard output as default. If `-o stdout` is defined, the output goes to the standard output.

-v fstdout j testvecsg

Saves the test vector sequence produced by **fastest** in compact form, which is only compatible to the SFSIM(sequential fault simulator) input format. It has the header to specifies PIs, SCANLs, and LATCHs vector position and other necessary information for the program to use this test vector sequence. If `-v` option is omitted, test vector file will not be generated.

-u udf_file

Saves the undetected **fault_list** separately in UW **fault_list** file format rather than verbose format as presented in file defined by `-o` option. If `-u` option is omitted, the undetected **fault_list** file will not be generated.

-s [init_state]

Optionally set the initial state (LATCHs) values defined in the **init_state** file. To define a LATCH value, "<LATCH

```
LATCH1out 0
LATCH2out 1
```

As usual, comment convention is accepted. NOTE that duplicated initialization is not checked. However, if there is a

-h

Prints out the statistics.

-r

Initialize all flip-flops to 0.

19.3 EXAMPLES

```
fastest -no -p option" ...
// just report total stat.
```

```
fastest -p ...
// report all activity.
```

```
fastest -p ... -o stdout
// report goes to terminal
```

```
fastest -p ... -no -o option""  
    // report goes to terminal
```

```
fastest -p ... -o outf  
    // report goes to "outf"
```

```
fastest ... -v stdout  
    // vector goes to terminal
```

```
fastest ... -no -v option""  
    // vector file will not be generated
```

```
fastest ... -v vecf  
    // vector goes to "vecf"
```

```
fastest ... -no -u option""  
    // undetected fault list will not be generated
```

```
fastest ... -u undf  
    // undetected fault list goes to "undf"
```

20 catpg

catpg - a combinational circuit test pattern generator

20.1 SYNOPSIS

```
catpg -n netlist [-f faultlist] [-t testvecs] [-u [udf file]]
```

20.2 DESCRIPTION

catpg¹⁵ is a test pattern generator for combinational circuits. The circuit is specified in **net_list** file format; and the fault list is specified in **fault_list** file format. The resulting tests generated by **catpg** is stored in the file **testvecs**, if specified. Note that the format of the tests generated is NOT the vector set file format as used in other UW tools. Therefore, the output tests are not directly usable by other tools. The following options are available for **catpg**.

-n netlist

Use the circuit description file **netlist** for simulation. The circuit description must be in the **net_list** file format.

-f faultlist

Use the fault list file **faultlist**. The format must be the **fault_list** file format. The comment line is the same as for **netlist**.

-t testvecs

Use the file **testvecs** for storing the generated tests.

-u [udf_file]

Write the undetected fault list in file **udf_file**. If no file is specified, the result will be printed on the standard output.

20.3 EXAMPLES

```
catpg -n n17 -f n17.f -t n17.t
// netlist file 'n17' is used with fault list file 'n17.f'.
// The resulting tests are stored in 'n17.t'.
```

¹⁵**catpg** was originally called **cat**. Due to the conflict with UNIX utility program **cat**, it was renamed **catpg**. This program was implemented by Byungse So of the ECE Department at the University of Wisconsin - Madison.

21 satpg

satpg - a sequential circuit test pattern generator

21.1 SYNOPSIS

```
satpg -n netlist [-f faultlist] [-t testvecs] [-u [udf file]]
```

21.2 DESCRIPTION

satpg¹⁶ is a test pattern generator for sequential circuits. The circuit is specified in **net_list** file format; and the **fault_list** is specified in **fault_list** file format. The resulting tests generated by **satpg** is stored in the file **testvecs**, if specified. Note that the format of the tests generated is NOT the **vector_set** file format as used in other UW tools. Therefore, the output tests are not directly usable by other tools. The following options are available for **satpg**.

-n netlist

Use the circuit description file **netlist** for simulation. The circuit description must be in the **net_list** file format.

-f faultlist

Use the fault list file **faultlist**. The format must be the **fault_list** file format. The comment line is the same as for **netlist**.

-t testvecs

Use the file **testvecs** for storing the generated tests.

-u [udf_file]

Write the undetected fault list into the file **udf_file**. If no file is specified, the result will be printed on the standard output.

21.3 EXAMPLES

```
satpg -n n17 -f n17.f -t n17.t
// netlist file 'n17' is used with fault list file 'n17.f'.
// The resulting tests are stored in 'n17.t'.
```

¹⁶**satpg** was originally called sat. For naming consistency reason, it was renamed satpg. This program was implemented by Byungse So of the ECE Department at the University of Wisconsin - Madison.

22 detectfaults

detectfaults - remove all faults from a list that are contained in another list and save the output

22.1 SYNOPSIS

```
detectfaults faultlist1.f faultlist2.f outlist.f [-r]
```

22.2 DESCRIPTION

detectfaults¹⁷ takes in two faultlists (A.f and B.f) and outputs another (Out.f). The output contains the “inverse union” of the two lists. It outputs all faults in A.f that do not exist in B.f. The **-r** option suppress any multiple occurring faults if they exist in A.f.

The main purpose of this tool is to give a detected faults list given a master fault list (A.f) and undected fault list (B.f) or UDF.

22.3 EXAMPLES

```
detectfaults main.f udf.f out.f
// remove from main.f all faults from udf.f and place into out.f
```

A.f	B.f	Out.f
0 0 0	0 0 0	1 0 1
1 0 1	1 2 0	2 1 0
1 2 0		
2 1 0		

¹⁷**detectfaults** was implemented by Spencer Millican of the ECE Department at the University of Wisconsin - Madison.