Atalanta-M 2.0 Manual

Modified by Petr Fišer, September 2005

****

Atalanta - Automatic Test Pattern Generator and fault simulator for stuck-at faults in combinational circuits

```
SYNOPSIS: atalanta-M [options] circuit_file
```

OPTIONS:

File specification

Note that all the desired outupt files have to be specified. If not, the file is not created.

-n *fn*	Name of the .bench file. The -n statement is optional, the benchmark name can be specified without any prequisite parameter.
-f *fn*	The fault file is read from *fn*. If not specified, all the s-a-faults are set as default.
-F *fn*	The processed fault list is written to a file *fn*.
-t *fn*	Test patterns are are written or read from the file *fn* (written in TPG mode, read in simulation mode).
-U *fn*	Atalanta writes the list of aborted (or undetected) faults to the given file name.
-v	Atalanta prints out all identified redundant faults as well as aborted fauts in a file. The -U option has to be specified.
-m *fn*	The fault mask is written to the file *fn*. The order of faults corresponds to the fault list.
-P *fn*	The ATPG/FS report is written to a file *fn*.
-W *n*	The output test file format (for TPG only). *n* = 0 - no test pattern output (default). *n* = 1 - PAT file (test vectors only). Do not use together with -D *n* option, since all the test vectors are put together. *n* = 2 - input + output. Do not use together with -D *n* option, since all the test vectors are put together. *n* = 3 - more test vectors for each fault, output. *n* = 4 - more test vectors for each fault, output, with fault mask. HOPE simulator is employed.

ATPG Options

These options have no sense in the simulation mode (-S)

-A	Atalanta derives all test patterns for each fault. In this option, all unspecified inputs are left unknown, and fault simulation is not performed. HOPE fault simulator is employed. *Note: it does not work properly. Not all existing test patterns are produced.*
-D *n*	Atalanta derives *n* test patterns for each fault. In this option, all unspecified inputs are left unknown, and fault simulation is not performed. If both -A and -D option are specified, -D option is applied. HOPE fault simulator is employed. *Note: it does not work properly. Not all existing test patterns are produced.*
-b *n*	The number of maximum backtracks for the FAN algorithm phase 1. (default: -b 10)
-B *n*	If -B *n* (*n* > 0) option is specified, atalanta generates test patterns in two phases. In phase 1, static unique path sensitization is employed. If the test generation for a target fault is aborted in phase 1, the test generation is tried in phase 2. In phase 2, dynamic unique path sensitization is employed. If *n*=0, phase 2 is not performed. If *n* > 0, phase 2 test generation is performed with the backtrack limit of *n*. (default: -B 0, i.e., phase 2 is not performed.)
-H	HOPE is employed for fault simulation. In this option, three logic values (0, 1 and X), instead of two logic values (0 and 1), are employed. Due to the embedding of the unknown logic value and the parallel fault fault simulation algorithm, the test generation time is slower than the default mode.) (default: FSIM, which is a parallel pattern fault simulator, is employed, and two logic values are used.)
-L	Static learning is performed. (default: no learning)
-c *n*	Atalanta compacts test patterns using two different methods: reverse order compaction and shuffling compaction. First, test patterns are applied in the reverse order and fault simulated (reverse order compaction). Second, test patterns are shuffled randomly and fault simulated (shuffling compaction). During the fault simulations, all the test patterns which do not detect a new fault are eliminated. The option -c *n* specifies the limit of shuffling compaction. If *n*>0, shuffling compaction is terminated if *n* consecutive shuffles do not drop a test pattern. If *n*=0, shuffling compaction is not included and compaction is done only by the reverse order fault simulation. (default: -c 2)
-N	Test compaction is not performed.
-r *n*	Random Pattern Testing (RPT) Session is included before deterministic test pattern generation session. The RPT session stops if any n consecutive packets of 32 random patterns do not detect any new fault. If *n*=0, the RPT session is not included. (default: -r 16)
-s *n*	Initial seed for the random number generator (random()). If *n*=0, the initial seed is the current time. (default: -s 0)
-Z	Atalanta derives one test pattern for each fault. In this option, no fault simulation is performed during the entire test generation (including random pattern test generation session, deterministic test generation session and test compaction session). All unspecified inputs are left unknown.
-0, -1, -X, -R	During test generation, some inputs can be unspecified. Atalanta provides various options to set these unspecified inputs into a certain value. (default: -R)

Fault Simulation Options

-S	Simulation mode is performed, instead of TPG. The pattern file has to be specified (-t option), if the -l option is not present.
-l *poly seed num*	Simulates *num* LFSR patterns. The LFSR polynomial and seed are specified in octal form (*poly, seed*).
-g *num fn*	Simulates *num* LFSR patterns. The polynomial and seed are generated randomly and reported to *fn* file.

Note: the limits can be extended upon request.

Examples

```
atalanta-M -t c432.pat -W 1 c432.bench
```

Generates test patterns for c432.bench, writes them to c432.pat.

```
atalanta-M -t c432.pat -W 1 -F c432.flt c432.bench
```

Generates test patterns for c432.bench, writes them to c432.pat. The complete fault list is written to c432.flt.

```
atalanta-M -t c432.pat -W 2 c432.bench
```

Generates test patterns for c432.bench, writes them to c432.pat. The correct CUT responses are computed as well.

```
atalanta-M -D 1 -t c432.pat -W 2 c432.bench
```

Generates test patterns for c432.bench, writes them to c432.pat. One test pattern is produced for each fault. Don't cares are present in the test. The correct CUT responses are computed as well.

```
atalanta-M -t c432.pat -f c432.flt -W 1 c432.bench
```

Generates test patterns for c432.bench, writes them to c432.pat. Only faults specified in c432.bench are considered.

```
atalanta-M -t c432.pat -P c432.rep -m c432.mask -W 1 c432.bench
```

Generates test patterns for c432.bench, writes them to c432.pat. The report file is written to c432.rep, the fault mask (after applying the whole test to CUT) is written to c432.mask.

```
atalanta-M -S -t c432.pat -P c432.rep c432.bench
```

Simulates vectors for c432.pat and writes the results to c432.rep.

```
atalanta-M -S -t c432.pat -P c432.rep -U c432.ud -v c432.bench
```

Simulates vectors for c432.pat and writes the results to c432.rep. All the undetected faults are written to c432.ud.

```
atalanta-M -S -t c432.pat -P c432.rep -m c432.msk c432.bench
```

Simulates vectors for c432.pat and writes the results to c432.rep. The fault mask is written to c432.msk.

```
atalanta-M -l 100040000001 765177704700 1000 -U c432.flt -v -P c432.rep c432.bench
```

Simulates 1000 LFSR vectors for c432.pat and writes the results to c432.rep. The LFSR generating polynomial is 100040000001 (in octal), the seed 765177704700 (in octal). The undetected faults list is written to c432.flt.

File Formats

BENCH Format

The input netlist format for atalanta is ISCAS89 netlist format except for the following two cases. The first line should be # followed by the name of the circuit. The lines beginning with # excluding the first line are comment lines and ignored. These comment lines may be put into any part of the netlist. It should be noted that the order of gates appearing in the netlist is not significant. The name of gates can be a string of alpha-numeric characters (0-9, A-Z, a-z, _, [, or]).

Supported Gates

```
-------------------------------------------------------
   syntax                  gate type
-------------------------------------------------------
   INPUT                   primary input
   OUTPUT                  primary output
   AND                     and gate
   NAND                    nand gate
   OR                      or gate
   NOR                     nor gate
   XOR                     2 input exclusive-or gate
   BUFF or BUF             buffer
   NOT                     inverter
-------------------------------------------------------
* Gate types can be also written in lower case.
```

EXAMPLE: ISCAS89 NETLIST FORMAT (c17.bench)

```
# c17
# 5 inputs
# 2 outputs
# 0 inverters
# 6 gates ( 6 NANDs )

INPUT(1)
INPUT(2)
INPUT(3)
INPUT(6)
INPUT(7)

OUTPUT(22)
OUTPUT(23)

10 = NAND(1, 3)
11 = NAND(3, 6)
16 = NAND(2, 11)
19 = NAND(11, 7)
22 = NAND(10, 16)
23 = NAND(16, 19)
```

Test Pattern Files

Simple PAT Format (-W 1) and the input test pattern file (for simulation)

Consists of test patterns only. The j'th bit of a test pattern is the value of the j'th input of the circuit (in terms of their appearance in the circuit). For example, c17 has five inputs named input1, input2, input3, input6 and input7 which appear in the order in the netlist. The first bit of a test pattern is the value for input1, the second for input2, ..., and the last for input7.

Example: Test Pattern File for C17

```
01010
11110
10101
00111
10010
00101
```

PAT Format with output values (-W 2)

Contains test patterns and correct CUT responses.

Example: Test Pattern File for C17

```
01010 11
00101 01
11111 10
10000 00
```

PAT Format with output values, more vectors for each fault (-W 3)

Contains test patterns and correct CUT responses.
The ordinal number of the test pattern for a particular fault preceeds the colon (:).
The test vectors are numbered in a reverse order, for simpler parsing.

Example: Test Pattern File for C17 (obtained by -D 5 option)

```
3: 100xx 0x
2: 0111x 00
1: 00xxx 0x
2: 1111x 10
1: 101xx 1x
4: 0110x 11
3: 010xx 11
2: 110xx 11
1: 1x1xx 1x
3: 110xx 11
2: 0110x 11
1: 010xx 11
1: 100xx 0x
1: 001xx 0x
1: 101xx 1x
1: 100xx 0x
4: 11100 11
3: 110xx 1x
2: 0110x 1x
1: 010xx 1x
5: 10111 10
4: 101x0 10
3: 100xx 0x
2: 0111x 00
1: 00xxx 0x
1: x111x x0
1: 000xx 0x
5: x00x1 01
4: 11101 11
3: 11100 11
2: 0110x 1x
1: x10xx 1x
2: x0011 01
1: x101x 1x
4: x0111 x0
3: 11111 10
2: 11110 10
1: 0111x 0x
1: 0110x 1x
3: x111x x0
2: x0111 x0
1: x0xx0 x0
2: x0101 x1
1: x00x1 01
```

```
4: x0101 x1
3: x00x1 01
2: x110x 11
1: x10xx 11
2: x1100 11
1: x10x0 11
1: xx111 x0
1: x00x0 00
```

The full PAT format (-W 4)

Contains test patterns, correct CUT responses and fault masks for each test pattern (for more information see the Fault Mask Format).
The ordinal number of the test pattern for a particular fault preceeds the colon (:).
The test vectors are numbered in a reverse order, for simpler parsing.

Example: Test Pattern File for C17 (obtained by -D 5 option)

```
3: 100xx 0x 1000100101010000000000
2: 0111x 00 1000011001100010100000
1: 00xxx 0x 1000000001000000000000
2: 1111x 10 0110001001100010100000
1: 101xx 1x 0110001000000000000000
4: 0110x 11 0011000010001001001000
3: 010xx 11 0011000010001000001000
2: 110xx 11 0011000010001000001000
1: 1x1xx 1x 0010000000000000000000
3: 110xx 11 0011000010001000001000
2: 0110x 11 0011000010001001001000
1: 010xx 11 0011000010001000001000
1: 100xx 0x 1000100101010000000000
1: 001xx 0x 1000010001000000000000
1: 101xx 1x 0110001000000000000000
1: 100xx 0x 1000100101010000000000
4: 11100 11 0010000010001001001100
3: 110xx 1x 0011000010001000001000
2: 0110x 1x 0011000010001001001000
1: 010xx 1x 0011000010001000001000
5: 10111 10 0110001001000010100010
4: 101x0 10 0110001001000000100000
3: 100xx 0x 1000100101010000000000
2: 0111x 00 1000011001100010100000
1: 00xxx 0x 1000000001000000000000
1: x111x x0 0000001001100010100000
1: 000xx 0x 1000000001010000000000
5: x00x1 01 1000000001011000011000
4: 11101 11 0010000000001001001000
3: 11100 11 0010000010001001001100
2: 0110x 1x 0011000010001001001000
1: x10xx 1x 0011000010001000001000
2: x0011 01 1000000101011100011000
1: x101x 1x 0011000110001100001000
4: x0111 x0 0000001001000010100010
3: 11111 10 0110001001100010100010
2: 11110 10 0110001001100010100000
1: 0111x 0x 1000011001100010100000
1: 0110x 1x 0011000010001001001000
3: x111x x0 0000001001100010100000
2: x0111 x0 0000001001000010100010
1: x0xx0 x0 0000000001000000100000
2: x0101 x1 0000000000001001011000
1: x00x1 01 1000000001011000011000
4: x0101 x1 0000000000001001011000
3: x00x1 01 1000000001011000011000
2: x110x 11 0010000000001001001000
1: x10xx 11 0011000010001000001000
2: x1100 11 0010000010001001001100
1: x10x0 11 0011000010001000001100
1: xx111 x0 0000001001000010100010
1: x00x0 00 1000000001010000100001
```

Fault List Format

```
------ EXAMPLE: A FAULT LIST FILE ----------------------
gate_A->gate_B /1
gate_A->gate_B /0
gate_A /1
gate_B /1
--------------------------------------------------------
```

In the above example, gate_A and gate_B are the name of gates. The first line, "gate_A->gate_B /1" describes the stuck-at 1 fault on the gate_B input line, which is connected to gate_A. Similarly, the second line, "gate_A->gate_B /0" describes the stuck-at 0 fault on the gate B input which is connected to gate_A. The third and fourth lines describe the stuck-at 1 faults on the gate_A output and the gate_B output, respectively.

Fault Mask Format

The fault mask is a string determining the state (coverage) of particular faults. One character is reserved for each processed fault. The order corresponds to the fault list order.
Meaning of the values:

0 - not detected
1 - detected
3 - redundant
4 - aborted

The Report File Format

Example. *The comments are not a part of the report file.*

```
gates: 6                                // number of CUT gates
iv: 5                                   // number of CUT primary inputs
ov: 2                                   // number of CUT primary outputs
i_patterns: 52                  // number of test patterns before compaction (or final, if no compaction)
patterns: 52                    // number of final test patterns (after compaction)
faults: 22                              // number of processed faults
d_faults: 22                    // number of detected faults
r_faults: 0                             // number of identified redundant faults
time: 0.001                             // CPU time [s]
```