IEEE 6th Workshop on RTL and High Level Testing, pp. 123-130, July 2005.

A Functional Test Method for State Observable FSMs

Toshinori Hosokawa † and Hideo Fujiwara ‡

† College of Industrial Technology, Nihon University
1-2-1, Izumicho, Narashino, Chiba 275-8575, Japan
‡ Graduate School of Information Science, Nara Institute of Science and Technology (NAIST)
8916-5, Takayama, Ikoma, Nara 630-0192, Japan
E-mail: † t7hosoka@cit.nihon-u.ac.jp, ‡ fujiwara@is.naist.jp

Abstract

Scan testing is the most popular test method for VLSIs. However, because the shift operation during the test causes over testing, the yield loss of VLSIs may occur. It is important to test VLSIs using the given function. On the other hand, it is well known that state-observable and completely specified FSMs are completely functionally tested by performing all possible state transitions. This method, however, drastically increases the test cost. This paper proposes two test methods, a fault independent functional test method and a fault dependent test generation method, for state-observable FSMs to reduce the yield loss while improving the test quality over scan testing at a reasonable test cost. Fault dependent test generation is based on a single pattern test for a logical fault model and a two-pattern test for a timing fault model. Experimental results using MCNC'91 benchmark circuits show that the proposed method reduces the average area by 13.2% while increasing the state transition coverage 4.3 times by using only 29.4% additional test sequences compared with the conventional non-scan test method for FSMs.

keywords *: State-observable FSMs, Constrained ATPG, Over testing, Fault dependent test generation, Fault independent functional test, State transition coverage*

1. Introduction

In recent years, VLSI (Very Large Scale Integrated circuit) testing has become more important because the number of gates on VLSIs is rapidly increasing and their complexity is growing with advances in semiconductor technology. Currently, scan testing for the stuck-at fault model [1, 2] is one of the most popular test methods for VLSIs. However, it has been reported that scan testing for the stuck-at fault model may not detect defective VLSIs [4] and delay testing [3] and at-speed functional testing can effectively improve test quality.

VLSI design methodologies employing hardware description languages have recently been adopted to reduce VLSI design time. VLSIs are designed at RTL (Register Transfer Level) and the RTL circuits consist of a data path part and a controller part. The data path contains a hardware element (e.g., registers, multiplexers, and operational modules) and signal lines and the controller is represented by an FSM (finite state machine). A non-scan based DFT (Design For Testability) method of data path parts is proposed in [5], while a non-scan based DFT method for controller parts is proposed in [6]. At-speed testing is easily applicable and test patterns for a stuck-at fault model are completely generated by the non-scan based DFT methods. As mentioned above, if at-speed functional testing and/or delay testing are applied to VLSIs with a non-scan based DFT, the test quality can be further improved. It was reported in [7] that state-observable and completely specified FSMs are thoroughly functionally tested by performing all of the state transitions. Thus, complete at-speed functional testing can be applied to FSMs whose states are made observable by performing all of the state transitions at-speed.

As mentioned above, scan testing is currently the most popular test method. Scan testing is not based on the function, but the structure of the circuit. In this method, the states of the circuits are transferred to invalid states by shift operation during the testing in order to detect faults. This method is considered over testing and a yield loss of VLSIs may occur. It is important to test using the function, and the test method described in [7] can functionally test state-observable FSMs completely, but at a dramatically increased test cost.

This paper proposes two test methods, a fault independent functional test method and a fault dependent test generation method, for state-observable FSMs in order to reduce the yield loss

caused by over testing while improving the test quality over that of scan testing within a reasonable test cost. This paper also proposes state transition coverage as the measure of test quality for functional testing. State transition coverage is defined as the ratio of the number of state transitions executed by a test sequence to the entire number of state transitions. Moreover, problems are formulated from the points of view of both fault independent functional tests and fault dependent tests for state-observable FSMs, and test generation methods are proposed to solve these problems.

This paper is organized as follows. In Section 2, a functional testing method for state-observable and completely specified FSMs is detailed. In Section 3, the optimization problems for both functional and fault dependent tests are formulated for state-observable FSMs, and their test generation methods are proposed. The proposed methods are applied to MCNC'91 FSM benchmarks [9] in Section 4, and the experimental results are presented. Finally, Section 5 concludes the paper and discusses future research possibilities.

2. Functional Test for State-Observable and Completely Specified FSMs

Fig. 1 shows an example of an FSM. In this figure, ST0 through ST5 and T0 through T11 show the states and the input values of the state transitions (the value of each primary input $\in \{0, 1, X\}$, where X denotes don't care), respectively. The DFT makes the outputs of the status registers in the FSM observable and a synchronous sequential circuit is synthesized from the FSM using logic synthesis. Fig. 2 shows the logic circuit model that corresponds to an FSM after logic synthesis. Because the PPI, which is the outputs of the status registers, is observable in this figure, the PPI is treated as the primary output. PI, PO, SR, PPI, PPO, and R denote primary inputs, primary outputs, status registers, pseudo primary inputs (the outputs of the status registers), pseudo primary outputs (the inputs of a the status registers), and a reset input, respectively. The combinational circuit part is called the test generation model because combinational test generation is applied to the circuit.

It has been reported that state-observable and completely specified FSMs are completely functionally tested by performing all of the possible state transitions [7]. Thus, complete at-speed functional testing can be applied to FSMs whose states are made observable by performing all of the state transitions at-speed.

3. A Test Method for State-Observable FSMs

From now on, state-observable and completely specified FSMs will only be called state-observable FSMs due to simpkification.

3.1 Preliminaries

(Definition 1: Functional test for state-observable FSMs)

In this test, the PI value is applied to a state-observable FSM, the state is then transferred from the current state to the next state, and the resulting PPI and PO values are observed. A series of these procedures are referred to as a functional test for state-observable FSMs.

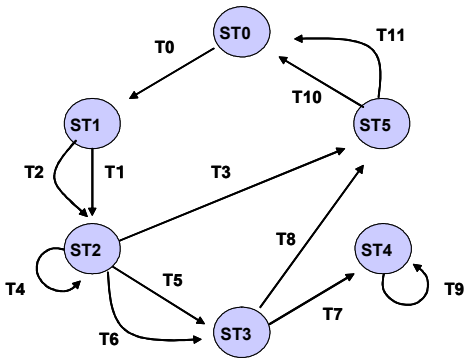

Fig. 1 An example of an FSM

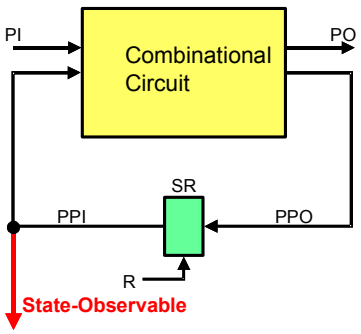

Fig. 2 A logic model for a state-observable FSM

Example 1: In Fig. 1, T0 is applied to state ST0 on the state-observable FSM and the state is transferred from ST0 to ST1. T1 is then applied, and the state is transferred from ST1 to ST2. Fig. 3 depicts the timing chart for this functional test for a state-observable FSM. R is activated and the values of the status registers are initialized to ST0 at the first cycle. At the second cycle, T0 is applied and the values of the POs for (PI, PPI)=(T0, ST0) are observed just before the rising edge of the clock. Here, (PI, PPI) denotes that the value of PI is applied to the PPI value (state) for a state-observable FSM. Moreover, the PPI value is observed after the rising edge of the clock. Thus, it is verified that the state is successfully transferred from ST0 to ST1. At third cycle, T1 is applied and the PO values for (PI, PPI) =(T1, ST1), which are observed just before the rising edge of the clock. The resulting PPI value is observed after the rising edge of the clock. Thus, it is verified that the state is successfully transferred from ST1 to ST2.

(Definition 2: FSM test generation graph)

An FSM test generation graph is a directed graph G(V, E, r, s) where a vertex v ∈ V denotes a state. Each vertex has a label s: V → A (A={$PPI_1PPI_2\cdots PPI_m$}, PPI_1, PPI_2, ⋯, PPI_m ∈ {0, 1}, where m denotes the number of status registers). The label s indicates the state assignment. For any vertices u, v ∈ V, an edge (u, v) ∈ E denotes the state transition from u to v, and each edge has a label r: E → B (B={$PI_1PI_2\cdots PI_n$}, PI_1, PI_2, ⋯, PI_n∈ {0, 1}, where n denotes the number of primary inputs). The label indicates the input values for a state transition.

Example 2: Fig. 4(a) shows an example of the state-observable FSM and Fig. 4(b) shows an example of an FSM test generation graph. A code is assigned to each state in Fig. 4(b), and input values containing "don't care" for a state transition are expanded into those without "don't care." (ST0, ST1) and XX, represented by a cube, are expanded into four edges and 00, 01, 10, and 11 are represented by vectors.

(Definition 3: State transition coverage)

State transition coverage is defined as the ratio of the number of state transitions executed by a test sequence to the total number of state transitions. When state transitions are executed by a test sequence, it is referred to as the state transitions are covered by the test sequence. In this paper, only state transitions specified in an FSM are used to calculate the state transition coverage. On state-observable FSMs, the input value for each state transition is represented using two types of notation. One notation is a cube that consists of 0's, 1's, and don't cares (Xs). The other notation is a vector that consists of 0's and 1's. While generating an FSM test generation graph, state transitions and their input values initially represented by a cube in a state-observable FSM are expanded into state transitions and input values that are represented by a vector. If the input value for a state transition includes k don't cares in an FSM, the state transition is expanded into 2^k state transitions in an FSM test generation graph. Thus, state transitions with an input value represented by a vector in an FSM test generation graph are used to calculate the state transition coverage. State transition coverage is then used as a measure of the test quality for a functional test.

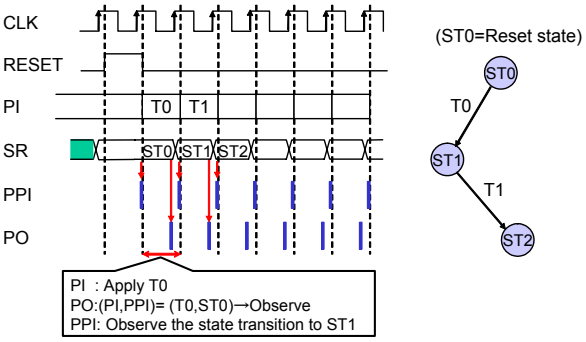

Fig. 3 A functional test for a state-observable FSM

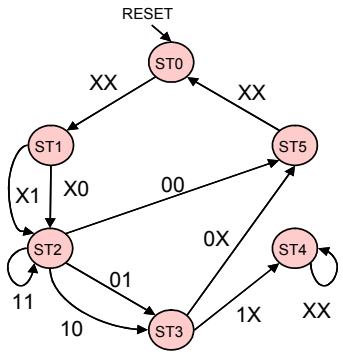

Fig. 4(a) A state-observable FSM

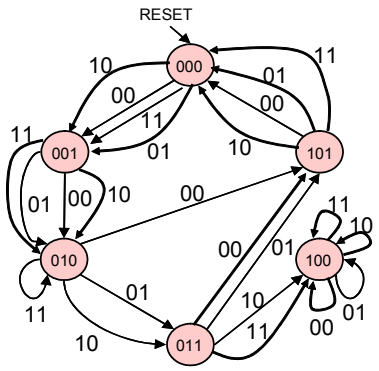

Fig. 4(b) An FSM test generation graph for a state-observable FSM

Example 3: When the test sequence (11, 00, 00) is applied to the FSM test generation graph in Fig. 4(b) on the current state 000, the state transition coverage is 12.5% (3/24).

(Definition 4: Detectable stuck-at faults on valid states)

When a state is defined in a state-observable FSM and is reachable from the reset state, the state is called a valid state. Stuck-at faults detected by performing a state transition on a current state are called detectable stuck-at faults on valid states.

Example 4: The input value for state transition 00 is applied to the valid state 001 shown in Fig. 4(b), and the subsequent state 010 and output value are observed. Stuck-at faults detected by the above-mentioned procedures are detectable stuck-at faults on valid states.

(Definition 5: Detectable delay faults on the transition between valid states)

The transition between valid states refers to performing state transitions between valid states in state-observable FSMs. After a transition between valid states, delay faults detected by performing a continuous state transition are defined as detectable delay faults on the transition between valid states.

Example 5: The input value for state transition 11 is applied to the valid state 000 shown in Fig. 4 (b), and the state is transferred to valid state 001. Next, the input value for state transition 00 is continuously applied to valid state 001, and the subsequent state 010 and output value are observed. Delay faults detected by the above-mentioned procedures are detectable delay faults on the transition between valid states.

3.2 Fault Independent Functional Test Method

State-observable FSMs are considered to be completely functionally tested for a specified function by performing all of the state transitions. The following two problems are formulated for a functional test of state-observable FSMs.

(Formulation 1a)

Input: a state-observable FSM

Output: a test sequence such that the state transition coverage is 100% (All state transitions in the FSM test generation graph are performed.)

Optimization: minimizing the test length

The test generation for Formulation 1a does not use ATPG (Automatic Test Pattern Generation) for a specific fault model. An FSM test generation graph is generated from the state-observable FSM and it searches the path so that all of the edges are traversed at least once. If the path length is minimized, the test length is also minimized. This test sequence can functionally test a state-observable FSM completely, although this method generates a huge amount of test sequences, causing the test cost to drastically increase.

(Formulation 1b)

Input: a state-observable FSM

Output: a test sequence such that state transitions with an input represented by a cube and state transitions with an input represented by a vector in a state observable FSM are covered (Hence, on each state transition with an input represented by a cube, at least one state transition from among the corresponding state transitions in the FSM test generation graph is performed.)

Optimization: minimizing the test length

The test of Formulation 1b aims at improving the quality of the functional test by formulating a more reasonable test length. The test generation for this method also does not use ATPG for a specific fault model. An FSM test generation graph is generated from a state-observable FSM. When the graph is generated, each state transition with an input represented by a cube is expanded into state transitions with inputs represented by vectors. The method then searches the path so that all of the original state transitions with an input represented by a vector and at least one state transition from among

the expanded state transitions are traversed at least once. If the path length is minimized, the test length is also minimized. This test generation is expected to have a longer test length, but the state transition coverage becomes higher than the fault dependent test generation formulated in the next sub-section.

3.3 Fault Dependent Test Generation Method

In this sub-section, problems are formulated for both one pattern testing for a logical fault model and two pattern testing for a timing fault model on state-observable FSMs. In this paper, a stuck-at fault model is used as the representative of one pattern testing and a delay fault model is used as the representative of two pattern testing. The purpose of these test generations is to reduce the yield loss caused by over testing while maintaining a high quality test sequence. Problems are formulated for a stuck-at fault model and a delay fault model in the following two test generations.

(Formulation 2a)

Input:

- a state-observable FSM
- a test pattern set with 100% stuck-at fault efficiency

Output: a test sequence for a state-observable FSM so that all detectable stuck-at faults on valid states are detected

Optimization: minimizing the test length

After logic synthesis, a combinational circuit part (test generation model) is extracted from the synthesized sequential circuit. The valid states are assigned to the PPI values as constraints. A constrained ATPG is performed on the stuck-at faults for a test generation model and a test pattern set is generated. After that, an FSM test generation graph is generated and test patterns are assigned to the corresponding edges of an FSM test generation graph. Finally, paths are searched on the FSM test generation graph so that all of the edges where test patterns are assigned are traversed at least once. If the path length is minimized, the test length is also minimized. For this test generation, it is expected that the state transition coverage decreases, but the test length become shorter than those for the functional test in Formulation 1b.

Example 6: Fig. 5 shows an FSM test generation graph that is generated from the state-observable FSM shown in Fig. 4(a) and a test pattern set with 100% stuck-at fault efficiency for the test generation model. Each test pattern is assigned to the corresponding edge in this graph. For example, t_1 is assigned to the edge that represents the state transition from the state 001 through the input value 11.

(Formulation 2b)

Input:

- a state-observable FSM
- a two test pattern set with 100% delay fault efficiency

Output: a test sequence for a state-observable FSM so that all detectable delay faults on the transition between valid states are detected

Optimization: minimizing the test length

After logic synthesis, a time expansion model [10] for two time frames is generated from a synthesized sequential circuit. The valid states are assigned to the PPI values as constraints. A constrained ATPG for delay faults is performed for the time expansion model and a two test pattern set is generated. Because the delay fault test generation for state-observable FSMs is a future work, the test generation method is not discussed in this paper.

4. Experimental Results

The test generation methods for Formulations 1a, 1b, and 2a were implemented and were applied to MCNC'91 benchmark circuits [9].

The platform used for the experiments is as follows.

CPU: Pentium 4
Frequency: 1.8 GHz
Memory: 1 Gbyte

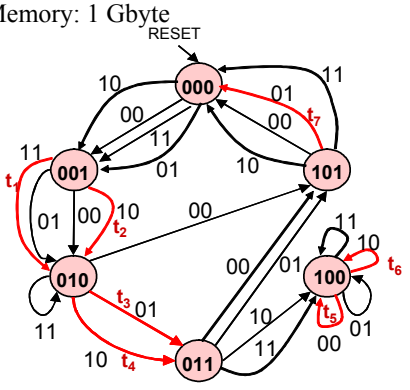

Fig. 5 An FSM test generation graph for stuck-at fault testing

The characteristics of MCNC'91 benchmark circuits are shown in Table 1. In this figure, Circuit, #PI, #PO, #Reg, #Vertex, #Edge, and TL denote the circuit names, the number of primary inputs, the number of primary outputs, the number of status registers, the number of states, the number of state transitions and the test length with 100% state transition coverage generated by the test generation of Formulation 1a, respectively. "NA" indicates that the test generation was not completed within 10 hours. It was discovered that extremely large test lengths and CPU times were required to perform a complete functional test. In these experiments, FSMs were made state-observable by DFT and the three test generations were performed for state-observable FSMs.

The logic syntheses for the FSMs were performed by the Synopsys Design Compiler® and the test generations for the combinational circuits were performed by Synopsys TetraMax®.

Table 2 compares the experimental results for the proposed methods with those for [6], denoted by "JETTA'00" in the table. In [6], testing for both valid and invalid states was performed for FSMs and stuck-at faults were completely tested. "Proposed methods" denotes the experimental results of the proposed methods. "1b" denotes the experimental results of the functional test method for Formulation 1b and "2a" denotes the experimental results of the stuck-at fault test generation method for Formulation 2a. Circuit, FE, Area, TL, #CE, STC, and FC denote the circuit name, the stuck-at fault efficiency, the area of synthesized sequential circuit with DFT that was calculated based on the standard library for logic synthesis, the test length, the number of edges covered by the test sequence, the state transition coverage, and the stack-at fault coverage, respectively.

Compared to "JETTA'00," "2a" reduced the area by 1.9 to 27.7% (average 13.2%), while increasing the state transition coverage from 1.5 to 10.8 times (average 4.3 times) by using -18.1 to 160.4% (average 29.4 %) additional test sequences. After the state is transferred to a test state in [6], the PI values of the test patterns are applied one after another while holding the PPI value in the status registers. Therefore, the path is searched so that all of the test states are traversed at least once. This method reduced the test length, making it smaller than that of "2a." On the other hand, because the stuck-at faults were tested by performing state transitions in "2a," the state transition coverage was higher than for "JETTA'00." As for the area, "2a" made the outputs of status registers observable by DFT, while "JETTA'00" made the inputs of status registers observable. Moreover, "JETTA'00" added the circuit that was going to be transferred to invalid test states, the multiplexers switched the valid state transitions and the invalid state transitions, and the hold function of status registers by DFT. Therefore, "2a" reduced the area compared to "JETTA'00."

Test length of "1b" compared "2a" was 1.3 to 13.3 times longer (average 3.4 times), and the state transition coverage was 0.5 to 8.2 times higher (average 1.8 times). "1b" detected all of the detectable stuck-at faults on the valid states for the FSMs, but the ratio of detected faults to detectable stuck-at faults on the valid states was 92.4 to 97.7% for cse, ex6, lion, opus, s386, and train11.

5. Conclusion

This paper proposed both a fault independent functional test method and a fault dependent test generation method for state observable FSMs. This paper also proposed state transition coverage as the measure of test quality for a functional test. The following conclusions were obtained from applied the proposed methods to MCNC'91 benchmark circuits.

(1) The test generation of Formulation 2a reduces the average area by 13.2% while increasing the state transition coverage by 4.3 times, while adding only 29.4% test sequences compared with [6].

(2) The functional test of Formulation 1b increases the average state transition coverage 1.8 times by using 3.4 times additional test sequences compared with that of Formulation 2a.

Future work includes proposing a test generation for a delay fault model in state-observable FSMs (Formulation 2b).

Acknowledgements

The authors would like to thank Mr. Seiji Hamamoto of System JD Co., Ltd. for his invaluable discussion and comments.

Table 1 FSM benchmark characteristics

Circuit	#PI	#PO	#Reg	#Vertex	#Edge	TL
bbara	4	2	4	10	160	3730
bbtas	2	2	3	6	24	46
beecount	3	4	3	7	56	671
cse	7	7	4	16	2048	NA
dk14	3	5	3	7	56	95
dk15	3	5	2	4	32	49
dk16	2	3	5	27	108	228
dk17	2	3	3	8	32	81
dk27	1	2	3	7	14	34
ex1	9	19	5	20	10240	NA
ex3	2	2	4	10	40	108
ex4	6	9	4	14	896	304949
ex5	2	2	4	9	36	103
ex6	5	8	3	8	256	1592
keyb	7	2	5	19	2432	NA
kirkman	12	6	4	16	65536	NA
lion	2	1	2	4	16	34
lion9	2	1	4	9	36	61
mc	3	5	2	4	32	95
opus	5	6	4	10	320	15185
planet	7	19	6	48	6144	NA
pma	8	8	5	24	6144	NA
s1	8	6	5	20	5120	NA
s1488	8	19	6	48	12288	NA
s1494	8	19	6	48	12288	NA
s208	8	2	5	18	4608	NA
s27	4	1	3	6	96	650
s298	3	6	8	218	1744	NA
s386	7	7	4	13	1664	NA
s420	8	2	5	18	4608	NA
s510	19	7	6	47	24641536	NA
s820	18	19	5	25	6553600	NA
s832	18	19	5	25	6553600	NA
sand	11	9	5	32	65536	NA
shiftreg	1	1	3	8	16	32
styr	9	10	5	30	15360	NA
tav	4	4	2	4	64	189
tbk	6	3	5	32	2048	NA
tma	7	6	5	20	2560	NA
train11	2	1	4	11	44	172
train4	2	1	2	4	16	28

References

[1] H. Fujiwara, “Logic Testing and Design for Testability,” The MIT Press, 1985.

[2] M. Abramovici, M. A. Breuer, and A. D. Friedman, “Digital systems testing and testable design,” IEEE Press, 1995.

[3] A. Krstic, and K.-T. Cheng, “Delay Fault Testing for VLSI Circuits,” Kluwer Academic Publishers, 1998.

[4] P.C. Maxwell, R.C. Aitken, R. Kollitz, and A. C. Brown, “IDDQ and AC Scan: The War Against Unmodelled Defects,” Proc. of IEEE Int. Test Conf., pp.250-258, Oct., 1996.

[5] H. Wada, T. Masuzawa, K.K. Saluja, and H. Fujiwara, “Design for strong testability of RTL data paths to provide complete fault efficiency,” Proc. of 13th Int. Conf. on VLSI Design, pp.300-305, 2000.

[6] S. Ohtake, T. Masuzawa, and H. Fujiwara, "A non-scan approach to DFT for Controllers Achieving 100% Fault Efficiency," Journal of Electronic Testing: Theory and Applications (JETTA), Vol. 16, No. 5, pp.553-566, Oct. 2000.

[7] H. Fujiwara, and K. Kinoshita, “Design of Diagnosable Sequential Machines Utilizing Extra Outputs,” IEEE Trans. on Computers, Vol. C-23, pp.138-145, Feb., 1974.

[8] T. Sasao, “Switching Theory for Logic Synthesis,” Kluwer Academic Pub., 1999.

[9] S. Yang, "Logic synthesis and optimization benchmarks user guide," Technical Report 1991-IWLS-UG-Saeyang, Microelectronics Center of North Carolina, 1999.

[10] T. Inoue, T. Hosokawa, T. Mihara,, and H. Fujiwara, “An optimal time expansion model based on combinational ATPG for RT level circuits”, IEEE Proc. Asian Test Symp., pp.190-197 , Dec. 1998.

Table 2 Experimental results

Circuit	JETTA'00[6]					Proposed methods									
							1b				2a				
	FE	Area	TL	#CE	STC	Area	TL	#CE	FC	STC	TL	#CE	FC	FE	STC
bbara	100.00%	207	40	9	5.63%	163	121	60	98.28%	37.50%	52	36	98.28%	100.00%	22.50%
bbtas	100.00%	83	16	5	20.83%	60	46	24	97.40%	100.00%	19	17	97.40%	100.00%	70.83%
beecount	100.00%	248	30	7	12.50%	226	63	33	98.31%	58.93%	33	23	98.31%	100.00%	41.07%
cse	100.00%	410	95	19	0.93%	394	278	95	94.31%	4.64%	155	130	100.00%	100.00%	6.35%
dk14	100.00%	213	41	9	16.07%	191	95	56	98.95%	100.00%	56	40	98.95%	100.00%	71.43%
dk15	100.00%	113	25	3	9.38%	105	49	32	100.00%	100.00%	28	25	100.00%	100.00%	78.13%
dk16	100.00%	507	93	34	31.48%	455	228	108	98.12%	100.00%	93	69	98.12%	100.00%	63.89%
dk17	100.00%	146	29	8	25.00%	134	81	32	100.00%	100.00%	41	24	100.00%	100.00%	75.00%
dk27	100.00%	80	21	8	57.14%	58	34	14	97.50%	100.00%	21	12	97.50%	100.00%	85.71%
ex1	100.00%	807	101	26	0.25%	728	396	156	99.18%	1.52%	128	123	99.18%	100.00%	1.20%
ex3	100.00%	185	46	13	32.50%	142	102	37	94.96%	92.50%	51	28	94.96%	100.00%	70.00%
ex4	100.00%	332	41	13	1.45%	297	138	35	98.07%	3.91%	43	42	98.07%	100.00%	4.69%
ex5	100.00%	177	43	10	27.78%	132	97	33	94.95%	91.67%	50	25	94.95%	100.00%	69.44%
ex6	100.00%	326	41	7	2.73%	314	69	35	95.86%	13.67%	43	41	100.00%	100.00%	16.02%
keyb	100.00%	386	94	18	0.74%	332	497	170	96.78%	6.99%	197	122	96.78%	100.00%	5.02%
kirkman	100.00%	573	73	15	0.02%	557	716	398	100.00%	0.61%	163	162	100.00%	100.00%	0.25%
lion	100.00%	45	13	3	18.75%	37	27	12	92.45%	75.00%	12	9	100.00%	100.00%	56.25%
lion9	100.00%	273	27	8	22.22%	216	58	34	95.30%	94.44%	26	18	95.30%	100.00%	50.00%
mc	100.00%	44	11	3	9.38%	36	16	10	100.00%	31.25%	9	8	100.00%	100.00%	25.00%
opus	100.00%	242	41	10	3.13%	200	113	30	94.30%	9.38%	59	55	96.14%	100.00%	17.19%
planet	100.00%	700	204	82	1.33%	602	723	115	97.70%	1.87%	196	194	97.70%	100.00%	3.16%
pma	100.00%	755	116	29	0.47%	689	339	97	98.81%	1.58%	147	143	98.81%	100.00%	2.33%
s1	100.00%	680	111	26	0.51%	606	245	107	97.67%	2.09%	139	135	97.67%	100.00%	2.64%
s1488	100.00%	1160	219	84	0.68%	1062	1997	251	97.96%	2.04%	262	260	97.96%	100.00%	2.12%
s1494	100.00%	1115	211	81	0.66%	1019	1903	250	98.07%	2.03%	244	242	98.07%	100.00%	1.97%
s208	100.00%	316	48	17	0.37%	246	1060	153	91.95%	3.32%	125	122	91.95%	100.00%	2.65%
s27	100.00%	132	27	6	6.25%	105	57	34	93.53%	35.42%	31	27	93.53%	100.00%	28.13%
s298	100.00%	4394	646	280	16.06%	4305	4162	1108	99.18%	63.53%	849	548	99.18%	100.00%	31.42%
s386	100.00%	329	59	15	0.90%	290	225	64	95.11%	3.85%	100	95	97.31%	100.00%	5.71%
s420	100.00%	319	48	17	0.37%	249	957	137	91.95%	2.97%	119	118	91.95%	100.00%	2.56%
s510	100.00%	390	121	46	0.00%	308	1110	1109	97.36%	0.00%	138	137	97.36%	100.00%	0.00%
s820	100.00%	661	133	30	0.00%	598	1351	1350	97.92%	0.02%	222	221	97.92%	100.00%	0.00%
s832	100.00%	670	131	32	0.00%	609	1347	1346	97.99%	0.02%	237	236	97.99%	100.00%	0.00%
sand	100.00%	904	181	44	0.07%	884	315	189	100.00%	0.29%	156	148	100.00%	100.00%	0.23%
shiftreg	100.00%	67	22	9	56.25%	55	32	16	100.00%	100.00%	24	14	100.00%	100.00%	87.50%
styr	100.00%	724	151	39	0.25%	685	492	167	98.95%	1.09%	212	211	98.95%	100.00%	1.37%
tav	100.00%	49	16	3	4.69%	41	63	49	100.00%	76.56%	14	13	100.00%	100.00%	20.31%
tbk	100.00%	1019	222	44	2.15%	999	2998	1568	100.00%	76.56%	224	191	100.00%	100.00%	9.33%
tma	100.00%	593	98	25	0.98%	526	203	63	98.52%	2.46%	106	92	98.52%	100.00%	3.59%
train11	100.00%	287	50	13	29.55%	251	71	36	90.85%	81.82%	46	33	98.17%	100.00%	75.00%
train4	100.00%	43	12	3	18.75%	35	28	16	100.00%	100.00%	11	10	100.00%	100.00%	62.50%