The Influence of Implementation Technology on Dependability Parameters

Jan Schmidt, Petr Fišer, and Jiří Balcárek
Dept. of Digital Design
Czech Technical University in Prague
Prague, Czech Republic
{jan.schmidt; petr.fiser; jiri.balcarek}@fit.cvut.cz

Abstract— **Circuits which are designed to be dependable are evaluated after gate-level design. To demonstrate the influence of implementation technology on dependability parameters, we developed a simple method which transforms the evaluation problem into conceptual hardware and then to SAT instances. The method can accommodate any combinational fault model. The performed evaluation demonstrated that the dependability parameters of the implementations correlate to a significant degree.**

Keywords-dependability; SAT; fault classification; generalized miter

I. INTRODUCTION

The principal way to improve the dependability of a circuit is to introduce redundancy. One possible strategy is to *detect* errors in output signals and take appropriate measures during circuit operation. This technique is called Concurrent Error Detection (CED [1]).

Redundancy must be introduced with care, as redundant blocks are also prone to faults, and the point of diminishing return can be reached easily. Numerous schemes were devised to balance redundancy and dependability. To evaluate variant designs, we need to know how much dependability we get for a given investment into redundant circuits. Initially, dependability meant roughly what is today called robustness. In this paper, we adhered to the original meaning from [2], where *dependability parameters* were introduced to quantify dependability.

The standard design flow is to design the circuit first, to construct its redundancy afterwards, and then to evaluate its dependability parameters. The underlying assumption is that the actual design, the technology used and its resulting fault models influence the dependability parameters to a large degree. In many studies, the ubiquitous stuck-at (S@) models were used.

Recent Automatic Test Pattern Generation (ATPG) programs [3] and procedures based on solving the Satisfiability Problem (SAT) [4] permit analysis with a variety of fault models suitable for a particular circuit implementation technology. This in turn enables us to see the influence of technology on dependability. In other words, we ask whether there are circuits hard to make dependable or technologies hard to make dependable.

To study this question, we needed a simple framework. Recently, two approaches to robustness analysis and other tasks dealing with faults exist. The first one, represented by [5] and [6], transforms the task instance to an ATPG task instance. The ATPG program then may or may not convert it internally to one or more SAT instances [7], [9]. The other approach, most notably represented by [4] and [10], converts an instance of the task to *conceptual hardware* (hardware which is not intended for synthesis) and then constructs SAT instances from that hardware.

Both approaches can be seen as special cases of a more general method, which can be summarized as follows. Firstly, transform the task instance into a piece of conceptual hardware together with *assertions* about the hardware. Secondly, use formal verification methods to prove or disprove the assertions (e.g. [3], [11]). If required, transform the assertions into conceptual hardware as well [8]. Thirdly, transform the answers back to the answers to the original task instance.

This is a powerful framework, which can even produce answers about sequential behavior in the presence of multiple faults. For our study, combinational circuits (or full-scan circuits) were sufficient. Furthermore, only fault classification was required, without the need to analyze multiple fault impact.

Therefore, we present a simple framework for this limited situation, which constructs conceptual hardware representing the circuit and the assertions directly. We borrowed the term *miter* from ATPG [7], although *monitor* from [8] and other sources has a similar meaning.

We present a method to determine the value of an arbitrary Boolean formula over input vectors, error-free output vectors, and error-stricken output vectors of a combinational circuit. The formula can be quantified over all input vectors or their subset.

Although we cannot overcome the exponential worst case complexity of the SAT problem, we have *practical* solvers for the Boolean Satisfiability Problem, which solve *both* satisfiable and unsatisfiable instances effectively. The method also benefits from the fact that the SAT instances encountered during ATPG, and, as it was discovered, robustness analysis, are far simpler to solve than the worst case [11], [12], [13].

We present the method as an extension of SAT ATPG first in its general form. Then we review the class of dependable circuits studied and present their fault classification. We demonstrate application of the proposed method on this problem, and finally show and discuss

classification results for two different implementation technologies.

II. Predicate Evaluation

A. The SAT-Based ATPG

Let the circuit in question realize a Boolean function $F(x)$ over input x. The circuit has n primary inputs and m primary outputs. In the application presented below, it is provided by the CED code, where an erroneous output is indicated by one extra output signal.

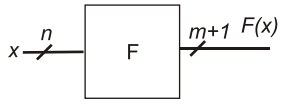

Figure 1. A circuit F with n inputs and m+1 outputs

Denote $F_{flt}(x)$ the Boolean function characterizing the circuit with a given fault. The question whether the fault can be detected is answered by the predicate

$$\exists x,\ F(x) \neq F_{flt}(x) \qquad (1)$$

This is understood as a circuit, see Figure 2. The fault-free and faulty circuits provide $F(x)$ and $F_{flt}(x)$, respectively. The predicate itself is also expressed as a circuit called the *miter* [3].

The characteristic function of the entire circuit is then constructed in Conjunctive Normal Form (CNF) and its satisfiability is solved. If the instance is unsatisfiable, the fault cannot be tested. If it is satisfiable, all solutions are input vectors testing the fault.

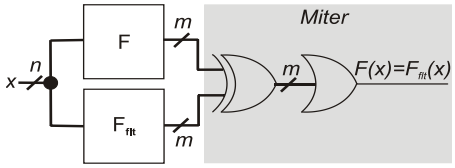

Figure 2. Circuit description of the ATPG SAT instance

For details on the SAT-based ATPGs, see [7], [8], [9].

B. General Predicates

Let x, $F(x)$ and $F_{flt}(x)$ have the same meaning as above. Let

$$\exists x,\ G(x, F(x), F_{flt}(x)) \qquad (2)$$

be any Boolean predicate over x, $F(x)$ and $F_{flt}(x)$. Then G can also be understood as a circuit, see Figure 3. As it has the same role as in ATPG or model checking, we call it the *generalized miter*. Its characteristic function can be constructed as in the ATPG case, and the SAT instance is solved.

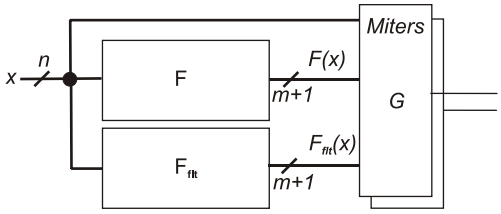

Figure 3. The generalized miter for predicate G

A universally quantified predicate

$$\forall x,\ H(x, F(x), F_{flt}(x)) \qquad (3)$$

can simply be converted to

$$\neg\exists x,\ \neg H(x, F(x), F_{flt}(x))) \qquad (4)$$

The construction of $\neg H$ might seem difficult. When seen as a circuit, however, it suffices to add an inverter. This causes one more variable and two clauses in the SAT instance, which is tolerable.

The predicate can be transformed to CNF by other methods as well; the above case illustrates the advantage of seeing it as a circuit.

The dependency of the general predicate on X is useful in situations where not every input vector is admissible. Let A be the set of admissible input vectors and $a(x)$ the predicate characterizing the set. Then

$$\exists x \in A,\ G(F(x), F_{flt}(x)) \qquad (5)$$

becomes

$$\exists x,\ a(x) \wedge G(F(x), F_{flt}(x)) \qquad (6)$$

This feature achieves the same effect as the input encoder in [5]. In the case solved there, code generator and detector for the codes in question are comparable in complexity. For other problems, however, to *produce* a vector may be more difficult than to *check* that vector.

III. The Analyzed Architecture

A. The Structure of the Dependable Block

The CED strategy proposed in [1], [14] is used in this paper to illustrate principles of the proposed SAT-based predicate evaluation and for the experimental evaluation.

The digital circuit D to be secured by a CED code is supplemented with a predictor P and a checker E, see Figure 4. The predictor can be understood as a copy of the functional circuit together with an encoder. The encoder transforms the vector on the primary outputs of the circuit into the redundancy bits of a selected error detection code. The primary outputs (POs) of the circuit to be secured and the predictor outputs form the code-word whose correctness is verified by the checker.

Any fault in the functional logic D either does not alter the output for a given input vector, or should be detected by the checker. Faults in the predictor and checker either do not affect the operation, or cause false alarms. This

architecture can be apprehended as a kind of modification of the well-known *duplex scheme* [14], [16].

For the purpose of this paper, *single parity* is used as the error detection code. Thus, the predictor is constructed as a copy of the original circuit supplemented with a XOR tree at its outputs, $k = 1$ in Figure 4.

The single parity code offers a low area overhead, however its error detection capabilities are limited. Therefore, the fault coverage can also be lower than in the case of the duplex system, and must be analyzed.

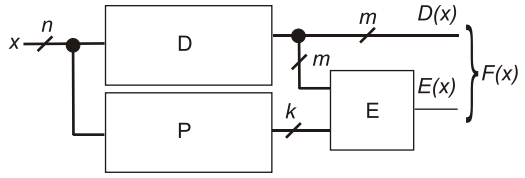

Figure 4. Basic concurrent error detection (CED) scheme

B. Fault Classification and Dependability Parameters

There are three basic dependability parameters in the field of CED (Concurrent Error Detection) [1], [2]:

- ***Fault security (FS)*** - probability that the erroneous outputs produced for a modeled fault do not belong to the output code-words.
- ***Self-testing property (ST)*** - probability that an input vector occurring during normal operation produces an output vector which does not belong to the code when a modeled fault occurs.
- ***Totally self-checking (TSC)*** - The FS and ST parameters of the circuit are equal to 100%. Totally Self-Checking property offers the highest level of protection.

The faults in the secured block cannot be classified only as detectable or undetectable, as for a common circuit. Their detectability by the checker must also be evaluated [4].

To compute these parameters, an approach based on a *fault classification* was presented in [4], [15]. The faults are classified into four groups (A, B, C and D) based on their observability on primary outputs of the circuit and detectability by the checker.

- ***Class A*** – These faults do not affect the circuit POs for any allowed input vector. This is the class of redundant (undetectable) faults. They have no impact to the FS property, but circuits with these faults cannot be ST.
- ***Class B*** – These faults are detectable by at least one input vector and do not produce an incorrect code-word (a valid code-word, but incorrect) for other input vectors. They have no negative impact on the FS and ST properties, since if such a fault occurs, it is detected by the checker.
- ***Class C*** – The faults that produce an incorrect codeword for at least one input vector and cannot be detected by any input vector. This is the class of faults, that can never be detected by the checker and that produce an erroneous output. The circuit with these faults is neither FS nor ST.
- ***Class D*** – these faults cause at least one detectable and one undetectable error on the POs. They are detectable, but also may produce an incorrect output, which is not detected by the checker. They do not satisfy the FS property.

The FS property can be computed from the number of faults in these classes as:

$$FS = (A+B) / (A+B+C+D) \cdot 100\ [\%] \qquad (7)$$

The ST property is computed in similar way as:

$$ST = (B+D) / (A+B+C+D) \cdot 100\ [\%], \qquad (8)$$

where A, B, C, and D are the numbers of faults in the respective classes.

IV. SAT-Based Fault Classification Technique

To apply the SAT-based classification on the above outlined architecture, we must characterize the classes by binary predicates and apply the general scheme form Figure 3.

A. Predicates

To compute the dependability parameters of the given architecture, each fault must be classified into one of the classes A, B, C, and D. Four classes need at least two binary predicates to distinguish. In this case, they are easy to derive from the specifications. In principle, the classes are defined by the ability of the fault to cause a detected or an undetected error, which can be formalized as follows:

- $J(x)$ is true iff the input vector x gives an erroneous output $D(x)$ of the faulty circuit and the error is detected ($E(x)$ is true.)
- $K(x)$ is true iff the input vector x gives an erroneous output $D(x)$ of the faulty circuit and the error is not detected ($E(x)$ is false.)

Then the given fault belongs to

- the class A, iff $\neg\exists x, J(x) \wedge \neg\exists x, K(x)$
- the class B, iff $\exists x, J(x) \wedge \neg\exists x, K(x)$
- the class C, iff $\neg\exists x, J(x) \wedge \exists x, K(x)$
- the class D, iff $\exists x, J(x) \wedge \exists x, K(x)$

Hence, two SAT instances must be solved to classify a fault.

B. Generalized Miters

To construct a miter for the J and K predicates, we have to apply the general process leading from the circuit in Figure 1. to the circuit in Figure 3. on the discussed

architecture. The output $F(x)$ is in our case decomposed into $D(x)$ and $E(x)$, giving the circuit in Figure 5.

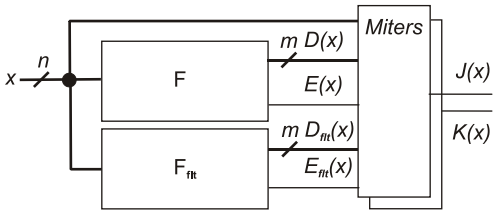

Figure 5. The general circuit for J and K evaluation

Bringing in the internal structure of F and F_{flt} from Figure 4. , we obtain the circuit in Figure 6.

The actual predicates apply to all input vectors x, therefore x does not enter into the miter circuits. Furthermore, we are interested in faults in the secured circuit D only, not in the predictor or checker. Therefore, we can omit $E_{flt}(x)$ from the miters and, therefore, P_{flt} and E_{flt} from the circuit. The final optimized circuit is in Figure 7.

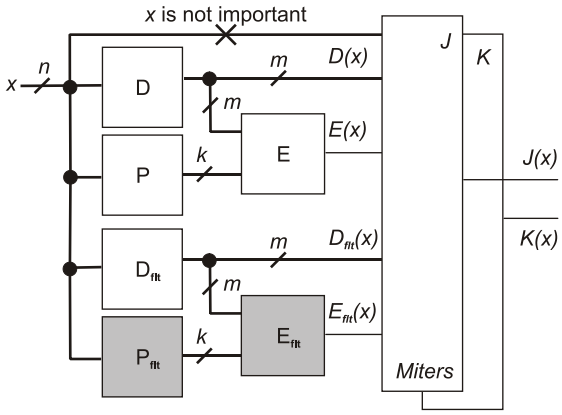

Figure 6. The unoptimized circuit for J and K

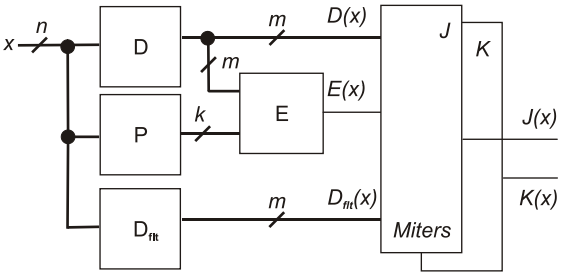

Figure 7. The optimized circuit for J and K

Using $D(x)$, $D_{flt}(x)$ and $E(x)$, we can implement the miters as

$$J(x) \equiv D(x) \oplus D_{flt}(x) \wedge E(x)$$
$$K(x) \equiv D(x) \oplus D_{flt}(x) \wedge \neg E(x) \qquad (9)$$

V. Experimental Technology Comparison

Using the above described framework, we compared robustness of a set of benchmarks, implemented either structurally (as a network of gates) with S@ faults, or implemented as a set of Look Up Tables (LUTs), considering Single Event Upset (SEU) in the LUT configuration memory as the primary fault mechanism.

The experiments have been performed on 65 ISCAS'85 [17], ISCAS'89 [18], ITC'99 [19] and LGSynth [20] benchmark circuits.

For the S@ faults, the original structural description was used. The fault lists were generated by Atalanta [21] and were free from dominated faults.

The LUT implementations were synthesized by ABC [22] using the command sequence *strash; dch; if; lutpack* as recommended by the authors.

A. Measurements and Metrics

The gate implementation and the LUT implementation of a circuit have different number of possible faults. To compare them in a practically relevant manner, we decided to count *points of vulnerability,* that is, the number of faults which can cause dysfunction of the circuit. The coefficients *FS* and *ST,* which indicate distance to the Totally Self Checking goal, are of minor importance here. The metrics used were *Not Fail Safe*

$$NFS = C+D$$

and *Not Self-Testing*

$$NST = C.$$

B. Measured Numbers of Faults

TABLE II. shows the number of faults classified by the above described method. The statistical properties are summarized in the following TABLE I. , using standard correlation and least square linear regression.

TABLE I. Statistical Properties of Fault Numbers

Quantity	Correlation	Lin. regression
Total faults	0.894	2.0
A	0.180	2.2
B	0.892	1.8
NST=C	0.934	1.77
D	0.947	1.15
NFS	0.949	0.73

It is apparent that the values, with the exception of Class A fault number, are correlated. The values *NFS, NST,* which give the number of points of vulnerability, are most tightly correlated. From the correlation it follows that the dependability, or, more precisely, the ability to become dependable using the MDS architecture, does not depend on architecture and fault model. Rather, it is a property of the circuit itself.

From the coefficient of total fault number, it would seem that the LUT technology has twice the number of potential faults. From the coefficient of the A Class faults, it would further seem that many of them are caused by redundancy. This comparison, however, is influenced by the construction of the fault list for gates. A single fault there can represent

more than one dominated fault and hence more than one point of vulnerability.

VI. Conclusions

A method for proving arbitrary predicates quantified over input vector of a combinational circuit has been presented. The method combines elements from SAT ATPG and SAT-based property checking. The Modified Duplex System architecture, which requires classification into four classes, has been selected for demonstration of the method.

A set of benchmark circuits was constructed using the MDS redundancy architecture. The circuits were implemented both in gates and LUTs. Their self-checking characteristics were evaluated by the described method under the stuck-at and single event upset fault models, respectively. The characteristics were found to be correlated, which suggests that the ability to become dependable under the MDS scheme is an intrinsic property of the circuit itself.

References

[1] S. Mitra, J. E. McCluskey, "Which concurrent error detection scheme to choose?", In: IEEE International Test Conference, 2000, pp. 985-994.

[2] D.K. Pradhan, "Fault-Tolerant Computer System Design," In: Prentice-Hall, Inc., New Jersey, 1996.

[3] A. Czutro et al., TIGUAN: Thread-parallel Integrated test pattern Generator Utilising satisfiability Analysis. Proc. Int. VLSI Design Conference, 2009

[4] G. Fey and R. Drechsler, "A Basis for Formal Robustness Checking," Proc. 9th Int. Symp. Quality Electronic Design (ISQED'09), March 2008, pp. 784-789.

[5] M. Hunger, S. Hellebrand: Verification and Analysis of Self-Checking Properties through ATPG; IEEE International On-Line Testing Symposium 2008 (IOLTS'2008), Rhodos, Greece, July, 2008, pp. 25-30

[6] M. Hunger, S. Hellebrand, A. Czutro, I. Polian, B. Becker: ATPG-Based Grading of Strong Fault-Secureness. Proc. 15th IOLTS, Sesimbra-Lisbon, June 24-26, 2009

[7] T. Larrabee, "Test Pattern Generation Using Boolean Satisfiability," IEEE Transactions on Computer-Aided Design, 1992, pp. 4-15.

[8] Drechsler, S. Eggersglüß, G. Fey, D. Tille, "Test Pattern Generation using Boolean Proof Engines," Publisher Springer Netherlands, ISBN 978-90-481-2360-5, 2009, XII, p. 192.

[9] P. Shawhney, G. Ganesh, A. K. Bhattacharjee: Automatic Construction of Runtime Monitors for FPGA based Designs. IEEE Int. Symposium on Electronic System Design, pp. 164-169, 201.

[10] G. Fey, A. Sulflow, S. Frehse, and R. Drechsler, "Effective robustness analysis using bounded model checking techniques," IEEE Trans. on CAD, Vol. 30, No. 8,Aug. 2011, pp.1239-1252.

[11] A. Biere and W. Kun, "SAT and ATPG: Boolean engines for formal hardware verification," In: International Conference on Computer Aided Design (ICCAD '02), 2002, pp. 782—785.

[12] B.R. Prasad, P. Chong, K. Keutzer, „Why is ATPG easy?", In Proc. of the 36th Annual ACM/IEEE Design Automation Conference, New Orleans, USA, June 21-25, 1999, pp. 22-28.

[13] P. Chong and M. Prasad, "Satisfiability for ATPG: Is it Easy?," 1998.

[14] S. Mitra and E. J. McCluskey, "Diversity Techniques for Concurrent Error Detection," In: Proc. of IEEE 2nd International Symposium on Quality Electronic Design, 2001, pp. 249-250.

[15] R. Dobiáš, P. Kubalík, H. Kubátová, "Dependability computations for fault-tolerant system based on FPGA," In Proc. of 12th IEEE International Conference on Electronics, Circuits and Systems, 2005, pp. 1-4.

[16] P. Kubalík, H. Kubátová, "Dependable design technique for system-on-chip," Journal of Systems Architecture. 2008, vol. 2008, no. 54, ISSN 1383-7621, pp. 452-464.

[17] F. Brglez, H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortan," In Proc. of International Symposium on Circuits and Systems, pp. 663-698, 1985.

[18] F. Brglez, D. Bryan, K. Kozminski, "Combinational Profiles of Sequential Bench-mark Circuits," Proc. of International Symposium of Circuits and Systems, pp. 1929-1934, 1989.

[19] F. Corno, R. Sonza, M. Squillero, "RT-Level ITC 99 Benchmarks and First ATPG Results," IEEE Design & Test of Computers, July-August 2000, pp. 44-53.

[20] K. McElvain, "LGSynth93 Benchmark Set: Version 4.0," Mentor Graphics, May 1993.

[21] H.K. Lee and D.S. Ha, "Atalanta: an Efficient ATPG for Combinational Circuits,", Technical Report, 93-12, Dep't of Electrical Eng., Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1993.

[22] Berkeley Logic Synthesis and Verification Group, "ABC: A System for Sequential Synthesis and Verification", http://www.eecs.berkeley.edu/~alanmi/abc/

TABLE II. FAULT NUMBERS IN TWO IMPLENENTATIONS

	Gates, S@						LUTs, SEU					
Circuit	**FAULTS**	***A***	***B***	***NTS=C***	***D***	***NFS***	**FAULTS**	***A***	***B***	***NTS=C***	***D***	***NFS***
5xp1	422	0	353	7	62	69	538	83	431	4	20	24
9symml	446	0	446	0	0	0	1036	275	761	0	0	0
9sym	713	0	713	0	0	0	1440	409	1031	0	0	0
al2	400	0	338	8	54	62	736	0	692	0	44	44
alcom	319	0	291	16	12	28	560	0	556	0	4	4
alu1	109	0	91	0	18	18	120	0	120	0	0	0
alu2	1132	117	383	9	623	632	1864	680	555	9	620	629
amd	842	0	632	23	187	210	2084	371	1546	16	151	167
b1	37	6	25	0	6	6	16	0	16	0	0	0
b9	366	2	205	45	114	159	560	6	412	64	78	142
br1	341	0	230	17	94	111	792	120	558	57	57	114
br2	296	0	185	20	91	111	564	58	406	18	82	100
c1355	882	0	704	0	178	178	1088	2	944	14	128	142
c17	22	0	12	0	10	10	32	0	32	0	0	0

Circuit	Gates, S@						LUTs, SEU					
	FAULTS	***A***	***B***	***NTS=C***	***D***	***NFS***	**FAULTS**	***A***	***B***	***NTS=C***	***D***	***NFS***
c1908	971	5	437	0	529	529	1252	118	614	6	514	520
c432	553	8	82	0	463	463	1088	128	222	17	721	738
c499	882	0	704	0	178	178	1088	2	944	14	128	142
c8	636	56	489	0	91	91	454	21	401	0	32	32
cc	219	13	172	4	30	34	328	24	296	0	8	8
chkn	918	0	806	0	112	112	1924	269	1607	0	48	48
cht	669	39	604	0	26	26	588	0	584	0	4	4
clip	1108	27	964	3	114	117	1068	205	793	8	62	70
clpl	38	0	14	0	24	24	84	0	52	0	32	32
cm138a	74	0	68	6	0	6	96	0	96	0	0	0
cm150a	245	23	222	0	0	0	160	8	152	0	0	0
cm152a	56	0	56	0	0	0	72	4	68	0	0	0
cm162a	166	6	113	0	47	47	172	18	130	0	24	24
cm163a	159	4	115	0	40	40	160	4	156	0	0	0
cm42a	76	0	68	2	6	8	160	0	160	0	0	0
cm82a	60	0	17	0	43	43	32	0	24	0	8	8
cm85a	131	0	107	0	24	24	148	0	148	0	0	0
cmb	141	6	92	0	43	43	228	22	182	0	24	24
con1	51	0	43	0	8	8	68	2	66	0	0	0
count	379	0	265	4	110	114	520	24	432	0	64	64
cu	164	6	100	27	31	58	208	11	159	28	10	38
dc1	120	0	85	7	28	35	112	0	112	0	0	0
dc2	255	0	201	3	51	54	422	49	361	4	8	12
decod	130	0	122	8	0	8	192	0	192	0	0	0
dist	796	0	712	4	80	84	2240	552	1686	0	2	2
duke2	1303	1	609	132	561	693	2476	482	914	264	816	1080
ex5	940	0	697	29	214	243	2768	917	1662	18	171	189
ex7	297	0	229	0	68	68	412	26	362	0	24	24
f51m	459	0	402	0	57	57	570	100	466	0	4	4
frg1	1049	0	1041	0	8	8	1248	118	1130	0	0	0
gary	1059	0	850	17	192	209	2508	445	1936	43	84	127
i1	129	1	89	2	37	39	162	0	138	0	24	24
ibm	492	0	354	0	138	138	1080	36	933	0	111	111
in0	1059	0	850	17	192	209	2416	415	1813	59	129	188
in2	1002	0	757	56	189	245	2060	289	1512	109	150	259
in4	1013	0	696	22	295	317	1928	230	1431	20	247	267
in5	802	0	549	16	237	253	1972	221	1539	17	195	212
in6	767	0	548	26	193	219	1384	159	1077	20	128	148
in7	311	0	167	12	132	144	668	75	441	57	95	152
jbp	1132	0	833	55	244	299	2222	168	1826	74	154	228
lal	414	0	295	11	108	119	374	15	279	0	80	80
ldd	278	9	164	45	60	105	404	71	238	42	53	95
luc	621	0	430	55	136	191	1192	182	819	108	83	191
m1	195	0	144	8	43	51	302	23	274	2	3	5
m2	543	0	442	7	94	101	956	170	729	16	41	57
m3	630	0	534	9	87	96	1546	340	1176	4	26	30
m4	973	0	844	8	121	129	2752	647	2044	14	47	61
majority	39	0	39	0	0	0	20	0	20	0	0	0
max46	380	0	380	0	0	0	744	110	634	0	0	0
max512	891	0	792	0	99	99	2788	716	2042	4	26	30
misex1	161	0	105	13	43	56	248	21	205	13	9	22
misex2	294	0	237	0	57	57	496	28	448	0	20	20
mlp4	694	0	590	12	92	104	1816	366	1437	0	13	13