SUPERB: Simulator Utilizing Parallel Evaluation of Resistive Bridges*

Piet Engelke[1] Bettina Braitling[1] Ilia Polian[1] Michel Renovell[2] Bernd Becker[1]

[1]Albert-Ludwigs-University
Georges-Köhler-Allee 51
79110 Freiburg im Breisgau, Germany
{engelke|braitlin|polian|becker}@informatik.uni-freiburg.de

[2]LIRMM – UMII
161 Rue Ada
34392 Montpellier, France
renovell@lirmm.fr

Abstract

A high-performance resistive bridging fault simulator SUPERB (Simulator Utilizing Parallel Evaluation of Resistive Bridges) is proposed. It is based on fault sectioning in combination with parallel-pattern or parallel-fault multiple-stuck-at simulation. It outperforms a conventional interval-based resistive bridging fault simulator by 60X to 120X while delivering identical results. Further competing tools are outperformed by several orders of magnitude.

Keywords: Resistive bridging faults, bridging fault simulation, PPSFP, SPPFP, fault mapping

1 Introduction

Conventional bridging fault models assume the short defect resistance of zero [1, 2, 3, 4, 5, 6]. Resistive bridging fault models take other resistance values into account [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]. State-of-the-art resistive bridging fault simulators are able to determine the detection status for all defect resistances simultaneously [13, 17]. This is achieved by considering intervals of resistances in which the defect influences the logic function of the circuit. However, successful acceleration techniques used in stuck-at simulators, such as utilizing multiple bits of a machine word, are not applicable to interval-based simulation.

In this paper, we propose the high-performance resistive bridging fault simulator SUPERB (Simulator Utilizing Parallel Evaluation of Resistive Bridges). It uses the sectioning-based approach [20] in combination with a parallel multiple-stuck-at fault engine. For a bridging fault between circuit nodes a and b, a section $[R_1, R_2]$ denotes a range of bridging defect resistances for which the logic behavior of the circuit is identical. In particular, if a bridging defect between nodes a and b with a resistance R_{sh} from section $[R_1, R_2]$ is detected by a test vector, then all bridging defects between a and b with resistances between R_1 and R_2 will be detected by the same vector.

*Parts of this work are supported by the German Research Foundation under grant BE 1176/14-1.

Given a section and an input vector, the faulty behavior of the circuit can be characterized by a multiple-stuck-at fault. Since the resistive bridging fault model considers pattern-dependency [16, 17], the faulty behavior of the same circuit under a different input vector may be described by a different multiple-stuck-at fault. We store the mapping of resistance sections to multiple-stuck-at faults considering the pattern dependency in a hash table which is calculated before actual simulation begins. It is possible to generate the hash table using SPICE simulations [13] or electrical equations [7, 8, 21]. Whenever the simulation arrives at a fault site, the adequate multiple-stuck-at fault is looked up in the hash table and handed to a multiple-stuck-at fault simulation engine which utilizes the usual speed-ups. We reiterate that there is no one-to-one mapping between a resistive bridging fault, even restricted to a section, and a multiple-stuck-at fault, because of pattern dependency.

Considering surrogate faults, including multiple-stuck-at faults and multiple single-stuck-at faults, to represent complex resistive bridging defects has been proposed in the past. Maeda and Kinoshita [22] utilized locally exhaustive testing of the bridge site. n-detection [23, 24] and its extensions [25, 19] demanded application of multiple tests for one single-stuck-at fault. The Unified Fault Model [26] considered all possible multiple-stuck-at faults at the nodes succeeding the bridge site. In contrast to these approaches, we retain the modeling accuracy of the original resistive bridging fault model. The fault coverages returned by SUPERB are not over- or underapproximations, they correspond exactly to the numbers generated by an interval-based resistive bridging fault simulator.

The remainder of the paper is organized as follows. In Section 2 we describe the fault sectioning approach and discuss its differences from the interval-based approach in context of simulation. In Section 3 we explain the simulator SUPERB which utilizes the sectioning technique to leverage the accelerations known for stuck-at fault simulation. In Section 4, experimental results are reported. Section 5 concludes the paper.

2 Fault Sectioning

In this section, the fault sectioning technique from [20] is formalized within our framework and illustrated by an example.

A *bridging fault* is given by two bridged nodes a and b. A *bridging defect* is given by two nodes a and b and the defect resistance R_{sh}. A bridging fault corresponds to an infinite number of bridging defects having different resistances.

For a bridging fault or a bridging defect between a and b, the two gates which drive a and b are called *preceding gates* and the gates driven by either a or b (including all their fanout branches) are called *succeeding gates*. If an input vector implies opposite logic values on nodes a and b, then a bridging defect will result in intermediate voltages V_a and V_b on the respective lines, i.e., $0\text{V} < V_a, V_b < V_{DD}$. For every input of each succeeding gate, a single *logic threshold* is assumed such that every voltage below this threshold is interpreted as logic 0 and every voltage above is interpreted as logic 1.

The resistance of a bridging defect which induces a voltage corresponding to a logic threshold of one of the inputs driven by the node is called *critical resistance*. An *analogue detection interval* (ADI) is the range of resistances for all bridging defects which are detected by a test vector, i.e., which produce a wrong logic value on at least one primary output (or other observable point). An ADI typically has the shape $[0, R]$ for some value of R but it can also be a union of disjoint sub-intervals [16]. The boundaries of the ADIs are always critical resistances. The number of different critical resistances depends on the fault site. The ADI is calculated at the bridge site based on an electrical analysis and propagated to the observable points of the circuit.

The interval $[R_1, R_2]$ is called a *section* if R_1 and R_2 are critical resistances and there is no critical resistance R_3 such that $R_1 < R_3 < R_2$. The number of sections is given by the number of different critical resistances. Let $R_1, R_2, \ldots, R_m$ be all the critical resistances for the bridging fault between nodes a and b, sorted in ascending order. The detection status of a bridging fault can be represented by the detection statuses of the sections $[0, R_1]$, $[R_1, R_2]$, $\ldots$, $[R_{m-1}, R_m]$, $[R_m, \infty]$. The detection status is uniform for every bridge resistance within a section, i.e., either all bridging defects with a resistance from a section are detected by a test vector or none is detected. If the detection status of all the sections is known, the ADI can be constructed as the union of all detected sections. This approach results in the same ADI as the technique based on interval propagation.

For illustration, we consider the circuit from Figure 1 [17] which is a part of a larger circuit shown in Figure 3. Solid curves in Figure 2 show possible voltage characteristics as a function of bridge resistance R_{sh} when logic values 0011 are applied to gates A and B preceding the bridge and dashed curves show the characteristics when values 0111 are applied. Logic thresholds of the succeeding gates are Th_C, Th_D and Th_E. The fault is not excited and no detection is possible if nodes a and b have the same logic value. For the sake of simplicity, we assume that the only logic value assignments to the inputs of the preceding gates leading to a fault excitation are 0011 and 0111. (Assignment 1011 can be assumed to have effect identical to 0111.) Critical resistances are R_C, R_E (for value assignment 0011), R'_C, R'_D and R'_E (for value assignment 0111). Note that there is no critical resistance R_D. Sections are $[0, R'_D]$, $[R'_D, R_C]$, $[R_C, R'_E]$, $[R'_E, R'_C]$, $[R'_C, R_E]$ and $[R_E, \infty]$. Note that the fault-free value is always assumed in the last section such as $[R_E, \infty]$ in this example and this section does not need to be considered explicitly.

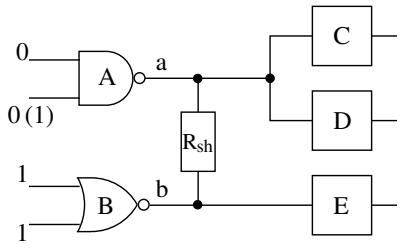

Figure 1: Example circuit: bridge site

Figure 3 shows how the fault effect is propagated for input values 0111 using the interval-based technique. The interval assigned to a node is shown over the node. For instance, interval $[0, R'_C]0/1$ means that the node assumes logic value 0 if the bridge resistance is between 0Ω and R'_C, and logic value 1 otherwise. Similarly, interval $[0, R'_E]1/0$ means that the node assumes logic value 1 if the bridge resistance is between 0Ω and R'_E, and logic value 0 otherwise. The intervals are propagated through the circuit taking the types of the gates and the values on the side inputs into account. For instance, no interval is propagated through the AND gate D because its side input has controlling logic value 0. The interval is complemented by inverting gates such as E. If two intervals reconverge, i.e., arrive at two inputs of a single gate such as XOR gate G preceding the output, intervals not starting at 0Ω or even having 'holes' may be produced. In the example, the fault-free logic value is 1, so the fault is detected by the applied test vector if R_{sh} is between R'_E and R'_C.

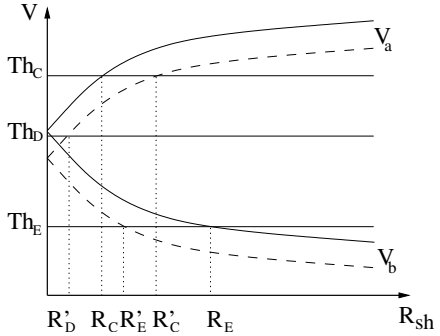

Figure 2: R_{sh}-V-diagram

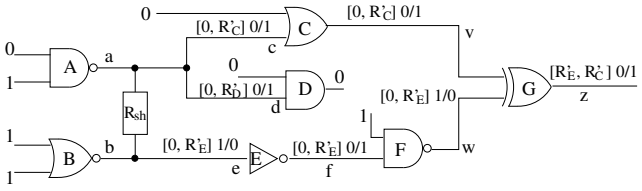

Figure 3: Example circuit: interval-based propagation

Circuit node	Fault-free value	Valued assumed in section $[0, R'_D]$	$[R'_D, R_C]$	$[R_C, R'_E]$	$[R'_E, R'_C]$	$[R'_C, R_E]$
c	1	0	0	0	0	1
d	1	0	1	1	1	1
e	0	1	1	1	0	0
f	1	0	0	0	1	1
v	1	0	0	0	0	1
w	0	1	1	1	0	0
z	1	1	1	1	0	1

Table 1: Section-based propagation in circuit from Figure 3

The logic values assumed by a node for individual sections are shown in Table 1. For example, the second input c of gate C is assumes logic value 0 for bridge resistances between 0Ω and R'_C, i.e., in sections $[0, R'_D]$, $[R'_D, R_C]$, $[R_C, R'_E]$, $[R'_E, R'_C]$, and logic value 1 in the remaining section $[R'_C, R_E]$ (section $[R_E, \infty]$ is not shown as explained above). The values for individual sections can be propagated through the circuit based on the logic functions of the gates. It is obvious that, if the sections are known, five bits representing the logic values for sections contain exactly the same information as the interval-based representation. In particular, the values at the output of the circuit are (1, 1, 1, 0, 1), i.e., only section $[R'_E, R'_C]$ assumes logic value 0 and all others assume logic value 1) corresponds to ADI $[R'_E, R'_C]0/1$ generated by the interval propagation.

Simulating a section under a given input vector (only the values assigned to the inputs of the preceding gates matter indeed) can be represented as simulating a multiple-stuck-at fault. In our example, simulating section $[0, R'_D]$ under input 0111 corresponds to simulating stuck-at faults c stuck-at-0, d stuck-at-0, and e stuck-at-1 injected simultaneously. Stuck-at faults corresponding to all sections and inputs are shown in Table 2.

The union of all ADIs on all outputs for all vectors is called *covered ADI* or C-ADI. C-ADI includes all the bridge resistances for which the fault has been detected by at least one test vector. C-ADI for the exhaustive test set is called *global ADI* or G-ADI, it represents all the resistances for which the fault could be detected. G-ADI can be calculated using ATPG techniques [27]. The *global fault coverage* G-FC is defined as the probability that a detectable fault f is detected:

$$G\text{-FC}(f) = 100\% \cdot \left(\int_{C\text{-ADI}} \rho(r)dr \right) / \left(\int_{G\text{-ADI}} \rho(r)dr \right),$$

where $\rho(r)$ is the probability density function of the short resistance r which can be derived from manufacturing data.

It is possible to approximate G-ADI by $[0, R_m]$, where R_m is the largest critical resistance [13]. R_m is much easier to calculate than G-ADI. The fault coverage which uses this approximation is called E-FC in [17]:

$$E\text{-FC}(f) = 100\% \cdot \left(\int_{C\text{-ADI}} \rho(r)dr \right) / \left(\int_{0}^{R_m} \rho(r)dr \right).$$

For multiple faults, average G-FC and E-FC are calculated. Redundant faults, i.e., faults with an empty G-ADI, are excluded.

3 SUPERB

Simulator SUPERB works in two stages. First, it reads in the list of resistive bridging faults and gate and process technology parameters, determines sections, and generates, for each section, a hash table which contains the information on equivalent multiple-stuck-at faults depending on logic values on preceding gates. Second, every section is fault simulated by a multiple-stuck-at engine. A section is mapped to a multiple-stuck-at fault taking into account the information stored in the hash table associated with the section. Then, either parallel-pattern or parallel-fault multiple-stuck-at simulation is performed. The subsequent sections provide details on the two stages of the simulation procedure.

3.1 Hash table generation

There is one hash table for every section of every considered resistive bridging fault. Let the fault f between nodes driven by gates a and b have a section $[R_L, R_U]$. The hash table for section $[R_L, R_U]$ of fault f contains all the logic value assignments to the inputs of gates a and b under which a resistive defect with a resistance between R_L and R_U has any faulty effect. Every such input value assignment serves as a key to look up the equivalent multiple-stuck-at faults. In the example from Table 2, the hash table for section $[0, R'_D]$

Input	Section	Stuck-at faults
0111	$[0, R'_D]$	c s-a-0, d s-a-0, e s-a-1
0111	$[R'_D, R_C]$	c s-a-0, e s-a-1
0111	$[R_C, R'_E]$	c s-a-0, e s-a-1
0111	$[R'_E, R'_C]$	c s-a-0
0111	$[R'_C, R_E]$	–
0111	$[R_E, \infty]$	–
0011	$[0, R'_D]$	c s-a-0, e s-a-1
0011	$[R'_D, R_C]$	c s-a-0, e s-a-1
0011	$[R_C, R'_E]$	e s-a-1
0011	$[R'_E, R'_C]$	e s-a-1
0011	$[R'_C, R_E]$	e s-a-1
0011	$[R_E, \infty]$	–

Table 2: Multiple stuck-at faults corresponding to resistive bridging faults

would have two entries: ($0111 \mapsto \{c$ s-a-0, d s-a-0, e s-a-1$\}$) and ($0011 \mapsto \{c$ s-a-0, e s-a-1$\}$). The hash table for section $[R'_C, R_E]$ would have one entry ($0011 \mapsto \{e$ s-a-1$\}$).

The identification of section boundaries (i.e., critical resistances) and the calculation of equivalent multiple stuck-at faults is done using the electrical equations from [7, 8]. It would also be possible to use SPICE simulations instead such as done in [13]. The keys, i.e., the input value assignments, are represented as 32-bit unsigned integers. This is valid because the length of a key cannot exceed $2 \cdot I_{\max}$ where $I_{\max}$ is the maximal number of inputs of a logic gate in the library and $I_{\max}$ is not supposed to exceed 16.

The maximal theoretical number of entries in a hash table is $2^{2 \cdot I_{\max}}$ but in practice the hash tables turn out to be quite small. The number of hash tables is the number of faults in the fault list multiplied by the number of sections per fault. The latter is typically a one-digit number, so the memory overhead for storing the hash tables is limited.

3.2 Fault simulation

SUPERB supports both parallel-pattern single-fault processing (PPSFP) and single-pattern parallel-fault processing (SPPFP). In PPSFP simulation, one section is fault-simulated for 32 test vectors $t_1, \ldots, t_{32}$ simultaneously. For this purpose, every node j is assigned a 32-bit string B_j represented using a machine word. The ith position of B_j corresponds to the logic value of node j under vector t_i with fault injected. If node j is an input of a gate succeeding the bridged lines, the (multiple-stuck-at) fault injection is done using two 32-bit masks: AND mask A_j and OR mask O_j. The ith position of A_j is set to 0 if a stuck-at-0 fault is injected at node j under test vector t_i and to 1 otherwise. The ith position of O_j is set to 1 if a stuck-at-1 fault is injected at node j under test vector t_i and to 0 otherwise.

The circuit is processed in topological order. A logic gate driving node j is simulated by applying its bit-wise logic function to the bit-strings of its inputs. In case of gates succeeding the bridged lines, a bit-wise AND operation with A_j and a bit-wise OR operation with O_j is performed first. For example, if a NOR3 gate succeeding the bridge drives node n and the gate's input bit-strings are B_k, B_l and B_m, then B_n is obtained as

$$B_n = \neg\left((B_k \wedge A_k \vee O_k) \vee (B_l \wedge A_l \vee O_l) \vee (B_m \wedge A_m \vee O_m)\right)$$

where $\neg$, $\vee$ and $\wedge$ represent bit-wise NOT, OR and AND operations, respectively.

The generation of masks A_j and O_j is done for all inputs of gates succeeding the bridging fault to which the simulated section belongs, using the hash table of that section. For a bit position i, the input value assignments of the preceding gates are evaluated and the corresponding key of the hash table is looked up. The ith component of A_j is set to 0 if the hash table entry contains a stuck-at-0 fault on node j. The ith component of O_j is set to 1 if the hash table entry contains a stuck-at-1 fault on node j. Note that, in contrast to traditional pattern-parallel simulation, the masks do not always contain identical values. This is because the faults to be injected depend on the input value assignments of the preceding gates.

The authors of [13] also call their technique PPSFP simulation. Their approach is interval-based. They maintain a list of pointers to the intervals, and if an interval to be created already exists, they store only the pointer, avoiding copying the interval. In contrast to SUPERB, they do not employ bit-parallelism for fault effect propagation.

In parallel-fault (SPPFP) simulation, an input vector is simulated under 32 sections, not necessarily belonging to the same fault. The AND and OR masks are set at all the succeeding nodes of the resistive bridging faults to which the 32 sections belong. This is done by looking up the 32 hash tables for the individual sections using the key corresponding to the input value assignment induced by the simulated vector. The processing of individual gates is identical to the parallel-pattern case.

4 Experimental Results

We applied SUPERB to ISCAS 85 and combinational parts of ISCAS 89 benchmark circuits. The fault set consisted of 10,000 randomly selected non-feedback faults, where available. We employed the density function ρ derived from one used in [13]. All measurements were performed on a 2GHz AMD Opteron Linux machine with 4 GB RAM.

Table 3 contains the numbers of bridging faults, the numbers of sections, the fault coverages (FC) and the run times for 1,000 random vectors, 10,000 random vectors and ATPG patterns from [27] for SUPERB in PPSFP and SPPFP configurations and the interval-based simulator from [17]. The number of sections per fault ranges between 1.91 and 5.73, the average number being 3.09. Hence, approximately three sections must be simulated on average to obtain the detection interval for a resistive bridging fault. The maximal number of sections per fault in a circuit was between 4 and 61. The fault coverage is G-FC for all circuits except c6288 for which G-ADI cannot be calculated and E-FC has to be used. We reiterate that identical fault coverage is computed by SUPERB and the interval-based simulator. The ATPG patterns have fault coverage of 100%.

The last row of Table 3 contains the sums of the run times over all circuits. Parallel-pattern simulation outperforms parallel-fault simulation, which is also typically observed for existing stuck-at simulators. The acceleration compared with the interval-based approach is around 62X for 1,000 vectors and 120X for 10,000 vectors (the measurement has been done on the same machine). The construction of the hash tables (the first stage of the fault simulation algorithm) takes between 0.02 and 0.59 seconds per circuit. The

Circuit	Bridges	Sections	1,000 random vectors				10,000 random vectors				ATPG vectors			
			FC	PPSFP	SPPFP	Interval	FC	PPSFP	SPPFP	Interval	Patterns	PPSFP	SPPFP	Interval
c0017	2	8	100.00	0.02	0.03	0.00	100.00	0.02	0.07	0.08	2	0.02	0.02	0.00
c0095	77	441	100.00	0.08	0.42	0.78	100.00	0.53	3.84	8.13	18	0.02	0.03	0.01
c0432	5253	18296	99.78	1.10	5.78	57.46	99.99	3.53	29.80	584.48	732	1.25	8.12	62.16
c0499	8985	16092	99.99	1.01	4.14	108.23	100.00	2.16	15.63	1079.19	54	0.72	0.89	7.47
c0880	10000	36074	98.88	2.55	18.41	108.66	99.54	11.47	116.67	1072.33	745	2.05	13.94	75.67
c1355	10000	32160	99.89	3.38	22.72	217.21	100.00	20.63	187.72	2154.24	178	1.74	4.52	45.94
c1908	10000	30961	99.25	2.63	13.71	281.63	99.98	7.89	64.33	2730.71	267	1.53	3.25	85.44
c2670	10000	28379	95.93	2.56	16.26	174.50	97.68	13.02	110.01	1751.03	366	1.21	4.37	64.19
c3540	10000	28115	99.19	2.55	15.62	231.15	99.91	9.40	81.58	2342.25	489	1.80	7.60	117.38
c5315	10000	28491	99.96	1.53	5.84	209.97	100.00	3.80	26.36	2193.35	384	1.30	4.21	76.16
c6288	10000	33425	91.88	5.66	46.30	634.95	91.89	41.47	446.95	6720.93	n/a	n/a	n/a	n/a
c7552	10000	31592	99.12	2.23	11.16	247.67	99.55	8.97	70.67	2545.09	357	1.42	3.85	94.46
cs00027	2	10	100.00	0.01	0.01	0.02	100.00	0.01	0.02	0.14	2	0.01	0.01	0.00
cs00208	3986	11508	99.56	0.98	7.87	19.26	100.00	5.64	60.38	195.03	124	0.35	1.21	2.52
cs00298	4468	12863	99.98	0.91	5.42	28.10	100.00	4.90	44.63	279.76	125	0.47	1.27	3.60
cs00344	7760	21229	99.97	2.02	15.10	55.42	100.00	14.48	139.52	552.15	206	0.84	3.88	11.40
cs00349	7881	21914	99.97	2.12	16.40	57.86	100.00	15.30	149.83	589.74	197	0.87	3.88	10.94
cs00382	7809	28578	99.96	1.71	10.02	53.26	100.00	7.90	68.08	532.40	255	1.04	3.56	13.69
cs00386	9384	17907	99.82	2.20	17.55	40.92	100.00	10.41	100.96	412.77	77	0.73	1.61	4.49
cs00400	8290	31563	99.96	1.94	11.43	58.62	100.00	9.06	77.34	595.00	245	1.14	3.79	14.27
cs00420	10000	29142	85.20	4.09	39.31	51.48	95.74	24.31	272.00	505.67	317	1.20	7.23	17.43
cs00444	10000	38922	99.98	2.79	19.07	70.42	100.00	14.53	145.12	721.01	312	1.58	7.06	22.15
cs00510	10000	43359	99.98	3.54	27.41	70.74	100.00	23.25	244.55	696.66	320	1.89	10.37	22.88
cs00526	10000	33954	99.44	3.26	23.16	71.90	99.95	14.95	140.48	724.64	372	1.82	8.39	27.70
cs00641	10000	17494	99.61	0.88	4.54	97.33	99.89	2.29	19.98	1001.55	304	0.63	1.84	27.10
cs00713	10000	19871	99.68	1.48	9.79	105.60	99.90	7.85	72.37	1060.36	332	0.89	3.91	32.70
cs00820	10000	39164	96.29	6.00	54.93	65.54	99.61	31.97	357.59	673.51	454	2.87	19.61	30.47
cs00832	10000	40010	96.13	6.28	58.09	65.83	99.28	34.11	372.92	654.15	445	2.85	19.83	30.98
cs00838	10000	29463	69.83	6.34	57.93	57.77	79.84	47.21	476.03	574.19	473	1.62	10.97	30.90
cs00953	10000	44768	97.33	5.63	47.19	85.06	99.75	34.60	348.82	835.06	422	2.71	17.25	37.57
cs01196	10000	33300	96.07	3.65	27.98	71.55	99.46	15.61	149.77	696.53	500	1.87	10.27	37.31
cs01238	10000	34365	96.49	3.90	31.23	72.53	99.58	17.61	168.19	729.29	508	2.03	11.64	39.66
cs01423	10000	28598	99.21	2.42	17.77	86.46	99.77	12.07	114.39	830.52	514	1.50	8.58	57.62
cs01488	10000	19587	99.65	2.02	14.50	62.30	99.97	8.32	73.70	624.18	234	0.98	3.64	16.19
cs01494	10000	19584	99.66	2.13	15.17	63.42	99.95	8.89	78.04	630.46	225	1.04	3.78	15.59
cs05378	10000	28812	98.68	2.20	14.47	156.82	99.80	6.99	60.02	1552.36	824	1.50	8.58	144.12
cs09234	10000	21819	90.92	3.76	29.85	173.96	97.24	16.15	149.64	1730.58	904	1.67	11.21	167.83
cs13207	10000	20354	95.90	2.25	18.43	498.34	98.49	10.82	106.68	4929.44	1228	1.56	12.10	564.55
cs15850	10000	20093	96.76	2.08	17.17	394.83	98.87	8.71	87.27	3956.68	1060	1.54	11.89	428.38
cs35932	10000	27198	100.00	2.79	23.28	519.59	100.00	18.50	189.48	5088.17	516	1.92	14.19	406.20
cs38417	10000	25893	97.73	3.02	27.23	887.68	98.14	17.58	194.05	8957.95	1178	2.65	25.27	1013.94
cs38584	10000	26383	92.42	3.26	28.53	654.05	97.62	15.88	163.38	6853.46	1822	3.84	41.97	1087.09
Σ				110.96	851.22	6968.87		582.79	5778.86	70365.22		58.67	339.59	4948.15

Table 3: Run times (in CPU seconds) in comparison with the interval-based simulator

maximal memory consumption of SUPERB was 33.59 MB (for cs35932 and 10,000 patterns).

Table 4 compares SUPERB with the sectioning-based tool from [20] and the interval-based tool PROBE [13]. For each tool, the number of bridging faults B (not sections), the number of used test vectors V and the reported run time T in seconds are given. Note that the original papers [20, 13] report the run time in minutes. To allow a comparison, we calculate the normalized run time as run time T divided by the product of the number of bridges B and the number of vectors V and report the result (in milliseconds) in columns T/BV.

The final row contains the sums of all run times and the average normalized run times. SUPERB is approximately 115 times faster than the tool from [20] and 37,500 times faster than PROBE [13]. The experiments in [20] were run on a 850 MHz Pentium III, so if we assume that our computer is three times faster the 'actual' speed-up is around 39X. The average number of sections per fault for the circuits in Table 4 was 3.61 for the tool from [20] and 3.11 for SUPERB, which may account for some 15% of the acceleration. The measurement in [13] was done on a Sun SPARC 5 with an unspecified clock frequency which should not have resulted in a slowdown of more than two orders of magnitude. The remaining difference must be attributed to the interval-based simulation (despite the acceleration through the shared pointers) and the implementation of PROBE in tcl/tk.

Circuit	Shinogi et al. [20]				PROBE [13]				SUPERB (PPSFP)			
	B	V	T (s)	T/BV (ms)	B	V	T (s)	T/BV (ms)	B	V	T (s)	T/BV (ms)
c432	n/a	n/a	n/a	n/a	157	10016	780.00	0.49602	5253	10000	3.53	0.00007
c499	n/a	n/a	n/a	n/a	136	10016	900.00	0.66071	8985	10000	2.16	0.00002
c880	1000	10000	18.00	0.00180	949	10016	6960.00	0.73223	10000	10000	11.47	0.00011
c1355	1000	10000	114.00	0.01140	639	10016	12960.00	2.02493	10000	10000	20.63	0.00021
c1908	2000	10000	78.00	0.00390	1662	10016	37680.00	2.26353	10000	10000	7.89	0.00008
c2670	5000	10000	840.00	0.01680	4294	10016	151020.00	3.51138	10000	10000	13.02	0.00013
c3540	5000	10000	720.00	0.01440	4431	10016	202860.00	4.57089	10000	10000	9.40	0.00009
c5315	8000	10000	540.00	0.00675	7121	10016	418500.00	5.86760	10000	10000	3.80	0.00004
c6288	4000	10000	1800.00	0.04500	3216	10016	603240.00	18.72750	10000	10000	41.47	0.00041
c7552	13000	10000	2700.00	0.02077	12106	10016	1194480.00	9.85108	10000	10000	8.97	0.00009
Σ, $\emptyset$			6810.00	0.01510			2629380.00	4.87059			122.34	0.00013

Table 4: Comparison with existing approaches

5 Conclusions

We presented an approach for fast simulation of resistive bridging faults. Sections of bridge resistances are mapped to multiple-stuck-at faults, thus enabling to employ acceleration techniques developed for stuck-at fault simulation. The mapping takes the pattern-dependent circuit behavior into account. While retaining the accuracy of the interval-based simulation, speed-ups of several orders of magnitude are achieved compared to both interval-based and sectioning-based tools proposed before. Possible extensions are support of features present in industrial circuits such as Z values and incorporation of the model for feedback faults from [28].

6 References

[1] K.C.Y. Mei. Bridging and stuck-at faults. *IEEE Trans. on Comp.*, C-23(7):720–727, July 1974.

[2] P. Banerjee and J.A. Abraham. A multivalued algebra for modeling physical failures in mos vlsi circuits. *IEEE Trans. on CAD*, 4(5):312–321, 1985.

[3] S.D. Millman and Sir J.P. Garvey. An accurate bridging fault test pattern generator. In *Int'l Test Conf.*, pages 411–418, 1991.

[4] P.C. Maxwell and R.C. Aitken. Biased voting: A method for simulating CMOS bridging faults in the presence of variable gate logic thresholds. In *Int'l Test Conf.*, pages 63–72, 1993.

[5] F.J. Ferguson and T. Larrabee. Test pattern generation for realistic bridge fault in CMOS ICs. In *Int'l Test Conf.*, pages 492–499, 1991.

[6] J. Rearick and J.H. Patel. Fast and Accurate CMOS Bridging Fault Simulation. In *Int'l Test Conf.*, pages 54–62, 1993.

[7] M. Renovell, P. Huc, and Y. Bertrand. CMOS bridge fault modeling. In *VLSI Test Symp.*, pages 392–397, 1994.

[8] M. Renovell, P. Huc, and Y. Bertrand. The concept of resistance interval: A new parametric model for resistive bridging fault. In *VLSI Test Symp.*, pages 184–189, 1995.

[9] H. Hao and E.J. McCluskey. Resistive shorts within CMOS gates. In *Int'l Test Conf.*, pages 292–301, 1991.

[10] M. Favalli, M. Dalpasso, P. Olivo, and B. Ricco. Analysis of dynamic effects of resistive bridging faults in CMOS and BiCMOS digital ICs. In *Int'l Test Conf.*, pages 865–873, 1993.

[11] H. Vierhaus, W. Meyer, and U. Glaser. CMOS bridges and resistive transistor faults: IDDQ versus delay effects. In *Int'l Test Conf.*, pages 83–91, 1993.

[12] Y. Liao and D.M.H. Walker. Fault coverage analysis for physically-based CMOS bridging faults at different power supply voltages. In *Int'l Test Conf.*, pages 767–775, 1996.

[13] C. Lee and D. M. H. Walker. PROBE: A PPSFP simulator for resistive bridging faults. In *VLSI Test Symp.*, pages 105–110, 2000.

[14] V. Sar-Dessai and D.M.H. Walker. Accurate fault modeling and fault simulation of resistive bridges. In *Int. Symp. Defect and Fault Tolerance in VLSI Systems*, pages 102–107, 1998.

[15] V. Sar-Dessai and D.M.H. Walker. Resistive Bridge Fault Modeling, Simulation and Test Generation. In *Int'l Test Conf.*, pages 596–605, 1999.

[16] M. Renovell, F. Azaïs, and Y. Bertrand. Detection of defects using fault model oriented test sequences. *Jour. of Electronic Testing: Theory and Applications*, 14:13–22, 1999.

[17] P. Engelke, I. Polian, M. Renovell, and B. Becker. Simulating resistive bridging and stuck-at faults. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 25(10):2181–2192, Oct. 2006.

[18] Z. Li, X. Lu, W. Qiu, W. Shi, and D.M.H. Walker. A circuit level fault model for resistive bridges. *ACM Trans. on Design Automation of Electronic Systems*, 8(4):546–559, 10 2003.

[19] I. Polian, I. Pomeranz, S. Reddy, and B. Becker. On the use of maximally dominating faults in n-detection test generation. *IEE Proceedings Computers and Digital Techniques*, 151(3):235–244, 5 2004.

[20] T. Shinogi, T. Kanbayashi, T. Yoshikawa, S. Tsuruoka, and T. Hayashi. Faulty resistance sectioning technique for resistive bridging fault ATPG systems. In *Asian Test Symp.*, pages 76–81, 2001.

[21] I. Polian, S. Kundu, J.M. Galliere, P. Engelke, M. Renovell, and B. Becker. Resistive bridge fault model evolution from conventional to ultra deep submicron technologies. In *VLSI Test Symp.*, pages 343–348, 2005.

[22] T. Maeda and K. Kinoshita. Precise test generation for resistive bridging faults of CMOS combinational circuits. In *Int'l Test Conf.*, pages 510–519, 2000.

[23] S.C. Ma, P. Franco, and E.J. McCluskey. An experimental chip to evaluate test techniques experimental results. In *Int'l Test Conf.*, pages 663–672, 1995.

[24] S.M. Reddy, I. Pomeranz, and S. Kajihara. Compact test sets for high defect coverage. *IEEE Trans. on CAD*, 16:923–930, 1997.

[25] M.R. Grimaila, S. Lee, J. Dworak, K.M. Butler, B. Stewart, H. Balachandran, B. Houchins, V. Mathur, J. Park, L.-C. Wang, and M. Ray Mercer. REDO-random excitation and deterministic observation–first commercial experiment. In *VLSI Test Symp.*, pages 268–274, 1999.

[26] G. Chen, S. Reddy, I. Pomeranz, J. Rajski, P. Engelke, and B. Becker. An unified fault model and test generation procedure for interconnect opens and bridges. In *European Test Symp.*, pages 22–27, 2005.

[27] P. Engelke, I. Polian, M. Renovell, and B. Becker. Automatic test pattern generation for resistive bridging faults. *Jour. of Electronic Testing: Theory and Applications*, 22(1):61–69, 2006.

[28] I. Polian, P. Engelke, M. Renovell, and B. Becker. Modeling feedback bridging faults with non-zero resistance. *Jour. of Electronic Testing: Theory and Applications*, 21(1):57–69, 2005.