Bit Parallel Test Pattern Generation for Path Delay Faults

Manfred Henftling Hannes Wittmann
Institute of Electronic Design Automation, Department of Electrical Engineering
Technical University of Munich, 80290 Munich, Germany

Abstract

A method to apply bit-parallel processing at all stages of robust and nonrobust test pattern generation for path delay faults is presented. Two different modes of bit-parallel processing are combined: fault parallel test pattern generation (FPTPG) and alternative parallel test pattern generation (APTPG). We discuss the problems that appear while exploiting bit-parallelity and we describe how to overcome them. Experimental results demonstrate a reduction of aborted faults and an acceleration up to a factor of nine.

1 Introduction

The increasing complexity of logic systems and the application of high performance semiconductor technologies have major consequences in view of test preparation. Tests targeted for stuck-at faults may be insufficient to guarantee an acceptable quality level, because some defects and/or random process variations do not change the steady state behavior of a circuit but do affect the dynamic behavior of a system. Considering path delay testing, not only the number of gates increases with the circuit size, but the number of structural paths does, too.

The *path delay fault model* [1] has been introduced to detect slow chips. It assumes delay faults on entire paths in a circuit. A lot of research has been devoted to the topic of path delay fault testing. Best suited algorithms for fault simulation [1, 2, 3, 4] have been proposed. All state-of-the-art tools use bit-parallelity for fault simulation. In the area of test pattern generation (TPG) efficient approaches [5, 6, 7, 8, 9, 10] have been presented. Contrary to fault simulation none of these tools exploits bit-parallelity. In fact, there is one attempt [11] to use up to four bits of a word for test pattern generation for stuck-at faults. However, the proposed methods result in a speed-up of only 1.2.

In spite of the discouraging results of [11], two interesting facts form a strong motivation to exploit the entire machine word length for automatic test pattern generation. First, there is enormous unused resource. Only a small fraction of a machine word representing a logic value is really active. Second, bit-parallelism has already been exploited successfully during fault simulation. The step from single pattern single fault propagation (SPSFP) to parallel pattern single fault propagation (PPSFP) [3, 12, 13] resulted in a speed improvement.

This paper shows that it is possible to use the whole machine word length for test pattern generation. For the very first time we perform successfully bit-parallel test generation at all stages of the algorithm. Two different modes of bit-parallel processing are possible: *fault parallel test pattern generation (FPTPG)* where L faults are treated simultaneously, and *alternative parallel test pattern generation (APTPG)* that allows the examination of L different pattern alternatives for a given fault. Starting with FPTPG and passing over to APTPG for still undetected faults we exploit the machine word length for easy- and hard-to-detect faults.

After a short introduction to Path Delay Fault Testing in the following section the bit-parallel TPG approach is explained in Section 3. Some interesting details that appear while using bit-parallelity are discussed in Section 4. The experimental results given in Section 5 show advantages and improvements of the new approach. Section 6 concludes the paper.

2 The Path Delay Fault Model

Smith [1] has introduced a hardware model for delay testing. The combinational circuit under test is embedded between a block of input latches and a block of output latches. All latches are assumed to be glitchless. At time T_1 the first vector V_1 is loaded into the input latches. At time T_2, after all signals in the circuit have reached stable values, the second vector V_2 is applied. The logic values of the primary outputs are sampled into the output latches at time $T_S = T_2 + T_C$, where T_C is the desired clock rate.

A path P has a nominal delay $d(P)$ and may have a delay fault called $\Delta(P)$. P is said to be faulty if the delay fault causes the wrong value of its primary output at time T_C, i.e. $d(P) + \Delta(P) > T_C$. A path delay fault is called robust detectable, if and only if, its detection is independent of all other delay faults in the circuit. If delays of arbitrary gates may invalidate the detection, the fault is nonrobust detectable. Obviously, robust detection implies nonrobust detection.

All signals that feed gates on a path and are not path signals are called off-path signals. The task of testing a path delay fault is to sensitize a given target path. All off-path signals of the path have to be set to logic values that allow the propagation of a transition at the primary input along the target path to its primary output. The test pattern generation algorithm consists of two steps. First, the path sensitization is performed, i.e., the required logic values are assigned to all path signals and all off-path signals. Second, all logic values at the off-path signals are justified by assigning appropriate logic values to the primary inputs of the circuit. Test pattern generation is performed for all paths in the circuit.

3 Bit Parallel Test Pattern Generation

With this section we will present our bit-parallel test pattern generation approach. Only for sake of explanation, we first restrict ourselves to nonrobust TPG. The extension to robust test generation is explained in the next section.

To generate nonrobust tests only the final logic values of the signals have to be considered. A three valued logic is convenient for this task: $0, 1,$ and X. As we have three values, we need $\lceil log_2 3 \rceil = 2$ bits for encoding. To exploit the whole machine word length L, we store L logic values in two words. Each bit level represents one logic value. A best suited implication procedure guarantees an efficient handling of the words while performing bit-parallel implications. Table 1 shows the encoding we have chosen.

logic value	0-bit	1-bit
0	1	0
1	0	1
X	0	0
conflict (C)	1	1

Table 1: Encoding for nonrobust TPG

The combination (1,1) is not used for encoding. Hence, it represents an illegal signal assignment, i.e., a conflict.

To illustrate the procedure we consider an example circuit that is shown in Figure 1 for FPTPG and in Figure 2 for APTPG, respectively. In the sequel we concentrate on bit-parallel TPG and consider path sensitization and backtrace in Section 4. We assume for our example a four-bit-computer, i.e., $L = 4$.

3.1 Fault Parallel TPG

During FPTPG we consider L faults simultaneously. First, L paths are sensitized and the resulting implications are performed. Now, there may be unjustified logic values in the circuit. As long as there is at least one logic value that is not justified, a backtrace procedure is performed and bit-parallel implications are made from the primary inputs. Consider our example of Figure 1. We want to perform

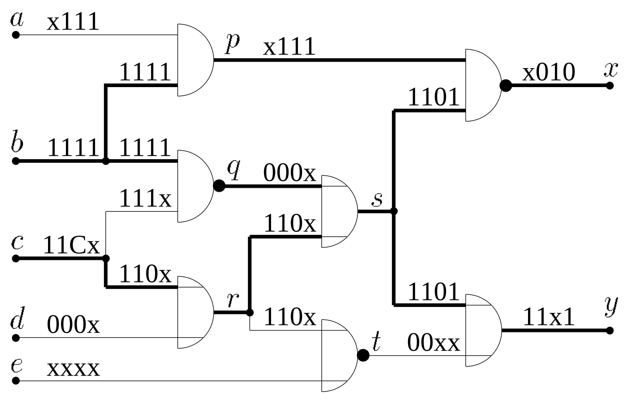

Figure 1: Performing FPTPG

FPTPG for 4 paths. We treat the paths $b-p-x, b-q-s-x, c-r-s-x,$ and $c-r-s-y$ in parallel on the bit levels 0 through 3. In our notation, bit level 0 is on the right hand side and bit level 3 is on the left hand side. Figure 1 shows the resulting logic values of the four bit levels after sensitizing the paths and performing the implications.

We can distinguish three cases. On bit level 2 and 3 all signal values are justified. Hence, the two corresponding paths are tested. On bit level 1 a conflict occurred at signal c (denoted by a ”C”). As no optional value assignments have been made, the path is redundant. Because the sensitization of subpath $b-q-s$ with a rising transition at b is impossible all paths containing this subpath are proved to be redundant, too. Finally, on bit level 0 no conflict occurred, but the value 1 at signal s is not yet justified. The result of the backtrace procedure is to assign a 1 to input d, and after the resulting implications, signal s is justified and a test pattern for path $b-p-x$ is found. The advantage of FPTPG is that it is possible to treat multiple paths simultaneously and not sequentially.

3.2 Alternative Parallel TPG

In order to examine several alternatives of test patterns simultaneously, APTPG is performed. If the result of backtrace indicates that at various primary inputs 0 and 1 is required, L alternatives can be considered simultaneously. Assume that we want to test path $a-p-x$ with a falling transition at a. The final values are shown in Figure 2. We sensitize the path at all L bit levels. Backtrace indicates to assign values to the primary inputs c and d. We examine all four possibilities in four bit-levels at one time. As there is at least one bit level without conflict the path is tested.

In general, we can consider all possible value assignments at $log_2 L$ primary inputs. As soon as we exceed this

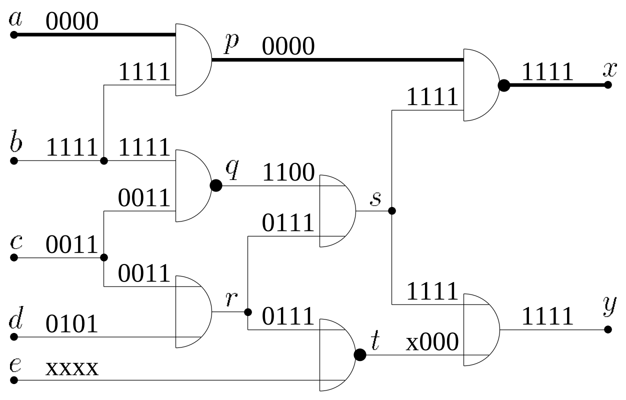

Figure 2: Performing APTPG

limit we proceed with conventional backtracking on all bit levels simultaneously.

3.3 Combination of FPTPG and APTPG

FPTPG and APTPG complete one another excellently. In order to treat easy-to-test faults as efficient as possible, we start with FPTPG and identify a lot of testable and redundant paths; this leads to a speed-up of test pattern generation. If backtracking is necessary, we dynamically pass over to APTPG. We avoid the effort of resensitizing a target path by simply flattening the active bit of a logic value to multiple bit levels. With the help of APTPG we examine hard-to-detect faults. As we examine several alternatives a time, backtracking is less probable and greater areas of the search space can be examined; our combined method results in less aborted faults, faster test pattern generation, and faster redundancy identification.

4 Details of Bit Parallel TPG

While developing bit-parallel TPG the single tasks of the algorithm were adapted. Bit-parallelity offers the possibility of an improved formulation of the path sensitization. Backtracking had to be adapted to bit-parallelity. The consideration of robust tests needed further examination. If robust tests for path delay faults are generated, not only the final value has to be considered, but some signals are required to be stable at their value. We use the seven valued logic of [5] to perform the generation. Our encoding of the seven values is shown in Table 2. In order to achieve efficient gate evaluations we use four bits for encoding. The general TPG-algorithm is the same as for nonrobust TPG. The difference is that the stable values have to be justified from the primary inputs. FPTPG and APTPG work in the same way as explained in Section 3. A detailed explanation of the mentioned procedures is given in [14].

logic value	0-bit	1-bit	stable-bit	instable-bit
$0s$	1	0	1	0
$1s$	0	1	1	0
$0\overline{s}$	1	0	0	1
$1\overline{s}$	0	1	0	1
$0x$	1	0	0	0
$1x$	0	1	0	0
X	0	0	0	0
conflict	1	1	x	x
conflict	x	x	1	1

Table 2: Encoding for robust TPG

5 Experimental Results

The bit-parallel test generation approach was implemented in "C" and is integrated in our test preparation tool TIP; currently, TIP consists of 6000 lines of code. We use global implications and perform parallel pattern fault simulation after every L generated test patterns (L is the machine word length); no further heuristics and speed-up-techniques are implemented. The approach is tested with the help of well-known benchmark circuits [15, 16, 17]. When sequential circuits are processed, only the combinational part is considered. Several experiments were performed to get an impression of the improvements of the new techniques.

The initial experiment we made shows the efficiency of the bit-parallel test pattern generator. We generated robust and non robust tests for the ISCAS85 benchmarks. These circuits are known to be hard to test due to their enormous number of paths and due to the difficulties in detecting their delay faults. We ran this experiment on a DECstation 3000/500 with a machine word length of $L = 64$. Table 3

Circuit	# faults	# tested	efficiency	time [s]
c432	583652	3730	100.00 %	951.33
c499	795776	133696	99.94 %	2284.40
c880	17284	16083	100.00 %	49.84
c1355	8346432	22782	99.96 %	14509.06
c1908	1458114	97495	99.98 %	86387.41
c2670	1359920	15370	99.99 %	755.02
c3540	57353342	88356	99.99 %	140135.93
c5315	2682610	81435	99.99 %	14457.13
c7552	1452988	86114	99.87 %	42895.47

Table 3: Robust ATPG for the ISCAS85 circuits

and Table 4 show the results, which demonstrate that we are able to handle all circuits in a reasonable amount of time[1]. $\#faults$ and $\#tested$ represent the number of functional paths in the circuit and the number of tested faults. The efficiency is given as $efficiency = (1 - \frac{\#aborted}{\#faults}) \cdot 100\%$, where $\#aborted$ is the number of aborted faults. Finally, the time for test generation is shown. Contrary to previ-

[1] except circuit c6288, containing 10^{20} functional paths

Circuit	# faults	# tested	efficiency	time [s]
c432	583652	15855	100.00 %	17.01
c499	795776	367744	100.00 %	446.91
c880	17284	16652	100.00 %	6.51
c1355	8346432	1110304	100.00 %	1124.25
c1908	1458114	355168	100.00 %	380.55
c2670	1359920	130626	100.00 %	114.52
c3540	57353342	1202584	100.00 %	9637.56
c5315	2682610	342117	100.00 %	1604.68
c7552	1452988	277244	100.00 %	2825.41

Table 4: Nonrobust ATPG for the ISCAS85 circuits

ously published approaches for nonrobust test generation, no aborted paths are left. In the case of robust test generation some aborted paths have been left for the ISCAS85 benchmarks; however the fraction of aborted faults is always lower than 10^{-3}. These two observations also hold for all sequential benchmark circuits.

In the main part of our experiment we compared the bit-parallel generator to a version that is restricted to one bit level. Of course, we carefully omitted any unnecessary overhead. The comparison is shown in Table 5 and Table 6. Thereby, t_{sens} is the time for sensitizing the

Circuit	t_{sens}	t_{single}	$t_{parallel}$	$\frac{t_{single}}{t_{parallel}}$
s713	0.92	1.63	0.37	4.4
s838	6.04	18.12	5.62	3.2
s938	10.14	14.17	1.59	8.9
s991	15.76	39.98	29.34	1.4
s1269	45.15	717.67	408.84	1.8
s1423	31.06	110.62	13.16	8.4
s3271	18.31	631.30	154.70	4.1
s5378	51.40	122.01	26.92	4.5
s9234	108.66	257.99	121.40	2.1
s13207	763.12	537.54	255.07	2.1
s15850	4770.00	41853.60	19579.88	2.1

Table 5: Comparison of bit parallel and single bit generation (robust ATPG)

paths (identical for single-bit and bit-parallel sensitization), t_{single} is the time required by the single-bit approach, and $t_{parallel}$ is the time used by the parallel method proposed in this paper. The given results impressively show the improvements of our bit-parallel approach. For all circuits a speed-up is achieved; the average acceleration is about five. Furthermore, contrary to the parallel approach, some aborted faults occurred while handling the circuits with the single-bit approach. Hence, we achieve both a speed-up of test generation and a reduction of aborted faults.

With the next experiments, we compared ourselves to three efficient and well-known state-of-the-art tools. TSUNAMI-D [8] represents an efficient BDD-based approach. BDDs are known to be best suited for test genera-

Circuit	t_{sens}	t_{single}	$t_{parallel}$	$\frac{t_{single}}{t_{parallel}}$
s713	1.32	1.00	0.15	6.7
s838	1.41	3.26	1.41	2.3
s938	2.33	2.41	0.54	4.5
s991	3.96	11.67	1.63	7.2
s1269	17.50	23.50	7.44	3.2
s1423	10.91	44.76	10.26	4.4
s3271	6.24	15.76	6.40	2.5
s5378	16.02	17.76	3.06	5.8
s9234	45.12	140.59	36.47	3.9
s13207	1551.86	1634.67	3836.50	2.3
s15850	30929.62	27602.17	5448.47	5.1

Table 6: Comparison of bit parallel and single bit generation (nonrobust ATPG)

tion as long as the BDD can be constructed. DYNAMITE [10] is a structural method that has been shown to perform very well. Since all published results were achieved on a DEC 5000/200 ($L = 32$) we computed our results for this machine, though only 32 instead of 64 bits can be exploited. The results for ten published sequential circuits can be seen in Table 7 and 8 for nonrobust and robust test generation, respectively. The tables show that the bit-parallel approach is able to generate complete test sets in a reasonable amount of time. For some circuits TIP performs slightly slower than TSUNAMI-D. For detecting non-robust tests TSUNAMI-D is based on a slightly deviated test class compared to TIP and DYNAMITE. For nonrobust test generation TIP is up to eight times faster than DYNAMITE, for robust test generation it is comparable. Please note, that up to now we use the suboptimal seven valued logic [5] instead of a ten valued logic [6] for generating robust tests. Furthermore no heuristics and speedup techniques are used. These resources are topics of our future work.

The comparison with NEST [9] is difficult to do. NEST targets on a fast and good estimation of the fault coverage, while we want to perform complete test generation. Results for the circuits presented in Table 8 can be found in [9]. The interested reader may perform a comparison, always keeping in mind the different intentions of the two tools.

6 Conclusion

We have shown with this contribution that it is possible to perform bit-parallel test pattern generation at any stage of path delay fault test generation. The experimental results presented show that, analogously to bit-parallel fault simulation, a speedup of up to nine is obtained and a reduction of aborted faults is possible. Our future research activity concentrates on further speed-up techniques and the application of bit-parallel test generation to further fault models, first of all the stuck-at fault model.

Circuit	TIP		TSUNAMI-D		DYNAMITE	
	# tested	time [s]	# tested	time [s]	# tested	time [s]
s641	2270	2.5	2096	6.7	2270	6.2
s713	4922	4.4	2066	15.1	4922	26.8
s1196	3759	13.5	3708	8.7	3759	24.9
s1238	3684	14.0	3663	9.0	3684	31.2
s1423	45198	60.3	33981	878.5	45198	208.2
s1494	1927	20.7	1926	2.4	1927	5.2
s5378	21928	57.8	19413	284.1	21928	111.9
s13207	476145	10195.8	162798	1566.0	476145	12274.6
s15850	10782994	125419.9	n.a.	n.a.	10782994	975240.0
s38584	334927	11991.7	170291	1934.7	334927	32364.9

Table 7: Comparison for nonrobust test generation (DEC 5000/200)

Circuit	TIP		TSUNAMI-D		DYNAMITE	
	# tested	time [s]	# tested	time [s]	# tested	time [s]
s641	1979	6.4	1979	8.2	1979	6.7
s713	1184	4.0	1184	20.6	1184	15.0
s1196	3581	69.0	3562	15.0	3579	32.3
s1238	3589	71.2	3654	16.6	3587	37.8
s1423	28696	162.1	23220	926.5	28696	315.0
s1494	1882	90.4	1882	3.0	1882	17.1
s5378	18656	227.6	18248	296.9	18656	198.0
s13207	27603	2813.8	27484	4286.0	27603	1984.0
s15850	182673	62138.1	n.a.	n.a.	182673	86040.0
s38584	92239	14060.0	90146	4790.2	92239	10636.0

Table 8: Comparison for robust test generation (DEC 5000/200)

References

[1] Gordon L. Smith. Model for Delay Faults Based Upon Paths. In *Proceedings IEEE International Test Conference*, pages 342–349, November 1985.

[2] Irith Pomeranz and Sudhakar M. Reddy. An Efficient Non–Enumerative Method to Estimate Path Delay Fault Coverage. In *Proceedings IEEE/ACM International Conference on Computer–Aided Design*, pages 560–567, November 1992.

[3] Franz Fink, Karl Fuchs, and Michael H. Schulz. Robust and Nonrobust Path Delay Fault Simulation by Parallel Processing of Patterns. *IEEE Transactions on Computers*, pages 1527–1536, December 1992.

[4] Manfred Henftling, Hannes C. Wittmann, and Kurt J. Antreich. Path Hashing to Accelerate Delay Fault Simulation. In *Proceedings IEEE/ACM Design Automation Conference*, pages 522–526, 1994.

[5] C. J. Lin and S. M. Reddy. On Delay Fault Testing in Logic Circuits. In *Proceedings IEEE/ACM International Conference on Computer–Aided Design*, pages 148–151, November 1986.

[6] Karl Fuchs, Franz Fink, and Michael H. Schulz. DYNAMITE: An Efficient Automatic Test Pattern Generation System for Path Delay Faults. *IEEE Transactions on Computer–Aided Design*, pages 1323–1335, October 1991.

[7] Kwang-Ting Cheng, Srinivas Devadas, and Kurt Keutzer. Robust Delay–Fault Test Generation and Synthesis for Testability Under a Standard Scan Design Methodology. In *Proceedings IEEE/ACM Design Automation Conference*, pages 80–86, June 1991.

[8] Debashis Bhattacharya, Prathima Agrawal, and Vishwani D. Agrawal. Delay Fault Test Generation for Scan/Hold Circuits using Boolean Expressions. In *Proceedings IEEE/ACM Design Automation Conference*, pages 159–164, June 1992.

[9] Irith Pomeranz, Sudhakar M. Reddy, and Prasanti Uppaluri. NEST: A Non–Enumerative Test Generation Method for Path Delay Faults in Combinational Circuits. In *Proceedings IEEE/ACM Design Automation Conference*, pages 439–445, June 1993.

[10] Karl Fuchs, Hannes C. Wittmann, and Kurt J. Antreich. Fast Test Pattern Generation for all Path Delay Faults Considering Various Test Classes. In *Proceedings European Test Conference*, pages 89–98, April 1993.

[11] Kewal Saluja and Kyuchull Kim. Improved Test Generation for High-Activity Circuits. *IEEE Design & Test of Computers*, pages 26–31, August 1990.

[12] J. A. Waicukauski, E. B. Eichelberger, D. O. Forlenza, E. Lindbloom, and T. McCarthy. Fault Simulation for Structured VLSI. *VLSI Systems Design*, pages 20–32, December 1985.

[13] Kurt J. Antreich and Michael H. Schulz. Accelerated Fault Simulation and Fault Grading in Combinational Circuits. *IEEE Transactions on Computer–Aided Design*, pages 704–712, September 1987.

[14] Manfred Henftling and Hannes Wittmann. A Bit Parallel ATPG Approach for Path Delay Faults. Technical Report TUM-LRE-6-94, Department of Electrical Engineering, Technical University of Munich, Germany, 1994.

[15] Franc Brglez and Hideo Fujiwara. A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran. In *IEEE International Symposium on Circuits and Systems; Special Session S6AB on ATPG and Fault Simulation*, pages 663–698, June 1985.

[16] Franc Brglez, David Bryan, and Krzysztof Kozminski. Combinational Profiles of Sequential Benchmark Circuits. In *Proceedings IEEE International Symposium on Circuits and Systems*, pages 1929–1934, May 1989.

[17] Franc Brglez. ACM/SIGDA Benchmark Electronic Newsletter DAC '93 Edition. Microelectronics Center of North Carolina (MCNC), June 1993.