A Hardware Unit for Backward Propagation of Output Vectors through a Combinational Circuit with Don't Care Justification

Martin Šťáva, Ondřej Novák

Abstract: *Backward propagation of output vectors through a combinational circuit lies at the core of many applications, including automatic test pattern generation (ATPG). This paper describes and evaluates an approach for the backward propagation using don't care justification. This approach gets high performance due to large amounts of fine-grain parallelism in the implication process. The results of this new method are compared with HW implementation of the basic backtrace algorithm proposed recently. The experimental results have been obtained for the ISCAS'85 benchmarks.*

Key words: *Backward propagation, backtrace, VLSI, ATPG, hardware, FPGA, vector generation, combinational circuit.*

1. INTRODUCTION

Many algorithms and heuristics to generate tests for combinational and sequential circuits have been proposed, such as D-algorithm, PODEM, RAPS, FAN, SOCRATES, EST and many others, e.g., [1]-[4], [7]. Improving techniques for deterministic test pattern generation, and a high-speed ATPG system for large scan designs are described in [5] and [10] respectively.

Our main task is to reduce the time complexity of scan-based sequential circuits. Within these circuits, a scan-chain divides a sequential part from combinational ones. So we have decided to combine test vectors scanned into the scan-chain with responses computed by combinational parts (*CP*s) in order to reduce the test length thus the number of clock cycles for applying the test. Such an approach puts a test generator together a signature analyser into a Highly Integrated Logic Design Observer, marked as *HILDO*. There are two main approaches how to consider HILDO values.

In the first approach, HILDO values are considered to be input vectors into corresponding CPs in time t and, consequently, to be responses computed by other CPs in time $(t+1)$. In this case, any CP input vector uniquely determines one response only. This determination is forward.

In the second approach, HILDO values are considered to be responses from corresponding CPs in time t and, before, to be input vectors resulted from other CPs in time $(t-1)$. In this case, one response can correspond to more input vectors or none. This determination is backward, and is unique as well. Because more input vectors can be resulted, so there are more possibilities to select a proper input vector for subsequent combining. That is why we have been dealing with a backward determination solved by a backtrace algorithm that we have implemented in HW, where the advantageous of parallel processing is implicitly used.

The Boolean satisfiability (SAT) problem is a well-known problem in computer-aided design of integrated circuits, such as test generation, logic verification and timing analysis. The idea is to translate the D-algorithm problem formulation into a characteristic equation and to solve the equation using a branch-and-bound search. A lot of algorithms using fast SAT solvers were recently proposed, e.g., in [6], [11]. A way of using configurable hardware for accelerating Boolean satisfiability is described and evaluated in [12], where the performance of hardware implementation of a certain SAT algorithm is compared to its software implementation.

The rest of the paper is organised as follows. A backward-determining circuit is described in Section 2. In Section 3, the conflict-based control is explained. Section 4 presents and evaluates the experimental results. Conclusions are given in Section 5.

2. BACKWARD-DETERMINING CIRCUITS

In the case of HW implementation of the backtrace algorithm, it is necessary to transform a CP structure into a backward-determining one. Such a method has been

developed and its principal scheme is shown in Fig. 1. The symbol *CP* has the same meaning as in the section above. The symbol *Rq* denotes requested values on internal signal lines, e.g. forced logic values to given signals. The symbol *TCP* represents a transformed CP and the symbol *CL* denotes control logic. Let the entire part (Rq + TCP + CL) be called a *backward-determining circuit*, marked as *BDC*. In the rest of the paper, the terms *OutVector* and *InpVector* denote an output vector applied to the input and an input vector obtained at the output respectively.

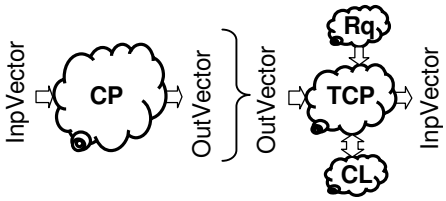

Fig. 1: Transformation of combinational circuits

2.1 ARCHITECTURE

We have proposed backward-determining circuit architecture, see Fig. 2, that was designed for working with three-value data signals. The architecture results from the basic scheme. Functions of the blocks used in the architecture are described further.

The *Input Block* and the *Output Block* register an output vector value and an input vector value respectively. Moreover, the Output Block registers values of status signals Valid and Done of which meanings follow. A generated input vector can be valid or invalid. It is signalised by the signal *Valid*. When all possible input vectors for an applied output vector have been computed, the signal *Done* is set to 1 otherwise to 0. Computation of input vectors is initialised by the signal *Start*.

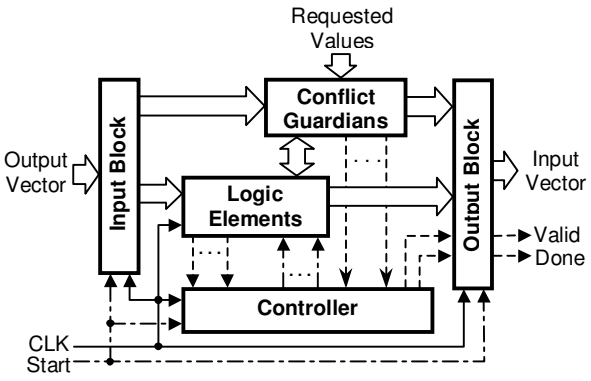

Fig. 2: Backward-determining circuit architecture

The block *Conflict Guardians* consists of guarding blocks *GB* that take care of conflicts among signal line values in signal fan-outs or between values of signal lines and *Requested Values* at these signal lines. It follows that each fan-out in the CP is converted to the GB block (see Fig. 5). This block performs matching operations with the binary intersection operator as shown in the table ∩. The *err* entry in the box means that a conflict between two values has occurred and it is signalised to the Controller as a status signal Conflict.

∩	0	1	x
0	0	*err*	0
1	*err*	1	1
x	0	1	x

The block *Logic Elements* consists of *LE* blocks that represent backward implication tables of gates (*BIT*s) or of greater combinational blocks. Also each gate in the CP is substituted by the corresponding block LE during transformation of a circuit into a BDC. A structure of the block LE for a 2-AND gate and a BIT corresponding to this block are shown in Fig. 3. The block *Controller*, of which reconfigurable structure is illustrated in Fig. 4, controls whole computation of InpVectors and contains the main logic of the backtrace algorithm. In this figure, a clock signal and data ones are drawn with a continuous line, status signals with a dashed line, and control signals with a dash-and-dot line. Function of the *Controller* block is described in Section 3 and its subsections.

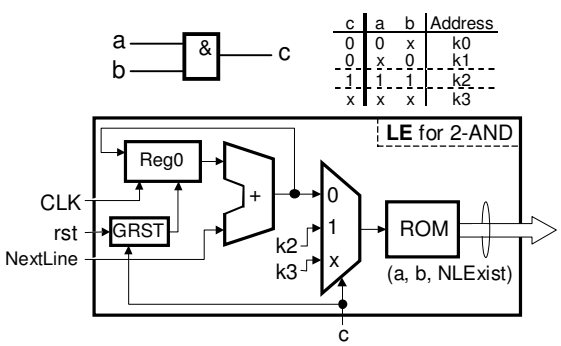

Fig. 3: The backward implication table and the Logic Element block for a 2-AND gate

The BDC architecture has been designed to generate one InpVector by one clock cycle and to be able to generate all valid InpVectors for an OutVector (introduced firstly in [9]). Let it be highlighted that the backtrace algorithms used in ATPG tools give one valid input vector only.

2.2 OUR PREVIOUS WORK

Recently, we proposed the basic backtrace algorithm (*BBA*) without any optimisations and heuristics to implement into HW. Some experimental results were obtained, and were stated in [8]. These results are also shown in Section 4 for comparison with other ones only.

Then, we were trying to find, propose and apply some heuristics or optimisation methods or advanced techniques to speed up the vector generation process. We were found some advanced techniques and just one of these is explained and described in the following section.

3. CONFLICT-BASED CONTROL

The new technique, introduced here, is based on the change of the LE block selection order that give the cause of conflict occurrence. In this technique, the non-reconfigurable LE blocks are used as well as in the BBA (for example, see Fig. 3). Now, reconfiguration logic is embedded in the controller – see Fig. 4, where the non-cross-hatched objects were added against the controller used in the BBA.

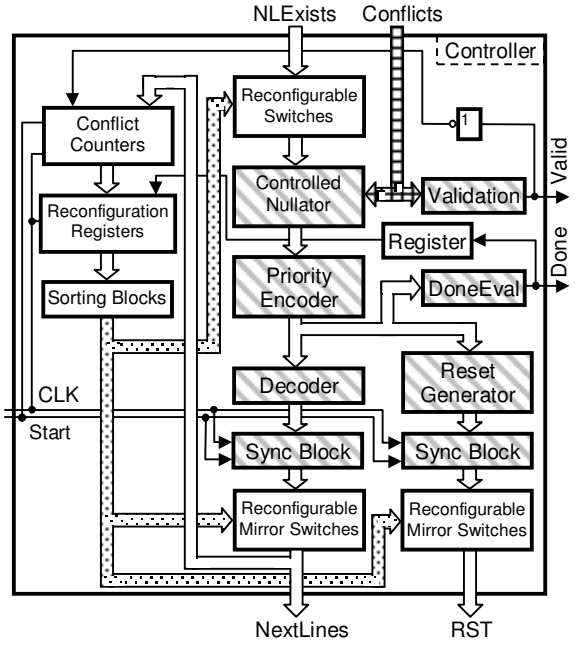

Fig. 4: A block diagram of the controller including on-line reconfiguration

The block *Controlled Nullator*, resetting some of signals *NLExists*, transfers these signals according to a priority of controlling signals *Conflicts* – by what way is shown on an example. A circuit before its transformation into a BDC is divided into gate levels according to fan-outs (see Fig. 5 (a)). Each level is consisted of gates lying between the nearest fan-outs, or between input/output and the nearest fan-out. Let us note that convention of signal labelling is following: $label_{level}^{index}$ where *label* is a signal name, *level* denotes the level to which the corresponding gates or fan-outs belong, *index* is a rank number in *level*. If several conflicts occur, those with the lowest level number take precedence over others, i.e. the controller processes them with a priority – a lower level has a higher priority. In the example, if both conflicts c_1^1 and c_2^1 occur, c_1^1 will be preferred. Thus values of the status signals *NLExists* = (*NLExist*$_1$, …, *NLExist*$_4$) with the level number lower than 1 are blocked, i.e. are set to 0. It follows that the occurred conflict can be solved by the selection of another BIT line in a chosen LE, or by the selection of another LE. Such LEs have to lie in the level equal to or less than 1. If c_2^1 occurs, which LE corresponding to one of {&2, &3} will be tried to select first? There are two main possibilities how that can be determined – static and dynamic approach. In the static one, the LE selection order is given by a method transforming the circuit into an appropriate BDC, and is fixed.

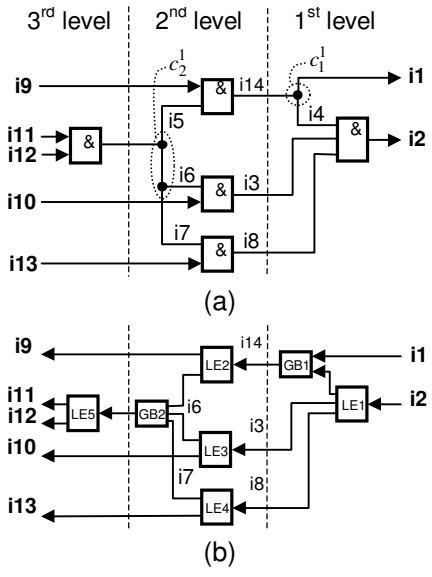

Fig. 5: (a) An example of a circuit, (b) a simplified backward-determining circuit corresponding to it

In the dynamic approach, the LE selection order can be reconfigured on-line in dependence on the number of conflicts that were caused by different values of some LEs in the same or higher level. Then, the conflict occurred in time *t* is imputed to the LE selected

by the appropriate control signal *NextLine* in time (*t*–1). This selective information is stored in the *Sync Block* (see Fig. 4) of which another function is to synchronise the status and control signals thus to break the closed combinational path, i.e. to break the combinational path: * → Reconfigurable Switches → Controlled Nullator → Priority Encoder → Decoder → Reconfigurable Mirror Switches → LE(Adder) → LE(MUX) → LE (ROM) → and again to *. And just the dynamic approach has been proposed and is presented in this paper.

3.1 RECONFIGURATION FUNCTION OF THE CONTROLLER

This subsection focuses on description of the reconfiguration part of the controller displayed in Fig. 4. Function of the basic controller part, i.e. that consisted of the blocks cross-hatched, is in detail analysed and shown on an example in [9].

The main idea rests in counting conflicts by *Conflict Counters* and in sorting their counts according to a size. These counters are attached to the signals NextLines of all LEs, which are divided into the levels as mentioned above. Counter values, that can be in the range from 0 to $min(\#occurred\_conflicts, 2^{width\_ConflictCounter})$, are arranged in ascending order in each level. It means that the LE after whose selection in some level at least conflicts have occurred is selected first. That is to say, the LE selection order has to be changed by re-mapping the signals *NLExists* in each level in order to select the LE causing at least conflicts. After selecting a proper LE the signals *NextLines* and *RSTs* have to be re-mapped back in order to choose the right LE, i.e. the *NLExist* and *NextLine* indexes have to be both same. An example of this process is shown in Fig. 6.

Clock enables signals of *Reconfiguration Registers* and *Conflict Counters* in time *t* are also driven by signal values in time (*t*–1). Conflicts are counted by the conflict counters and registered by the reconfiguration registers.

In the *Sorting Block*, there is a non-blocking sorting network in order to sort the number of conflicts in ascending order. This network is composed of elements having two inputs and two outputs and performing function "compare & exchange". The element compares values at both inputs and connects them to its outputs as follows: the input with a less or equal value to the top output and the remaining input to the bottom output. So the element can be configured in one of two states. In the first one, called a straight-through state, the top input is connected to the top output and the bottom input to the bottom output. This state is signalised by the logic value 0 at the information output, drawn by a broken line in Fig. 6. In the second state, called an exchange state, the outputs are exchanged, so the top input is connected to the bottom output and the bottom input is connected to the top output. This state is signalised by the logic value 1 at the information output.

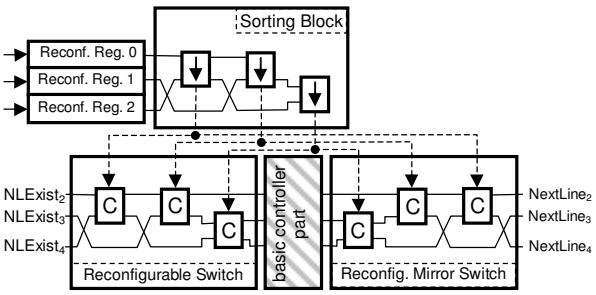

Fig. 6: The reconfiguration part for the 2nd level of the example displayed in Fig. 5 (b)

In the *Reconfigurable Switches* and *Reconfigurable Mirror Switches*, there is a non-blocking switching network in order to re-map selection addresses of the BIT lines. This network is composed of switches having two inputs and two outputs and being controlled by a switching signal. So the switch can be set by the switching signal into the same states as the element in the Sorting Block.

4. EXPERIMENTAL RESULTS

Let a backward-determinig circuit (BDC) without the conflict-based control and a BDC with that (proposed) be marked as BDC_BASIC and BDC_L respectively.

The results of experiments carried out for the ISCAS'85 benchmarks are presented in Table 1. Its columns marked as 'a-o' show an area overhead expressed in the thousands of equivalent gates (*GE*) for original benchmarks and transformed circuits. Columns 2 and 6 contain clock speed values for the BDC_BASICs and BDC_Ls. The columns marked as

'avg_#cycles for 1st VIV' denote an average number of clock cycles for generating the first valid input vector. In order to determine the average number of clock cycles we applied all of 3^m output vectors and observed the first occurrence of the valid input vector, if any, for each output one. The average time to compute the first VIV was simply calculated as $\frac{10^{-6}}{CLK\_speed} * VIV$ seconds and is stated in columns marked as 'avg_time for 1st VIV'.

Table 1: Results for the ISCAS'85 benchmarks and corresponding backward-determining circuits (BDCs)

	BDCs – without confl.-based control				BDCs – with that					
									Summary	
Bench-marks	CLK speed [MHz]	a-o in #GE [10^3]	avg_#cycles for 1st VIV	avg_time for 1st VIV [µs]	CLK speed [MHz]	a-o in #GE [10^3]	avg_#cycles for 1st VIV	avg_time for 1st VIV [µs]	a-o increase [%]	speed up [%]
c17	82.6	1	1.6	0.02	81.5	2	1.5	0.02	119.5	7.4
c432	12.4	28	47.7	3.84	11.3	68	39.7	3.50	141.3	8.8
c499	10.6	34	31.6	2.99	9.5	57	25.9	2.72	69.9	9.1
c880	6.4	60	67.0	10.52	5.6	88	54.5	9.67	47.3	8.0
c1355	4.7	76	138.8	29.71	4.1	156	104.8	25.75	104.9	13.3
c1908	3.0	99	206.3	68.27	2.6	214	141.0	54.72	116.3	19.9
c2670	2.3	158	243.2	107.15	2.0	255	154.5	77.60	61.4	27.6
c3540	1.6	211	310.2	188.32	1.4	306	218.5	150.80	45.2	19.9
c5315	1.2	296	431.8	358.58	1.1	385	280.3	251.60	30.0	29.8
c6288	1.2	329	780.0	674.78	0.9	519	492.7	520.87	57.7	22.8

In comparison with BDC_BASIC, the average growths of an area overhead are $\frac{\sum_{i=1}^{10} \frac{ao(BDC\_L)_i - ao(BDC\_BASIC)_i}{ao(BDC\_BASIC)_i} * 100}{10}$ = 79.4 % and the average speeds up are $\frac{\sum_{i=1}^{10} \frac{avg\_time(BDC\_BASIC)_i - avg\_time(BDC\_L)_i}{avg\_time(BDC\_BASIC)_i} * 100}{10}$ = 16.7 % for the BDC_Ls.

As a target platform, we have used the Xilinx FPGA of the Spartan 2E series, in the concrete xc2s600e. All circuits have been synthesised by Xilinx XST of version 6.3i.

The values from Table 1 are displayed in the graphs. In Fig. 7, the average number of clock cycles for the first valid input vector generation by the backward-determining circuits is shown. As shown in Fig. 8, the area overhead curve of the transformed circuits grows by the 6th order polynomial. It means that the asymptotical complexity of the area

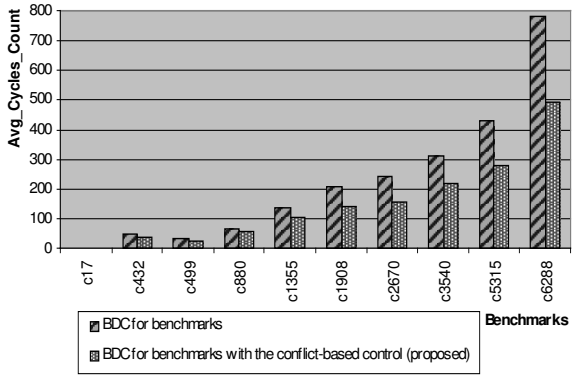

Fig. 7: The average number of clock cycles for the first valid input vector generation by the backward-determining circuits (BDCs)

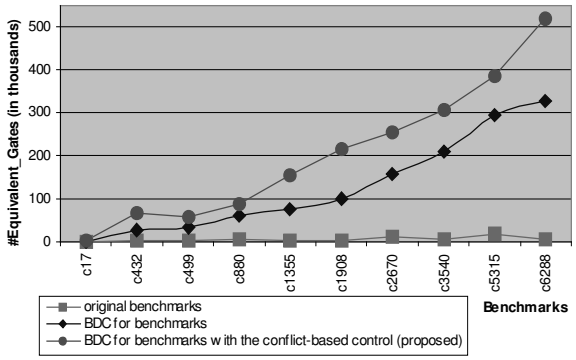

Fig. 8: The area overhead for the benchmarks and their backward-determining circuits (BDCs)

overhead is polynomial for the selected set of the benchmarks. So we can assume that these results would be accepted for any combinational circuit generally.

5. FUTURE WORK

We are going to implement the basic ATPG algorithms and heuristics containing various forms of backtrace procedure in software. Then we will be able to compare the complexities between HW and SW solution. However, we have to note that the ATPG backtrace algorithms find the first valid assignment of values to primary inputs only, but our HW implementation of backtrace algorithm finds all possible assignments. Therefore, we will modify the basic backtrace algorithms and implement them in SW. Finally it will be possible to compare these solutions as well.

Another possibility how to reconfigure the block Logic Element is replacing ROM by RWM and changing not the addresses of BIT lines but the values in BIT lines.

We are going to reduce the operational complexity of testing sequential circuits with a scan-chain by using the backward determination as introduced in Section 1.

6. CONCLUSION

We have proposed the conflict-based on-line control technique to use in the backtrace into hardware, strictly speaking into Xilinx FPGA. That of series Spartan 2E has been our target architecture but we are going to take advantage of Xilinx FPGA of series Virtex-II.

As a result of experiments, for the backtrace algorithm implemented in HW, we can evaluate the conflict-based on-line control using the conflict counters and re-mapping the LE block selection order. It can be summarised that the usage of that implementation of that control is not effective enough because the average growth of an area overhead is about 80 % but the average speed up is about 17 % only.

ACKNOWLEDGEMENT

This research has been supported by the grant GA102/04/2137 and by the institutional research plan MSM6840770014.

REFERENCES

[1] Abramovici, M., M. A. Breuer, A. D. Friedman. Digital Systems Testing and Testable Design. IEEE Press, 1990, 652 pages.

[2] Bushnell, M. L., V. D. Agrawal. Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits. Kluwer Academic Publishers, 2000, 690 pages.

[3] Fujiwara, H., T. Shimono. On the Acceleration of Test Generation Algorithms. IEEE Trans. on Comp., December 1983, pp. 1137-1144.

[4] Goel, P. An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits. IEEE Tran. on Comp., March 1981, pp. 215-222.

[5] Hamzaoglu, I., J. H. Patel. New Techniques for Deterministic Test Pattern Generation. Proc. of IEEE VLSI Test Symp., April 1998, pp. 446-452.

[6] Larrabee, T. Test Pattern Generation Using Boolean Satisfiability. IEEE Trans. on Computer-Aided Design, January 1992, pp. 4-15.

[7] Roth, J. P. Diagnosis of Automata Failures: A Calculus and a Method. IBM Journal of Res. and Develop., vol. 10, July 1966, pp. 278-291.

[8] Šťáva, M., O. Novák. HW Implementation of the Backtrace Algorithm. 11th IEEE Europ. Test Symp., 2006, in press.

[9] Šťáva, M., O. Novák. An Illustrative Case Study of Backward Determination of Input Vectors by HW. Proc. of MOSIS, 2006, in press.

[10] Tsai, K. H., R. Tompson, J. Rajski et al. STAR-ATPG: A High Speed Test Pattern Generator for Large Scan Designs. Proc. of the Inter. Test Conf., September 1999, pp. 1021-1030.

[11] Zhang, L., S. Malik. Conflict Driven Learning in a Quantified Boolean Satisfiability Solver. Proc. of the Intl. Conf. on Computer Aided Design, November 2002, pp. 442-449.

[12] Zhong, P., P. Ashar, S. Malik, M. Martonosi. Using Reconfigurable Computing Techniques to Accelerate Problems in the CAD Domain: A Case Study with Boolean Satisfiability. Proc. of the Design Automat. Conf., June 1998, pp. 194-199.

ABOUT THE AUTHORS

Ing. Martin Šťáva, PhD Student, Department of Computer Science and Engineering, Czech Technical University in Prague, Czech Republic, Phone: +420 22435 7225, E-mail: stavam1@fel.cvut.cz.

Prof. Ing. Ondřej Novák, CSc., Department of Computer Science and Engineering, Czech Technical University in Prague, Czech Republic, Phone: +420 22435 7237, E-mail: novako3@fel.cvut.cz.