Using a Hierarchical DfT Methodology in High Frequency Processor Designs for Improved Delay Fault Testability

Michael Kessler[1], Gundolf Kiefer[2], Jens Leenstra[1], Knut Schünemann[1], Thomas Schwarz[2], Hans-Joachim Wunderlich[2]

[1]IBM Deutschland Entwicklung GmbH
Schönaicherstr. 220, 71032 Böblingen
Germany

[2]University of Stuttgart
Breitwiesenstr. 20/22, 70567 Stuttgart
Germany

Abstract

In this paper a novel hierarchical DfT methodology is presented which is targeted to improve the delay fault testability for external testing and scan-based BIST. After the partitioning of the design into high frequency macros, the analysis for delay fault testability already starts in parallel with the implementation at the macro level. A specification is generated for each macro that defines the delay fault testing characteristics at the macro boundaries. This specification is used to analyse and improve the delay fault testability by improving the scan chain ordering at macro-level before the macros are connected together into the total chip network. The hierarchical methodology has been evaluated with the instruction window buffer core of an out-of-order processor. It was shown that for this design practically no extra hardware is required.

1. Introduction

The increasing speed and complexity of modern VLSI circuits emerge a need for very high fault coverage for both stuck-at and delay faults. Deep-submicron technologies introduce new performance-related defect types, and the increasing clock frequencies in high-speed designs impose aggressive timing margins. While the internal clock frequencies have risen by 30% per year, the accuracy of external test equipment has improved at a rate of only 12% per year [1]. Hence, it is becoming increasingly difficult to do performance-related testing using external test equipment. The test for delay faults becomes of practical importance especially for high frequency processors, since many paths are critical.

Proper implementations of existing design-for-test methodologies like scan design [2] and Built-In Self-Test (BIST) may achieve high coverage for static faults (e. g. stuck-at faults) [3, 4], and in general, any improvement for the stuck-at fault coverage also improves the delay fault coverage. In addition, the circuit can be clocked at system speed enabling to test for delay or transition faults as described in [5, 6].

For BIST, the STUMPS architecture based on pseudo-random pattern generation [7] is widely used. Pseudo-random patterns can efficiently be generated on-chip using linear feedback shift registers (LFSRs), eventually combined with phase shifters for reducing pattern correlations [8]. However, pseudo-random patterns cannot guarantee complete or sufficient fault coverage, therefore a number of techniques for improving the fault coverage have been published. On the one hand the circuit under test may be modified, e. g. by test point insertion [9-12]. On the other hand, a more sophisticated test pattern generator for weighted random patterns [13-15], pseudo-exhaustive patterns [16-18] or deterministic patterns [19-22] can be used.

Most of these schemes can be used or extended in order to target delay faults. However, testing for delay faults requires two patterns, an initialisation pattern, which sets the circuit to a predefined state, and an activation pattern, which triggers a transition. In a standard test-per-scan scheme, many delay faults may be untestable because of latch-adjacency problems, which do not allow to apply the needed pair of test patterns to test the given delay fault. This problem can be addressed by using an enhanced scan chain in which additional scan latches are inserted or in which the individual scan elements are enhanced such that they can store two independent values [31, 32]. Besides the additional hardware overhead for the scan cells such a scheme may require additional clock and control signals.

Two main techniques are known to apply pattern pairs to a standard scan design. When using *scan shifting* [23], the scan path is operated in shift mode, and the second pattern is generated out of the first one

0-7803-7169-0/01 $10.00 © 2001 IEEE

by this single bit shift. When using *functional justification* the circuit is operated in system mode for two clock cycles, so that the second pattern is calculated out of the first one by the circuit itself.

In this work, we concentrate on the scan shifting approach using standard scan design. While for functional justification the dependencies are mainly determined by the system functionality of the design, the delay fault testability with scan shifting can be improved by reordering the scan chain.

Several algorithms have been described in the literature that all try to improve the delay fault coverage by scan chain reordering [24-26]. By adding a minimal number of additional latches afterwards [24, 27] the delay fault coverage can be increased even further at the cost of chip area. In [10] it is proposed to insert observation points in order to increase the path delay fault testability.

In general, these techniques operate on the complete design, so that the complexity may increase considerably for large designs. Furthermore, the scan chain reordering/latch insertion can only be performed after the complete design is available and completed.

In this paper, a hierarchical design-for-test method is presented where the scan modifications are applied locally at macro level. Directly after the design has been partitioned into macros, conflict matrices defining constraints on scan chains outside a given macro are added to the macro interface description. After the macros have been designed individually, the scan ordering of each macro, as required for improved delay fault testability for the chip, is determined using the interface conflict matrices of the adjacent macros.

In Section 2 the design method will be discussed as it is in use for the design of high frequency processor systems. In Section 3, the new hierarchical DfT method and its integration into the design flow is described. In Section 4, the instruction window buffer will be introduced as a case study to which the hierarchical method was successfully applied. Experimental results presented in Section 5 will show that the delay fault testability, but also the random pattern fault coverage increases considerably for the critical parts of the instruction window buffer. Finally, Section 6 concludes the paper.

2. High frequency processor design and test

Currently almost all design methods for high performance processors are optimised to reach a high processor frequency. Increasing the frequency of the processor has been found to be more effective than designing for a reduced CPI at a lower frequency to meet the performance goals.

One of the techniques applied to increase frequency is to add additional pipeline stages. In other words, the depth of the combinational logic, which can be measured by the number of fanout-4, balanced (FO4) inverters that fit in a cycle decreases for each generation of processor designs [28]. This eases the testing of the combinational logic but it increases the number of latches. Furthermore, to meet a high frequency target all kinds of physical aspects have to be taken into account from the beginning. For example, wiring delays can no longer be neglected and can take up a substantial part of the cycle. Hence, processor architects, circuit designers as well as the physical designers play an equally important role from the start. The high frequency design method in use by IBM therefore consists of a top-down definition phase followed by a bottom-up implementation phase.

The first phase of the high frequency process is the partitioning of the design into units followed by the partitioning of the design into macros. A floorplan is developed concurrently with the partitioning of the design into units/macros, and the location of the pins of the macros and their timing specification is defined. Furthermore, so-called cross sections are developed at the circuit level that implement the critical paths of the critical macros. This is done to investigate if the timing specification can be met. So at the end of the first phase a validated floorplan results in which the design has been partitioned into units/macros. Each macro has a so-called "contract" in which the size, pin location, power consumption, test strategy, custom or synthesis implementation method etc. has been specified. In this way, it is possible to design each macro to a large extent in isolation (and in parallel) with respect to all other macros.

After the definition phase the implementation of the macro starts by writing a VHDL (or Verilog) description for the macro. In parallel, the custom macros are implemented on the transistor level. Since each macro has a contract, it is possible to verify if each macro meets the specification of its contract independent of the rest of the processor. Hence, all requirements are verified for the macro individually including the testability of each macro.

Once that the implementation of all macros has been completed, the same process continues at the unit level/chip level. Furthermore, several iterations are introduced in which the macro contracts can be modified to optimise the overall processor frequency

and to get a balanced design in which all macros are equally critical.

In such a design method it is important that the testing for all stuck-at faults as well as delay faults can be addressed first at the macro level instead of at the unit/chip level only. Once the chip level implementation becomes available, mostly the timing becomes predominant and less time is left to address testing problems. Addressing the stuck-at faults at the macro level is relatively easy since in high frequency designs the logic cone depth is limited and the test for the stuck-at faults can be applied through the macros' scan latches that are either at their inputs or outputs. In other words, running test generation for the macro in isolation gives a good indication for the stuck-at fault detection for the macro at the unit/chip level. However, for delay faults this characteristic is no longer valid since a two pattern test is required and it depends on the scan path ordering of other macros driving the macro inputs if such a two pattern test can be applied.

The design methodology described above was for example used to design a high frequency instruction window buffer (IWB) core for an out-of-order processor. This IWB processor core will be discussed in more detail in Section 4.

3. Incorporating DfT for delay faults into the design method

3.1. Overview

Figure 1 shows a part of the design and DfT flow, the bold lines identify the additional DfT steps. The first new step in the hierarchical approach is a testability analysis and test pattern generation for each macro in isolation. This gives the delay fault coverage for the macro if there are no external constraints so that all two pattern tests can be applied. The next step is the composition of a so-called *interface conflict matrix (ICM)* which specifies constraints on the scan path order(s) of neighbouring macros. The ICMs are added to the macro contracts and are thus made available to the designers of other macros. In the next iteration, for each macro the ICMs of its neighbours are combined to a *combined conflict matrix (CCM)*, based on which the scan elements are reordered locally at macro level. The following sections describe the additional DfT steps. For the time being two levels of hierarchy (macro level and design level) are assumed. The approach can be extended to more levels, but this is not covered in detail in this paper.

3.2. Properties of the macros

The design flow as described in Section 2 implies the following properties of the design:

1. The design is partitioned into macros.
2. Each macro contains latches at all outputs (see Figure 2).
3. All latches are lined up to a scan chain.

The hierarchical reordering method makes use of these properties in order to improve the delay fault testability when using scan shifting. No restrictions are imposed on internal latches, which are not connected to an output. Their ordering within a scan chain can be determined while designing the macro itself, they do not depend on any other macro.

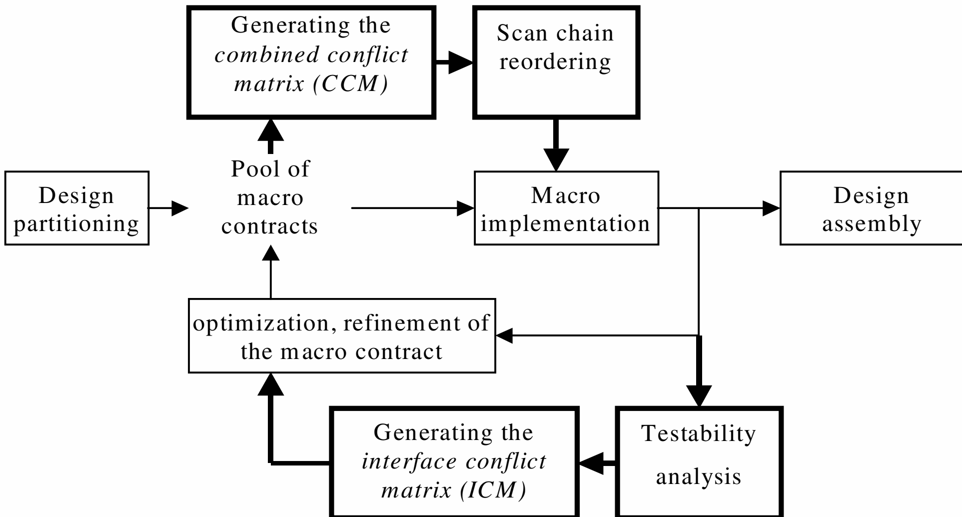

Figure 1: Integration of the DfT steps into the macro design flow

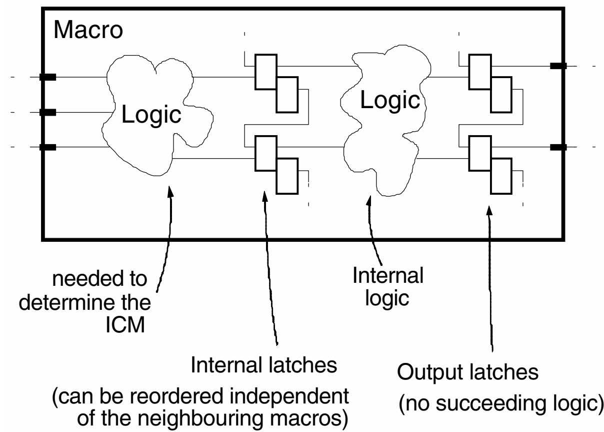

Figure 2: Structure of a macro

The partitioning of the circuit in this way enables the ordering of latches early in the design process. For the inputs of each macro, the interface conflict matrix (ICM) is computed as described in the next section. It summarizes the restrictions that this macro imposes on the scan chain of any preceding macro in order to be completely delay fault testable. The ICM is added to the interface description of the macro, which may additionally contain layout information, timing specifications, pin placement information etc. (see Figure 3).

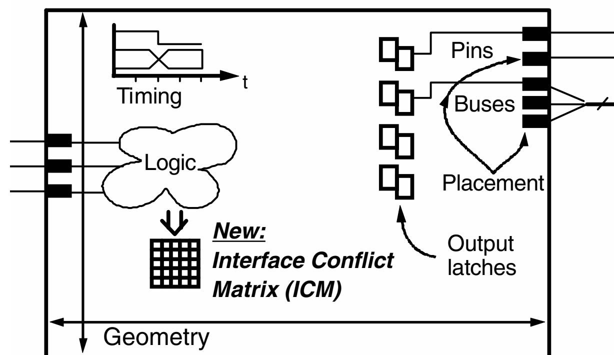

Figure 3: Elements of a macro contract

3.3. Computing the interface conflict matrix

Ideally, two consecutive latches do not feed the same logic making them independent of each other. We use a conflict matrix [24, 26, 27, 29] to model the dependencies between scan path latches. Each row and each column of the matrix refers to one latch of the scan path. If the value of the entry at row X and column Y of the matrix is larger than zero then problems are expected if we put latch Y after latch X into the scan chain, i.e. there are some faults, which cannot be detected. The higher the value the more faults are expected to be not detectable. Figure 4 shows an example of a conflict matrix and the estimated number of conflicts for two scan path orderings.

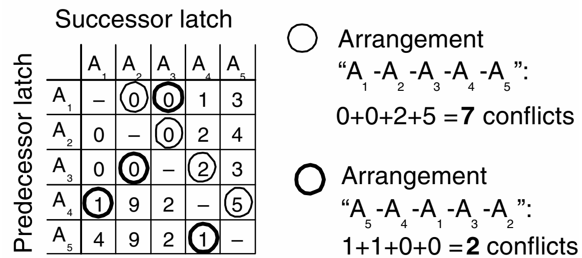

Figure 4: Example for the evaluation of a latch order

The quality of any given latch arrangement can be approximated and efficient latch arrangements can be calculated with a conflict matrix. It can be constructed in several ways, which are more or less precise. Firstly logic cones [27, 29], which consist of all paths from a certain starting or ending point, can be intersected, and latches whose intersection of output cones is nonempty are considered to cause a conflict. Another option is to calculate a correlation measure of each pair of latches [26]. This takes into account at which position in the circuit different paths meet and how many other paths are involved there.

We used test patterns [24] to construct an interface conflict matrix. Given a pattern set targeting delay faults, for each pair of latches the number of test pattern pairs is counted which cannot be generated in the circuit by scan shifting at the given ordering of the latch pair. A pattern pair cannot be generated if the value of some latch in the initialisation pattern and the value of its successor latch in the activation pattern are incompatible, i.e. '0' and '1' or vice versa.

Several different kinds of test patterns can be used to construct a conflict matrix. The most precise results are achieved if a dedicated test pattern is generated for every single fault modelled in the circuit.

Generating test patterns for every fault however takes a long time, even if faults are dropped that are detected by a pattern, which has already been generated for another fault. One approach for reducing the size of the pattern set is to generate deterministic patterns for hard-to-detect faults first. Those faults may require pattern pairs with many specified bits, which in turn may detect many other faults that can be excluded from pattern generation. The hard-to-detect faults are determined by computing signal probabilities. Our experiments have shown that the gain in speed is small

and the calculated scan orderings sometimes leads to worse results than the ones calculated on the basis of test patterns for every single fault.

A faster way is to use compacted test patterns that detect several faults at once. They are constructed by merging as many test patterns for single faults as possible into one pattern and by filling up the remaining "don't care" positions with random values. However, a lot of non-existing conflicts are counted in the conflict matrix due to the pattern compaction and the replacements of don't cares.

We have used both compacted and single test patterns in our experiments.

3.4. Local combination of multiple conflict matrices

After the interface conflict matrices of all macros have been calculated, for each macro the *combined conflict matrix (CCM)* is computed based on the ICMs of the succeeding macros, and an optimal ordering of the output latches is calculated using this combined conflict matrix (see Figure 5). The initial combined conflict matrix is set to all zeros. Then parts of the interface conflict matrix of each succeeding macro are added to the combined conflict matrix according to the connections between both macros.

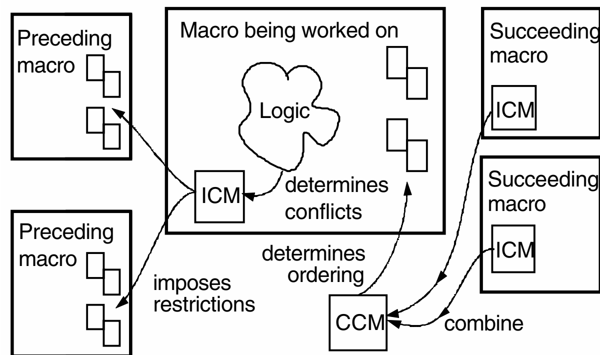

Figure 5: Relationships between the conflict matrices

Assume the current macro has n outputs labelled $O_1, \ldots, O_n$, and the inputs of a succeeding macro are labelled $I_1, \ldots, I_m$, where m can be different for each succeeding macro. The combined conflict matrix CCM is an $n \times n$ matrix and an interface conflict matrix ICM is a $m \times m$ matrix. The rows and columns are marked with the names of the inputs or outputs they belong to. Each interface matrix ICM is converted into an $n \times n$ addition matrix ICM' depending on the connections between the current macro and the succeeding macro:

$$ICM'[i,j] = \begin{cases} ICM[k,l] & \text{if } O_i = I_k \text{ and } O_j = I_l \\ 0 & \text{otherwise} \end{cases}$$

The addition matrices of all succeeding macros are added to the combined conflict matrix. Figures 6 and 7 sketch an example for a macro with two succeeding macros.

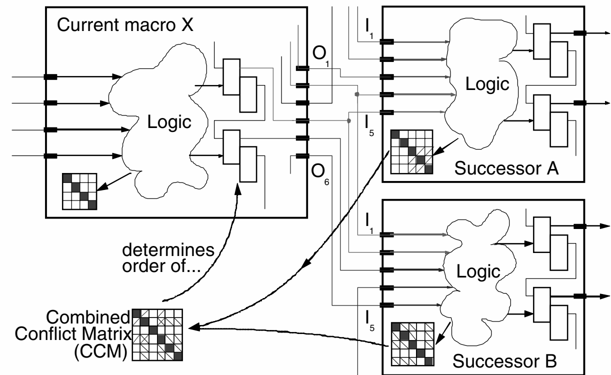

Figure 6: Example for a macro connected to two successors

3.5. Calculating a scan path ordering

The problem of finding an optimal scan path ordering for a given conflict matrix is equivalent to a travelling salesman problem and is thus NP-complete [25]. In our experiments we used three different heuristics.

The *"greedy serial"* algorithm [24] constructs the scan chain incrementally by starting with the first latch and subsequently appending new latches to the current chain which show the smallest number of conflicts with the current tail latch of the chain.

The *"greedy hardest first"* algorithm iteratively selects a previously unprocessed latch with a maximum sum of conflicts (row or column). This latch is considered to be "hard" and is connected to another latch with the least number of conflicts as long as no cycle in the scan chain is introduced.

The *"greedy least conflicts"* algorithm sorts all entries of the conflict matrix and incrementally connects latches with the smallest number of conflicts as long as no cycles in the scan chain are introduced.

The above methods can be extended such that additional latches are inserted if the number of conflicts cannot be reduced sufficiently by reordering only. However, additional scan latches increase the chip area and were not necessary in our case.

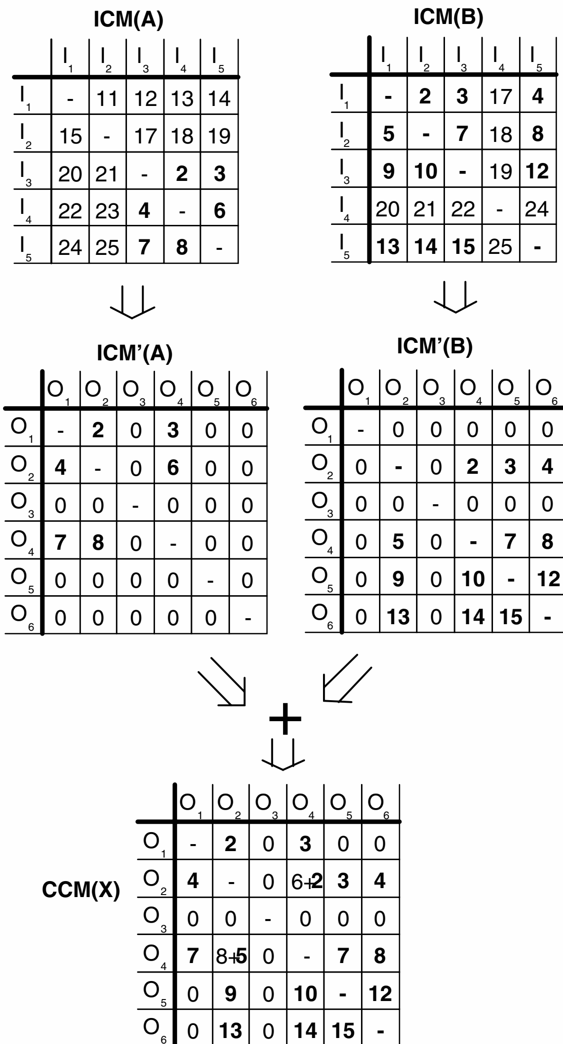

Figure 7: Determining a combined conflict matrix based on the interface matrices of succeeding macros

4. Case study: instruction window buffer

The instruction window buffer (IWB) [30] is shown in Figure 8. It was partitioned into 9 full-custom macros. Each cycle, up to 4 dispatched instructions are renamed and allocated in consecutive order (with wrap-around) in the 64 entry IWB. The rename process is done for each of the three source operands of an instruction. After renaming, the dispatch process stores two of the three sources in the "RS src data" macro at the allocated entries. These two sources each contain 64 data and 8 parity bits. The third source operand ("src") is stored in the "RS condition code" (cc) macro containing a 2 bit condition code field and a parity bit. Up to 4 instructions can be issued in each cycle to the instruction execution units (IEUs). The issue process is controlled by the "RS select" macro containing a window manager and priority filter functionality that selects the oldest instructions for which all sources are available out of the active IWB entries. For load instructions the address is calculated by one of the 3 fixed point IEUs. Data returned from the data cache ("D$") or the load store unit (LSU) is aligned by one of the 4 storage execution units (SEUs). Up to 8 results (64 bit data, 2 bit condition code) are returned in each cycle. The writing of result data into the "RS/ROB data" macros is controlled in a data flow approach by comparing the result tag with the tag of each "ROB result"/"RS operand" field. Finally, the renaming state is check pointed for a partial removal of the instructions for miss-predicted branches.

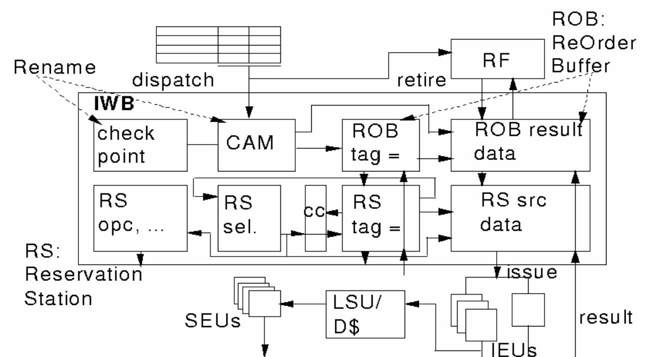

Figure 8: Structure of the instruction window buffer (IWB)

The macros making up the critical path are the "RS src data", "RS tag compare" ("RS tag =") and the "RS select" macro that consists of 4 issue filters and the window manager. These macros have been fabricated on a test chip [30].

5. Experiments

When applying our hierarchical DfT method on the instruction window buffer we concentrated on the timing-critical macros which were "RS src data", "RS tag compare", "RS issue filter" and "RS window manager" (see above). All those macros are completely testable with respect to stuck-at faults.

In a first series of experiments we investigated the scan chain reordering technique described in Sections 3.3 and 3.5. The results are shown in Table 1. For each macro we determined the achievable gate delay fault coverages using the original, unmodified scan chain ordering (column "Unmodified scan path"), the fault coverages for the macro in isolation assuming unconstrained access to the macro inputs

("Unconstrained access") and the fault coverages after applying our scan chain reordering technique. For all these cases we used *TestBench*[1] to generate three types of test patterns. Pseudo-random and weighted random patterns can be generated on-chip when using BIST, but these types of patterns cannot guarantee the detection of all faults. Deterministic patterns are precomputed patterns for all faults, so fault coverage numbers below 100% indicate the presence of untestable faults due to circuit redundancies or scan dependencies.

Macro	Patterns	Org.	Unconstr. access	After reordering
RS window mgr.	pseudo-random	98,00%	99,92%	99,52%
	weighted random	98,00%	99,92%	99,61%
	det.	98,50%	100,00%	99,61%
RS issue filter	pseudo-random	10,75%	11,90%	15,87%
	weighted random	73,81%	100,00%	100,00%
	det.	75,10%	100,00%	100,00%
RS tag comp.	pseudo-random	38,93%	39,04%	39,45%
	weighted random	98,84%	99,02%	99,03%
	det.	99,39%	99,62%	99,62%
RS src data	pseudo-random	82,05%	82,41%	82,66%
	weighted random	99,89%	99,93%	99,93%
	det.	99,97%	100,00%	100,00%

Table 1: Delay fault coverage for the critical macros

For deterministic patterns, the "unconstrained access" numbers represent upper bounds for the achievable fault coverage, and in three of the four macros this upper bound has been achieved after the reordering. For the "RS window manager", our heuristics failed to calculate an optimal ordering due to a small number of inputs (only 13). For the "RS tag compare" macro, even independent inputs do not achieve complete fault coverage because of the ordering of an internal scan chain, which has not been changed in our experiments.

It is noticeable that by the scan chain reordering also the random pattern fault coverage has been increased. By using weighted random patterns all macros are almost completely testable.

The complexity of the procedure for computing the interface conflict matrices depends on the way the test patterns are generated that are used to determine possible conflicts. Computing a compact test is less time consuming but also less exact than generating single patterns for every fault (see Section 3.3). Table 2 shows a comparison between these two approaches when generating deterministic patterns. For each case we tried the heuristics "greedy serial", "greedy hardest first", and "greedy least conflicts" for computing a scan path ordering out of a conflict matrix (see Section 3.5), and the best result together with the corresponding heuristic are listed in the table.

In all cases with single patterns per fault better results are obtained than by using compact test sets. This is due to the don't care bits which have been replaced by random values in the compact test set and from which non-existing conflicts are derived.

Macro	Single pattern	Compact test set
RS window manager	99,62% (hardest first)	98,31% (least conflicts)
RS issue filter	100,00% (serial)	97,30% (least conflicts)
RS tag compare	99,63% (hardest first)	99,58% (least conflicts)
RS src data	100,00% (serial)	99,99% (hardest first)

Table 2: Impact of the ICM computation on the delay fault coverage

A comparison between the three ordering heuristics has shown that the "greedy serial" and the "greedy hardest first" algorithms produce equally good results with test patterns for every single fault. If compacted test patterns are used for calculating a conflict matrix then the "greedy least conflicts" algorithm is the best choice.

Figure 9 sketches the connections between some of the macros. In order to calculate the fault coverage for the Issue Filter 3 the scan paths in the preceding macros, which are the "RS window manager" and the "RS tag compare", have to be reordered. For reordering these scan paths the interface conflict matrices (ICMs)

[1] *TestBench* is a UNIX[2]-based set of test design automation tools developed by IBM for internal use. It was made commercially available in 1994 by the IBM Microelectronics Division.

[2] Trademark or registered trademark of the Open Group or X/Open Company Ltd.

of their successor macros, which are the "issue filters" 0 through 3, the "RS src data" 0 and 1 and another macro, have to be combined using the method described in Section 3.

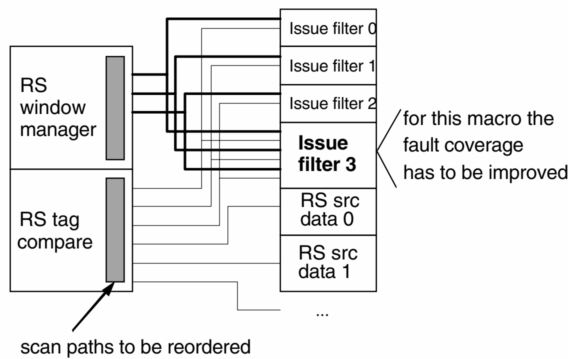

Figure 9: Connections between the example macros

Table 3 shows the results for the "Issue filter 3" with different scan path orderings in its preceding macros. Similar to Table 1, the column "Unmodified scan path" shows the original fault coverage, the next column shows the achievable fault coverage of the macro with unrestricted access, and the last column shows the results after applying the hierarchical reordering technique. Again, we determined the fault coverage achievable by pseudo-random, weighted random and deterministic patterns for each case.

Patterns	Unmodified scan path	Unconstr. access	After reordering
pseudo-random	3,30%	3,36%	3,15%
weighted random	52,43%	70,33%	80,83%
deterministic	68,84%	100,00%	99,88%

Table 3: Delay fault coverages in "Issue filter 3" after reordering the scan chains in its preceding macros

With the reordered scan path almost the best possible fault coverage is achieved which is far better than the initial fault coverage before reordering the scan path. In this example, the fault coverage achieved by weighted random patterns is even better for the reordered scan path than for unconstrained inputs.

6. Conclusion

In this paper we presented a new design-for-test method that hierarchically reorders the scan path latches for improving the delay fault testability by using scan shifting. An arrangement of the latches in the scan path is calculated by first constructing an interface conflict matrix (ICM) as part of each macro contract. Next a hierarchical approach is used whereby the scan chain order of a macro is calculated by combining the ICM's of all macros connected to the outputs of the macro. Ordering the scan path with this approach has the advantage that the delay fault testability can be addressed for each macro individually and that therefore the complexity of the NP-complete scan chain reordering problem remains limited. Furthermore, it can be applied during the design phase before the complete chip netlist becomes available.

The IWB case study showed that the reordering of scan chains using the proposed approach is very efficient. The delay fault coverage was improved significantly and was close or even equal to the upper bounds obtained for the unconstrained scan path case assuming that any generated two pattern test can be applied. The method was described for a two level hierarchy, however it can be extended to a multi level hierarchy as well.

The main restriction of the proposed approach is that the design has to be partitioned into macros of which all macro outputs are directly latch outputs (buffers/inverters between latches and outputs are allowed). Our experience is that high frequency processor designs are often partitioned in such macros. However, further research will concentrate on how macros without an output latch boundary can be handled as well and how the ICM calculation method can be further refined by taking into account the macro connections as available after the initial top down design phase.

References

[1] Semiconductor Industry Association (SIA), International Technology Roadmap for Semiconductors (ITRS), 1999 edition.

[2] E.B. Eichelberger, T.W. Williams: "A Logic Design Structure for LSI Testability", Proc. 14th Design Automation Conference, 1977, pp. 462-468

[3] G. Hetherington, T. Fryars, N. Tamarapalli, M. Kassab, A. Hassan, J. Rajski: "Logic BIST for Large Industrial Designs: Real Issues and Case Studies", Proceedings of the International Test Conference, pp. 358-367, 1999

[4] G. Kiefer, H. Vranken, E. J. Marinissen, H.-J. Wunderlich: "Application of Deterministic Logic BIST on Industrial Circuits", Proceedings of the IEEE International Test Conference, 2000, pp. 105-114

[5] Z. Barzilai, B. Rosen: "Comparison of AC Self-Testing Procedures", Proceedings of the International Test Conference, pp. 89-94, 1983

[6] G.L. Smith: "Model for Delay Faults Based Upon Paths", Proc. International Test Conference, 1985, pp. 342-349

[7] P.H. Bardell, W.H. McAnney: "Parallel Pseudo-random Sequences for Built-In Test", Proceedings International Test Conference, IEEE, 1984, pp. 302-308

[8] J. Rajski, N. Tamarapalli, J. Tyszer: "Automated Synthesis of Large Phase Shifters for Built-In Self-Test", Proceedings of the International Test Conference (ITC) 1998, pp. 1047-1056

[9] J.P. Hayes, A.D. Friedman: "Test Point Placement to Simplify Fault Detection", IEEE Transactions on Computers, Vol. C-33, July 1974, pp. 727-735

[10] N. Mukherjee, T. Chakraborty, S. Bhawmik: "A BIST Scheme for the Detection of Path-Delay Faults", Proceedings of the International Test Conference, 1998, pp. 422-431

[11] B.H. Seiss, P.M. Trousborst, M.H. Schulz: "Test Point Insertion for Scan-Based BIST", Proceedings of the European Test Conference, IEEE, 1991, pp. 253-262

[12] N. Tamarapalli and J. Rajski, "Constructive Multi-Phase Test Point Insertion for Scan-Based BIST," Proceedings International Test Conference, IEEE, 1996, pp. 649-658

[13] F. Muradali, V.K. Agarwal, B. Nadeau-Dostie: "A New Procedure for Weighted Random Built-In Self-Test", Proceedings of the International Test Conference, pp. 660-669, 1990

[14] J.A. Waicukauski, E. Lindbloom, E.B. Eichelberger, O.P. Forlenza: "A Method for Generating Weighted Random Test Patterns", IBM Journal of Research & Development, Vol. 33 (2), pp. 149-161, 1989

[15] H.-J. Wunderlich: "Self Test Using Unequiprobable Random Patterns", Proceedings of the 17th International Symposium on Fault-Tolerant Computing, pp. 258-263, 1987

[16] S. B. Akers: "On the use of Linear Sums in Exhaustive Testing", Proc. Of the 15th Int. Symp. On Fault-Tolerant Computing (FTCS), 1985, pp. 148-153

[17] S. Hellebrand, H.-J. Wunderlich, O. F. Haberl: "Generating Pseudo-Exhaustive Vectors for External Testing", Proc. IEEE Int. Test Conf. (ITC), 1990, pp. 670-679

[18] L.T. Wang, E.J. McCluskey: "Circuits for Pseudo-Exhaustive Test Pattern Generation", Proceedings of the International Test Conference, 1986

[19] S. Hellebrand, B. Reeb, S. Tarnick, H.-J. Wunderlich: "Pattern Generation for a Deterministic BIST Scheme", Proceedings International Conference on Computer-Aided Design, IEEE, 1995, pp. 88-94

[20] G. Kiefer, H.-J. Wunderlich: "Deterministic BIST with Multiple Scan Chains", Proceedings International Test Conference, IEEE, 1998, pp. 1057-1064

[21] N.A. Touba, E.J. McCluskey: "Altering a pseudo-random bit sequence for scan-based BIST", Proceedings International Test Conference, IEEE, 1996, pp.167-175

[22] H.-J. Wunderlich, G. Kiefer: "Bit-Flipping BIST", Proceedings International Conference on Computer-Aided Design, IEEE, 1996, pp. 337-343

[23] J.A. Waicukauski, E. Lindbloom, B. Rosen, V. Iyengar: "Transition Fault Simulation", IEEE Design and Test, April 1987, pp. 32-38

[24] K.-T. Cheng, S. Devadas, K.Keutzer: "A Partial Enhanced-Scan Approach to Robust Delay-Fault Test Generation for Sequential Circuits", Proceedings of the International Test Conference, pp. 403-410, 1991

[25] W. Mao, M.D. Ciletti: "Arrangement of Latches in Scan-Path Design to Improve Delay Fault Coverage", Proceedings of the International Test Conference, pp. 387-393, 1990

[26] W. Mao, M. D Ciletti: "Correlation-Reduced Scan-path Design to improve Delay Fault Coverage", 28th ACM/IEEE Design Automation Conference, pp. 73-79, 1991

[27] J. Leenstra, M. Koch, T. Schwederski: "On Scan Path Design for Stuck-Open and Delay Fault Detection", IEEE European Test Conference, pp. 201-210, 1993

[28] D. H. Allen et al.: "Custom Circuit Design as a Driver of Microprocessor Performance", IBM Journal of Research and Development, Vol. 44, No. 6, Nov. 2000, pp. 799-822

[29] J. Savir, R. Berry: "At-Speed Test is not necessarily an AC Test", Proceedings of the International Test Conference, pp. 722-728, 1991

[30] J. Leenstra et al.: "A 1.8GHz Instruction Window Buffer", IEEE International Solid-State Circuits Conference (ISSCC), 2001, pp. 314-315

[31] S. DasGupta, R. G. Walther, T. W. Wiliams, E. B. Eichelberger: "An Enhancement to LSSD and some Applications of LSSD in Reliability, Availability and Serviceability", Dig. 11th Annual Int. Symp. on Fault-Tolerant Computing, pp. 32-34, 1981

[32] B. I. Dervisoglu, G. E. Strong: "Design for Testability: Using Scanpath Techniques for Path-Delay Test and Measurement", Proceedings of the International Test Conference, pp. 365-374, 1991