

Sequential ATPG Using Combinational Algorithms

Miron Abramovici

Circuits and Systems Research Lab
Agere Systems - Murray Hill, NJ
miron@research.bell-labs.com

Xiaoming Yu and Elizabeth Rudnick

Center for Reliable and High-Performance Computing
University of Illinois - Urbana, IL
{xiaoming,liz}@crhc.uiuc.edu

Abstract: We use a DFT technique based on clock freezing and clock partitioning, to model a sequential circuit as a set of overlapping pipelines. Our DFT technique does not introduce any delay penalty and has small area overhead. We present a novel sequential ATPG algorithm that uses this model to detect most faults by using combinational techniques. Preliminary results are encouraging.

1. Motivation

Automatic test-pattern generation (ATPG) for sequential circuits is an extremely expensive computational process, so that ATPG algorithms working on complex circuits can spend many hours of CPU time and still obtain poor results in terms of fault coverage. Among the factors that make this problem difficult [Abramovici et al. 1990] are:

- the need to work with a model consisting of an iterative array of time-frames whose number is, in the worst case, an exponential function of the number of flip-flops (FFs) in the circuit;
- the existence of illegal states, which may cause the ATPG algorithm to waste a lot of time trying to justify them;
- the existence of untestable faults, which the ATPG algorithm identifies as untestable by failing to generate test sequences that detect them; for each untestable target fault the algorithm must complete an exhaustive search.

Because of the difficulty of the sequential ATPG problem, the electronics industry has given up the idea that complex circuits can be tested without intrusive design for testability (DFT) techniques, such as scan design, which significantly modify the design to make its buried FFs more controllable and observable in test mode. However, scan-type DFT techniques introduce delay penalties resulting in performance degradation, and area overhead causing additional power consumption and decreased yield; moreover, scan tests are not directly compatible with at-speed testing.

In this paper, we use a DFT technique based on clock control and we introduce a novel testing paradigm that allows the circuit to be modeled as a set of overlapping pipelines. We also present a new ATPG algorithm that can

take advantage of the resulting structure, so that most faults are detected with combinational techniques, which are several orders of magnitude faster than the sequential ones. Our DFT technique does not introduce any delay penalty, has small area overhead, and the generated tests may be applied at speed. Our preliminary results are encouraging.

2. Relevant Prior Work

In the conventional way to test a sequential circuit, the clock is activated and the state is changed with every applied input vector, so in any state only one vector is applied. But there may be several vectors that detect different faults in the current state. For example, several fault effects could have been propagated to flip-flops (FFs), and they may be propagated to primary outputs (POs) by different vectors. Similarly, the current state may be necessary to activate several other faults that may need different vectors to propagate their fault effects to POs. But with conventional testing, we can apply only one vector before the state is changed, so many fault-detection opportunities may have to wait for the next time when the current state will be reached. The *clock freezing* method, introduced in [Abramovici et al. 1992] and further enhanced in [Santoso et al. 1999], temporarily suspends the sequential behavior of the circuit by freezing its clock, and several vectors may be applied without changing the current state. These vectors can be generated only by combinational ATPG algorithms. In this way, the current state is fully exploited before it is changed. This method does not require any DFT. However, in general, a circuit has many more FFs than primary inputs (PIs), and because all FFs are frozen, the ATPG algorithm working with one time-frame is quite constrained in what it can accomplish. After all the faults that can be detected in the current frozen state have been targeted, the sequential behavior of the circuit is enabled again, and the application of the next vector is accompanied by a clock pulse. Note that clock freezing does not change the difficulty of controlling the state transitions of the sequential circuit.

In the normal operation of a sequential circuit, the same clock usually controls a large number of FFs. *Clock partitioning*, introduced in [Agrawal et al. 1991] and further developed in [Einspar et al. 1993][Baeg and Rogers 1993][Einspar et al. 1996][Rajan et al. 1996][Einspar et al.

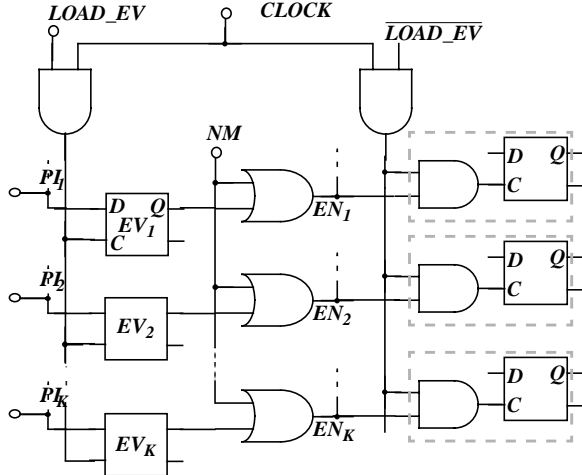


Figure 1. Clock control logic

1999], is a DFT technique that divides the FFs sharing the same clock into several groups, so that in test mode each group can be independently clocked. Figure 1 illustrates the clock control logic, derived from the techniques presented in [Rajan et al. 1996]. In normal mode $NM=1$ and $LOAD_EV=0$, so that $CLOCK$ propagates to all FFs. To avoid clock skewing, clock gating occurs within every FF, so the routing of $CLOCK$ to FFs is not changed. (FFs that are always clocked will have their enabling input tied to logic 1.) In test mode $NM=0$, and the clock propagation is under the control of K enabling signals EN_i . The FFs controlled by the same enabling signal are in the same group (Figure 1 shows only one FF per group). The enabling values EV_i that determine the clock partitioning in test mode are stored in a separate set of K FFs. These FFs are loaded from a set of K PIs when the $LOAD_EV$ signal is asserted. At the same time, the clocking of all other FFs is disabled, so the K PIs that supply the enabling values may be arbitrarily selected. This scheme can provide a large number of enabling signals with only one additional pin.

For a circuit with N FFs, the main component of the area overhead of this DFT technique is the N clock-enabling AND gates, which is less than the area overhead in scan design. This overhead is not needed if clock skewing in test mode can be controlled [Rajan et al. 1996]. In addition, we have K FFs and K OR gates, but usually $K \ll N$, so this overhead is negligible. Note that the operating frequency of the circuit is not affected by the clock control mechanism, since no delays are introduced in data paths. Unlike scan design, clock control DFT is fully compatible with at-speed testing.

Clock partitioning increases the testability of the sequential circuit, by reducing dependency among FF values and introducing many more state transitions in the state transition graph, thus making states that are illegal or difficult to reach easier to reach in test mode. As a result, some faults that were impossible or difficult to detect in the original cir-

cuit become easier to test. Clock partitioning is also useful for delay-fault testing [Fang and Gupta 1994]. Although the sequential circuit with partitioned clocks is easier to test, ATPG for the transformed circuit still requires sequential algorithms.

3. The New Approach

We combine clock partitioning with clock freezing to decompose the original circuit into several overlapping pipelines, so that the test generator works with one pipeline at a time. The main advantage is that all or most faults in a pipeline can be detected only by combinational algorithms [Gupta et al. 1990].

To test a fault in the pipeline shown in Figure 2a, we apply one input vector, then we clock twice; as a result, the effect of the input vector propagates throughout the entire circuit. Since the registers serve only to transmit the data, we can make them *transparent* for ATPG, and model the pipeline by a combinational circuit as shown in Figure 2b. Thus the application of every combinational vector generated on this model has to be accompanied by clocking of all transparent registers; the number of required clocks is equal to the sequential depth of the pipeline.

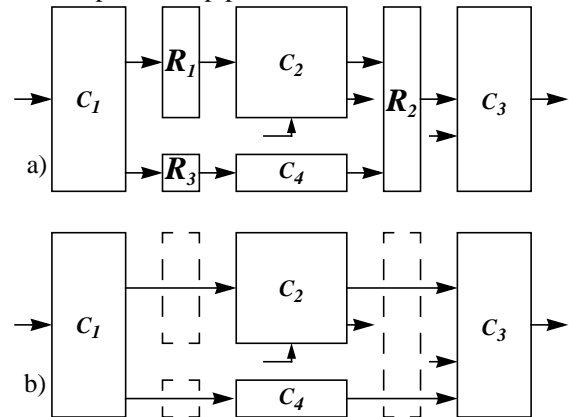


Figure 2. Pipeline and ATPG model

The pipeline in Figure 2 is *balanced*, as every path between two combinational blocks C_i and C_j goes through the same number of registers. In a balanced pipeline, a combinational ATPG algorithm can detect every fault detectable in the original circuit. In an unbalanced pipeline, some faults untestable in the combinational model may be detected in the original pipeline.

We will illustrate our approach using the circuit shown in Figure 3a [Gupta et al. 1990] as an example. Here we do clock partitioning by allocating an independent clock to each register in the block diagram. To create pipelines we have to cut all the feedback loops in the original circuit. Any loop-cutting algorithm used to select scan FFs for partial scan design can be used here. Instead of cutting a feedback loop

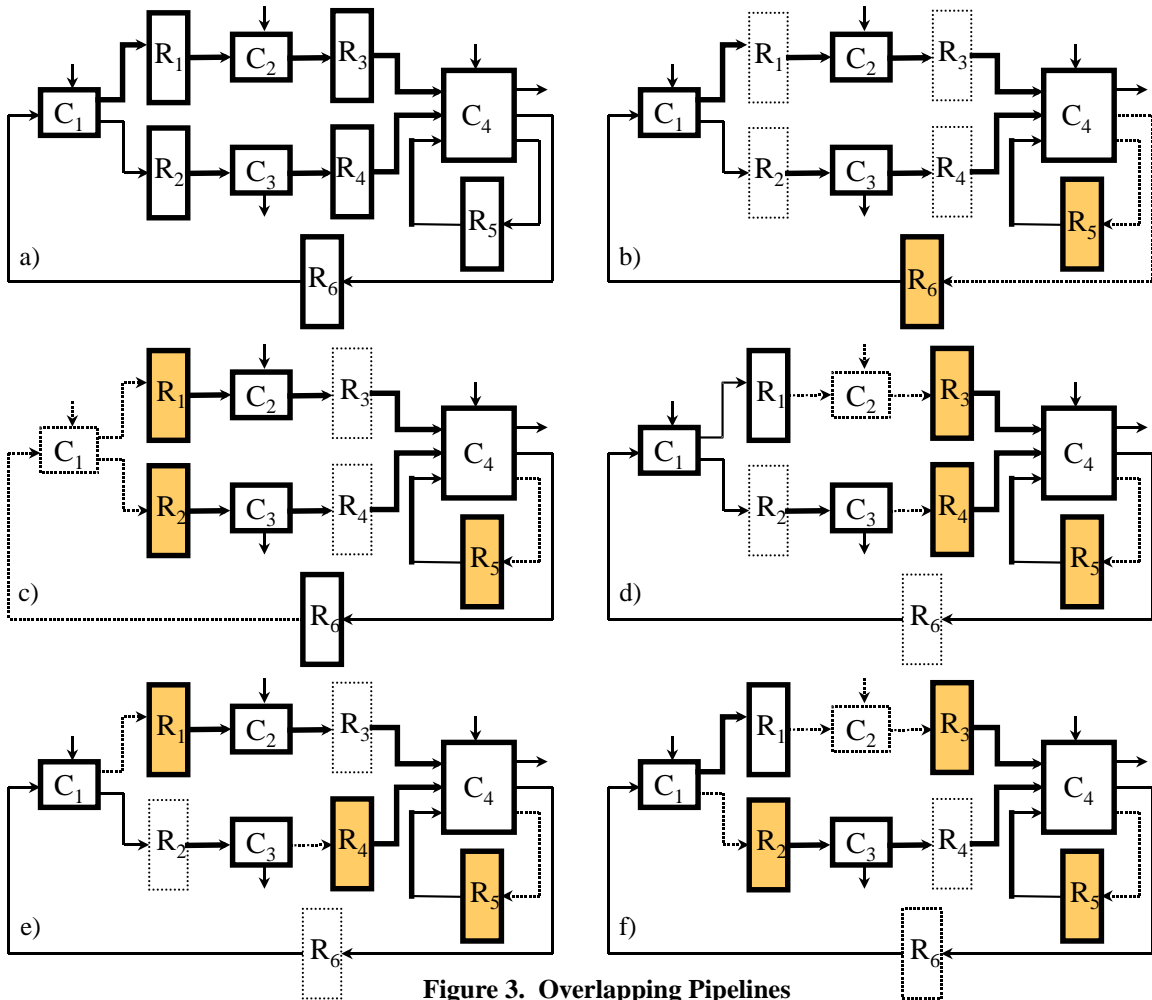


Figure 3. Overlapping Pipelines

by scanning registers, we temporarily freeze the clocks for some of the registers along the loop; these registers are said to be *frozen* and are shown as dark in Figure 3. In this circuit we have five different ways to cut the loops, so we can create five different (balanced) pipelines. Note that R_5 is frozen in every pipeline, while any other register is frozen only in at most two pipelines. In the same way as partial scan can be used to create balanced pipelines [Gupta et al. 1990], we can also use selective clock freezing to achieve the same result.

The test generator works with one pipeline at a time. Given its frozen registers, the model for a pipeline is constructed as follows:

1. Treat the frozen FFs as PIs with frozen values (that cannot be changed by the test generator).
2. Exclude the “invisible” combinational logic that directly feeds only frozen registers.
3. Model any register that is not frozen and feeds POs (directly or indirectly) as transparent.
4. Any other register (not frozen and not transparent) is flagged as a *capture register*.

For example, in Figure 3c, R_1, R_2 , and R_5 are frozen registers, R_3 and R_4 are transparent, and R_6 is a capture register. A capture register feeds only invisible logic (C_1), so it cannot be used to detect faults in the current pipeline. But we can use R_6 to capture fault effects to be propagated in the future in a pipeline where it will be frozen (Figure 3b). Similarly, in Figure 3d, R_3, R_4 , and R_5 are frozen registers, R_2 and R_6 are transparent, and R_1 is a capture register. Figure 4 illustrates the model of a pipeline (the invisible logic is shown as shaded triangles); here one time frame corresponds to d conventional time frames, where d is the sequential depth of the pipeline. It is important to note that a register frozen in one

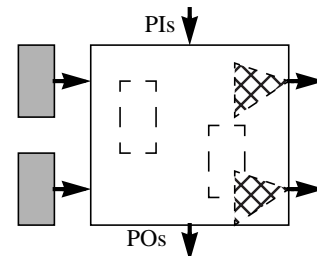


Figure 4. ATPG model with one time-frame

pipeline may be transparent in another one; thus applying a combinational vector in one pipeline may change the state of the frozen registers in all other pipelines.

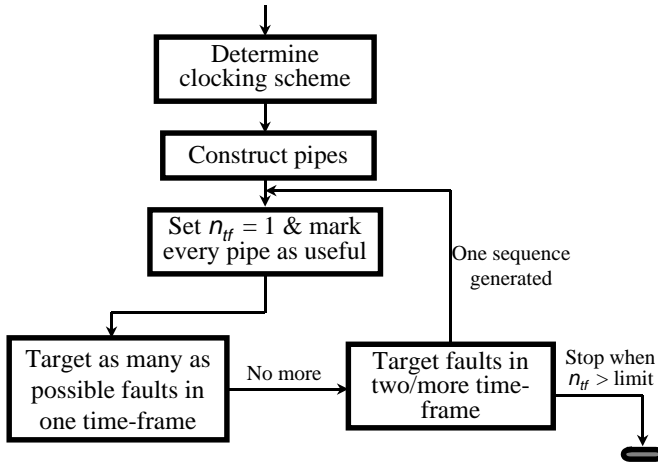


Figure 5. High-level structure of PIPEXPRESS

Figure 5 outlines the high-level structure of our algorithm, called PIPEXPRESS. The first block determines the clocking scheme by grouping together the FFs that will share one of the independent clocks. Then the pipelines are constructed as illustrated in Figure 3. n_{ff} represents the number of time frames in the current pipeline and is initially set to 1. A pipeline is said to be *useful* if it has undetected faults that may be detected in the presence of input values already assigned. Initially the only assigned input values are the values of the FFs in the frozen registers of the pipeline. PIPEXPRESS is opportunistic so that in every step it tries to detect as many faults as possible using the shortest possible test sequence. Test generation takes place in the two blocks at the bottom of Figure 5: the block on the left (detailed in Figure 6) generates as many as possible combinational vectors

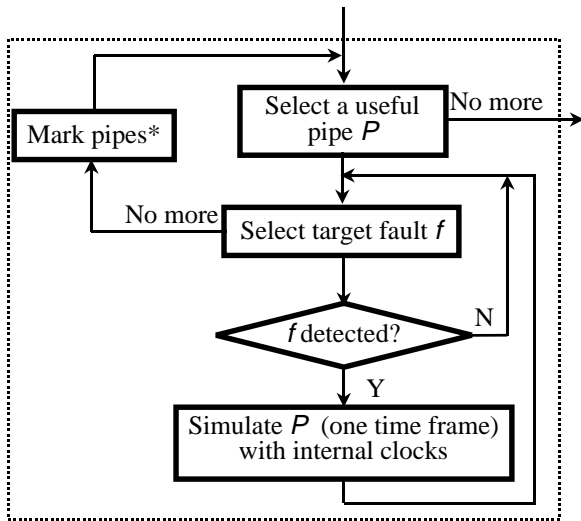


Figure 6. Generating combinational vectors

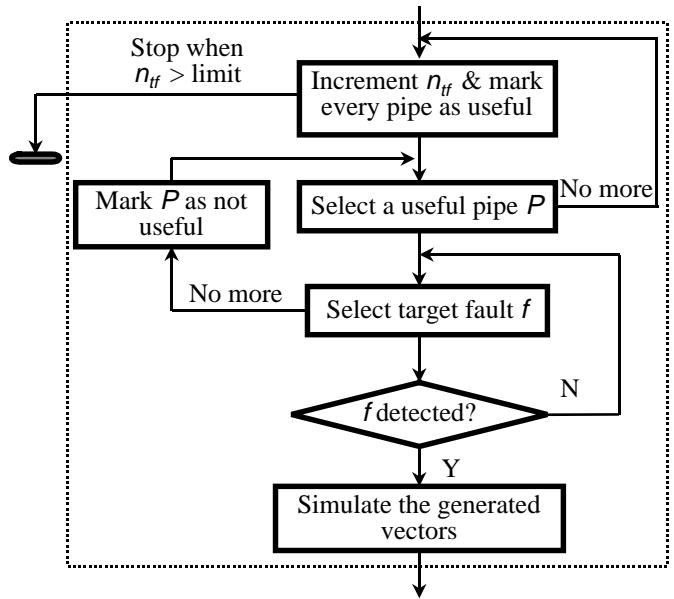


Figure 7. Generating a sequence

tors, and the block on the right (detailed in Figure 7) generates one sequence of two or more vectors. The entire process is repeated after one such sequence has been generated.

In Figure 6, PIPEXPRESS first selects a useful pipeline and generates as many test vectors as possible. Every such combinational vector is simulated, dropping the detected faults and keeping track of the faults still possible to detect in one time frame. When no more target faults are found, the current pipeline is marked as not useful. At the same time, any pipeline whose frozen register is a transparent or a capture register of the current pipeline is checked for usefulness, since its state has been changed, and new fault effects may be stored in its FFs or the new state may allow new faults to be detected. The entire process is repeated until no more useful pipeline can be found.

In Figure 7, PIPEXPRESS is first trying to generate a sequence of two vectors for a useful pipeline, using the model shown in Figure 8 for two time-frames (the number of vectors is equal to the number of time-frames n_{ff}). If not successful, n_{ff} is incremented and the entire process repeated. The transition between time-frames involves firing at least

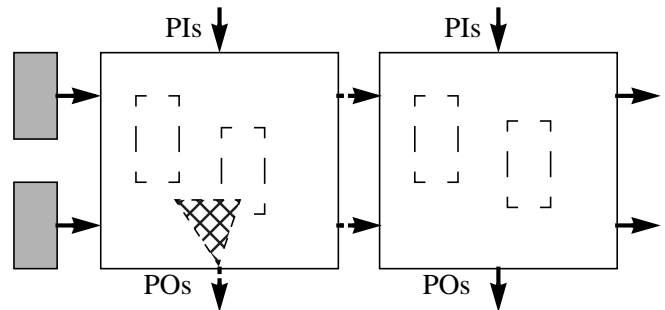


Figure 8. ATPG model with two time-frames

one of the frozen clocks of the pipeline. PIPEXPRESS determines the frozen clock(s) to be fired. Note that the state of a register that is frozen in every pipeline, such as R_5 in Figure 3, can be changed only in this way. Additional clock partitioning for the FFs of such a frozen register would increase the testability of the circuit. Like before, the application of a combinational vector in one time frame requires clocking of all the transparent registers of the pipeline, and we also clock the capture registers. When one sequence with n_{ff} vectors has been successfully generated, the state of the current pipeline, as well as of any other pipeline whose frozen register is transparent or a capture register in the current one, has been changed; then PIPEXPRESS returns to trying to generate only combinational vectors (Figure 6).

4. Implementation Details

While PIPEXPRESS works with one pipeline at a time, the generated vectors are expanded to account for clocking of the transparent registers, and the resulting sequences are fault simulated for the entire circuit using the PROOFS fault simulator [Niermann et al. 1992]. Note that in the test generation model, the POs are observed only after all the transparent registers have been clocked, but in the fault simulation model they are observed after every clock. This is equivalent with applying several random vectors in between the “real” vectors, and these vectors may detect additional faults. The results of PROOFS are ported back into the test generator to enable fault dropping and reporting the fault effects stored in registers. The two programs communicate via *sockets*.

PIPEXPRESS is similar to the FAST algorithm [Abramovici et al. 1986]. When working with several time frames, it borrows concepts from the FASTEST algorithm [Kelsey et al. 1993]. The most important is backtracing all objectives to PIs (or backtrace-stop lines) without stopping at time-frame boundaries; this scheme avoids the state justification problem and hence it never has to deal with illegal states.

The existence of frozen values at the PIs of a pipeline may preclude the activation and the observation of several still undetected faults; PIPEXPRESS removes these faults from the set of target faults for the pipeline, so that only potentially detectable faults will be targeted. Faults whose effects are stored in observable frozen FFs are also considered potentially detectable. The number of potentially detectable faults determines the *usefulness* of the pipeline. Following its opportunistic strategy, PIPEXPRESS always selects the most useful pipeline as the next one to work on.

As preprocessing steps, we use: 1) the algorithms FIRES [Iyer et al. 1996] and FUNI [Long et al. 2000] to identify combinational and sequentially untestable faults; 2) the combinational test generator ATOM [Hamazoglu and Patel 1998] running on a full-scan model of the circuit to identify the rest of the combinational untestable faults.

Every time we remove the identified untestable faults from the set of target faults. The same faults would be much more computationally expensive to target in test generation.

5. Preliminary Results

We compare a preliminary implementation of PIPEXPRESS with GATEST [Rudnick et al. 1997] and with a commercially available sequential ATPG that we will refer to as S_ATPG. Both PIPEXPRESS and GATEST were run on a Pentium III 550 computer with 256Mb memory, while S_ATPG was run on a SPARC SUNW Ultra-Enterprise computer; the two machines have comparable performance.

The circuits we used are *piir80* and *pcont2*, taken from [Chickermane et al. 1992]. *piir80* is an optimized finite-impulse response filter (see Figure 9), while *pcont2* is a controller circuit used in DSP applications. Table 1 gives the circuits data. For *piir80*, freezing the 8 FFs in the marked register is always required to cut the feedback loops, so we have a single pipeline with 8 frozen FFs and 48 transparent FFs. (Note that a single pipeline is the worst structure for our method, since all the faults in the invisible logic can be detected only using models with several time frames.) Each one of the frozen FFs has its independent clock, and the transparent registers are always clocked. Similarly, for *pcont2* we have a single pipeline, with 16 frozen FFs with independent clocks and 8 transparent FFs.

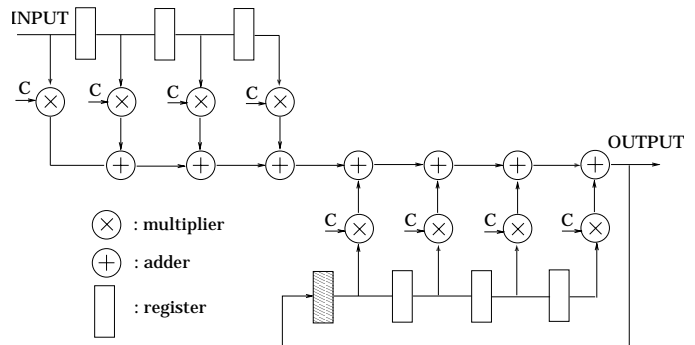


Figure 9. *piir80* block diagram (C = constant)

Table 1. Circuit data

Circuit	PIs	POs	Pipes	FFs	Frz.	Faults
<i>piir80</i>	9	8	1	56	8	19,936
<i>pcont2</i>	9	8	1	24	16	11,272

Table 2 summarizes the main experimental results and Table 3 presents details on the faults detected by PIPEXPRESS, showing the original number of faults, the faults identified as untestable in preprocessing, and the faults detected by sequences of different length. For *piir80*, PIPEXPRESS and GATEST detect comparable number of faults (almost twice as many as S_ATPG); PIPEXPRESS is

Table 2. Results

Circuit	GATEST			S_ATPG			PIPEXPRESS		
	Fault cov.	Time	Vectors	Fault cov.	Time	Vectors	Fault cov.	Time	Vectors
<i>piir8o</i>	15,083/15,288	59:33	455	8,251/19,918	232:28	17	15045/15288	21:57	692 (173X4)
<i>pcont2</i>	6,810/7,426	12:20	143	4,874/11,254	131:07	20	7,338/7,426	6:46	224 (112X2)

Table 3. PIPEXPRESS fault detection

Circuit	Original	ATOM	FIRES	1-vector seq.		2-vector seq.		3-vector seq.		4-vector seq.		Total vectors
				No.	Detected	No.	Detected	No.	Detected	No.	Detected	
<i>piir8o</i>	19,936	4,591	57	72	5,150	28	1675	15	8,220	0	0	173
<i>pcont2</i>	11,272	3,615	231	23	4,113	6	1,465	19	1,709	5	51	112

about 3 times faster than GATEST, and about 10 times faster than S_ATPG. Each one of the 173 generated vectors has to be clocked 4 times to propagate through the transparent registers. For *pcont2*, PIPEXPRESS detects 8% more faults than GATEST and about 50% more than S_ATPG, while being twice as fast as GATEST and more than 20 times faster than S_ATPG.

6. Conclusions

Our method has the potential of providing a very significant breakthrough in the field, as it offers a solution to an important problem that until now has been considered impossible to solve without significantly changing the structure of the circuit in test mode. Our DFT technique is based on clock freezing and clock partitioning, introduces no delay penalties and small area overhead, and it is compatible with at-speed testing.

References

[Abramovici et al. 1986] M. Abramovici, J. J. Kulikowski, P. R. Menon, and D. T. Miller, "SMART and FAST: Test Generation for VLSI Scan-Design Circuits," *IEEE Design & Test of Computers*, August, 1986

[Abramovici et al. 1990] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*, IEEE Press, 1994

[Abramovici et al. 1992] M. Abramovici, K. Rajan, and D. T. Miller, "FREEZE!: A New Approach for Testing Sequential Circuits," *Proc. 29th Design Automation Conf.*, June, 1992.

[Agrawal et al. 1991] V. D. Agrawal, S. C. Seth, and J. S. Deogun, "Design for Testability and Test Generation with Two Clocks," *Proc. 4th Intn'l. Symp. on VLSI Design*, pp. 112--117, January 1991

[Baeg and Rogers 1993] S.H. Baeg and W.A. Rogers, "A New Design for Testability Method: Clock Line Control Design", *Proc. Custom Integrated Circuits Conf.*, pp. 26.2.1-26.2.4, 1993.

[Chickermane et al. 1992] V. Chickermane, J. Lee, and J.H.

Patel, "A Comparative Study of Design for Testability Methods Using High-level and Gate-Level Descriptions," *Proc. Intn'l. Conf. on CAD*, pp. 620-624, November 1992.

[Einspar et al. 1993] K. L. Einspahr, S. C. Seth, and V. D. Agrawal, "Clock Partitioning for Testability," *Proc. 3rd IEEE Great Lakes Symp. on VLSI*, pp.42--46, March 1993

[Einspar et al. 1996] K. L. Einspahr, S. C. Seth, and V. D. Agrawal, "Improving Circuit Testability by Clock Control," *Proc. 6th IEEE Great Lakes Symposium on VLSI*, pp.288--293, March 1996

[Einspar et al. 1999] K. L. Einspahr, S.K. Mehta, and S.C. Seth, "A Synthesis for Testability Scheme for Finite State Machines Using Clock Control," *IEEE Transactions on CAD*, Vol. 18, No. 12, Dec., 1999

[Fang and Gupta 1994] W-C. Fang and S.K. Gupta, "Clock Grouping: A Low Cost DFT Methodology for Delay Testing," *Proc. 31st Design Automation Conf.*, pp. 94-99, 1994

[Gupta et al. 1990] R. Gupta, R. Gupta, and M. A. Breuer, "The Ballast Methodology for Structured Partial Scan Design," *IEEE Trans. on Computers*, Vol. 39, no. 4, pp. 538-544, April, 1990

[Hamazoglu and Patel 1998] I. Hamzaoglu and J. H. Patel, "New Techniques for Deterministic Test Pattern Generation," *Proc. VLSI Test Symp*, pp. 446-452, April 1998.

[Iyer et al. 1996] M. Iyer, D.E Long, and M. Abramovici, "Identifying Sequential Redundancies Without Search," *Proc. 33rd Design Automation Conf.*, June, 1996.

[Kelsey et al. 1993] T.P. Kelsey, K.K. Saluja, and S.Y. Lee, "An Efficient Algorithm for Sequential Circuit Test Generation," *IEEE Trans. on Computers*, Vol. 42, no 11, pp. 1361-1371, November 1993.

[Long et al. 2000] D.E Long, M. Iyer, and M. Abramovici, "FILL & FUNI: Algorithms To Identify Illegal States and Sequentially Untestable Faults," *ACM Trans. on Design Automation of Electronic Systems*, July, 2000.

[Niermann et al. 1992] T. M. Niermann, W. -T. Cheng, and J. H. Patel, "PROOFS: A Fast, Memory-Efficient Sequential Circuit Fault Simulator," *IEEE Trans. CAD*, vol. 11, no. 2, pp. 198--207, February, 1992.

[Rajan et al. 1996] K. B. Rajan, D. E. Long, and M. Abramovici, "Increasing Testability by Clock Transformation (Getting Rid of Those Darn States)," *Proc. VLSI Test Symp.*, April, 1996

[Rudnick et al. 1997] E. Rudnick, J.H. Patel, G.S. Greenstein, and T.M. Niermann, "A Genetic Algorithm Framework for Test Generation," *IEEE Trans. CAD*, vol. 16, no. 9, pp. 1034-1044, Sept. 1997.

[Santoso et al. 1999] Y. Santoso, M. Merten, E. Rudnick, and

M. Abramovici, "FreezeFrame: Compact Test Generation Using a Frozen Clock Strategy," *Proc. Design Automation and Test in Europe Conf.*, March 1999