16th IEEE VLSI Test Symposium, Monterey, CA, 1998

Fast Self-Recovering Controllers

Andre Hertwig, Sybille Hellebrand and Hans-Joachim Wunderlich
Computer Architecture Lab
University of Stuttgart, Germany

Abstract

A fast fault-tolerant controller structure is presented, which is capable of recovering from transient faults by performing a rollback operation in hardware.

The proposed fault-tolerant controller structure utilizes the rollback hardware also for system mode and this way achieves performance improvements of more than 50% compared to controller structures made fault tolerant by conventional techniques, while the hardware overhead is often negligible. The proposed approach is compatible with state-of-the-art methods for FSM decomposition, state encoding and logic synthesis.

Keywords: *FSM synthesis, fault-tolerance,checkpointing, performance-driven synthesis*

1 Introduction

The widespread use of microelectronics for safety critical applications leads to an increasing interest in synthesis techniques for reliable systems. But with the advances in circuit miniaturization coupling problems, increasing current densities, and ionizing radiation become important issues and make the circuits more susceptible to transient faults [2,8]. Since transient faults are hard to detect by periodical tests, microelectronics in safety critical applications have to implement an on-line test accompanied by fault isolation or recovery mechanisms. The target structure presented in this paper provides a hardware-based recovery mechanism in the controller part of controller/ datapath-systems in the presence of transient faults.

In complex systems, the controller is usually part of the critical path and any additional delay introduced by fault-tolerance techniques further decreases the system performance. But as the integrity of the system strongly depends on the controller, on-line test and fault-tolerance techniques are indispensable for reliable systems [6,7,12,13,14,16].

In contrast to conventional techniques, which insert checkers and isolation circuits in the communication paths between different modules, the presented target structure allows to perform checks in parallel with intermodule communication, and thus avoids severe performance degradations. More precisely, the signals at the module inputs are processed immediately when they become available, the checker operates in parallel and errors are indicated with a latency of one clock cycle only. Changes to the state of the system due to erroneous information are corrected by rolling back the system to the state that existed before the error first occurred.

The hardware resources (storage elements, multiplexers) for the rollback mechanism are utilized also for system functions, which provides performance improvements, comparable with those obtained by controller decomposition used to implement high-speed controllers [1,3,10]. But while controller networks made fault-tolerant by conventional techniques lead to a tremendous hardware overhead, as each submachine has to implement its own fault-tolerance mechanism, the proposed target structure keeps the hardware overhead very low.

The rest of this paper is organized as follows. Section 2 reviews the basic recovery concept and extends it to classical finite state machines. Section 3 presents the fault-tolerant target structure for high-speed applications, and a tailored state assignment is given in section 4. Section 5 describes the synthesis flow for the proposed fault-tolerant controllers, and experimental results in section 6 conclude this paper.

2 Basic concept

A rollback restores a state of a system, which had been reached before. This requires to store the state of the system at some cycle boundaries. The stored state, often called checkpoint, is used to overwrite the state of the system when a rollback is performed. It is determined by the contents of all storage elements which carry useful information across cycle boundaries. This simple rollback concept has been applied successfully in transaction systems for many years [9].

Figure 1 shows an example for a rollback operation. An error occurs during cycle 3 which is detected one cycle later. After detecting the error, the rollback mechanism

This work was supported by the DFG under grant Wu 245/1-1

restores the state of the system that had been reached at the end of cycle 2. This is done by overwriting the current state with the checkpoint, stored at the end of cycle 2. In the next clock cycle the operation originally performed during cycle 3 is performed again.

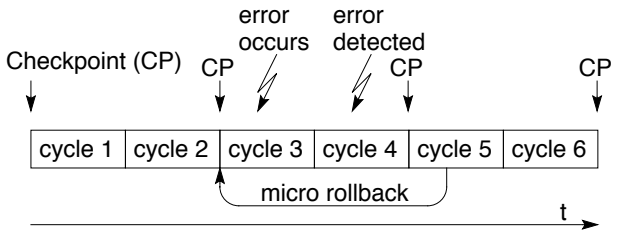

Figure 1: Rollback of a module

To reload a register with a state, that had been reached a few cycles before, a hardware rollback mechanism as sketched in figure 2 can be used as a straightforward implementation. The lower part shows the components for a conventional controller, and the upper part shows the extension to implement the rollback mechanism. In normal operation the logic block computes the output and next state function of the controller and the register R_0 stores the state of the system. The additional registers R_1 and R_2 on the top are configured as first-in-first-out storage (FIFO), storing checkpoints, and the checker initiates rollbacks. The flipflop connected to the checker output removes the checker from the critical path, and this way prevents performance degradations. The register R_2 is necessary to compensate the additional error-detection latency introduced this way.

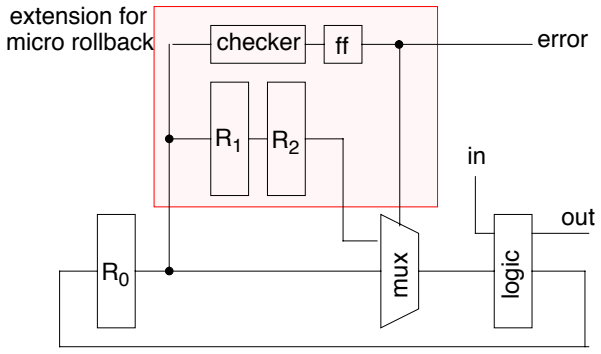

Figure 2: FSM with extension for rollback

Figure 3 shows the timing diagram for an example. The state register stores the state sequence S_0, S_1, S_2, S_3. The active *error* signal indicates that the previous state was erroneous and has to be computed again. The controller must rollback to the predecessor state S_2, which is stored in the FIFO storage. This is done by the multiplexer feeding the logic block, which switches the checkpoint register to the logic inputs in the case of an error.

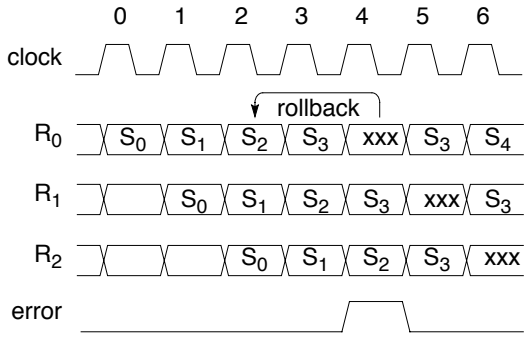

Figure 3: Timing diagram for a rollback operation

The rollback of the controller part must be accompanied by a rollback of the datapath, which has to restart its operation and recompute its status signals for the controller. The compatible hardware-based recovery scheme for datapaths, proposed by Tamir et al. [18,19], triggers on the error signal, too. Similar as proposed for the controller part Tamir suggest FIFO-like storage elements, called delayed write back buffers (DWB) for rollback purposes. The length of the FIFO depends in both cases on the worst case error detection latency. Other rollback schemes for datapaths are proposed by Karri and Orailoglu [7].

The straightforward implementation, sketched in figure 2, meets the functional requirements, but chip area and propagation delay increase because of the additional rollback circuitry. The additional delay is crucial, as the controller is usually part of the critical path and any additional delay slows down the entire system. The rollback circuitry is not profitably used for system functions by the straight forward implementation, whereas the target structure presented in the next section exploits the rollback hardware also for the system function and increases the controller performance distinctly by this way. The performance is comparable with that of controller networks [1,3,10].

3 Target structure

In a rollback scheme for controllers, the need for checkpoint registers may be supported by synthesis techniques for high-speed controllers relying on decomposition and pipeline techniques. For the bypass pipeline proposed in [6], the set of states is partitioned, such that state transitions occur only between, but not within different partition blocks. This way the state transitions can be implemented by different logic blocks, each one implementing only a subset of the state transitions. Thus, a simpler function and a shorter critical path can be expected.

Furthermore, the single state register in a conventional controller structure is replaced by a set of registers, such that at each time only one of the registers represents the current state of the controller. The remaining registers can therefore be used as checkpoint registers. Figure 4 shows the resulting controller structure when the set of states is partitioned into three groups.

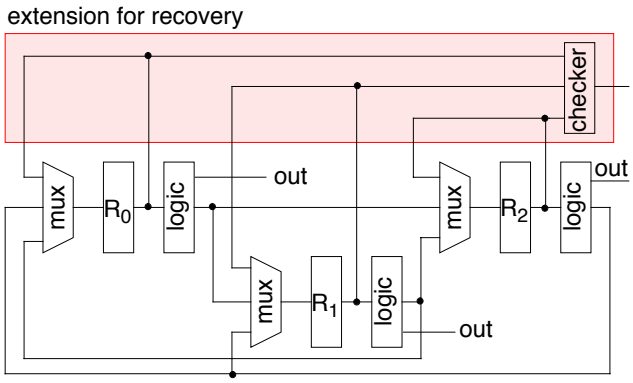

Figure 4: Target structure

As at each time step only one of the registers represents the state of the controller and the state transition diagram is free of self-loops, one register never represents the state of the controller for two consecutive clock cycles. Thus, the register containing the state information of the controller during the present clock cycle can store rollback information during the next clock cycle, because it is not used to encode the state of the system then. To store a checkpoint the register storing the state of the controller reloads its own contents during the following clock cycle again. This way at each time one checkpoint is available.

Figure 5 shows the course of events within the target structure, and illustrates that at each time one register contains the state the controller had reached one cycle before. The

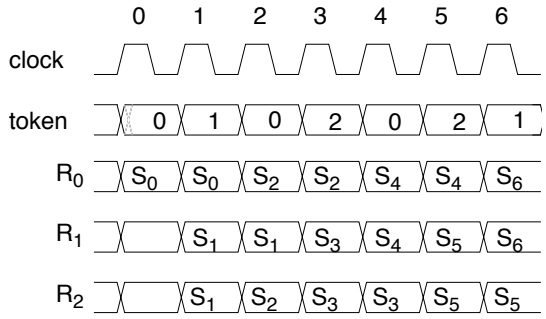

Figure 5: Timing diagram for normal operation

signal denoted *token* determines the register which holds the state information during the present clock cycle. E.g., during clock cycle 1 register R_1 contains the state information. Since register R_0 contained the state information one cycle before, register R_0 stores the checkpoint during cycle R_1. In the case that an error were detected during cycle 1, the checkpoint register R_0 would remain unchanged during cycle 2 and would represent the state of the controller.

At each clock cycle only one of the registers is responsible for the state of the controller and must be monitored together with the primary outputs at the checker. The multiplexers determine the path to that register which will store the next state. To control the multiplexers a mechanism is needed to identify the registers representing the state of the controller and the check point, respectively. For this purpose, one bit t_i is reserved in each register and the remain-

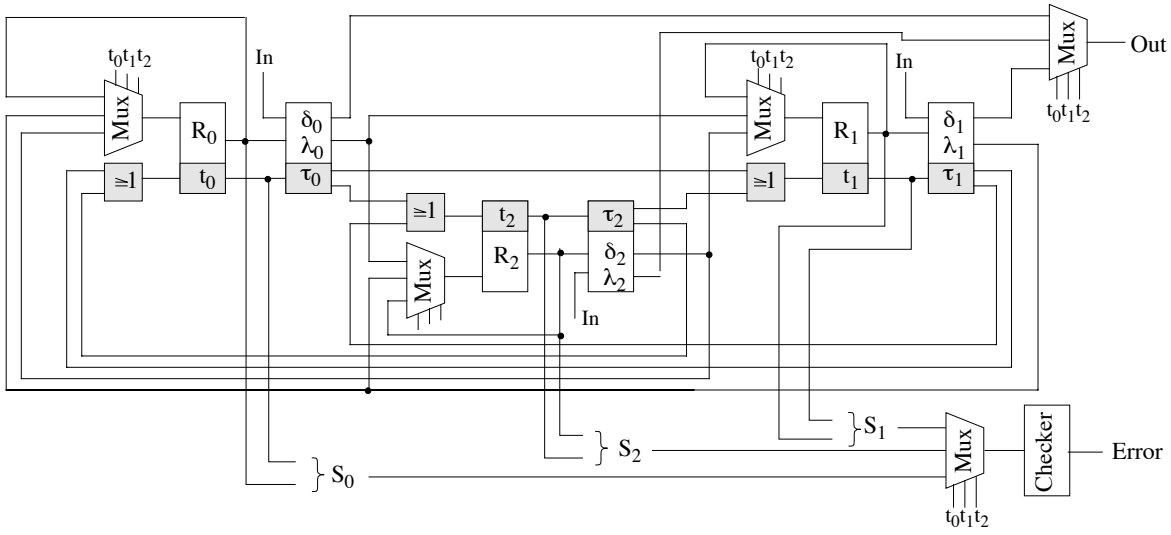

Figure 6: Token bit architecture.

ing bits can be used for an arbitrary strategy of error-detecting state encoding. This bit is called token bit and controls the multiplexers in the target structure as shown in figure 6.

Figure 7 shows a single stage of the target structure in detail. The token bit t_0 is stored in a single flipflop implemented in a master-slave style for timing issues [6], whereas the register R_0 contains single-edge-triggered flipflops only. The logic block implements three functions, the output function λ, the next state function δ and the token function τ, which evaluates the state and input signals to compute the stage which will receive the token bit during the next clock cycle. The signals $t_{i,j}$ connect the token output of the i-th stage with the token input of the j-th stage. The AND-gates connected to the outputs of the token function τ abort the regular token distribution in the case of an error, signalized by an active input cp_i (*check point i*).

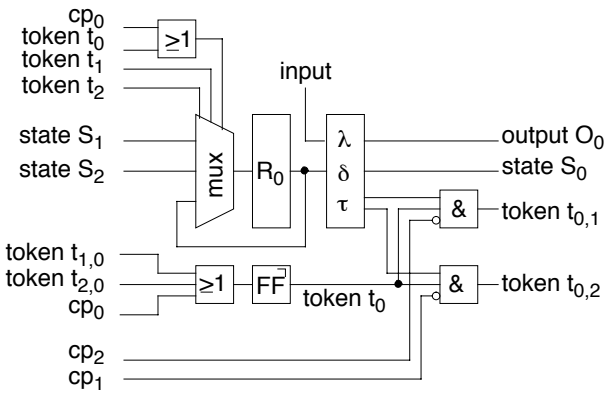

Figure 7: Token bit distribution

The signals cp_0 *(check point 0)* through cp_2 (*check point 2)* also indicate which of the registers contains the checkpoint to be reloaded in the case of an error indication. The easiest way to generate these signals is to store the token for an additional clock cycle as shown in figure 8. When the checker detects an error, the check point signal corresponding to the previously active register is set to one and reactivates the token bit in this register.

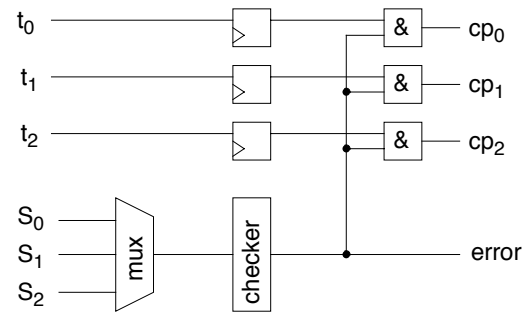

Figure 8: Checkpoint identification

The timing diagram in figure 9 illustrates the sequence of events in the case of a rollback. During each clock cycle a different register represents the state of the controller, determined by the token. As depicted in the timing diagram the relevant register becomes the checkpoint register in the next clock cycle and remains unchanged. An error, in the example state 3^f is erroneous, is detected with a latency of one clock cycle and is indicated by the checkpoint signals. As a result the token is distributed to the checkpoint register and the former erroneous operation is performed again. The error signal indicates that the datapath must perform a rollback as described in [19].

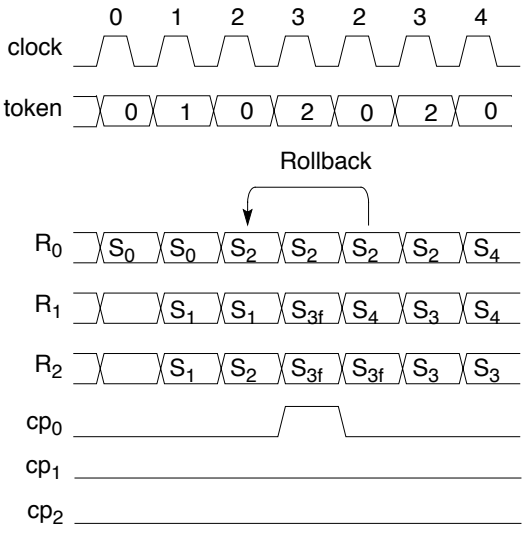

Figure 9: Timing diagram for micro rollback

In the case of a permanent fault the rollback operation is carried out in an endless loop. A simple automaton can be used to distinguish permanent and transient faults. Figure 10 shows the state transition graph for a permanent fault detector, which identifies a fault as permanent, if the same logic block computes erroneous outputs in two consecutive cycles or if the contents of the checkpoint register becomes erroneous.

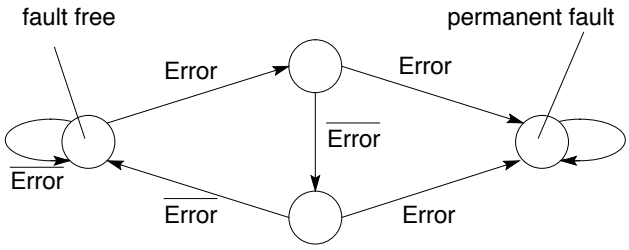

Figure 10: State transition diagram for permanent fault detector

4 State and output encoding

As explained in the previous sections a checker is used to initiate a rollback operation. Hence, state and output signals must be encoded with error-detecting codes, e.g. parity codes, hamming codes, berger codes, m-out-of-n codes, etc..

The target structure is compatible with all common error-detecting codes, but as the encoding has strong impact on the performance of the controller, not all codes fit for high-speed applications. Separable codes, e.g. parity codes, are most appropriate for the bypass pipeline. Parity codes are widely used in digital systems, because of their low impact on area and performance. But a single parity bit does not always guarantee the required fault coverage, since single faults within the circuit often lead to multiple errors at the outputs. For a higher fault coverage a cross-parity sec/ded code with the minimum number of parity bits is used in the experiments reported in section 6 [15].

To guarantee sufficiently high error detection capabilities for the target structure, it must be ensured that errors in the multiplexer which feeds the checker are detected, too. This can be achieved by a checking scheme assigning code words to "valid" states only. In this case, the checker will also detect all errors in the multiplexer which result in passing a wrong input to the output. The token bits in the checker structure, depicted in figure 7, indicate the validity of the register contents.

Figure 11 shows an example for a checking scheme to detect errors in the multiplexer which result in passing a wrong input to the output. The scheme is based on a parity code, and the prediction logic only takes the state bits as inputs. The token bit is not considered for this computation. The multiplexer feeding the parity checker switches the state signals, the parity bit and the token bit. Since the token bit is one for each valid code word and the token is taken into account by the checker, the checker and the prediction logic compute the complementary parity bits for valid states and the identical parity bits for invalid states.

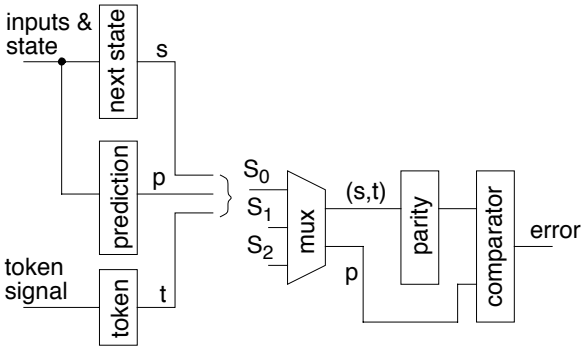

Figure 11: Detecting non-valid code words

Moreover, the correct configurations of token bits form a one-out-of-three code, and the checking scheme of figure 11 can easily be extended to check these configurations, too.

5 Synthesis procedure

To implement the target structure it is necessary to partition the states of the controller into three groups, such that state transitions occur only between, but not within groups. This state partitioning is equivalent to coloring the state transition graph with three colors. But obviously not all state transition graphs are three-colorable, so that a series of equivalent graph transformations are necessary to guarantee an admissible coloring. An appropriate coloring algorithm is proposed in [6] for data dominated applications. This algorithm leads to colorable transition graphs, which are only slightly larger than the original transition graphs.

After partitioning the set of states the symbolic truth tables for the different logic blocks are derived from the colored state transition graph. All state transitions starting in nodes with the same color form the truth table for a single logic block. Partitioning the set of state transitions facilitates state encoding and logic synthesis significantly, as each combinational block is dealt with independently. The well known encoding techniques are applied directly with small modifications which ensure that states of a certain color are assigned only to one register. Results reported in the next section are based on the encoding strategies of NOVA with additional check bits [17].

6 Experimental results

The proposed target structure has been implemented and compared to the straightforward implementation. The checkpoint register of the target structure remains unchanged when a transient fault occurs for several consecutive clock cycles. Also the straightforward structure can implement this feature if the FIFO reloads the checkpoint register in the case of an error. Figure 12 shows the investigated implementation.

The results presented in this section have been achieved for the benchmark set distributed for the workshop on Logic Synthesis 1993 [13]. In contrast to all the other performance-driven synthesis methods, the algorithm has its best results for the largest circuits which cannot be dealt with by the known methods. For small controllers the constant hardware overhead for the token control logic is dominating. As 3 flipflops are already needed for the token control we only investigated FSMs which have at least 33 states and require more than 5 flipflops.

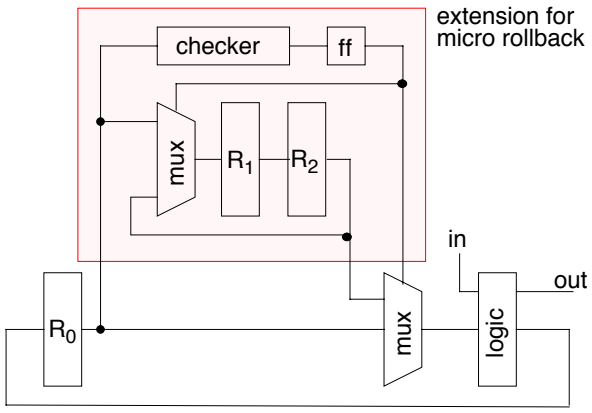

Figure 12: Investigated standard structure

Table 1 shows the characteristics of the investigated benchmark circuits. The columns two through four report the number of states | S |, inputs | I | and outputs | O | of the original specification. To implement the bypass pipeline usually equivalent graph transformations are necessary, resulting in additional states. The column | S_{total} | shows the number of states after the graph transformation and the three rightmost columns show the number of states in each register of the target structure.

Circuit	| S |	| I |	| O |	|S_{total}|	| S_0 |	| S_1 |	| S_2|
planet	48	7	19	49	15	18	16
planet1	48	7	19	49	15	18	16
s1488	48	8	19	49	14	17	18
s1494	48	8	19	49	18	17	14
scf	121	27	56	122	36	48	38
s298	218	3	6	219	64	83	72
s510	47	19	7	71	24	24	23

Table 1: Characteristics of investigated benchmarks

It is interesting that in most cases only a few transformations of the state transition graphs are sufficient, to implement the target structure. It should be noted that the total number of states has increased distinctly after graph transformation only for controller s510. This controller is not part of a data dominated application, and as shown in [6] the algorithm is not expected to provide the best results here.

The first experiment investigates the quality of the proposed fault-tolerance strategy. For both structures, the straightforward and the target structure, a parity code and a cross-parity single error correcting/double error detecting (sec/ded) code have been used to encode the state and output signals [15]. Not all faults are detected by the checkers, since some faults lead to multiple errors at the outputs. The percentage of undetected non-redundant faults in the next state and the output logic of the controllers is used as a metric for the quality of the encoding scheme. For all experiments the single stuck-at fault model has been assumed.

Table 2 shows in the left part the results for the straightforward implementation and in the right part the corresponding results for the bypass pipeline. An encoding with one parity bit often leads to better results for the target structure than for the straightforward implementation. This is explained by the smaller logic blocks in the target structure. Smaller circuits are usually less densely meshed than lager ones, so that less faults lead to multiple errors at the outputs.

As expected, a single parity checker cannot provide the required fault coverage. A cross-parity sec/ded code was applied to achieve high encoding qualities. The results are reported in table 2. In all cases less than one third percent of the non-redundant faults in the combinational part are masked by the checker, independent of the investigated structure.

Circuit	Standard structure with 1 parity bit	Std. structure with sec/ ded code (check bits)	Target structure with 1 parity bit	Target structure with sec/ ded code (check bits)
planet	10.5 %	0.0 % (10)	10.5 %	0.0 % (10)
planet1	10.5 %	0.0 % (10)	10.5 %	0.0 % (10)
s1488	15.4 %	0.0 % (10)	8.9 %	0.0 % (10)
s1494	16.1 %	0.2 % (10)	11.3 %	0.1 % (10)
scf	19.6 %	0.0 % (16)	14.4 %	0.0 % (16)
s298	10.8 %	0.1 % (8)	11.3 %	0.0 % (8)
s510	15.8 %	0.0 % (8)	14.2 %	0.1 % (8)

Table 2: Undetected non-redundant faults in the combinational part of the controller

The fault-tolerance capabilities of both structures are comparable, but there are major differences with respect to the operation frequency. Table 3 shows the propagation delays for all investigated self-recovering controller structures, mapped to a 0.7 µm CMOS process. The speed of the bypass-pipeline is determined by the delay of the slowest logic block plus the delay of the additional multiplexers and registers. It is shown that the total delay of the bypass-pipeline is significantly less than the delay of the standard solutions. Mainly we have operation frequencies for the target structure which are 50% higher than for the standard structure.

The prediction logic and the logic for the next state/output function are implemented in two disjoint logic blocks and the next state/output function is the same for both encoding schemes. Hence the propagation delay of the parity code and the sec/ded code is the same, if the next state/output function is part of the critical path.

Circuit	Standard structure with parity bit	Standard structure with sec/ded code	Target structure with parity code	Target structure with sec/ded code
planet	8.10	8.10	5.18	5.18
planet1	8.10	8.10	5.18	5.18
s1488	7.23	8.58	4.78	4.78
s1494	7.25	7.26	5.18	5.18
scf	11.17	11.17	6.50	7.40
s298	13.47	14.32	9.63	9.63
s510	4.48	5.36	5.06	5.06

Table 3: Propagation delays for fault tolerant structures

Both self-recovering structures use one checker, one comparator, three registers and multiplexers to implement the rollback mechanism. Since the target structure shares the check circuitry among all units and utilizes the register and multiplexer also for the system function, the hardware overhead remains very low for most of the investigated circuits. Table 4 shows the transistor count for the investigated implementations.

Circuit	Standard structure with parity bit	Standard structure with sec/ded code	Target structure with parity code	Target structure with sec/ded code
planet	6100	9496	6442	10154
planet1	6100	9496	6442	10154
s1488	7796	12626	8258	12038
s1494	7808	12510	8838	13182
scf	8334	9694	11162	17686
s298	23110	35868	31340	42092
s510	2844	3940	6002	9150

Table 4: Transistor counts for fault tolerant structures

7 Conclusion

A novel approach for fast self-recovery controllers has been presented which utilizes the rollback hardware also for the system function to increase the operation frequency. By sharing the check circuitry among different units the proposed target structure minimizes the hardware overhead.

8 References

[1] P. Ashar, S. Devadas, A. R. Newton; Optimum and Heuristic Algorithms for an Approach to Finite State Machine Decomposition, in: IEEE Trans. on CAD, Vol. 10, No. 3, March 1991, pp. 296-310

[2] B. Davari, R. H. Dennard, G. G. Shahidi; CMOS Scaling for High Performance and Low Power - The Next Ten Years; Proc. of the IEEE, Vol. 83, No. 4, April 1995, pp. 595-606

[3] S. Devadas, A. R. Newton; Decomposition and Factorization of Sequential Finite State Machines; in: IEEE Trans. on CAD, Vol. 8, No. 11, November 1989, pp. 1206-1217

[4] B. Eschermann; On Combining Off-Line BIST and On-Line Control Flow Checking; in: Proc. 22nd Fault Tolerant Computing Symposium (FTCS), 1992, pp. 298-305

[5] S. Hellebrand, H.-J. Wunderlich; An Efficient Procedure for the Synthesis of Fast Self-Testable Controller Structures; in: Proc. ACM/IEEE Int. Conf. on CAD, San Jose, Ca., 1994, pp. 110-116

[6] A. Hertwig, H.-J. Wunderlich; Fast Controllers for Data-Dominated Application; in: Proc. European Design and Test Conf. (ED&TC), Paris, 1997, pp. 84-89

[7] R. Karri, A. Orailoglu; Scheduling with Rollback Constraints in High-Level Synthesis of Self-Recovering ASICs; Proc. 22nd Intl. Symposium on Fault-Tolerant Computing (FTCS); 1992, Boston, Massachusetts; pp. 519-526

[8] I. Koren, A. D. Singh; Fault Tolerance in VLSI Circuits; IEEE Computer, Vol. 23, No. 7, July 1990, pp. 73-83

[9] H. F. Korth, A. Silberschatz; Database System Concepts; 2nd Edition, London, McGraw-Hill, 1991

[10] K. Lam, S. Devadas; Performance-Oriented Decomposition of Sequential Circuits; in: Proc. IEEE Int. Symposium on Circuits and Systems, New Orleans, LA, 1990, pp. 2642-2645

[11] H. K. Lee and D. S. Ha; On the Generation of Test Patterns for Combinational Circuits; Technical Report No. 12/93, Dep't of Electrical Eng., Virginia Polytechnic Institute and State University, 1993

[12] R. Leveugle, G. Saucier; Optimized Synthesis of Concurrently Checked Controllers; in: IEEE Trans. on Coputers, Vol. 37, No. 2, Feb. 1990, pp. 419-425

[13] K. McElvain; IWLS'93 Benchmark Set: Version 4.0, distributed as part of the IWLS'93 benchmark distribution

[14] R. A. Parekhji, G. Venkatesh, S. D. Sherlekar; Concurrent Error Detection Using Monitoring Machines; in: IEEE Design & Test of Computers, Vol. 12, No. 3, Fall 1995, pp. 24-31

[15] T. R. N. Rao, E. Fujiwara; Error-Control Coding for Computer Systems; Prentice-Hall, Englewood Cliffs, 1989

[16] K. Wilken, J. P. Shen; Continous Signature Monitoring: Low-Cost Concurrent-Detection of Processor Control Errors; in: IEEE Trans. on CAD, Vol. 9, No. 6, June 1990, pp. 629-641

[17] T. Villa, A. Sangiovanni-Vincentelli; NOVA: State Assignment of Finite State Machines for Optimal Two-Level Logic Implementations; in: IEEE Trans. on CAD, Vol. 9, Sep. 1990, pp. 905-924

[18] Y. Tamir, M. Liang, T. Lai, M. Tremblay; The UCLA Mirror Processor: A Building Block for Self-Checking Self-Repairing Computing Nodes; Proc. 21st Intl. Symposium on Fault-Tolerant Computing (FTCS), 1991, Montreal, Canada

[19] Y. Tamir, M. Tremblay; High-Performance Fault-Tolerant VLSI Systems Using Micro Rollback; IEEE Trans. on Computers, Vol. 39, No. 4, April 1990, pp. 548-554