Architecture of FPGA-based Concurrent Checking FSM[1]

Ilya Levin[1], Mark Karpovsky[2], Vladimir Sinelnikov[3]

[1]*Tel-Aviv University, School of Education, Ramat-Aviv 69978, Israel,*
ilia1@post.tau.ac.il

[2]*Boston University, Department of Electrical, Computer and System Engineering, Boston, MA 02215*

[3]*Academic Technological Institute, Department of Computer Science, Golomb, 52, Holon, Israel*

Abstract. A new technique for on-line checking of FPGA-based sequential devices defined by their finite state machines (FSMs) is presented. The technique utilizes specific properties of FSMs for achieving the totally self-checking goal with a low overhead. The proposed technique does not require any redundant encoding of output words and uses a one-rail design, thereby drastically decreasing the required overhead. The paper presents results for benchmarks for the proposed architecture.

I. Introduction

Existing approaches for design of self-checking FSMs are based on either duplication, or application to them of specific error detecting codes (Berger code, constant weight code, etc.). In most cases, these approaches require a hardware overhead of more than 100 percent.

Major difficulties in designing of self-checking devices are related to the complexity of decoding (i.e. verification that a given output is a codeword). Outputs of a self-checking circuit are usually encoded by codewords of a code, which detects unidirectional errors [10]. For example, in [10] it was shown that stuck-at fault, cross-point faults and shorts in MOS PLAs and ROMs result in unidirectional errors at their outputs. Applications of the self-checking concept to Control Units and microprocessors were presented in [13]. Several works deal with synthesis of totally self-checking (TSC) Control Units [6, 11], design for testability of controllers [2] and self-checking control networks [5]. Paper [9] presents several schemes for on-line checking of microprogram control units, which are based on computing of a set of signatures and inserting of these signatures in a microprogram code at specific locations. Papers [4] and [10] are also dedicated to the problem of synthesis of self-checking microprogrammed control units. In [4] a design of a special monitor circuit enabling to detect a specified fault set is proposed. In [10], duplication of a microprogram sequencer was proposed to achieve the totally self-checking property.

Paper [3] describes a special technique for decomposing the initial FSM to achieve both on-line checking and on-line testing. The concurrent testing and checking allows decreasing the overhead in comparison to traditional on-line checking approaches. The technique, which is presented in [8] allows detection of input faults with the aid of unordered input vectors.

---

[1] This research was supported by BSF under grant No. 9800154.

Known approaches for synthesis of Mealy-type FSMs are based on Berger encoding of outputs and m-out-of-n state assignment [5, 11]. For these architectures checkers detect presence of a fault by examining whether an output vector belongs to the corresponding code.

To the best of our knowledge, authors of the above-mentioned works did not try to design on-line checking controllers by utilizing specific FSM properties. We will focus on properties of the FSM, which are typical for controllers, and also have a considerable impact on the resulting overhead. One of such properties is that the number of possible FSM output codewords is much smaller than 2^N (where N- is a number of output line) while the set of possible codewords is known in advance.

This property was used in [6] for designing checkers. The authors show that the checker of an FSM controller can be efficiently implemented in the form of "sum of minterms" (SOM) of output functions of the controller. An unordered code for output vectors is used because for these codes any unidirectional error cannot transform one codeword into another codeword. In [6], the Berger code was used as an unordered code. Note that SOM-checker examines whether an output vector belongs to the set of possible codewords, and not to the Berger code as in the case of standard design [1, 7].

An efficient unordered code was developed by Smith [12] specifically for the case when the number of possible codewords is sufficiently smaller then 2^N. Taking into account that this property is valid for FSM based controllers, the code proposed in [12] could be applied instead of the Berger code for additional overhead reduction.

We will use Field Programmable Gate Arrays (FPGAs) as a basis for FSM controller's implementations. The approach described in [5] for synthesis of self-checking circuits by FPGAs is based on dual-rail implementations of the hardware to be checked. Using this approach FPGA-based FSM controller can be implemented as a combination of the dual-rail controller and the dual-rail SOM-based checker (SOM-checker). In this case, output vectors of the FSM controller have to be encoded by a code detecting all unidirectional errors (such as the Berger code). Needless to say, that such an implementation is critical from the point of view of resulting overhead due to both the dual-rail design and the Berger encoding.

In the present paper we propose a new architecture that does not require any encoding of output vectors and allows a single-rail design of FSMs.

This paper is organized as follows. Section 2 introduces basic definitions and a review of related works. In Section 3 we describe the proposed architecture of the self-checking FSM controller. Benchmarks results are presented in Section 4. Conclusions are presented in Section 5.

II. Frameworks

In this section we remind basic definitions related to FSMs and fault models.

Table 1. Table of FSM

a_m	a_s	$X(a_m, a_s)$	$Y(a_m, a_s)$	h
a_1	a_2	$x_1 x_2$	y_2, y_3	1
	a_4	$x_1 \overline{x_2} x_3$	y_4	2
	a_1	$x_1 \overline{x_2} \overline{x_3}$	–	3
	a_3	$\overline{x_1}$	y_2	4
a_2	a_4	1	y_1, y_4	5
a_3	a_1	$x_4 x_1$	y_1, y_3	6
	a_4	$\overline{x_1}$	y_1, y_4	7
	a_4	$\overline{x_4}$	y_1, y_4	8
a_4	a_5	x_2	y_5, y_6	9
	a_1	$\overline{x_2}$	–	10
a_5	a_1	1	y_1, y_3	11

A. FSM Controllers

We will consider a system as a combination of a control unit (controller) and a data-path.

FSM, being a natural formal model of the controller, can be defined by its transition table. The transition table for an exemplary FSM is shown by Table 1.

In this table: a_m and a_s present state and next state correspondingly, $X(a_m, a_s)$ - transition function, i.e. a Boolean function which is equal to 1 when FSM makes the transition from state a_m to state a_s. $Y(a_m, a_s)$ – micro-instruction, the list of output signals which are equal 1 on the transition of the FSM from a_m to a_s.

B. Definitions and Assumptions

We now recall some basic definitions from the theory of design of Totally Self Checking (TSC) sequential circuits.

A finite state machine FSM is self-testing if, for every fault in a fault's set, there is such an input/state pair in the circuit that a non-code output is produced. A state machine is fault-secure if, for every fault from a faults set, the machine never produces an incorrect code output for a code input. A state machine is totally self-checking if it is both self-checking and fault-secure. [5].

C. Fault Model

As it has been mentioned, the basis of target implementation of the FSM controller is LUT-based FPGA comprising Configurable Logic Blocks (CLBs). The fault model used in this paper is general model of single cell faults. We assume that at most one CLB can produce a faulty output. The circuit primary inputs are considered to be fault-free.

III. Match Detector based architecture of a self-checking controller

We propose a new architecture for the FSM controller that does not require any encoding of output vectors and consequently allows reduction of the required overheads. We call this architecture a Match Detector (MD) architecture since it is based on using a Match Detector within the checker. A checker that includes the Match Detector will be called MD-checker.

A schematic diagram of the MD-architecture is shown in Figure 2. The proposed architecture consists of two portions: a self-checking FSM and MD-checker. In turn, each of these portions contains two main blocks: an "evolution" block and an "execution" block [7]. Besides the FSM contains a Product Terms Compressor (PTC). The function of the PTC is to form *1-out-of-M* code of the output codeword (M – number of the micro-instructions).

The evolution block of the FSM (EvFSM) implements all of product terms while the execution block of the FSM (ExFSM) implements outputs and next state functions of the controller. The evolution block of the checker (EvCh) implements product terms corresponding to the output codeword of the controller while the execution block of the checker (ExCh) collect all these product terms for realization of the error function.

The main idea of the proposed approach is based on the property that both a vector that enters into ExFSM and a vector that enters into ExCh are equal to the *1-out-of-M* code of the corresponding output codeword. These vectors have to be equal in the case of proper functioning of the controller and are different in the case of a fault. Comparison of these two vectors by the Match Detector (MD) enables achieving the TSC property of the controller.

A. Self-checking FSM

Inputs of the evolution block of the FSM (EvFSM) comprise working inputs X of the FSM and output memory signals $T = \{t_1, \ldots, t_R\}$. Outputs of the EvFSM correspond to product terms $P = \{p_1, \ldots, p_H\}$. The Product Terms Compressor (PTC) transforms the vector of product terms into "*1-out-of-M*" code $\delta(Y)$.

The EvFSM is implemented as a tree, wherein each of the nodes is either a pre-designed Configurable Logic Block (CLB), or a fan-out. Each CLB is designed for implementation of either a sum of two product terms of g variables, or an AND-function of $2g$ variables. We use the Xilinx-4000 series FPGAs [13] for implementation of the proposed self-checking scheme. In this case the number of inputs of the CLB is equal to 8, which means $g = 4$.

PTC consists of CLBs each programmed for implementation of one of two functions: "*OR*" and "*1-out-of-2g*". If two product terms p_i and p_j, relating to the same output vector, are orthogonal (p_i & $p_j = 0$) they will be combined by the "*1-out-of-2g*" CLB. Otherwise these terms will be combined by the "*OR*" CLB.

The execution block of the FSM (ExFSM) implements OR-assembling of PTC outputs. Outputs of ExFSM are output signals Y of the FSM and input memory signals $D = \{d_1, \ldots, d_R\}$. The memory signals are coded by codewords of the "*1-out-of-R*" code.

B. MD-checker

The MD-checker consists of the evolution block (EvCh), the execution block (ExCh) and the Math Detector (MD) between them. EvCh implements all minterms, while ExCh assembles these minterms to implement the checker's function.

The EvCh is built as a self-checking tree with "AND" nodes for implementation of "long" product terms and "fork" nodes actually implemented by regular fan-outs. The ExCh comprises either "*1-out-of-2g*", or "*(2g -1)-out-of-2g*" cells (CLBs) combining all minterms, coming from the EvCh.

It is proposed to use the following pre-designed A-Cells and O-cells for implementation of the above-mentioned nodes of the checker.

A-Cell implements nodes of the "AND" type. It has two inputs, four functional inputs for implementation of minterms, and two cascade outputs.

O-Cell implements nodes of either "*g-out-of-2g*" or "*(2g -1)-out-of-2g*" types. These cells have 8 inputs and 1 output.

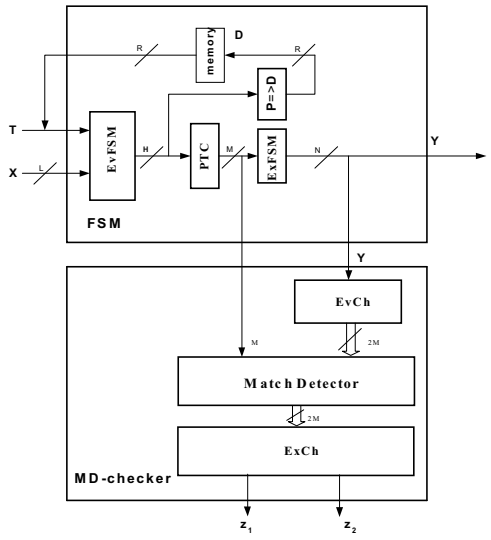

Fig 2. The MD-architecture of the totally self-checking controller

EvCh is a self-checking two-rail tree comprising a number of A-Cells. This tree is constructed in such a way that in the case of proper functioning of both the controller and the checker one and only one dual-rail output (S_i, V_i) will have value (1,0). All the remaining outputs will have value (0,1). Outputs of the EvCh tree are inputs of the ExCh of the checker. Each of the two-rail outputs of the EvCh corresponds to a certain output vector of the original FSM.

ExCh comprises two components consisting of O-Cells. All S-outputs of EvCh serve as inputs of the first component of the ExCh. This component is implemented as a converging "*1-out-of-M*" multilevel tree. All V-outputs of the EvCh are inputs to the second component of the ExCh, which is implemented as a converging "*(M-1)-out-of-M*" multilevel tree.

The Match Detector compares outputs of the PTC and outputs of the evolution block of the checker (EvCh). Any output vector of PTC is formed by M binary one-rail outputs. Output vectors of EvCh are M dual-rail-coded outputs. In Figure 2, the checker is shown as MD-checker. If the two compared vectors are equal, the resulting vector will be equal to the EvCh output vector. If they are not, the ExCh will receive a predetermined faulty dual-rail vector.

An example of the match detection function is shown in Table 2. In this table: $S^1(i)$, $V^1(i)$ - dual-rail code of bit *i* of an output vector of the EvCh; $S^0(i)$, $V^0(i)$ - dual-rail code of bit *i* of the corresponding output vector the match detector (MD); $\delta(i)$ - the state of bit *i* of the PTC single-rail output vector $\delta(Y)$.

Table 2. Truth table of the Match Detector.

$\delta(i)$	$S^1(i)$, $V^1(i)$		$S^0(i)$, $V^0(i)$	
1	**1**	**0**	**1**	**0**
0	**0**	**1**	**0**	**1**
1	**0**	**1**	**1**	**1**
0	**1**	**0**	**0**	**0**
-	**0**	**0**	**0**	**0**
-	**1**	**1**	**1**	**1**

For the MD-architecture at any clock, an input vector initiates "*1-out-of-M*" code of Y_m. This code is introduced both into the ExFSM and the match detector. Output vectors produced by ExFSM are transformed into the same $\delta(Y_m)$ code that has been produced by the PTC. The match detector checks whether these codes are the same, and if the codes differ from one another, the MD-checker will produce the error signal.

IV. Benchmarks results

We applied the synthesis approach described above to several MCNC benchmarks to compute FPGA implementations for Xilinx-4000 series FPGSs [13] Results for benchmarks are presented in Table 3. In this table: *L* - number of input lines, *N* - number of output lines, *H* - number of product terms (transitions), *R* - number of states of the FSM, *M* - number of output vectors;

S^{FSM} and S^{MD} - complexities (numbers of CLBs) of the initial FSM and the MD-checker correspondingly, $\Omega^{MD}=\frac{S^{MD}}{S^{FSM}}*100\%$ S^B, Ω^B - complexity (numbers of CLBs) and overhead (in %) for FSM self-checking implementations based on the Berger coding architecture [1, 6].

Table 3. Overheads results for FSM benchmarks implemented by Xilinx-4000 series FPGAs

NAME	*R*	*L*	*N*	*H*	*M*	S^{FSM}	S^b	Ω^b	S^{MD}	Ω^{MD}
bbsse	13	7	7	53	14	37	38	103%	22	59%
cse	13	7	7	98	12	60	51	85%	24	40%
Dk-14	7	5	3	56	13	38	31	82%	22	58%
Dk-15	4	5	3	30	16	27	26	96%	21	78%
styr	32	10	9	161	25	110	77	70%	52	47%
saund	32	12	11	134	27	116	79	68%	58	50%
S1488	48	19	8	236	64	213	201	94%	153	72%
S1	20	6	8	109	20	116	38	33%	33	28%
pma	24	8	8	120	24	91	83	91%	49	54%
planet	51	19	7	118	54	82	151	184%	135	165%
S820	24	18	18	199	22	175	91	52%	57	33%
Ex6	8	8	5	36	14	45	46	102%	34	76%
Ex1	20	23	9	154	60	74	171	231%	133	180%
tav	4	4	4	49	11	26	23	88%	21	81%
big	18	28	17	185	17	124	87	70%	71	57%

One can see from this table that the proposed approach for design of totally self-checking microcontrollers results in overheads, which are about 65%. Our approach results in about 25-30% reduction of overhead as compared to known implementation [1, 6].

V. Conclusions

We have proposed a novel technique for the synthesis of self-checking FPGA-based FSMs. By utilizing intrinsic features of the FSMs, the proposed architecture allows implementation of controllers by a single-rail scheme without any additional encoding of output words. This results in considerable reduction of the required overhead. Benchmarks results indicate that the proposed approach for the design of self-checking FSMs is efficient from the point of view of required overheads.

References

[1] C. Bolchini, R. Montandon, F. Salince, and D. Sciuto. "Design of VHDL-Based Totally Self-Checking Finite State Machines and Data-Path Descriptions", IEEE Transaction on Very Large Scale Integration (VLSI) Systems, Vol. 8, No. 1, 2000.

[2] B. Eschermann and H. Wunderlich, "Optimized Synthesis of Self-Testable Finite State Machines," Intl. Conf. on Fault Tolerant Computation, 1990.

[3] A. Hertwig. "Utilizing Off-line BIST Circuitry for the On-line Test of FSMs", 4-th International On-line Testing Conference, Capri, 1998, Compendium of papers, pp. 42-46.

[4] V. S. Iyengar, L. L. Kinney. "Concurrent Fault Detection in Microprogrammed Control Units", IEEE Transactions on Computers, Vol. C-34, No. 9, September 1985, pp. 810-821.

[5] P. Lala. Self-checking and Fault-Tolerant Digital Design. Morgan Kaufmann Publishers, San-Francisco/San-Diego/New York/Boston/London/ Sydney/Tokyo, 2000.

[6] I. Levin, M. Karpovsky. "On-line Self-Checking of Microprogram Control Units", 4-th International On-line Testing Conference, Capri, 1998, Compendium of papers, pp. 153-159.

[7] I. Levin, V. Sinelnikov. "Self-checking of FPGA based Control Units", Proceedings of 9th Great Lakes Symposium on VLSI, Ann Arbor, Michigan, 1999, IEEE press, pp. 292-295.

[8] A. Yu. Matrosova, S. A. Ostanin. "Self-Checking FSM Networks Design", 4-th International On-line Testing Conference, Capri, 1998, Compendium of papers, pp. 162-166.

[9] M. Namjoo. "Design of Concurrently Testable Microprogramming Control Units", Proc. Of the 15 Annual Workshop on Microprogramming, Palo Alto, CA., pp. 173-180, Oct. 1982.

[10] A. M. Paschalis, C. Halatsis, G. Philokyprou. "Concurrently Totally Self-Checking Microprogram Control Unit with Duplication of Microprogram Sequencer", Microprocessing and Microprogramming, 20, 1987, pp. 271-281.

[11] M. G. Sami, D. Sciuto, R. Stefanelli, "Concurrently Self-Checking Structures for FSMs". Microprocessing and Microprogramming, 39 (1993) 237-240, North-Holland.

[12] J. Smith, “On Separable Unordered Codes”, IEEE Transaction on Computers, Vol. C-33, No. 8, August 1984.

[13] Xilinx, "The Programmable Logic", Data Book, 1996.