Cascade Scheme for Concurrent Errors Detection[1]

Ilya Levin, Vladimir Ostrovsky
Tel Aviv University, Ramat Aviv, Tel Aviv 69978, Israel,
i.levin@ieee.org, vladio@post.tau.ac.il

Osnat Keren, Vladimir Sinelnikov
Bar Ilan University, Ramat Gan 52900, Israel,
kereno@eng.biu.ac.il, sinel@hait.ac.il

Abstract

The paper deals with synthesis technique for designing circuits with cascade errors detection. The proposed technique is based on partitioning a scheme into a number of cascades followed by parity checking their output logic. The algorithm for partitioning the scheme into cascades is provided.

An universal scheme of Finite State Machine (FSM) with the cascade errors detection is presented and investigated. The scheme does not require any redundant coding variables. Benchmark results are presented and show significantly low overhead requirement.

1. Introduction

Systematic error-detecting coding is one of the most effective instruments for concurrent error detection. This type of coding often utilize separate codes, since such codes allow preserving informational bits of the binary words to be coded, while complementing the binary words by check bits. The coded binary words form a set A of codeword. The set A can be defined either by a list of the codewords, or by a specific property distinguishing the codewords from non-codewords. For example, parity of the sum of binary values of all bits of the codeword can serve such a property. In order to ensure that the codewords differ from non-codewords, each non-codeword $\hat{a}$, formed at an output of a scheme instead of a codeword $a \in A$ due to a specific type of fault, should either not to belong to the set A, or to be equal to a word a. Models of distortion of codewords are usually built taking into account characterizing features of the stream of faults and of the scheme under checking.

As it is accepted in relevant papers, we will consider that faults are manifested by pins signals of "0" or "1" on input or output contacts of logical elements forming the scheme to be checked. The faults can be temporary or permanent. It is traditionally accepted that a time interval between occurrences of two adjacent faults is sufficient for coping with the earliest fault. Therefore, only a single fault can present simultaneously in a scheme under checking. This fact is usually considered when building models of acceptable distortions of output codewords. Most of relevant publications use the following two models of distortions.

The first model is based on an assumption that a system of functions, which reflects conversion of information in a scheme under check, is monotonous. For example, schemes that do not comprise invertors satisfy the mentioned assumption. In such schemes, any single fault may only result in so-called unidirectional faults of output code words [1]. The Berger code [2], and sometimes the Smith code [3] are used for detecting unidirectional faults. A number of algorithms are known [4, 5, 6, 7], which allow converting an arbitrary scheme in such a manner that the Berger code could be used for its checking. To this end, the scheme can be modified in such a way, that only its input variables become negated [4]. The works [5, 6] propose algorithms of converting the scheme under check by duplication of some of its elements. The paper [7] describes a combination of several approaches.

The second model is based on an assumption that a number k of distorted bits in a codeword is not greater than a predetermined threshold t. The paper [8], based on a study of a great number of benchmarks, shows that in most of the cases a single fault results in errors in two or less bits of a codeword $(k \le 2)$

[1] This research was supported by Israeli Science Foundation under grant No. 545/04.

In order to detect faults in a scheme satisfying the assumptions of both the first model and the second model, codes of Bose-Lynn [9] are used. The paper [10] describes an algorithm and a program that allows determining the maximal value of k for an arbitrary scheme, and therefore allows simplifying the checking scheme so that the overhead can be lowered by up to 25%.

Schemes having the threshold $t = 1$ are of a special interest, since they allow the parity checking. However, in this case, the variables to be checked must be implemented by schemes that do not comprise common elements. One example of converting an arbitrary scheme to the mentioned scheme is investigated in [5]. The study shows that independent implementation of output variables in a scheme complicates the scheme approximately twice. The works [5, 11, 12, 13, 14] suggest solutions based on dividing the set of variables under check into groups. Each of the groups comprises the variables, which can be implemented by separate schemes. These groups are complemented with one checking variable and are checked by parity.

And, finally, there is an approach according to which no limitations of possible distortions of a code word are set. In other words, it is assumed that an erroneous codeword $\hat{a}$, obtained as a result of a single fault, may be equal to any other codeword from A. For the above approach, no code exists which could detect such faults. Usually, duplication of the scheme can be useful in such cases. The paper [15] suggests partitioning the set of variables to be checked into groups in such a manner that each group is implemented by a separate independent scheme. Since the schemes, implementing variables belonging to different groups, do not have common elements, only variables of one of the groups can be distorted. This fact is used for detecting faults.

Among the criteria for evaluating various methods of detecting faults, the criterion of reducing the scheme overhead is one of the most important criteria. Simplification of the scheme means not only the cost reduction, the size reduction and reduction of energy consumption; it also means reduction of probability of faults, i.e., improving the scheme reliability. The paper [13] comprises the most complete comparative study of various known methods of checking. According to [13], the minimal overhead can be achieved by the duplication and by the parity check. The difference between the respective results can be of about 10% in the saved overhead. It is noted that sometimes conversion of a scheme for parity check is comparable, by complexity, with duplication of the scheme.

The present paper studies whether the above conversion can be simplified by preliminary partitioning the scheme under check into a number of sequential sub-schemes (cascades) in such a manner, that any constant fault in a cascade would result in distorting of no more than a single bit at the output of the cascade. In this case, the scheme to be checked is additionally equipped only by circuits forming one additional parity bit. Naturally, if parity of the code is constant and known in advance, for example if the cascade is a decoder, no additional bits and circuits for such bits are required. One of the disadvantages of the cascade checking is that the parity check is required not only for output variables of the whole scheme, but also for output variables of each of the cascades.

In order to obtain the total estimation of the proposed method, it was applied for checking sequential circuits described by Finite State Machine (FSM) model. A cascade FSM (CFSM) was developed and investigated. The scheme comprises three cascades, the output variables are checked only at two of the cascades.

The paper is organized in the following order. Section 2 describes, in detail, the methods of partitioning a scheme into cascades. FSM with the cascaded checking is presented in Section 3. Section 4 comprises results of studying overhead of the CFSMs.

2. Partitioning schemes into cascades

Our task is to represent a combinational scheme in the form of cascades connected in sequence, and in such a manner that any single fault in a cascade results in distortion of no more than one bit at the output of the cascade. In the paper, the fault will be understood as a constant stuck-at fault at an input or at an output of an element. Let us consider the task in two versions. In the first version, the scheme to be checked is given and the only problem is to distribute its component elements between the cascades. Let us call such first version *the structural cascade decomposition.* In the second version, the scheme to be checked is unknown but its functional description is available. The problem is to divide the scheme into cascades while designing it. Such a second version will be called *the functional cascade decomposition.*

The structural cascade decomposition. Let us consider a scheme having no feedback connections. En exemplary scheme is illustrated in Fig. 1.

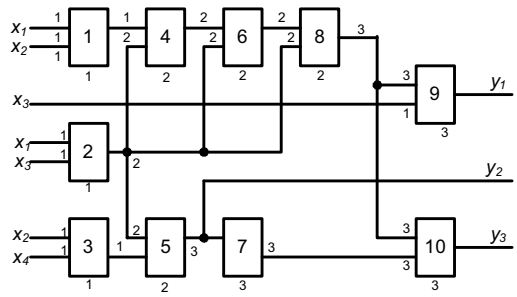

Figure 1. Exemplary scheme of the structure cascade decomposition.

The numbering of input and output pins of the scheme belongs to the proposed decomposition method and will be explained below.

It should be noted that, for proper conversion of the scheme, functions implemented by elements of the scheme are not important; only connections there-between are important. The basic condition of the algorithm of the structural cascade decomposition is as follows: an element which has outputs with a splitting coefficient two and more then two can be situated only at the output of the cascade. It is obvious that if the above condition is satisfied, all outputs of the cascade will be implemented by independent schemes and, consequently, any fault may affect the value of maximum one output variable.

In order to partition the scheme into cascades, let us number the component elements and their outputs according to the following rules:

- assign to each specific element a number equal to the maximal number among those assigned to the inputs of the element; this number is also the cascade level.
- assign a number c to each specific non-fan-out output of an element having number c;
- assign a number $c+1$ to each split output of an element having number c;
- assume that the number of an input of an element is equal to the number of an output connected to that input.

Let us start the numeration from input of the scheme, assuming that number "1" is assigned to input pins of the scheme. After running the algorithm, the number of an element is the same of its cascade. An example of the numeration is shown in Fig. 1, and the result of the cascade decomposition – in Fig. 2.

As for the checking, the scheme should be accomplished with a circuit for parity prediction and with a checker which comprises an EXOR block and a two-rail checker.

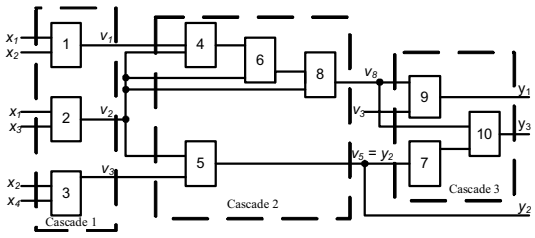

Figure 2. The result of the cascade decomposition of scheme from Figure 1.

Note that the cascade allows parity checking even if it comprises fanouts. Therefore, the condition of absence of fanouts is sufficient but not necessary. However, in order to remove the mentioned limitation, a specific functional analysis should be performed.

The functional cascade decomposition. Let we have a system of Boolean functions $y_i = F(x_1,\ldots,x_L), i = 1,\ldots,N.$ The task is to implement the system in the form of a scheme divided into cascades which are checked by parity. In other words, the necessity of cascade operation should be taken into account at the stage of developing the scheme. The task does not have a single solution.

We consider only two-level implementation of a Sum-of-Products representation of a logic function.

***Theorem* 1**. If output variables y_i and y_j of a two-level logic scheme are mutually disjoint $(y_i \,\&\, y_j = 0)$, and both have values "0" in a codeword, there is no such a fault in the scheme, which would result in changing values of the both variables to "1" in the same codeword.

The Proof of the Theorem follows from the fact that any two output variables y_i and y_j are disjoint and, thus, don't include any common products.

***Consequence* 1**. If the two-level scheme allows occurrence of only unidirectional faults, and its outputs are coded by $1\text{-}out\text{-}of\text{-}N$ code, a single fault may result in changing value of no more than one output variable.

Proof of the consequence is obvious. If the conditions of the consequence are fulfilled, the parity of all the code words is the same, and no parity prediction is required for checking additional check bits. Implementation of the above definitions is illustrated by the example of FSM having a cascade checking arrangement.

3. FSM with Cascaded Checking

We propose a new architecture for the FSM and a checker. The architecture does not require any encoding of output vectors and consequently allows reduction of the required overheads.

The FSM consists of three blocks: an "evolution" block, an "execution" block and a Product Terms Compressor (PTC) [16]. A schematic diagram of the self-checking FSM is shown in Figure 3.

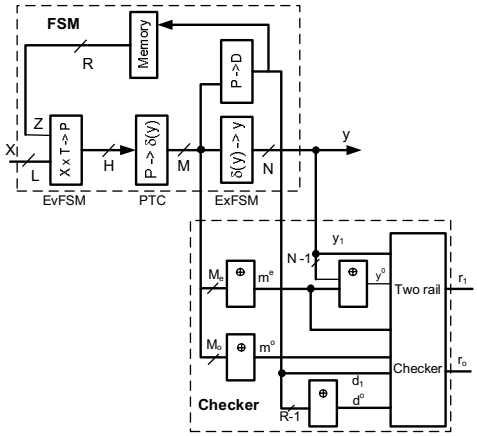

Figure 3. A schematic diagram of the self-checking FSM.

Inputs of the evolution block of the FSM (EvFSM) comprise working inputs X of the FSM and output memory signals $Z = \{z_1, \ldots, z_R\}$. Outputs of the EvFSM correspond to product terms $P = \{p_1, \ldots, p_H\}$.

Outputs $\{m_1, \ldots, m_M\}$ of the PTC correspond to of output codewords. At each clock, one and only one product term is equal to 1, which means that EvFSM outputs are codewords of the $1-out-of-H$ code. The EvFSM is denoted in Figure 3 as $X \times Z \to \delta(P)$. The Product Terms Compressor (PTC) transforms $1-out-of-H$ code $\delta(P)$ into $1-out-of-M$ code of $\delta(Y)$. The number of codewords is essentially smaller than the number of product terms for the typical FSM.

The execution block of the FSM (ExFSM) implements OR-assembling of the EvFSM outputs. Notice that each of output signals is formed by its own independent logic circuit. 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.

The checker comprises a number of EXOR based adders and a two-rail checker. M outputs of PTC are partitioned into two subsets. The first of the subsets includes outputs corresponding to codewords with even number of ones, while the second subset corresponds of codewords with odd number of ones. Signals of each of the subsets are assembled by EXORs. As a result, indicators "even" m^e and "odd" m^o are generated.

Obviously, unequal combinations of the checker's outputs m^e and m^o correspond to codeword occurrence on the FSM's outputs. According to Consequence 1, any fault may affect to the value of not more than one of PTC outputs and, consequently, leads to equality of outputs m^e and m^o.

In order to check output signals, one of the signals (for example, y_1) is distinguished while the rest of the signals are EXOR-summed with even signal m^e: $y^o = y_2 \oplus \ldots \oplus y_N \oplus m^e, y^1 = y_1$. Signals $d_1, \ldots, d_R$ are checked in similar manner taking into account that their codewords belong to $1-out-of-R$ code: $d^o = d_2 \oplus \ldots \oplus d_R, d^1 = d_1$. Codewords is indicated by inequality of three pairs of signals: $(m^o, m^e), (d^0, d^1), (y^0, y^1)$. This fact is checked by the two-rail checker.

Example

Let us illustrate the above description on an example of a reversible counter. A table of transitions/outputs of the counter is shown as Table 1.

Table 1. Transition/output table for the example

$x_2\, x_1$	S_1	S_2	S_3
0 0	S_1/11	S_1/00	S_1/11
0 1	S_2/00	S_3/00	S_1/01
1 0	S_3/10	S_1/00	S_2/00

The scheme of the FSM, partitioned into cascades, is presented in Fig 4. The scheme of a checker is presented in Fig. 5.

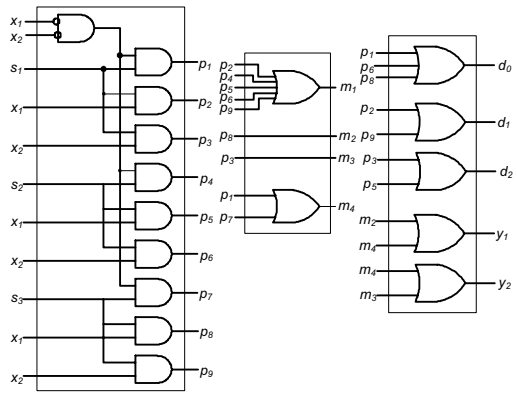

Figure 4. Exemplary cascade FSM.

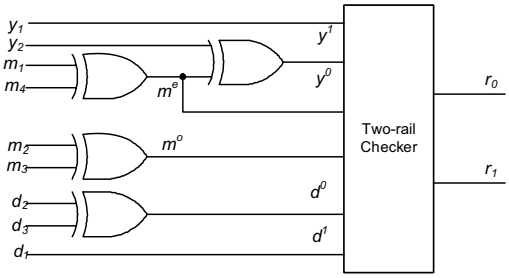

Figure 5. The checker for the cascade FSM.

Let us define and prove the property of the cascade checking to detect any errors in an output code word caused by faults of the considered class.

***Theorem* 2**. CFSM is fault secure with respect of faults of the considered class.

Proof: According to the consequence 1, no more than one bit can be erroneous in the binary word: $m_1,\ldots,m_M$. If such an error takes place, the variables m^o and m^e will take equal values, which will result in manifestation of the fault on the output of the two-rail checker, independently of values of other signals on its input. If the word $m_1,\ldots,m_M$. is free of errors, then an error in the output word $y_1,\ldots,y_N$ is also possible only in one single bit thereof. Similarly to the previous case, if a single bit error occurs in the output word, it will lead to the fact that $y^1 = y^0$, and $r_0 = r_1$. It can be seen that the system detects any errors in the word $y_1,\ldots,y_N$. Signals $d_1,\ldots,d_R$ for checking flip-flops are checked in a similar way as the output signals. If a flip-flop is faulty, it will be detected in one of the next clocks, in the form of erroneous (different from $1-out-of-R$) values of the words $m_0,\ldots,m_M$. and/or $d_1,\ldots,d_R$.

Notice, that the self-testing property of the CFSM is obvious, and thus the CFSM is *totally self-checking*.

4. Experimental results

In order to examine the proposed approach for design of self-checking CFSMs a number of experiments on standard MCNC benchmarks were provided. The experiments were made by using Leonardo Spectrum Synthesis tool. LUT-based FPGA Xilinx Spartan-3 were used as a basis for the implementation of CFSMs. Results of the experiments are presented in Table 2.

Table 2. Experimental results

	NAME	*N*	*M*	*R*	W(FSM)	W(CFSM)	W(Ch)	W%
1	bbsse	7	11	13	37	47	13	62
2	bbtas	2	4	6	12	16	8	100
3	beecount	4	4	7	31	29	8	19
4	cse	7	11	16	72	90	15	46
5	Dk14	5	12	7	48	47	11	21
6	Dk15	5	11	4	14	24	10	143
7	Dk16	3	5	27	64	73	4	20
8	Dk17	3	5	8	22	24	9	50
9	Dk512	3	4	15	18	21	11	78
10	Ex1	19	60	20	104	195	36	122
11	Ex4	3	5	3	6	6	7	117
12	Ex6	8	12	8	43	46	12	35
13	keyb	2	3	19	71	86	11	17
14	planet	19	54	48	136	182	45	67
15	pma	8	24	24	122	119	22	16
16	S1	6	20	20	95	85	18	8
17	S208	2	4	18	36	48	13	69
18	S298	6	5	218	549	570	80	18
19	S386	7	11	13	43	54	13	56
20	S420	2	4	18	36	48	12	67
21	S510	7	13	47	78	74	25	27
22	S820	19	22	25	91	132	25	72
23	S832	19	22	25	88	122	26	68
24	S1488	19	64	48	172	264	47	81
25	sand	9	27	32	141	151	26	25
26	Scf	54	39	121	147	194	75	83
27	sse	7	11	16	42	54	13	59
28	styr	10	25	30	133	148	21	27
29	tbk	3	5	32	256	297	17	23

The three columns in the table, which follow the FSM benchmark's capture, respectively show characteristics of its complexity:

N – the number of the outputs, M – the number of codewords, R – the number of states. The columns $W(FSM)$ and $W(CFSM)$ respectively characterize the complexity of the initial and the cascade schemes, presented as the number of LUTs. The column $W(Ch)$ corresponds to the complexity of the LUT-based checker. Finally, the last column presents the scheme's overhead W in percents.

As can be seen from the table the overhead changes in relatively broad limits. Large values of the overhead (100% and higher) can usually be found in relatively simple FSMs, while a small complication of about 10-20 LUTs results in essential increase of the overhead. Approximately for one third of the benchmarks, the overhead does not exceed 30%, and for half of the benchmarks – it does not exceed 50%. Notice, that the relative overhead decreases with increase of complexity of the FSMs.

For some benchmarks, the CFSM implementation is simpler than FSM implementation (say, for *beecount, Dk14, pma*, and some more). The meaning of such a feature is that the partitioning of a sequential scheme into cascades creates additional possibilities for simplification of the scheme which possibilities are not utilized by standard tools.

When estimating the obtained results, it can be concluded that the proposed method allows achieving the totally self-checking property for a very essential group of the FSMs and by relatively low overhead.

5. Conclusions

The paper proposes a method designing self-checking FSMs. The method is based on partitioning of a scheme to be checked into cascades, and in such a manner that any single fault in any cascade could result in distortion of no more that a single variable at the output of the cascade. The proposed method suggests checking all output variables of all cascades of the scheme EXOR. The algorithm for partitioning the scheme into cascades has been developed.

Based on the proposed method the authors obtained and studied a universal FSM-scheme with the cascade checking. The universal scheme comprises three cascades and does not use any coding variable.

The benchmark study has shown that the method does not require large overhead. For example, in one half of the cases the overhead did not exceed 50%, and in some cases it was even lower than 20%.

Notice, that the proposed cascade structure allows equalizing signal delays in various circuits and also allows increasing the efficiency owing to the use of pipeline processing of operation**.**

6. References

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

[2] Berger, J. M., "A Note on Error Detection Codes for Asymmetric Channels", *Information and Control,* Vol. 4, 1961, 68-73.

[3] J. E. Smith, "On separable unordered codes", *IEEE Trans. Computers*, vol. C-33, no. 8, Aug., 1984, 741-743.

[4] N.K. Jha and S.-J. Wang, "Design and Synthesis of Self-Checking VLSI Circuits", *IEEE Transaction CAD,* Vol. 12, No. 6, 1993, 878–887.

[5] Kaushik De., Chitra Natarajan, Devi Nair, Prithviraj Banerjee, "RSYN: A System for Automated Synthesis of Reliable Multilevel Circuits", *IEEE Transaction on Very Large Integration (VLSI) Systems,* Vol. 2, 1994, No. 2, 186-195.

[6] V.V. Saposhnikov, A. Morosov, Vl. V. Saposhnikov, M. Gössel., "A New Design Method for Self-Checking Unidirectional Combinational Circuits", *Journal of Electronic Testing: Theory and Applications,* 12, 1998, 41-53.

[7] C. Metra, S. Francescantonio, M. Omana., "Automatic Modification of Sequential Circuits for Self-Checking Implementation", *Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'03),* 2003, 417-424.

[8] I. Pomeranz and S.M. Reddy., "Recovery During Concurrent On-Line Testing of Identical Circuits"., *Proceedings of the 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'05*), 475-483.

[9] Bose, B. and D. J. Lin, "Systematic Unidirectional Error-Detecting Codes", *IEEE Trans. Comp.,* Nov. 1985, 1026-1032.

[10] M. Omana, O. Losco, C. Metra, A. Pagni. "On the Selection of Unidirectional Error Detecting Codes for Self-Checking Circuits Area Overhead and Performance Optimization"*., Proceedings of the 4-th IEEE International On-line Testing Symposium,* pp. 163-168.

[11] E. S. Sogomonyan, Design of Built-in Self-Checking Monitoring Circuits for Combinational Devices, *Automation and Remote Control, vol. 35, No. 2, 1974, 280-289.*

[12] N.A. Touba and E.J. McCluskey, "Logic Synthesis of Multilevel Circuits with Concurrent Error Detection", *IEEE Transactions on Computer-Aided Design,* Vol. 16, No. 7, 1997, 783-789.

[13] S. Mitra and E. J. McCluskey. "Which Concurrent Error Detection Scheme to Choose?" *Proceedings International Test Conference,* 2000, p. 98.

[14] V. V. Saposhnikov, A. Morosov, VL. V. Saposhnikov, M. Gössel. "Design of Self-Checking Unidirectional Combinational Circuits with Low Area Overhead", *Proc. 2nd Int. On-Line Testing Workshop, 1996.*

[15] V. Ostrovsky, I. Levin. "Implementation of Concurrent Checking Circuits by Independent Sub-circuits", *Proceedings of the 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'05*), 2005, 343-351.

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