Self-Checking of FPGA-based Control Units

Ilya Levin and Vladimir Sinelnikov
Department of Computer Science
Center for Technological Education
Affiliated with Tel Aviv University
52 Golomb St., P.O.Box 305
Holon 58102, Israel
ilia1@post.tau.ac.il
sinel@barlay.cteh.ac.il

Abstract

The paper introduces a new technique for on-line checking of FPGA based Control Units (CUs). This technique is based on the architecture comprising two portions: a self-checking CU and a separate totally self-checking (TSC) checker. Each of these portions is implemented as a combination of an Evolution block and an Execution block. Comparison of code vectors being transferred between the blocks of the portions enables providing a totally self-checking property. The self-checking CU is implemented in a form of one-rail network of interconnected pre-designed LUT-based configurable logical blocks. The self-checking checker is a Sum-Of-Minterms based checker. The proposed technique: a) does not require any encoding of output words; b) uses one-rail design, thereby drastically decreasing the required overhead.

1. Introduction

A control part of a digital system is usually the most critical part from the testability point of view. Irregularity and complexity of the control structure on one hand, and its central role in functioning of the whole digital system to be controlled on the other hand, puts the problem of synthesis of self-checking Control Units (CU) onto the theoretical and practical agenda. Most of the faults that occur in VLSI circuits and systems are transient/intermittent in nature. The self-checking property allows both the transient/intermittent and permanent faults to be detected, thus preventing data contamination.

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

According to [1], in the synthesis of self-checking Control Units, some basic properties of Finite State Machines (FSM) being a formal model of CUs can be utilized for minimizing a resulting overhead. Particularly, a self-checking checker for the CU can be implemented as a Sum-of-Minterms (SOM) of a Boolean function $z(y_1,\ldots,y_N)$ of the CU output variables $y_1,\ldots,y_N$. Function z is equal 1 if the CU output word is a codeword, and equal 0 if the CU output word is not a codeword. Each of minterms of $z(y_1,\ldots, y_N)$ corresponds to a specific codeword of the CU.

In [1] a method of PLA-based implementation of the above idea was investigated. The remarkable property of a CU, i.e. the orthogonality of its transition functions, was utilized. It allowed minimizing resulting overhead.

The only paper [3] dealing with on-line self-checking for the cases of FPGA. Its idea is based on the dual-rail design allowing on-line detection of faults within a FPGA. An additional circuitry that introduced into the scheme enables to produce identical error detecting outputs in a case of a fault. The methodology allows implementation of Boolean expressions using sets of pre-designed cells that have complementary outputs. If a fault occurs within a cell or on any single line, identical outputs will be produced. Cells are cascaded and combined into a final cell.

This methodology has some limitations. Firstly, it is inapplicable for implementations of systems of logical functions. Secondly, it does not provide detection of faults on inputs of configurable logic blocks (CLBs). Alternatively, if the approach [3] would be applied for implementation of the separated SOM-based checker,

0-7695-0104-4/99 $10.00

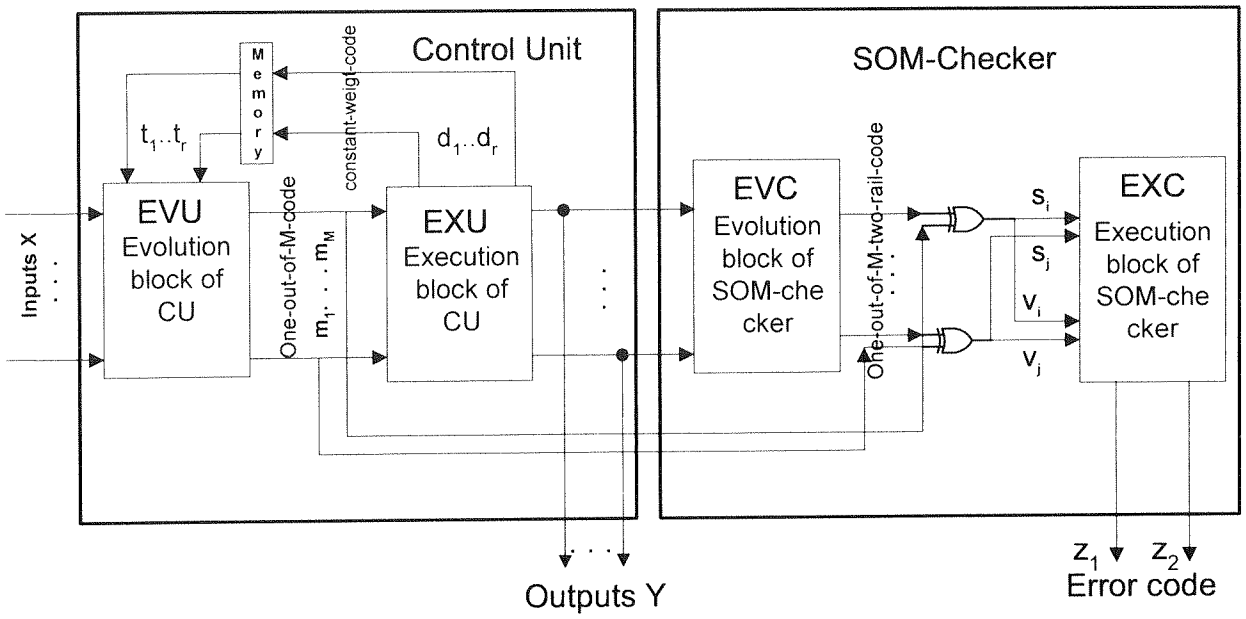

Figure 1. A schematic diagram of the architecture of the totally self-checking Control Unit.

this disadvantage would be prevented since any input fault will be interpreted as a non-code word and, consequently, would be detected.

Summarizing one can say that the FPGA-based self-checking CU can be implemented in a form of combination of dual-rail CU and the SOM-based checker. In this case output words of the CU have to be encoded by the code detecting unidirectional errors. Needless to say that this implementation is critical from the point of resulting overhead due to both the dual-rail design and the Berger encoding.

In this paper we suggest a new solution, suitable for FPGA implementation of CU, which allows substantial reduction of a required overhead. The proposed self-checking CU does not require any encoding of codewords and can be implemented by using a single-rail design.

Our technique is based on a novel architecture of the self-checking Control Unit. According to this architecture the totally self-checking control unit (TSCU) consists of the following two portions: a) the self-checking CU and b) the SOM checker. Each of these portions can be considered as a combination of two blocks: an Evolution block and an Execution block.

The main idea, which allows a sufficient decrease of the overhead is an idea to compare a binary vector being transferred at the time between the evolution and the execution blocks of the CU with a binary vector which passes at the time between the Evolution and Execution blocks of the SOM-checker. These two vectors have to be a) equal, b) belong to 1-out-M code. The comparison operation between these two vectors can be implemented with a very small overhead.

We propose to implement the Evolution block of the CU by using a one-rail scheme. The SOM-based self-checking checker (SOM checker) is implemented in a form of a network of interconnected pre-designed cells. This network is constructed in such a way that any fault occurring within the checker will be indicated by appearance of a fault signal at the output of the checker.

2. Architecture of the self-checking Control Unit

Let the CU to be implemented has the set of inputs $X=\{x_1, \ldots, x_L\}$, the set of outputs (microoperations) $Y=\{y_1, \ldots, y_N\}$. The set of possible output words of the CU constitutes the set of microinstructions. Each of the microinstruction is a subset of Y. The number of these microinstructions will be denoted M.

A schematic diagram of the proposed architecture of a totally self-checking Control Unit (TSCU) is shown in Fig. 1. The TSCU comprises two portions: the self-checking CU and the SOM-checker. In turn each of these portions consists of two blocks: the Evolution block and the Execution block.

Output vectors of the Evolution blocks of the mentioned portions are compared by comparing of corresponding components thereof. The resulting vector forms inputs of the Execution block of the SOM-checker.

Let us move to a detailed description of the proposed architecture.

2.1. Self-checking CU

Inputs of the Evolution block of the CU (EVU) comprise working inputs of the TSCU X and memory signals T. Outputs of the EVU are signals of microinstructions. At each clock, one and only one microinstruction is equal 1, which means that the 1-out-M code is implemented on EVU outputs.

We use the Xilinx-4000 series architecture [2] to illustrate the proposed method of designing a

self-checking FPGA-based CU. We implement the EVU in a form of a tree having nodes, which are pre-designed CLBs and fan-outs. Each CLB is designed for implementation of either AND-function of eight variables or a sum of two product terms of four variables.

The Execution block of the CU (EXU) comprises OR-assembling of EVU outputs. Outputs of EXU are output signals Y of the CU and memory signals D. These memory signals are coded by a "constant weight code".

2.2. SOM-based TSC checker

The Evolution block of the SOM-checker (EVC) implements all minterms each corresponding to a codword of the CU. The Execution block of the checker (EXC) implements the sum of the minterms.

Inputs of the EVC are outputs of the initial CU. The EVC is implemented in the dual-rail style. Output codes of the EVC belong to the 1-out-M two-rail code.

If at a certain clock the CU implements microinstruction m_i (m_i =1), then corresponding dual-rail output of EVC implements code (10). Each dual-rail output of the EVC is controlled by a corresponding single-rail output of the EVU. It implemented by a system of two logical functions S_iV_i, defined by the following truth Table 1.

Table 1. Truth table of the comparing element

Output of EVCU	m_i	S_iV_i	Comments
10	1	10	active state
01	0	01	passive state
10	0	00	faulty states
01	1	11	
00	-	00	
11	-	11	

The EVC is built as a self-checking dual-rail tree. It cnodes of two different types: "fork" nodes and "AND" nodes. The EXC consists of two parts having outputs z_1 and z_2, implementing functions 1-out-M and (M-1)-out-M correspondingly.

We propose to use pre-designed cells 1, 2, 3 for implementation of the above mentioned nodes of the SOM-checker.

1. Cell 1 implements node of the "AND" type and constitutes a CLB, pre-designed as a dual-rail "AND" function of four variables.
2. Cell 2 implements node of the 1-out-M function and constitutes CLB of eight variables.
3. Cell 3 implements node of the (M-1)-out-M function of eight variables.

CLB's of the last level of the EVC tree implement both "AND" function of three variables and comparison function according to Table 1.

The Execution block of the SOM-checker comprises cells 2, 3 which act to combine all S_i by the checker's function $z_1(y_1 \ldots y_N)$ and all Vi by the checker's function $z_2(y_1 \ldots y_N)$.

2.3. Analysis of the totally self-checking property

In a case of a fault free functioning one and only one microinstructiojn m_i is performed on any clock. It means that:

1. $(m_i = 1) \& \forall (j \neq i)\ m_j = 0$;
2. $((S_i = 1) \& (V_i = 0))$ (*active state*) $\& \forall (j \neq i)$ $((S_j = 0) \& (V_j = 1))$ (*passive states*);
3. $z_1(y_1 \ldots y_N) = 1$, $z_2(y_1 \ldots y_N) = 0$.

Faults of the EVU are unidirectional. Consequently, any fault in this block will lead to a non-1-out-M codeword at the outputs of the EVU. This fault will be detected by appearance of $z_1, z_2 \neq (10)$.

Faults of the EXU will lead to appearance of a non-codeword at the outputs of CU. Consequently, no path of the EVC will be activated and checker's outputs will be equal to (01).

Faults of the EVC lead to appearance of non-1-out-of-M two-rail codeword at the outputs of the EVC, or to such a 1-out-of-M codeword that S_i=1.V_i=0, but $m_i \neq 1$. In both cases, the checker's outputs will be unequal (10). There are no single faults in the EVC, which cannot be detected at the outputs of the EVC, since EVC is realized as a two-rail circuit and all LUT's and interconnected faults are detected at the EXC as non-two-rail-code.

Any fault at the EXC is detected by appearance at the outputs of the checker z_1, z_2 of such a value, which is not equal (10) (except output's faults s-a-1, s-a-0, which may lead to appearance code (10) at the checker's outputs). Fortunately, this situation is not critical since the checker continues functioning properly even with the presence of this fault. Indeed, if there is no fault both in the CU and in the EVC, the checker works properly. Only multiple faults (if occur simultaneously in more then two blocks) can be missed. Based on the above, the proposed architecture of the self-checking CU, has the totally self-checking property.

3. Estimation of the Overhead

Results for benchmarks are presented in Tab. 2. In this table: L, N, R, H and M are defined as follows:

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 possible codewords (microinstructions).

S_0 and S_E are numbers of CLBs required for the initial CU and its self-checking version, correspondingly, $\Omega_E=(S_E-S_0)/S_0*100\%$, $\Omega_L=(S_L-S_0)/S_0*100\%$. S_L and Ω_L refer to the corresponding parameters of the [3] scheme, implemented as SOM based checker.

Table 2. Results for FSM benchmarks implemented by FPGAs

Name	L	N	R	H	M	S_0	S_E	Ω_E	S_L	Ω_L
big	18	28	17	185	17	124	195	60%	251	100%
bs	19	13	17	185	17	125	168	34%	202	62%
acdl	16	27	22	214	23	158	247	56%	317	101%
cow	49	24	24	261	18	262	346	32%	402	53%
vl_6	14	18	17	169	17	74	119	60%	146	97%
vl_10	15	18	18	264	18	102	151	48%	187	83%
vl1_20	14	29	18	367	17	110	181	65%	247	125%

The above results indicate that overheads in the self-checking CU according to our approach may be essentially lower than overheads required for implementation of the method proposed in [3].

4. Conclusion

We have proposed a new approach in the area of synthesis of self-checking FPGA-based Control Units (CU). In spite of tremendous strides made in the theory of self-checking devices design efficient synthesis procedure for design of self-checking CU by FPGA has not been developed. We have tried to fill this vacuum by proposing a novel architecture of self-checking CU. Utilizing several intrinsic features of CU, the proposed architecture allows implementation the CU by one-rail scheme and without any additional encoding of output words. These two facts enable obtaining good solutions from the point of resulting overhead.

References

1. 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.

2. Xilinx, "The Programmable Logic", Data Book, 1996.

3. A. L. Burress and P. K. Lala. "On-line Testable Logic Design for FPGA Implementation", Proc. of 1997 International Test Conference, pp. 471-478.

4. S. Baranov. Logic Synthesis for Control Automata. Kluwer Academic Publisher. Dordrecht/Boston/London. 1994.