# DESIGN AUTOMATION OF FPGA BASED RELIABLE CHECKERS

V. Ostrovsky, S. Perelman, I. Levin,
Tel-Aviv University, Ramat Aviv, 69978, Israel

This work presents a CAD tool incorporating several approaches to the design of reliable checkers for totally self checking control units. Some of the approaches are original. To achieve the required properties, unidirectional error detecting codes are applied to the outputs of the control units. Traditionally, minimum check bits in the code serves as the criterion for its selection. The new proposed approaches to construction of the codes aim at reducing the resulting hardware overhead. The hardware architecture is FPGA, namely dual-LUT 4-input 2-output CLBs. A fault model for the checker is a failure of a single LUT, which changes its output function.

## 1. Introduction

Construction of checkers (circuits designed to detect errors) using LUT-base field-programmable gate arrays (FPGA) requires special handling and makes automated design worth considering.

A popular opinion exists that there is no way to implement self-testing checkers by FPGA, since a fault in any of the LUTs of the checker can result in an arbitrary function on the LUT output, that is not the intended function. This fault model makes the method of multiple module redundancy less reliable. The redundancy, however, significantly increases complexity of the checker, which makes the problem of minimization of its hardware especially important.

Unordered codes are used to detect unidirectional errors. This work deals with separable codes only (codes that explicitly contain information bits in code words): the universal Berger code [1], the context-oriented (aimed at a specific codeword set) Smith code [2] and a *context-oriented m-out-of-n* code [3]. The first two of the above codes are widely known. Each of them is optimal in their own code class that means that they achieve the unordered property by using a minimal number of additional (check) bits.

In the present paper we propose a *context-oriented m-out-of-n* code. It was shown [3] that, in a number of cases, this code allows not only simplifying the structure of the checker, but also reducing the overall hardware overhead required to detect the faults and errors of the considered types. The simplification is achieved by the use of a greater number of check bits (compared with the Smith code).

Applying the universal Berger code does not require any special checker design. Standard solutions exist, and the hardware they require only depends on the number of information bits in the output word of the checked device.

In turn, a context-oriented code requires: a) adaptation to a pre-defined set of codewords and b) development of a special checker, which is only applicable to checking this specific pre-defined set. The resulting hardware overhead depends significantly both on a specific coding and on the checker design. In the focus of the present work is automation of designing such checkers.

We present a software tool that allows finding the best one among the three above mentioned codes, to encode a given set of words, and to construct the checker by a LUT based FPGA.

This paper is built as follows. Section 2 describes general principles of unidirectional error detection and checkers construction. Section 3 introduces the *context-oriented m-out-of-n* code. The structure of the checkers and methods of their design are presented in Section 4. Results of a specially developed software tool for the design automation, on a number of benchmarks, are provided in Section 5.

## 2. Unidirectional errors and their detection

Let $y_1,\ldots,y_\ell$ be the (binary) variables at the output of the device to be checked. In general, it is possible that output vectors take not all $2^\ell$ possible values, but only a subset of them. Denote the subset of the output vectors in use as $W = \{X_1, X_2, \ldots, X_M\}, M \le 2^\ell$.

We assume that a single fault in the controlled device leads to a unidirectional error in the checked vector, i.e. to the corruption of either "0"s only, or "1"s only [4]. In order to allow the detection of such errors, the vector is encoded by a unidirectional error detecting code. The encoded vectors are pairwise unordered. Therefore, the set of encoded legal vectors and the set of vectors obtainable from them through unidirectional errors are disjoint. This property is utilized for errors detection of the above type.

Traditionally, the design of self-checking devices consists of two steps. First, the vectors are encoded, and thus $k$ check bits $c_1,\ldots,c_k$ are added to the $\ell$ information bits, for a total of $n = \ell + k$ bits. According to the acceptable assumption that fewer bits correspond to a simpler and smaller checker, codes with fewer check bits are favorable. A code which achieves the required properties using the minimum number of check bits is called optimal.

In our case, there are two optimal codes: 1) the universal Berger code [1]

($k = \lceil \log_2(\ell+1) \rceil$) if $M = 2^{\ell}$, i.e. all bit combinations are possible at the output; 2) a context-oriented code, if $M << 2^{\ell}$. In this case the information redundancy may be used to reduce the number of check bits *k*. Naturally, universality of the code will be lost, and it can only check this particular set of vectors. We call this code *context-oriented (CO)*. The optimal CO unidirectional error detecting code was proposed by Smith [2].

It so happens, that a shorter code does not ensure a smaller checker. The dropping of the intermediate criterion of minimum check bit have led to the creation of a so-called *separable context-oriented m-out-of-n* code. In some cases this code allows simplifying the checker's structure, compared to the Smith and the Berger codes. The code is described in Section 3.

The second step of the design of self-checking devices is construction of the checker. In general, the checker should be *self-testing* (ST), i.e. for any fault there is at least one legal input to the checker, which results in an illegal output. The ST property is not achievable for checkers implemented in FPGA, unless a special fault model is used to limit the types of LUT failures. This means that we shall use the double-module redundancy in our checker [5]. Let the checker consist of two identical blocks, the first one having dual-rail outputs $r_{11}$, $r_{12}$, the second $r_{21}$, $r_{22}$. If both the checked device and the checker are operating correct, then $r_{1j} = r_{2j}, r_{i1} \neq r_{i2}, \forall i, j \in \{1,2\}$. We know that the checked device is malfunctioning when either $r_{11} = r_{12}$ or $r_{21} = r_{22}$. If $r_{11} \neq r_{21}$ or $r_{12} \neq r_{12}$, there is something wrong with the checker.

## 3. Context-oriented m-out-of-n code

Let we have a set of binary vectors $X_1,\ldots,X_M$, in which the bit positions are variables $y_1,\ldots,y_{\ell}$. The following compatibility relations between pairs of the variables can be defined:

**Definition A**: *Two variables $y_i$ and $y_j$ are compatible if there is a vector X in which both $y_i$ and $y_j$ are nonzero. Otherwise, they are incompatible.*

**Definition B**: *A subset $V_i \subseteq V$ is a subset of (in-) compatible variables if any two variables in that subset are (in-) compatible.*

Clearly, if a particular $V_i$ is a subset of incompatible variables, then each vector in *V* may contain at most one variable from this subset (one or none at all). Based on the above definitions, we design a new code, which allows the detection of any unidirectional error in the circuit.

**Definition C**: Let *V* be partitioned into disjoint subsets of incompatible binary variables:

$$V = \{V_1, V_2, \ldots, V_m\}$$
$$V_i \cap V_j = \emptyset, \forall i, j \in \{1,2,\ldots,m\}, i \neq j$$

In the code, we assign each subset $V_i$ with one code bit $c_i$, the value of $c_i$ is ‘1’ in the vectors which contain no variables from $V_i$, and in these vectors only. Clearly, each thus encoded output word will have exactly $m$ ‘1’ bits, equal to the number of subsets $V_i$. This is why we call this code the context-oriented m-out-of-n code.

Let's illustrate the above with an example. Table 1 shows the code for the benchmark MCNC, dk15. The original output bits are in columns $y_1$-$y_5$, the check bits are in columns $c_1$-$c_3$ (for *CO m-out-of-n* code), $s_1$-$s_2$ (for Smith code) and $b_1$-$b_3$ (for Berger code).

Table 1. The context-oriented m-out-of-n code for benchmark dk15

| | *information bits* | | | | | *CO m-out-of-n* | | | *Smith* | | *Berger* | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $y_1$ | $y_2$ | $y_3$ | $y_4$ | $y_5$ | $c_1$ | $c_2$ | $c_3$ | $s_1$ | $s_2$ | $b_1$ | $b_2$ | $b_3$ |
| $X_0$ | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| $X_1$ | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| $X_2$ | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| $X_3$ | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| $X_4$ | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| $X_5$ | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| $X_6$ | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $X_7$ | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| $X_8$ | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| $X_9$ | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| $X_{10}$ | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |

The *compatibility* relation between the pairs of variables is defined by the given set of words and are presented in figure 1. On the graph, an arc connects two nodes $y_i$ and $y_j$ if and only if the $y_i$ and $y_j$ are compatible.

The partition upon which the encoding is based is $\{y_1, y_2, y_3, y_4, y_5\} = \underbrace{\{y_1, y_2\}}_{V_1} \cup \underbrace{\{y_3, y_4\}}_{V_2} \cup \underbrace{\{y_5\}}_{V_3}$. We call this the coding partition.

After the partitioning, the encoding procedure is simple: for all *i=1,2,3*, $c_i$=*‘1’* for a codeword that has no ‘1’ bits from the subset $V_i$, and $c_i$=*‘0’* otherwise. This way, in each codeword there is a single ‘1’ bit from each of the sets $V_i \cup c_i$.

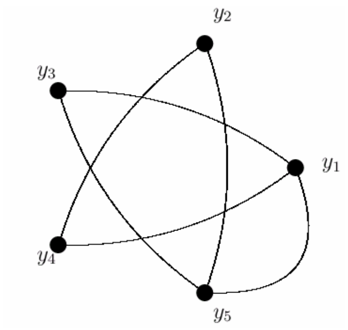

Figure 1. A graphical representation of the compatibility relations among bits of benchmark dk15.

## 4. Construction of checkers and automation of the process

In this section, the checkers for each of the considered codes are described, and the algorithms of their construction are presented. The software tool allows analyzing any and all of the three methods of coding, and selecting the simplest one based on the results.

### 4.1 Checker for the Berger code

The checker consists of a bit counter (which counts the number of '1's or '0's among the information bits) and a dual-rail checker [6]. These are generic modules, which do not depend on the checked vectors, but only on the number of bits *n*.

### 4.2 Checker for the Smith code

The set of output words is *randomly* partitioned into two disjoint subsets $V_1$ and $V_2$. One of the checker outputs, $r_1$, is '1' for inputs from $V_1$ and '0' for inputs from $V_2$, the other, $r_2$, vice versa. The checker is then synthesized by Altera's Max+Plus II CAD tool.

### 4.3 Checker for the CO m-out-of-n code

The checker has two layers. The first one consists of dual-rail-output checkers for each of the 1-out-of-*k* codes, which correspond to the subsets $V_i \cup c_i, i = 1,\ldots,m$. The partition of bits into subsets is done using the Monte Carlo iterations on a random greedy partition algorithm. The second layer is a dual-rail checker for the *m* pairs resulting from the first. Figure 2 shows the checker for benchmark dk15.

The first layer is constructed using symmetric functions. These functions are a convenient form for presentation of a logical function. The notation for symmetric functions is $F_{\langle cond\rangle}(X)$, where the subscript condition defines how many of the bits in X should be equal '1'. For example, $F_{=1}(X)$ is the "1-hot" function. The outputs of a dual-output STC for the 1-out-of-*k* code are $\begin{cases} r_1 = F_{\geq 1}(X_1) \vee F_{\geq 2}(X_2) \\ r_2 = F_{\geq 2}(X_1) \vee F_{\geq 1}(X_2) \end{cases}$, where $X_1 \cup X_2 = X, X_1 \cap X_2 = \varnothing$.

The second layer is a dual-rail *m*-pair checker, for which *m-1* 4-input 2-output CLBs are sufficient.

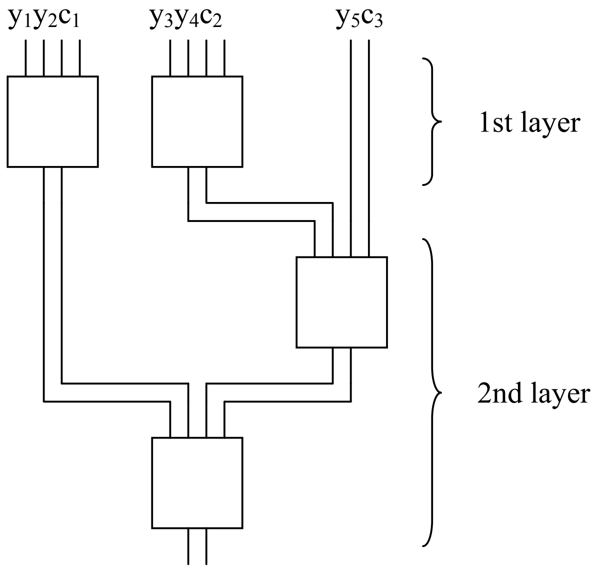

Figure 2. A checker for the separable m-out-of-n code for benchmark dk15.

## 5. Results

The software tool was used to encode output words and to construct the respective checkers for specific MCNC benchmarks. As has been suspected, the random selection mechanism allows finding a solution that is close to optimal, after only several attempts. The results are shown in Table 2. The CLB counts were produced using the procedure described in sections 3 and 4. The first three columns list the name of the benchmark, the number of information bits in the output word (IB), and the legal words number (WN). The rest of the columns show the number of code bits, the number of CLBs required for the checker, and the improvement in hardware overhead (in percents) over using a single code, for each of the codes: Berger (B), Smith (S) and the context-oriented *m-out-of-n* code (M).

Table 2. Test Results

| benchmark | IB | WN | code bits | | | checker CLBs | | | improvement over | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | B | S | M | B | S | M | B | S | M |
| bbara_bbtas | 2 | 3 | 2 | 1 | 1 | 4 | 2 | 2 | 50% | - | - |
| bbsse | 7 | 11 | 3 | 2 | 2 | 12 | 8 | 6 | 50% | 25% | - |
| bbtas | 2 | 4 | 2 | 2 | 2 | 4 | 2 | 2 | 50% | - | - |
| cse | 7 | 11 | 3 | 2 | 3 | 12 | 12 | 8 | 33% | 33% | - |
| dk14 | 5 | 12 | 3 | 2 | 3 | 10 | 10 | 8 | 20% | 20% | - |
| dk15 | 5 | 11 | 3 | 2 | 3 | 10 | 9 | 8 | 20% | 11% | - |
| dk16 | 3 | 5 | 2 | 2 | 2 | 4 | 4 | 4 | - | - | - |
| dk17 | 3 | 5 | 2 | 2 | 2 | 4 | 4 | 4 | - | - | - |
| dk512 | 3 | 4 | 2 | 1 | 1 | 4 | 2 | 2 | 50% | - | - |
| ex4 | 3 | 5 | 2 | 2 | 3 | 4 | 4 | 4 | - | - | - |
| ex6 | 8 | 12 | 3 | 2 | 6 | 14 | 18 | 14 | - | 22% | - |
| keyb | 2 | 3 | 2 | 1 | 1 | 4 | 2 | 2 | 50% | - | - |
| s1 | 6 | 20 | 3 | 2 | 3 | 12 | 14 | 6 | 50% | 57% | - |
| s208 | 2 | 4 | 2 | 2 | 2 | 4 | 2 | 2 | 50% | - | - |
| s298 | 6 | 5 | 3 | 1 | 2 | 12 | 5 | 6 | 58% | - | 17% |
| s386 | 7 | 11 | 3 | 2 | 3 | 12 | 10 | 8 | 33% | 20% | - |
| s420 | 2 | 4 | 2 | 2 | 2 | 4 | 2 | 2 | 50% | - | - |
| s510 | 7 | 13 | 3 | 3 | 4 | 12 | 16 | 12 | - | 25% | - |
| sse | 7 | 11 | 3 | 2 | 2 | 12 | 10 | 6 | 50% | 40% | - |
| tav | 4 | 12 | 3 | 2 | 3 | 7 | 9 | 6 | 14% | 33% | - |
| tbk | 3 | 5 | 2 | 2 | 2 | 4 | 4 | 4 | - | - | - |

It is easy to observe that, in most cases, the minimal overhead is achieved with the context-oriented m-out-of-n code. However, in some cases additional constraints may be put on the solution, such as the number of check bits or the code universality. The developed software tool allows selecting a more suitable solution for each of these cases.

## 6. Summary

We proposed the guiding principles of a software tool, which automates the design of checker circuits aimed at the detection of unidirectional errors at the output of a device to be checked. The tool allows encoding the output words using the traditional Berger or Smith codes, or our original context-oriented m-out-of-n code. For each of these codes, a checker is synthesized and the hardware overhead is analyzed. As the test results show, it is possible to reduce the checker hardware by as much as 57%.

## R e f e r e n c e s

[1] J. Berger. A note on error detection codes for asymmetric channels. *Inform. Control*, 4:68–73, March 1961.

[2] J. E. Smith. On separable unordered codes. *IEEE Transactions on Computers*, C-33:741–743, August 1984.

[3] V. Ostrovsky, I. Levin, S. Ostanin. "Synthesis of Self-checking Controllers Based on Modified (m, n)-code", *Automatic Control and Computer Science, 2, pp. 48-55, 2003*.

[4] P. K. Lala. *Self-Checking and Fault-Tolerant Digital Design*. Academic Press, San Diego, CA, USA, 2001.

[5] A. L. Burrass and P.K. Lala. On-line Testable Logic Design for FPGA Implementation. *Proc. of 1997 Int. Test Conference, pp. 471-478*.

[6] S. J. Piestrak. *Design of self-testing checkers for unidirectional error detection codes*. Oficyna Wydawnicza Politechniki Wroclawskiej, Wroclaw, 1995.