Duplication Based One-to-many Coding for Trojan HW Detection

Osnat Keren
School of Engineering
Bar-Ilan University
Ramat-Gan, Israel
kereno@eng.biu.ac.il

Ilya Levin
School of Education
Tel-Aviv University
Tel-Aviv, Israel
ilia1@post.tau.ac.il

Mark Karpovsky
Dep. of Electrical and Computer Engineering
Boston University
Boston, Massachusetts
markkar@bu.edu

Abstract

A functional unit having multiple errors on its output may be considered as a computational channel, in which there is no restriction on the direction and the number of bit-flips. This model characterizes the distortion of the output caused by an active Trojan hardware. In this paper, we present a Concurrent Error Detection (CED) scheme that detects an active Trojan with high probability within a given time. The suggested CED scheme consists of a duplication-based *one-to-many* code that protects the most probable, valuable, or vulnerable words that may appear on the output of the system.

I. Introduction

Economic motivations led the Integrated Circuit (IC) market to outsource the fabrication process. This trend makes the chips vulnerable to attacks and rises the concern that a "Trojan" hardware will be inserted to the chip [9]. A Trojan hardware is a malicious modification of the circuitry of an integrated circuit. A Trojan hardware is created by an adversary that adds or deletes any transistors or gates to the original design and thus changes its functionality. This motivates researchers to classify, model, and analyze Trojans and their effect on the behavior of the system, as well as to develop methods to detect the Trojans [2], [6], [7], [11], [18], [23], [26]. Trojans can be classified into three major types: a) Trojans which broadcast to the attacker some internal signals, b) Trojans that distort the functionality of the system, and c) Trojans which destroy the chip. In this paper we focus on designing system which can detect Trojans of the second type. When such a Trojan becomes active, the output of the system is erroneous. In this sense, an active Trojan can be modeled as an arbitrary error vector that is added on top the correct output vector and distorts it. Since the system functions correctly when the Trojan is not activated, it is difficult to detect a Trojan by off-line testing. Therefore, it is desired to design a Concurrent Error Detection (CED) mechanism, that can cope with faults (permanent or transient) in the circuitry, and at the same time, to detect Trojans.

In general, CED codes are designed to cope with all the error cases defined by a specific fault model [4], [5], [16], to detect a *subset* of protect faults, or protect the system from a *subset* of errors. The subset consists of faults which are most likely to cause damage [25], or faults whose detection requires a reasonable hardware (HW) overhead [3], [8], [14]. All these methods require primary information about the circuit's features, or they may require preprocessing for gathering information about sensitive areas, error-cones, or paths in a specific circuit [10], [12], [20], [22], [24], [25]. This circuitry-depended approach to code design provides fault secureness with relatively small HW overhead. Nevertheless, its performance become questionable in the presence of the Trojan HW, since in this case, the erroneous output does not results from faults in the circuit.

A different approach, which is likely to provide better protection at the presence of a 'lightweight' Trojan HW, considers a circuit as a *black-box* that, due to some reason, may produce arbitrary values on its output. This black-box approach requires the functional unit to be implemented as two (or more) independent sub-circuits. Otherwise, there is no code that can detect all the error cases [17]. Codes for a context-orientated systems that were designed without analyzing the actual implementation are presented in [17], [13].

The majority of CED codes may be considered as *one-to-one* codes that map between a k-bit vector at the output of the functional unit and an n-bit codeword. A different approach to concurrent error detection, called *one-to-many* coding was presented in [13]. In *one-to-many* coding, each codeword comprises a predefined set of words. The functional unit is referred to as an encoder. Each activation the encoder can map an information word (i.e. original output vector) to a different word in the corresponding set. This flexible mapping between an information word and a word in the set of words, allows to decrease the HW overhead in the functional unit and its checker.

In this paper, we present a modification of the *one-to-many code* to the case of the universal set of possible output words. We show that under some assumptions on the behavior of the Trojan HW, the suggested approach can detect also the Trojan HW with relatively low cost.

The paper is organized as follows: Section II presents the Trojan characteristics and the system's design requirements. Section III presents the concept of *one-to-many* coding. Section IV describes the structure of the suggested CED system. Experimental results are provided in Section V. Section VI concludes the paper.

II. Trojan model and design requirements

In this paper, we consider Trojan HW that distorts the functionality of the system. We model the effect of the Trojan HW as a certain erroneous behavior of the system. This erroneous behavior is reflected on the system's output as an *arbitrary-error* that distorts the correct values of the output signals. The arbitrary-error model refers to a computational channel, in which there is no restriction on the direction and the number of bit-flips in the output, and no information is available on the error-cone. Nevertheless, we can make some assumptions concerning the Trojan design and behavior.

- The Trojan HW designer is not familiar with the functionality of the circuit. That is, the Trojan HW does not generate (for at least a period of time) a legal sequence of outputs vectors. Therefore, its effect can be modeled as an arbitrary error. This assumption is essential - since if the Trojan generates only legal outputs it cannot be detected by a CED code.
- If the circuit contains two (or more) identical blocks and one block is contaminated by a Trojan HW, then, it is most probable that another block will have the same Trojan. This assumption implies that an error detection mechanism based on straightforward duplication cannot detect a Trojan HW.
- When a functional unit is implemented as a set of different sub-circuits, at most one sub-circuit contains an active Trojan. This assumption is used in next section to justify the use of CED codes with minimum distance that equals two.
- The Trojan HW is 'lazy'. That is, when the Trojan HW is activated its effects the outputs for a period of time, and then, it may disappear. We are not dealing with a Trojan that is activated for a single cycle (and becomes invisible for a long period of time). This assumption implies that the efficiency of the Trojan detection technique can be measured in terms of the probability of the undetected error over a certain time interval.

Additionally, we assume that there is no mandatory requirement that the Trojan will be detected immediately - at the first cycle it was activated. Base on this, we study methods for designing a low-cost CED mechanism that is capable of detecting a Trojan with high probability over a predefined time interval. This goal can be achieved by protecting just a sub-set of the possible legal output words.

We also assume that the probability distribution of the words at the functional unit's output is *not uniform* . That is, there are words that appear more frequently than others. This assumption was tested and verified by the authors of [25] for control logic modules in an industrial processor using sample segments of real applications. Note that sequential circuits implementing controllers, tend to have this property [1].

III. One-to-Many coding

Consider a functional unit that has m inputs and k outputs. The functional unit can be represented as a multi-output function $Y = f(X)$ where $X = (x_{m-1}, \ldots, x_0)$ and $Y = (y_{k-1}, \ldots, y_0)$. In this paper, we call the binary output vectors *information words*. Assume that the logic unit can produce M distinct information words out of the 2^k possible combinations, that is, $M \leq 2^k$.

Most of the CED codes are based on *one-to-one* codes. That is, codes that map between a set of information words $\{Y_i\}_{i=1}^{M}$ and a set of binary codewords $\{Z_i\}_{i=1}^{M}$. The set of codewords $\{Z_i\}_{i=1}^{M}$ forms a *one-to-one* code; Each information word $Y_i = (y_{k-1}^{(i)}, \ldots, y_0^{(i)})$, is encoded (mapped) to a *single* codeword $Z_i = (z_{n-1}^{(i)}, \ldots, z_0^{(i)})$, by adding $n-k$ redundant bits to the k information bits. Consequently, the redundancy bits are treated as completely specified switching functions.

Assume that the functional unit is implemented by $S = 2$ independent circuits. Without loss of generality, let the first circuit realize the first n_1 bits of Z, that is, $c_1 = (z_{n_1-1}, \ldots, z_0)$, and the second circuit realize the next n_2 bits, where, $n_1 + n_2 = n$. The output of the j'th circuit is a binary vector c_j of length n_j. This vector can be referred to as a symbol from an alphabet of size 2^{n_j}. Usually the finite field $GF(2^{n_j})$ is used, that is $c_j \in GF(2^{n_j})$. The output of the overall functional unit, can be referred to as either a binary codeword Z of length n, or as a codeword $C = (c_2, c_1)$ over a mixed alphabet. In this paper we use the latter notation.

Another approach to encoding, called *one-to-many*, was presented in [13]. In the *one-to-many* coding, an information word is mapped to a *set of words* and not necessarily to a single codeword. The mapping between an information word and one of the words in the corresponding set depends on the input vector X. The mapping is determined in order to reduce the implementation cost of the functional unit as well as the implementation cost of the checker. The *one-to-many* code is defined as follows:

Table I
INFORMATION WORDS IN EX. 1

x_2	x_1	x_0	Y	y_4	y_3	y_2	y_1	y_0	Y
0	0	0	Y_1	1	0	0	1	1	(4,3)
0	0	1	Y_2	0	0	0	0	0	(0,0)
0	1	0	Y_3	0	1	0	0	1	(2,1)
0	1	1	Y_2	0	0	0	0	0	(0,0)
1	0	0	Y_4	1	1	0	0	1	(6,1)
1	0	1	Y_2	0	0	0	0	0	(0,0)
1	1	0	Y_2	0	0	0	0	0	(0,0)
1	1	1	Y_5	0	0	1	0	1	(1,1)

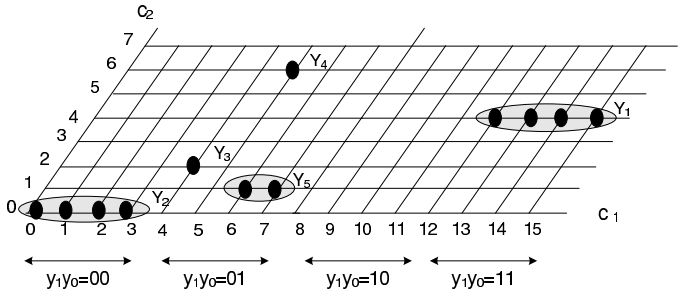

Figure 1. Two-dimensional representation of the *one-to-many* code in Ex. 1.

Definition 1 (One-to-many systematic code for $S = 2$): *One-to-many* encoding maps each information word $Y_i, i = 1, \ldots M$, to a product (cube) Z_i. In each product, there are k literals (which correspond to the k information bits $\{y_j^{(i)}\}_{j=0}^{k-1}$) and at most $n - k$ literals (which correspond to the redundancy bits). The product is represented as a codeword $C_i = (c_2^{(i)}, c_1^{(i)})$, where, $c_s^{(i)} \subseteq GF(2^{n_s})$ and $n_1 + n_2 = n \geq k$.

Clearly, any *one-to-one* code is a *one-to-many* code, but not vice-versa.

Definition 2 (Minimum distance of a one-to-many code): The distance between two codewords, C_i and C_j, in a $\mathcal{C}(n, M, d_{min})$ code is

$$d(C_i, C_j) = |\{s | c_s^{(i)} \cap c_s^{(j)} = \Phi, s = 1, 2\}|,$$

and the minimum distance of the code is,

$$d_{min} = \min_{C_i, C_j \in \mathcal{C}, i \neq j} d(C_i, C_j).$$

A functional unit, implemented as two independent circuits, is secure in terms of its k information bits from multiple errors (i.e. an arbitrary error vector) that distort the outputs of a single sub-circuit, iff, the minimum distance of the code $\mathcal{C}(S = 2, M, d_{min})$ equals two. In other words, any erroneous vector that appears on the output lines of a single sub-circuit due to its incorrect functioning - is detectable.

Example 1: : Consider the 3-inputs 5-output function that is specified by Table I. The function has $M = 5$ words $\{Y_i\}_{i=1}^{5}$. Assume that the function is realized as two independent circuits where the first circuit realizes the information bits (y_1, y_0), and the second circuit realizes (y_4, y_3, y_2). To simplify the writing we address the binary vectors by their integer value. For example, the information word Y_1 is represented as

$$Y_1 = (100, 11) = (4, 3).$$

The information words can be referred to as words of length two over a mixed alphabet ($GF(2^2)$ and $GF(2^3)$). The minimum distance between the information words is one,

$$1 \leq d_{min} \leq d(Y_3, Y_4) = 1.$$

Therefore, there may be a single fault in the second circuit that cannot be detected.

Now let us encode the five information words by a *one-to-many* code. First, notice that the distance between the information word Y_1 and all the other information words equals two. Additional redundancy bits cannot change this distance.

Consequently, it is possible to encode Y_1 not to a single word (as is done in *one-to-one* encoding) but to a set of binary words that forms a *Boolean cube*. In particular, let the information word Y_1 be mapped to to the set

$$C_1 = \{(4,12),(4,13),(4,14),(4,15)\},$$

that is $c_2^{(1)} = \{4\}$ and $c_1^{(1)} = \{12,13,14,15\}$. The codeword C_1 can be represented as a product

$$(z_6 z_5' z_4') \cdot (z_3 z_2) = y_4 y_3' y_2' y_1 y_0.$$

Fig. 1 shows the codewords as points on an $2^3 \times 2^4$ matrix. The x-axis corresponds to c_1 and the y-axis corresponds to c_2. A fault in the circuit that realizes c_1 may shift a point along the x-axis, while a fault in c_2 may shift a point along the y-axis. The four words that comprise C_1 are represented as a set of four adjacent points on the $c_1 \times c_2$ plane; the other codewords C_2, C_3, C_4 and C_5 are shown in the figure as well. Notice that each codeword consists of a different number of elements. Additionally, an erroneous values on the output of a single circuit may shift a point within its set, or shift it outside the set. It cannot shift points between sets. Therefore, the code shown in Fig. 1 can detect any arbitrary error vector that distorts the output of a single circuit.

The implementation cost of this *one-to-many* code is smaller than the implementation cost of the *one-to-one* code since the functional unit is not completely specified. Namely, there are several ways to map an input X to the actual word that will appear on the output lines.

IV. The proposed CED architecture

Let us assume that the probability distribution of the output vectors is known. Namely, each output vector is associated with a *cost*. The cost may indicate reliability or importance of a word, or it may be referred to as an a-priory information on the occurrence ratio of a word. For example, a word instructing to invalidate the content of the cache is more important than a word that indicates a miss, and hence, this word may be associated with a higher cost.

Let an information word Y_i $(i = 1,\ldots,M)$ be associated with a cost $p(i)$, such that $\sum_i p(i) = 1$. Without loss of generality, let

$$0 \le p(M) \le p(M-1) \le \cdots p(1) \le 1.$$

Divide the set of information words into two sets, a set $\mathcal{Y}_1 = \{Y_i\}_{i=1}^{M_s}$ of the words that will be protected by a *one-to-many* code, and a set $\mathcal{Y}_2 = \{Y_i\}_{i=M_s+1}^{M}$ of the remaining words. As for now, we assume that the number of protected words, M_s, is given.

Let the set $\mathcal{Y}_1$ of M_s information words be encoded by a *one-to-many* $\mathcal{C}_1$ code. The code defines the partition of the k information bits into two sets of size k_1 and k_2, ($k_1 \le n_1, k_2 \le n_2$). This partition determines how to implement the system as two sub-circuits. The structure of the system is shown in Fig. 2. The functional unit comprises two sub-circuits. Each sub-circuit i $(i = 1,2)$ produces: k_i information bits, $n_i - k_i$ redundancy bits and a valid bit v. The valid bit v indicates whether the output is a codeword of $\mathcal{C}_1$. It is set to "1" if the output is a codeword of $\mathcal{C}_1$, otherwise, it equals "0".

During an error free operation, the functional unit produces either an information word from $\mathcal{Y}_1$, or an information word from $\mathcal{Y}_2$. In the first case, a codeword from $\mathcal{C}_1$ will appear on the output of the system and the valid bit in both sub-circuits will be set to "1". If the information word is from $\mathcal{Y}_2$, then the information bits will carry a word from $\mathcal{Y}_2$, both valid bits will be zero, and redundancy bits will carry arbitrary values. The arbitrary values have to be determined to reduce the implementation cost of the overall system.

The checker receives two valid bits and two symbols of the codeword. The checker is activated when the two valid bits are set to one. When the checker is activated, it checks whether a codeword of $\mathcal{C}_1$ appears on its input lines. The checker notifies a problem if

- only one valid bits (out of the two bits) is set, or,
- if both valid bits are set but the received word is not a codeword.

Denote by $P_{\mathcal{Y}_1}$ the probability that a word is protected, that is $P_{\mathcal{Y}_1} = \sum_{i=1}^{M_s} p(i)$. The probability $P_{ud}(N)$ that a Trojan will not be detected by the checker within N cycles from the time it was activated is upper bounded by $(1 - P_{\mathcal{Y}_1})^N$. The number of protected words M_s is chosen so to provide the desired undetected error probability within a window of size N.

Now we turn to the construction of the code $\mathcal{C}_1$. The key idea is to design a CED scheme whose implementation cost is inherently not larger than the implementation cost of conventional duplication.

The principle of the duplication-based *one-to-many* code is the following. Without loss of generality assume that $\{y_1,\ldots,y_j\}$ are information bits generated by the first circuit, and that $\{y_{j+1},\ldots y_k\}$ are the information bits generated by of the second circuit. Methods for dividing the information bits between the two circuits, so to reduce the implementation

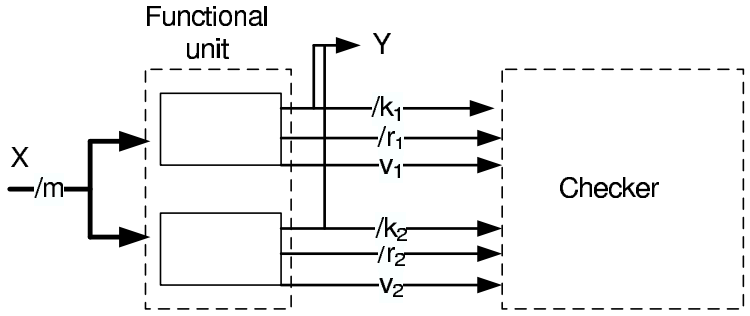

Figure 2. On-line checking scheme for arbitrary errors

Table II
DUPLICATION BASED *one-to-one* AND *one-to-many* CODES FOR EX. 2

Y	$y_4y_3y_2z_1z_0$	$z_4z_3z_2y_1y_0$	C	$y_4y_3y_2z_1z_0$	$z_4z_3z_2y_1y_0$	C
Y_1	10011	10011	(19,19)	$100--$	$---11$	(4,3),(4,7)(4,11),(4,15)
Y_2	00000	00000	(0,0)	$000--$	$---00$	(0,0),(0,4)(0,8),(0,12)
Y_3	01001	01001	(9,9)	$010--$	$00-01$	(2,0)
Y_4	11001	11001	(25,25)	$110--$	$10-01$	(6,9)
Y_5	00101	00101	(5,5)	$001--$	$-1-01$	(1,5),(1,13)

cost, were discussed in [17], [13]. Denote by $\{z_{j+1}, \ldots z_k\}$ the redundancy bits appended to the first set of information bits, and by $\{z_1, \ldots, z_j\}$ the bits appended to the second set. Let $z_i = y_i$. Clearly this setting is equivalent to duplication.

Let $u_j(z_i)$ stands for the value of the redundancy bit z_i in the codeword Y_j. In duplication $u_j(z_i)$ may take the values 0 or 1. However, the value of the redundancy bit $u_j(z_i)$ may be considered as *don't care* if z_i is not required for increasing the distance between the codeword Y_j and the other codewords. Assigning a don't care value to $v_j(z_i)$'s makes it one to many code.

The following example demonstrates this idea.

Example 2: : Consider the five information words of Ex. 1. The left hand side of Table II shows the conventional duplication solution (which is equivalent to *one-to-one* code), the right hand side of the table shows the duplication based *one-to-many* code. The variables z_2, z_1 and z_0 carry only *don't cares* and thus can be removed. Namely, only two redundancy bits (z_3 and z_4) are required to separate between the codewords. Notice that the variables z_4 and z_3 define a *one-to-many* code. The code is shown in Fig. 1.

Clearly, the order of deciding upon the values of $\{v_j(z_i)\}_{j=1,i=0}^{M_s,n-1}$ determines the actual number of redundancy bits, the codewords, and consequently, the implementation cost. Optimal encoding procedure, is not in the scope of this paper. Here we use a greedy procedure that goes from $i = 0$ to $n - 1$. For each i it considers all codewords and determines the value of $\{v_j(z_i)\}_{j=1}^{M_s}$.

V. EXPERIMENTAL RESULTS

In this section, we present results obtained from experiments with a number of ISCAS89 benchmarks. We used the combinatorial part of these sequential circuits which inherently have the context orientation property. The set of codewords was obtained by using the Espresso minimization package with the flag "-Ddisjoint".

In general, the cost associated with a codeword reflects the reliability or importance of a word, or the a-priory information on the occurrence ratio of a word. In this section, we defined the cost of a word as its occurrence ratio measured over 10^6 cycles from boot.

The results of the experiments are presented in Table III. The table shows the implementation cost of the proposed method in respect to the cost of the original functional unit. The implementation cost is measured in terms of the number of Look-Up-Tables (LUTs) required to implement the function by *SPARTAN3 xcs200ft256* as computed by LeonardoSpectrum.

The first two columns of Table III contain the name of the benchmark function, and the number of inputs m. The remaining columns are divided to four sets groups. Each set corresponds to a different $\hat{P}_{\mathcal{Y}_1}$, where $\hat{P}_{\mathcal{Y}_1}$ is (a lower bound on) the probability that the valid bits are set to one. That is, $\hat{P}_{\mathcal{Y}_1} \leq P_{\mathcal{Y}_1}$. The first set corresponds to the original system ($\hat{P}_{\mathcal{Y}_1} = 0$), in this case the functional unit is implemented as a single circuit and no redundancy bits are added and no valid bits are implemented, that is, $n - k$. The next sets correspond to $\hat{P}_{\mathcal{Y}_1} = 0.5, 0.75$ and 1.0. For each set we show:

- The number of protected information words M_s. The value of M_s is determined as the minimal number of information words that satisfy $\sum_{i=1}^{M_s} p(i) \geq \hat{P}_{\mathcal{Y}_1}$. Note, that the costs $p(i)$ were obtained by simulation, therefore, in several cases,

Table III
BENCHMARK RESULTS

		Orig.			$\hat{P}_{\mathcal{Y}_1}=0.5$			$\hat{P}_{\mathcal{Y}_1}=0.75$			$\hat{P}_{\mathcal{Y}_1}=1$		
	m	M	k	LUT	M_s	(n_1,n_2)	LUT	M_s	(n_1,n_2)	LUT	M_s	(n_1,n_2)	LUT
bbara	8	12	6	24	3	(6,2)	46	4	(6,3)	47	12	(6,6)	48
bbsse	11	19	9	48	3	(9,2)	68	5	(9,4)	72	19	(7,9)	77
dk14	6	26	8	27	10	(6,6)	44	17	(6,7)	46	26	(7,7)	49
dk16	7	75	8	56	29	(8,7)	122	48	(8,8)	130	75	(8,8)	113
ex6	8	13	11	55	2	(6,5)	64	5	(8,3)	62	13	(7,7)	57
planet	13	91	22	209	19	(18,11)	258	40	(11,20)	278	91	(12,20)	292
sand	16	73	11	158	12	(9,7)	202	25	(9,9)	216	73	(10,11)	275
sse	11	19	9	47	3	(7,3)	64	5	(7,5)	64	15	(7,9)	72
s298	17	332	20	155	2	(20,1)	190	6	(19,4)	212	204	(15,16)	718
s386	13	23	13	60	3	(13,2)	85	5	(13,4)	86	20	(11,12)	102
s510	25	73	13	67	27	(9,9)	102	47	(9,12)	138	73	(9,13)	129
s820	23	70	24	129	3	(24,2)	138	6	(22,5)	150	31	(23,15)	177
s832	23	70	24	132	3	(24,2)	140	6	(22,5)	149	30	(23,15)	181
s1488	14	168	25	209	3	(24,2)	215	5	(17,10)	211	100		307
s1494	14	168	25	209	3	(23,3)	216	5	(16,10)	209	98	(16,22)	281
Avrg.			1	1		1.24	1.37		1.36	1.45		1.55	1.80

some information words were associated with zero cost. In those cases, the number of protected information words M_s is smaller than M for $\hat{P}_{\mathcal{Y}_1}=1$. See for example the benchmark function '*s332*'.

- The length n of a codeword, $n=n_1+n_2$, is represented as (n_1,n_2).
- The implementation cost (in LUTs) of the overall system, that is the two sub-circuits that implement the information bits, the redundancy bits and the two valid bits. Note that when M_s equals M there is no need in valid bits. See for example the benchmark function '*s510*'.

The last row in the table shows the average ratio n/k, and the average normalized implementation cost. The experimental results show that, in average, the ratio between the length k (in bits) of the information word and the length n of the coded word $1/1.24, 1/1.36$ and $1/1.55$, for $\hat{P}_{\mathcal{Y}_1}=0.5, 0.75$ and 1.0. This ratio is significantly smaller than a straightforward duplication $(1/2)$.

The implementation cost of the duplication based *one-to-many* coding is (in average) larger by $37\%, 45\%$, and 80% in comparison to the implementation cost of the original functional unit, for $\hat{P}_{\mathcal{Y}_1}=0.5, 0.75$ and 1.0. Note that the implementation cost streigthforward duplication is twice the implementation cost of the original functional unit.

The undetected error probability $P_{ud}(N)$ decreases as the window size increases. Therefore, by using a window of length two and $\hat{P}_{\mathcal{Y}_1}=0.75$ it is possible to increase the probability of detection of an active Trojan HW to be larger than 93.75%. This can be done by increasing (on average) the implementation cost by 45% and the number of outputs by 36%. Alternativly, the same detectction probability can be achived by choosing a time window of length four and using $\hat{P}_{\mathcal{Y}_1}=0.5$. The latter solution increases 9on average) the implementation cost by 37% and the number of outputs by 24%.

VI. CONCLUSIONS

Concurrent error detection plays an important role in modern logic design. It is usually used to increase the immunity of the system against faults and transition errors. In this paper, we introduced a CED scheme that detects a Trojan HW whose behavior can be modeled by a computational channel with arbitrary errors. The proposed scheme is based on a *one-to-many* code over a mixed alphabet. It provides a low undetected error probability within a given time window by protecting a subset of all the possible output words. The set of protected words can be chosen according to various criteria. The authors believe that the presented study opens the way to future research in using CED scheme for simultaneous detection of errors caused by faults and Trojans.

REFERENCES

[1] B. Abramov, O. Keren, I. Levin and V. Ostrovsky, "Constructing Self-testing Circuits with the Use of Step-by-step (Cascade) Control," *Automation and Remote Control*, vol. 70, no. 7, pp. 1217-1227, 2009.

[2] A. Adamov, A. Saprykin, D. Melnik, O. Lukashenko,"The problem of Hardware Trojans detection in System-on-Chip," *The 10th International Conference on CAD Systems in Microelectronics - CADSM 2009*, pp. 178 - 179, 2009.

[3] S. Almukhaizim, P. Drineas and Y. Makris, "Entropy-driven parity-tree selection for low-overhead concurrent error detection in finite state machines," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 8, pp. 1547- 1554, 2006.

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

[5] B. Bose and D. J. Lin., " Systematic Unidirectional Error-Detecting Codes," *IEEE Transaction on Computers*, vol. 34, pp. 1026-1032, Nov. 1985.

[6] R.S. Chakraborty, S. Bhunia, "Security against hardware Trojan through a novel application of design obfuscation," *IEEE/ACM International Conference on Computer-Aided Design -ICCAD 2009* , pp. 113-116,2009.

[7] R.S. Chakraborty, S. Narasimhan, S. Bhunia, "Hardware Trojan: Threats and emerging solutions," *IEEE Inter. High Level Design Validation and Test Workshop -HLDVT 2009* , pp. 166-171, 2009.

[8] M. R. Choudhury and K. Mohanram, "Approximate logic circuits for low overhead, non-intrusive concurrent error detection," *Proceedings of DATE 2008*,pp. 903-908, 2008.

[9] DARPA BAA06-40, Trust for Integrated Circuits,
http://www.darpa.mil/mto/solicitations/baa06-40/index.html

[10] N.K. Jha and S.J. Wang, "Design and Synthesis of Self-Checking VLSI Circuits," *IEEE Transaction CAD*, vol. 12, no. 6, pp. 878-887, Jun. 1993.

[11] Y. Jin, N. Kupp, Y. Makris, "Experiences in Hardware Trojan design and implementation," *IEEE International Workshop on Hardware-Oriented Security and Trust -HOST '09*, pp. 50 - 57, 2009.

[12] 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, no. 2, pp. 186-195, June 1994.

[13] O. Keren, "One-to-many: Context-Oriented Code for Concurrent Error Detection," *JETTA-Journal of Electronic Testing. Theory and Applications*, Springer ISSN. 1573-0727 (Online), Dec. 2009.

[14] P. Kubalik, P. Fiser and H. Kubatova, "Fault tolerant system design method based on self-checking circuits," *Proceedings of the 12th IEEE International On-Line Testing Symposium*, pp. 185 -186, 2006.

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

[16] F.J.MacWilliams and N.J.A.Sloane, *The Theory of Error-Correcting Codes*, North-Holland, 1977.

[17] V. Ostrovsky, I. Levin, O. Keren, B. Abramov, "Designing Concurrent Checking Circuits by using Partitioning," *International Journal of Highly Reliable Electronic System Design*, vol. 1 no.1, pp. 1-11 , 2007.

[18] M. Potkonjak, A. Nahapetian, M. Nelson, T. Massey, "Hardware Trojan horse detection using gate-level characterization," *The 46th ACM/IEEE Design Automation Conference - DAC '09* , pp. 688-693, 2009.

[19] V.V. Saposhnikov, A. Morosov, Vl.V. Saposhnikov, M. Gossel, "Design of Self-Checking Unidirectional Combinational Circuits with Low Area Overhead," *Proceeding of the 2nd IEEE International On-line Testing Workshop*, pp. 56-67, Jul. 1996.

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

[21] J. E. Smith , "On separable unordered codes," *IEEE Transaction on Computers*, vol. 33, no. 8, pp. 741-743, Aug. 1984.

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

[23] M. Tehranipoor, F. Koushanfar, "A Survey of Hardware Trojan Taxonomy and Detection," *IEEE Design and Test of Computers*, vol. 27, no. 1, pp. 10-25, 2010.

[24] N.A. Touba, and E.J. McCluskey, "Logic Synthesis of Multilevel Circuits with Concurrent Error Detection," *IEEE Trans. Computer-Aided Design of Integrated Circuits and System*, vol. 16, pp. 783-789, 1997.

[25] R. Vemu, A. Jas, J.A. Abraham, S. Patil, R. Galivanche, "A low-cost concurrent error detection technique for processor control logic," *Proc. of Design, Automation and Test in Europe (DATE)*, pp. 897-902, 2008.

[26] W. Xiaoxiao, H. Salmani, M. Tehranipoor, J. Plusquellic, "Hardware Trojan Detection and Isolation Using Current Integration and Localized Current Analysis," *IEEE Inter. Symposium on Defect and Fault Tolerance of VLSI Systems, DFTVS '08*, pp. 87 - 95, 2008.