VLSI DESIGN
1998, Vol. 5, No. 4, pp. 347–356
Reprints available directly from the publisher
Photocopying permitted by license only

Embedded Parity and Two-Rail TSC Checkers with Error-Memorizing Capability

STEFFEN TARNICK *

SATCON GmbH, Satellitenkommunikationsgesellschaft, Potsdamer Str. 7-9, 14513 Teltow, Germany

In self-checking systems, checkers usually do not receive all code words during normal operation. Missing code words may prevent a checker from achieving the totally self-checking property. The paper presents a novel approach to the design of embedded parity and two-rail checkers that allows a checker to receive all code words irrespective of the set of code words that is provided by the functional circuit. A checker gets all code words by an LFSR while at the same time it monitors the output of the functional circuit. Additionally, the LFSR is able to capture the error patterns of noncode words. Captured error patterns will be modified since they will cycle through the LFSR. Thus, noncode words that are not detected due to undetected checker faults can be detected in later instances of time. The proposed method can be extended to the design of checkers for linear codes.

Keywords: Embedded TSC checkers, Error-Memorizing checkers, Cyclic codes, Parity checkers, Two-Rail checkers, Hardware test pattern generation, Linear feedback shift registers, Complete feedback shift registers, Controllable checkers

1 INTRODUCTION

One of the problems when designing self-testing, totally self-checking, strongly code-disjoint and strongly self-checking checkers[1–4] is that the checkers will receive only a subcode of the code they are designed for, so that they can loose their self-testing, totally self-checking, strongly code-disjoint or strongly self-checking property.

The code words can be regarded as test vectors for the checker. If some of the code words are not provided by the circuit under check (CUC) then it is possible that some checker faults will not be detected. If a checker contains an undetected nonredundant fault then some noncode words will not be detected, i.e., the checker will not remain code-disjoint anymore. This problem will be even more severe if the checking process is performed for a subset of CUC output code words only[5].

Several methods have been discussed in the literature to solve or avoid the problem of missing code words. The first method consists of arranging the components of the checker according to the available code words[6,7] or the dependencies of

* Tel.: (+49) 3328 303 897. Fax: (+49) 3328 427 356. E-mail: tarnick@satcon.de.

the input signals[8]. Another possibility to obtain the required set of code words for all subcheckers of a global checker is to add one or several pairs of control inputs which are independent from all other inputs of the checker[8,9]. A suggestion for generating additional code words for two-rail checkers is the insertion of delay elements at selected input pairs of the checker[10]. However, with this approach a delay of the error indication will be introduced. Another possibility is to provide the checker with internal off-line test capability[11,12]. Here, the checker contains an internal test pattern generator which can provide the missing code words and additional noncode words. The disadvantage of this approach is that the checker has to be placed into a test mode, and cannot monitor the function of the CUC during the test. However, an off-line test of checkers also allows to detect multiple stuck-at and bridging faults in the checker[13,14] or to locate checker faults[15].

In this paper we present a novel approach to the design of embedded checkers for parity, two-rail, and linear codes in order that they receive all necessary code words. A checker designed according to this approach will have the following advantages over other known checker designs:

- The checker will receive all code words irrespective of which and how many code words are provided by the CUC.
- The checker will be tested by a linear feedback shift register (LFSR) without going into a test mode.
- In the case of a parity checker the checker can be provided not only with all code words, but with all possible input words such that it will be tested exhaustively (while at the same time it is monitoring the output of the CUC).
- A noncode word that is not detected due to an undetected checker fault can be detected in later instances of time because the error pattern of the noncode word will be memorized and modified (the error pattern will cycle through the LFSR).

The fact that the set of code words received by the checker is independent of the set of code words provided by the CUC makes it possible to use every checker design for the respective code, especially the classical designs.

The rest of the paper is organized as follows. In Section 2 we first define the basic notions related to self-checking checkers. The generation of the test patterns is mainly based on the theory of cyclic codes. We therefore consider the generation of cyclic code words by linear feedback shift registers. This will be done in Section 3. In Sections 4 and 5 we discuss the design of embedded parity and two-rail checkers using an internal linear feedback shift register generating cyclic code words. In Section 6 we extend the proposed method to the design of embedded linear code checkers. The design of controllable checkers with internal LFSR will be addressed in Section 7. Finally, concluding remarks are given in Section 8.

2 TSC CHECKERS—BASIC DEFINITIONS

In this section we provide the basic definitions for self-checking circuits that will be used throughout this paper[1,2]. We denote the input code space of a circuit by X and the output code space by Y. The output of a circuit is defined by a Boolean vector function F. Let Φ be a set of physical faults and let φ be a fault of Φ. We denote the function of the circuit containing the fault φ by $F(x, \varphi)$. The fault-free function will be denoted by $F(x, \emptyset)$.

DEFINITION 1 A circuit is called *self-testing* with respect to Φ if and only if

$$\forall \varphi \in \Phi \; \exists x \in X\colon F(x, \varphi) \notin Y.$$

DEFINITION 2 A circuit is called *fault-secure* with respect to Φ if and only if

$$\forall x \in X \; \forall \varphi \in \Phi\colon F(x, \varphi) \notin Y \quad \text{or} \quad F(x, \varphi) = F(x, \emptyset).$$

DEFINITION 3 A circuit is called *totally self-checking* (TSC) with respect to Φ if and only if it is self-testing and fault-secure with respect to Φ.

DEFINITION 4 A circuit is called *code-disjoint* if and only if

$$\forall x \in X\colon F(x,\emptyset) \in Y \quad \text{and} \quad \forall x \notin X\colon F(x,\emptyset) \notin Y.$$

DEFINITION 5 A circuit is called a *totally self-checking checker* if and only if it is totally self-checking and code-disjoint.

A TSC network always produces a noncode word as the first erroneous output due to a fault. This behaviour is referred to as the *TSC goal*[2]. This goal is achieved under the following hypotheses[2].

HYPOTHESIS 1 Each fault can be modeled as a member of Φ.

HYPOTHESIS 2 Faults occur one at a time and between the occurrence of any two faults a sufficient time elapses such that all code word inputs are applied to the network.

In this paper we consider Φ to be the set of all single stuck-at faults. The fault-secure property is not really necessary for checkers if we consider only single faults[3,16,17]. Since a valid input code word should produce a valid output code word from the checker, it is not important which correct output is produced. Therefore, for single faults the self-testing property is sufficient for a checker. For sequences and sets of faults the property of a checker to be *strongly code-disjoint*[3] or *strongly self-checking*[4] is of more interest.

In the paper we only discuss the design of totally self-checking (TSC) embedded checkers. Since the TSC property is the strongest requirement, the same design method can be used if embedded checkers have only to be self-testing, strongly code-disjoint, or strongly self-checking.

3 LFSRs AND CYCLIC CODES

An n-stage linear feedback shift register (LFSR) as shown in Fig. 1 is represented by its feedback polynomial $h(x) = x^n + \sum_{i=0}^{n-1} h_i x^i$. Let $s(t)$ be the state vector of the LFSR at time t. Then $s(t+i)$ can

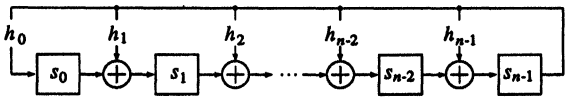

FIGURE 1 Linear feedback shift register.

be obtained by $s(t+i) = M^i s(t)$, where

$$M = \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 & h_0 \\ 1 & 0 & \cdots & 0 & 0 & h_1 \\ 0 & 1 & \cdots & 0 & 0 & h_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & h_{n-2} \\ 0 & 0 & \cdots & 0 & 1 & h_{n-1} \end{pmatrix}$$

is the *companion matrix* of the polynomial $h(x)$. The inverse matrix of M, M^{-1}, always exists if $h_0 = 1$.

An (n,k) binary cyclic code is a k-dimensional subspace of an n-dimensional vector space[18]. A code word $c = (c_0, c_1, \ldots, c_{n-1})$ can be associated with the polynomial $c(x) = c_0 x^{n-1} + c_1 x^{n-2} + \cdots + c_{n-2}x + c_{n-1}$, which is called the *code polynomial* of c. An (n,k) cyclic code is defined by its *generator polynomial* $g(x)$ of degree $n-k$. A polynomial is a code polynomial if it is divisible by $g(x)$.

THEOREM 1[19] Let $g(x)$ be a generator polynomial of an (n,k) cyclic code, and let $p(x)$ a primitive polynomial of degree k. Then the LFSR with the feedback polynomial $h(x) = p(x)g(x)$ generates all nonzero code words of the (n,k) cyclic code.

A cyclic code is also defined by its *generator matrix* G. If i is the information bit vector then the cyclic code word corresponding to i is $c = iG$. Given the generator matrix G, the generator polynomial $g(x)$ of the cyclic code with $c(x) = i(x)g(x)$ can easily be derived.

As cyclic codes are linear codes they have the property that the sum of two code words is a code word too. Additionally, a cyclic shift of a code word also results in a code word. These are the basic properties the new checker designs are based on.

4 EMBEDDED TSC PARITY CHECKERS

The basic properties of cyclic codes can be used to provide additional code words for a checker. Since a checker does not check *which* code word it gets but *whether it is* a code word, we can sum up two code words, one provided by the CUC, the other by an LFSR. The sum will be again a code word which is sent to the checker. Suppose that the CUC generates a code word c that belongs to a linear code. If we add another code word c', provided by an LFSR, then the sum $c \oplus c'$ will again be a code word. If an erroneous word $c \oplus e$ is produced by the CUC, and if we add again c' then the resulting vector will have the same error pattern e. Therefore, a noncode word produced by either the CUC or the LFSR will cause the sum to be a noncode word too that will be detected by the checker.

If the number of code words produced by the CUC is too small to make the checker totally self-checking then the pairwise sum of the code words provided by the CUC with code words generated by the LFSR will produce all necessary code words for the checker. First we consider the generation of (even) parity code words.

OBSERVATION 1 The parity code is a cyclic code. This property can be proved easily since a cyclic shift of a code word does not change the parity of this word.

Therefore, all nonzero parity code words can be generated by an LFSR (see Section 3).

Example 1 We consider the parity code of length $n=4$, i.e. a $(4,3)$-cyclic code. A generator polynomial of the parity code is $g(x)=x+1$. According to Theorem 1 we choose a primitive polynomial of degree $k=3$, e.g. $p(x)=x^3+x+1$. Therefore, an LFSR with the feedback polynomial $h(x)=$

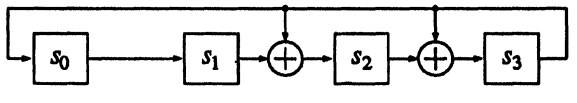

FIGURE 2 LFSR with feedback polynomial $x^4+x^3+x^2+1$.

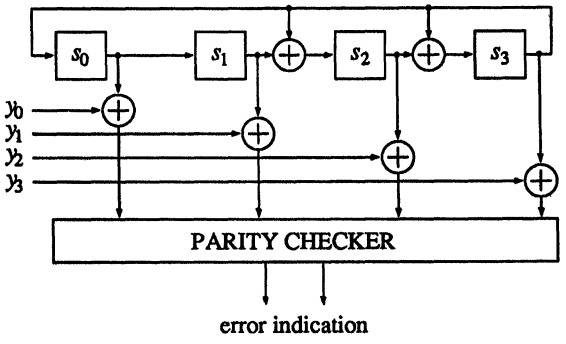

FIGURE 3 4-bit parity checker with LFSR.

$p(x)g(x)=x^4+x^3+x^2+1$ can generate all nonzero 4-bit parity code words (Fig. 2).

The sequence of code words generated by the LFSR of Fig. 2 with the initial state $s(0)=c_1=$ 1100 is

$$
\begin{array}{lcccc}
c_1 & = & 1 & 1 & 0 & 0 \\
c_2 & = & 0 & 1 & 1 & 0 \\
c_3 & = & 0 & 0 & 1 & 1 \\
c_4 & = & 1 & 0 & 1 & 0 \\
c_5 & = & 0 & 1 & 0 & 1 \\
c_6 & = & 1 & 0 & 0 & 1 \\
c_7 & = & 1 & 1 & 1 & 1 \\
c_1 & & & & & \\
\vdots & & & & &
\end{array}
$$

The LFSR of Fig. 2 can now be used as a test pattern generator (TPG) for a parity checker as shown in Fig. 3.

OBSERVATION 2 If the CUC produces at least two different code words then it is possible to provide each parity code word for the parity checker.

If we assume that the CUC provides only one code word c then the checker will get all parity code words, except $c \oplus 0 = c$. The checker can also get the code word c if the CUC can generate at least one additional code word different from c. Assuming statistical independence of the code words produced by the CUC and the code words generated by the LFSR it is possible to provide every parity code word for the checker. Under Hypothesis 2 the parity checker is totally self-checking if the CUC produces at least two different parity code words.

A modification of the checker of Fig. 3 is shown in Fig. 4. The code words provided by the CUC are now fed directly into the LFSR. This has the advantage that the error pattern of a noncode word will be captured by the LFSR, and it will be modified since it will cycle through the LFSR. Therefore, if a noncode word cannot be detected due to an undetected checker fault then it can be detected one (or maybe more) clock cycles later.

The hardware required for the checker of Fig. 4 is the same as for the checker shown in Fig. 3. To distinguish the two checker designs we call this type of checker an *error-memorizing* embedded self-checking checker. Regarding the number of code words the parity checker in Fig. 4 will receive, we can make a statement similar to that in Observation 2.

THEOREM 2 Let the LFSR that is used as internal test pattern generator generate all nonzero parity code words. If the CUC generates at least two different output words then it is possible to provide each parity code word for the checker.

Proof Let M be the companion matrix of the feedback polynomial $h(x)$ of the LFSR. Assume that the CUC produces only one output response y. Let $s(t)$ be the state of the LFSR at time t, which is a code word. Then the state of the LFSR at time $t+1$ is $s(t+1)=M(s(t)\oplus y)$. In general, state $s(t+i)$ is given by

$$s(t+i) = M^i s(t) \oplus \sum_{j=1}^{i} M^j y. \tag{1}$$

Since $y \oplus M^{-1}y$ is a code word there must be a number b such that $M^{-b}y = y \oplus M^{-1}y$. Therefore

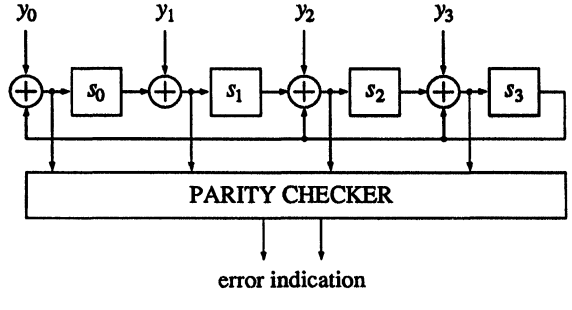

FIGURE 4 Error-memorizing 4-bit parity checker.

we have $y = M^b y \oplus M^{b-1}y$. Let $a = M^b y$. This yields

$$y = a \oplus M^{-1}a$$

Because $My \oplus a = Ma$ we have $M^i y = M^i a \oplus M^{i-1}a$ and therefore

$$\sum_{j=1}^{i} M^j y = M^i a \oplus a. \tag{2}$$

With (1) and (2) we obtain

$$s(t+i) \oplus a = M^i(s(t) \oplus a). \tag{3}$$

Equation 3 indicates that the parity checker will get all code words except a. If the CUC produces at least one additional code word different from y then the parity checker also can get the code word a. Assuming statistical independence of the code words produced by the CUC and those generated by the LFSR it is possible to provide every code word for the parity checker. ◁

Example 2 Let's consider again the LFSR of Fig. 2. Assume that c_1 is the only code word provided by the CUC. If we start the LFSR sequence again with $s(0)=c_1$ then with $s(t+1)=M(s(t)\oplus y)$ we obtain the following sequence.

$$\begin{array}{llllllll}
c_1 & = & 1 & 1 & 0 & 0 & = & c_5 \oplus c_6 \\
c_0 & = & 0 & 0 & 0 & 0 & = & c_6 \oplus c_6 \\
c_2 & = & 0 & 1 & 1 & 0 & = & c_7 \oplus c_6 \\
c_5 & = & 0 & 1 & 0 & 1 & = & c_1 \oplus c_6 \\
c_7 & = & 1 & 1 & 1 & 1 & = & c_2 \oplus c_6 \\
c_4 & = & 1 & 0 & 1 & 0 & = & c_3 \oplus c_6 \\
c_3 & = & 0 & 0 & 1 & 1 & = & c_4 \oplus c_6 \\
c_1 & & & & & & & \\
\vdots & & & & & & &
\end{array}$$

The only code word that cannot be generated is $a = c_6$ since $b = 5$. □

The design of the code word generator can be further improved with respect to the number of generated code words if we use *complete feedback shift registers* (*CFSRs*) instead of LFSRs. A CFSR can additionally cycle through the all-zero state. There is an efficient method to derive a CFSR from

the corresponding LFSR[20]. From Observation 2 and Theorem 2 we can derive the following corollary.

COROLLARY 1 If a parity checker is provided with a CFSR that generates all parity code words then it is possible to provide all code words for the parity checker irrespective of which and how many code words are produced by the CUC.

The use of an LFSR (or the corresponding CFSR) assures that the parity checker will receive all parity code words. If we use an LFSR to support a parity checker then we are also able to provide the checker with *all* input words, i.e., all code words and all noncode words (which would not be possible if the checker gets the code words directly from the CUC). If we can provide all input words to the parity checker then the checker will be tested *exhaustively*, and therefore we can detect more checker faults as in the case where only code words are provided.

This can be easily achieved if we use an LFSR of length $n+1$, where n is the number of input lines of the parity checker. An exhaustively tested 4-bit parity checker is shown in Fig. 5. Instead of a 2-output parity checker we can use a single-output checker which is simply an XOR-tree. Single-output checkers can be used if it can be guaranteed that the checker will receive both, code words and noncode words, and therefore a stuck-at 'correctly' fault at the checker output can be detected[12,17]. The second output line of the whole checker is provided by the rightmost flip-flop of the LFSR. This yields an XOR-type controllable checker[5] consisting of a single-output checker and one control line.

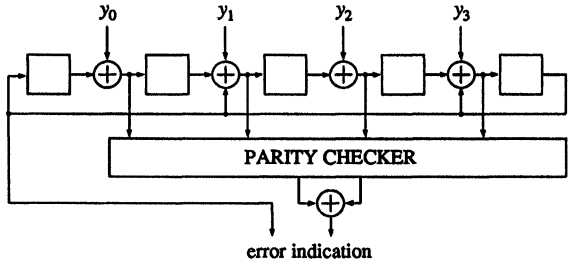

FIGURE 5 Exhaustively tested 4-bit parity checker.

Since the LFSR can generate all nonzero $n+1$-bit parity code words it can generate every n-bit word, where the n bit positions can be an arbitrary subset of the $n+1$ bit positions. (Similar LFSRs often have been used as TPGs for the pseudo-exhaustive test, because they can generate exhaustively all w-dimensional subspaces of an $n+1$-dimensional space[21–23].) It can easily be seen that a noncode word received from the CUC will be detected by the checker.

The proposed design of embedded parity checkers relies on additional hardware (LFSR and XOR gates) in order to guarantee that the checkers get all necessary words to be totally self-checking. Since this additional hardware is part of the checker we also have to ensure that faults inside this checker part will be detected. Faults that can occur in the LFSR always depend on how the flip-flops and the XOR gates are realized. We therefore only consider single stuck-at faults at the inputs and outputs of flip-flops and XOR gates. It is easy to verify that all single stuck-at faults of the flip-flops and the XOR gates will influence the parity of the code words that the checker finally gets. Therefore, errors caused by these faults will be detected immediately. In CFSRs the detection of all single stuck-at faults cannot be guaranteed because some faults never change the parity of the generated words. However, the use of CFSRs is only of theoretical interest. In practical applications an LFSR will be sufficient since it is reasonable to expect that the CUC produces more than only one output response.

Summarizing up to this point we can state the following. We presented a design method for embedded parity checkers that have the following properties. The checker will get every parity code word irrespective of which and how many code words are provided by the CUC. The checker will get the missing code words by an LFSR that serves as test pattern generator while at the same time it is monitoring the output of the CUC. The checker can be tested with all possible input words, i.e., code words and noncode words, while it is monitoring the CUC. This corresponds to an exhaustive test of the checker, performed on-line. The TPG is able to

capture and modify the error pattern of a noncode word such that noncode words can be detected by the checker even in presence of undetected checker faults.

We only discussed the design of embedded checkers for the even parity code. The same checker design can be used for the odd parity code. We only have to invert one output of the CUC, i.e. odd parity code words are converted to even parity code words before they are sent to the checker.

5 EMBEDDED TSC TWO-RAIL CHECKERS

A two-rail code word consists of n bit pairs a_i, a_{n+i}, $i=1,\ldots,n$, with $a_i = a_{n+i} \oplus 1$. We denote this code by C_{TRC}. The two-rail code is not linear since with two code words c_i and c_j the sum $c_i \oplus c_j$ is not a two-rail code word. A code that is similar to the two-rail code is the duplication code consisting of those words for which $a_i = a_{n+i}$, $\forall i$, holds. This code will be denoted by C_{EQU}. It is easy to see that for $c_1 \in C_{TRC}$ and $c_2 \in C_{EQU}$ we have $c_1 \oplus c_2 \in C_{TRC}$. Obviously, C_{EQU} is a cyclic code since the $n \times 2n$ generator matrix has the form

$$G = \begin{pmatrix} 1 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 1 \end{pmatrix}.$$

An LFSR with the feedback polynomial $h(x) = (x^n + 1)p^{(n)}(x)$ can generate all nonzero words of C_{EQU}, where $p^{(n)}(x)$ is a primitive polynomial of degree n. Similarly to Observation 2 it can be shown that a two-rail checker provided with such an (external) LFSR can be provided with every nonzero two-rail code word. An alternative possibility consists of inverting the check bits of each two-rail code word. In this way words of C_{TRC} will be converted to words of C_{EQU}. With $c_i \in C_{EQU}$ and $c_j \in C_{EQU}$ we have $c_i \oplus c_j \in C_{EQU}$, and after inverting again the check bits (or the information bits) we can check the obtained result by a two-rail checker.

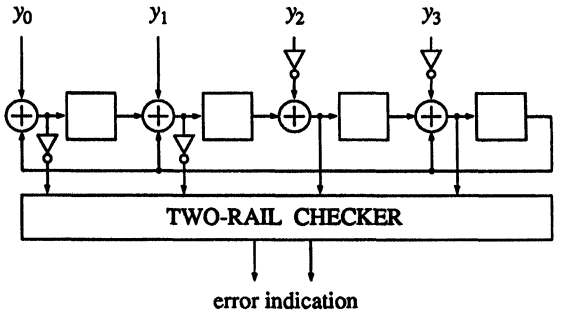

FIGURE 6 Error-memorizing 4-bit two-rail checker.

This alternative approach can also be used to design the error memorizing version of a two-rail checker. An example of such a design is given below. The structure of this checker is similar to that of an error-memorizing parity checker as shown in Fig. 4.

Example 3 Figure 6 shows a 4-bit two-rail checker with internal LFSR that generates all nonzero 4-bit code words of C_{EQU}. The LFSR feedback polynomial $h(x)$ is determined by $h(x) = (x^2 + 1)p^{(2)}(x)$, where $p^{(2)}(x)$ is a primitive feedback polynomial of degree 2. With $p^{(2)}(x) = x^2 + x + 1$ we have $h(x) = x^4 + x^3 + x + 1$. Since the check bits y_2 and y_3 are inverted the LFSR will receive words of C_{EQU}. After inverting the information bits of the words generated by the LFSR the two-rail checker will receive words of C_{TRC}. □

THEOREM 3 Let the LFSR that is used as test pattern generator generate all nonzero words of C_{EQU}. If the CUC generates at least two different output words then it is possible to provide each two-rail code word for the checker.

The prove of Theorem 3 is similar to that of Theorem 2. If the corresponding CFSR is used as code word generator then it is possible to provide every two-rail code word for the checker, irrespective of which and how many code words are produced by the CUC.

6 EMBEDDED LINEAR CODE CHECKERS

We can now extend the results of Sections 4 and 5 to the design of checkers for arbitrary linear codes.

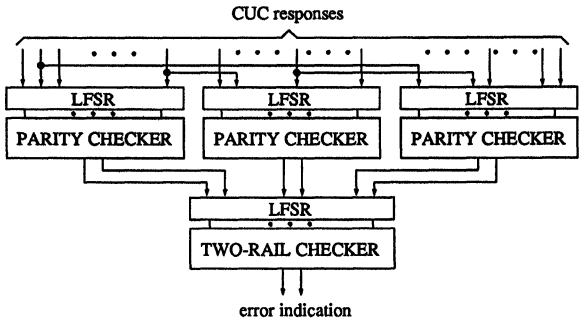

FIGURE 7 General design of an embedded linear code checker.

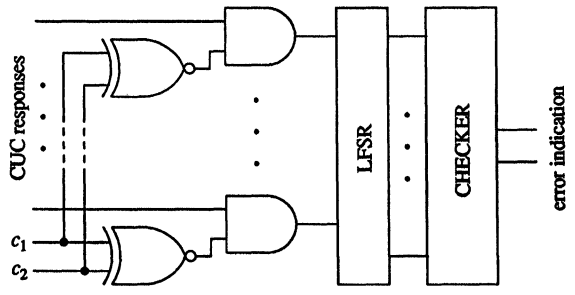

FIGURE 8 Controllable checker with internal LFSR.

Let C be an (n, k) linear code. If C is a cyclic code or can be converted into a cyclic code then we can directly apply the results of Observation 2 and Theorem 2. Otherwise, we consider the (n, k) linear code as a collection of $n-k$ parity codes given by the parity check equations of the (n, k) code. For each of the $n-k$ parity codes we design an embedded checker as shown in Section 4. The $n-k$ parity code checkers have a two-rail encoded output each. An embedded two-rail checker which can be designed as shown in Section 5 will map the $n-k$ two-rail signals to a single two-rail signal. The general scheme of an embedded linear code checker is shown in Fig. 7. The figure shows an $(n+3, n)$ linear code checker.

7 DESIGN OF CONTROLLABLE CHECKERS

Controllable checkers can be used in systems where the output of the functional circuit is monitored only for a subset of input words. The function of a controllable checker is defined as follows[5]. The checker performs its usual check function if it is instructed to check (it monitors the output of the functional circuit). If the checker is instructed not to check it gives a fault-free indication regardless of the output response it receives from the functional circuit.

As for conventional code checkers the main problem when designing controllable checkers is the lack of code words to make the checkers totally self-checking. The design of checkers with internal LFSR as proposed in this paper for parity, two-rail, and linear code checkers also allows a new design possibility of controllable checkers for these codes. The general scheme of a controllable checker with internal LFSR is shown in Fig. 8.

For a 'CHECK ON' signal, $c_1c_2 \in \{00, 11\}$ the checker monitors the output of the functional circuit. In the 'CHECK OFF' mode, $c_1c_2 \in \{01, 10\}$, where the output response of the functional circuit is usually a noncode word this circuit response is prevented from being loaded into the LFSR. The checker just receives a new code word generated by the LFSR from its previous state. The use of one XOR gate for every circuit output instead of a joint XOR gate for all outputs avoids an undetected stuck-at 'CHECK OFF' fault.

The checkers with internal LFSR are in system mode and in test mode at the same time whereas a controllable checker with LFSR is in both modes at the same time only for a 'CHECK ON' signal. For a 'CHECK OFF' signal it is only in the test mode.

8 CONCLUSIONS

In this paper we presented a novel method for the design of embedded TSC checkers for parity, two-rail, and linear codes. The main advantage of the checkers designed according to this method is that they will receive all code words irrespective of which and how many code words are provided by the functional circuit. The checker will receive the

code words by an LFSR without going into a real off-line test mode, i.e., on-line and off-line test are perfectly merged. Another advantage over conventional checker designs is that the test pattern generator is able to capture the error patterns of noncode words such that noncode words that are not detected due to undetected checker faults can be detected in later instances of time.

Most of the presently known checkers are TSC under the hypothesis that a fault is detected before another fault occurs. This hypothesis is true only with a certain probability[24]. The capability of a checker to memorize errors allows to increase this probability.

Compared to conventional checkers the proposed checker designs require some additional hardware in order to generate the required code words. This hardware is very simple—only LFSRs are needed. The additional hardware overhead is smaller than for self-exercising checkers[12] since we do not need any logic to distinguish code words from noncode words.

If two (or several) functional circuits in the system generate responses of the same code (parity, two-rail, or linear code) then these responses can be combined to one word of this code by adding them up and the result is sent to the checker. Especially, in some cases one functional circuit (or a collection of circuits) can serve as test pattern generator for a non-error memorizing checker if it can provide enough code words, supposed that at most one circuit response is erroneous at a time.

It is also possible to use the proposed checkers in built-in self-test (BIST) applications[25]. In this case, the test pattern generator of the checker can be employed as signature analyzer for the CUC. The checker can reduce the masking of errors during BIST. On the other hand, BIST can detect errors that cannot be detected by the checker only, i.e. wrong code words.

The proposed method for designing embedded checkers using LFSRs as code word generators and the XOR operation for combining code words to new ones is limited to error control codes that are related to cyclic codes. However, the general design principle can be extended to other codes as well. Embedded checker designs for unidirectional error detecting codes (e.g. m-out-of-n codes, Berger codes) and arithmetic codes (e.g. AN codes, residue code) are currently under investigation.

Acknowledgements

The author would like to thank M. Gössel and E.S. Sogomonyan for the discussions on code word generation for checkers, and the anonymous reviewers for their helpful comments and suggestions. This research was performed while the author was with Max Planck Society, Fault-Tolerant Computing Group at the University of Potsdam, Germany.

References

[1] D. A. Anderson and G. Metze, "Design of Totally Self-Checking Check Circuits for m-out-of-n Codes," *IEEE Transactions on Computers*, vol. C-22, no. 3, pp. 263–269, March 1973.

[2] J. E. Smith and G. Metze, "Strongly Fault Secure Logic Networks," *IEEE Transactions on Computers*, vol. C-27, no. 6, pp. 491–499, June 1978.

[3] M. Nicolaidis, I. Jansch and B. Courtois, "Strongly Code Disjoint Checkers," *Proc. 14th International Symposium on Fault-Tolerant Computing*, Orlando, FL, pp. 16–21, June 1984.

[4] N. K. Jha, "Fault Detection in CVS Parity Trees with Application to Strongly Self-Checking Parity and Two-Rail Checkers," *IEEE Transactions on Computers*, vol. 42, no. 2, pp. 179–189, Feb. 1993.

[5] S. Tarnick, "Controllable Self-Checking Checkers for Conditional Concurrent Checking," *Proc. 12th IEEE VLSI Test Symposium*, Cherry Hill, NJ, pp. 144–150, April 1994.

[6] J. Khakbaz, "Totally Self-Checking Checker for 1-out-of-n Code using Two-Rail Codes," *IEEE Transactions on Computers*, vol. C-31, no. 7, pp. 677–681, July 1982.

[7] J. Khakbaz and E. J. McCluskey, "Self-Testing Embedded Parity Checkers," *IEEE Transactions on Computers*, vol. C-31, no. 8, pp. 753–756, August 1984.

[8] M. Nicolaidis, "Fault Secure Property Versus Strongly Code Disjoint Checkers," *IEEE Transactions on Computer-Aided Design*, vol. 13, no. 5, pp. 651–658, May 1994.

[9] E. Fujiwara and K. Matsuoka, "A Self-Checking Generalized Prediction Checker and Its Use for Built-In Testing," *IEEE Transactions on Computers*, vol. C-36, no. 1, pp. 86–93, January 1987.

[10] S. Kundu and S. M. Reddy, "Embedded Totally Self-Checking Checkers: A Practical Design," *IEEE Design and Test of Computers*, pp. 5–12, August 1990.

[11] M. Nicolaidis, *Design of Specific Checkers*, IMAG/TIM3 Technical Report RR 535, Grenoble, May 1985.

[12] M. Nicolaidis, "A Unified Built-In Self-Test Scheme: UBIST," *Proc. 18th International Symposium on Fault-Tolerant Computing*, Tokyo, pp. 157–163, June 1988.

[13] W.-B. Jone and C.-J. Wu, "Multiple Fault Detection in Parity Checkers," *IEEE Transactions on Computers*, vol. 43, no. 9, pp. 1096–1099, Sept. 1994.
[14] S. M. Reddy, I. Pomeranz, and R. Jain, "On Codeword Testing of Two-Rail and Parity TSC Checkers," *Proc. 24th International Symposium on Fault-Tolerant Computing*, Austin, TX, pp. 116–125, June 1994.
[15] S. C. Seth and K. C. Kodandapani, "Diagnosis of Faults in Linear Tree Networks," *IEEE Transactions on Computers*, vol. C-26, no. 1, pp. 29–33, Jan. 1977.
[16] Y. Tamir and C. H. Sequin, "Design and Application of Self-Testing Comparators Implemented with CMOS PLA's," *IEEE Transactions on Computers*, vol. C-33, no. 6, pp. 493–506, June 1984.
[17] S. D. Millman and E. J. McCluskey, "Bridging, Transition, and Stuck-Open Faults in Self-Testing CMOS Checkers," *Proc. Int. Symposium on Fault-Tolerant Computing, FTCS 21*, Montréal, pp. 154–161, 1991.
[18] W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*, MIT Press, Cambridge, 1972.
[19] M. Y. Hsiao, A. M. Patel, and D. K. Pradhan, "Store Address Generator with On-Line Fault-Detection Capability," *IEEE Transactions on Computers*, vol. C-26, no. 11, pp. 1144–1147, November 1977.
[20] L. T. Wang and E. J. McCluskey, "Complete Feedback Shift Register Design for Built-In Self-Test," *Proc. Int. Conf. on Computer-Aided Design, ICCAD*, pp. 56–59, 1986.
[21] C. L. Chen, "Exhaustive Test Pattern Generation Using Cyclic Codes," *IEEE Transactions on Computers*, vol. 37, no. 2, pp. 225–228, Febr. 1988.
[22] L.-T. Wang and E. J. McCluskey, "Circuits for Pseudo-Exhaustive Test Pattern Generation," *IEEE Transactions on Computer-Aided Design*, vol. 7, no. 10, pp. 1068–1080, Oct. 1988.
[23] T. Damarla and A. Sathaye, "Applications of One-Dimensional Cellular Automata and Linear Feedback Shift Registers for Pseudo-Exhaustive Testing," *IEEE Transactions on Computer-Aided Design*, vol. 12, no. 10, pp. 1580–1591, Oct. 1993.
[24] J.-C. Lo and E. Fujiwara, "A Probabilistic Measurement for Totally Self-Checking Circuits," *Proc. IEEE Int. Workshop on Defect and Fault Tolerance in VLSI Systems*, Venice, pp. 263–270, Oct. 1993.
[25] S. K. Gupta and D. K. Pradhan, "Can Concurrent Checkers Help BIST?," *Proc. International Test Conference*, Baltimore, MD, pp. 140–150, Sept. 1992.

Author's Biography

Steffen Tarnick received the diploma degree in mathematics from the Technical University of Dresden, Germany, in 1989, and the Dr. rer. nat. (Ph.D.) degree from the University of Potsdam, Germany, in 1995. From 1989 to 1991 he was a Research Assistant at the Central Institute of Cybernetics and Information Processes of the East German Academy of Sciences in Berlin. Then he spent one year as visiting scientist at the TIMA/IMAG Laboratory in Grenoble, France. From 1992 to 1995 he was with the Max Planck Society Group for Fault-Tolerant Computing at the University of Potsdam, Germany. He is currently with SATCON GmbH in Teltow, Germany. His main research interests include built-in self-test, self-checking circuits design and cryptographic authentication systems. Steffen Tarnick is a member of the IEEE.