Comput. Math. Applic. Vol. 16, No. 9, pp. 705–714, 1988
Printed in Great Britain.

0097-4943/88 $3.00 + 0.00

A t-UNIDIRECTIONAL ERROR-DETECTING SYSTEMATIC CODE

N. K. JHA
Department of Electrical Engineering, Engineering Quad, Princeton University, Princeton, NJ 08544, U.S.A.

M. B. VORA
Yojna Inc., Detroit, MI 48018, U.S.A.

(*Received* 10 *February* 1988)

Communicated by E. Y. Rodin

Abstract—A systematic code consists of codewords in which the check symbol is appended to the information symbol. Thus, data manipulation and encoding/decoding can be done in parallel. The Berger code is a well-known optimal systematic code for detecting all unidirectional errors. In VLSI circuits most of the errors are found to be unidirectional in nature. However, in many applications it may not be necessary to detect all unidirectional errors. Most faults, unless catastrophic in nature, do not cause errors in all the bits of the information and check symbol. Therefore, it may be enough to guarantee detection of every unidirectional error in t or fewer bits of the codeword, if t is reasonably large. In this paper we present such a t-unidirectional error-detecting code.

1. INTRODUCTION

Due to the ever-increasing complexity of modern computers, ensuring the reliability of the data in the computer system has become very important. One way to achieve this aim is to use redundancy in the data bits. With the help of redundant bits it is possible to detect and/or correct errors.

Extensive research has been done in the area of symmetric error detecting/correcting codes [1–4]. In a symmetric error model, errors of type 1-to-0 and 0-to-1 are considered to be equally likely. However, it has been shown that the errors in VLSI circuits are of the unidirectional type [5, 6]. A unidirectional error model assumes that even though both 1-to-0 and 0-to-1 errors are allowed, only one type of error occurs in a particular data word.

Some codes have been found which detect all unidirectional errors in the data word [7–9]. For the codes given in Refs [7, 8] the information symbol can be separated from the check symbol. This enables the data manipulation and encoding/decoding to be done in parallel. The Berger code [7] is optimal if all 2^k information symbols occur in the code, where k is the number of bits in the information symbol. However, when all the information symbols are not present, Smith's code [8] is optimal. In Ref. [9] Frieman gave a non-systematic constant-weight code to detect all unidirectional errors. He showed that $\lfloor n/2 \rfloor$-out-of-n codes are the least redundant block codes. However, the disadvantage of non-systematic codes is that decoding is necessary to get the information symbol from the codeword.

The three codes mentioned above are optimal when all unidirectional errors are required to be detected. But when we need to detect unidirectional errors in only any t or fewer bits in the codeword, these codes are not optimal. In Ref. [10] systematic t-unidirectional error-detecting (t-UED) codes were presented which require a fixed number of checkbits independent of the number of information bits. These codes are capable of detecting 2, 3 and 6 undirectional errors when 2, 3 and 4 checkbits are used respectively. For these cases the codes are shown to be optimal. For $r \geqslant 5$, where r is the number of checkbits, codes capable of detecting up to $5 \times 2^{r-4} + r - 4$ unidirectional errors were presented. However, as can be seen from the code presented in this paper, it is possible to detect a much larger number of unidirectional errors for the same r, depending on that the value of k is.

In Ref. [11] modified Berger codes were given to detect t-unidirectional errors. However, the codes presented in Ref. [10] and our code have higher code-detecting capabilities.

In Ref. [12] Borden proved that the set of codewords with weight $\lfloor n/2 \rfloor \bmod (t+1)$ forms the optimal code among all t-UED codes of length n. However, these codes are non-systematic in nature.

In this paper we present a systematic t-UED code which performs better than the Bose–Lin code [10] for certain ranges of k. The organization of the paper is as follows. In Section 2 we discuss the different error classes. In Section 3 the capabilities of binary block codes are discussed. In Section 4 we give the encoding technique and other results for our systematic code. In Section 5 we discuss the error-detecting capability of this code. In Section 6 we discuss a self-checking checker for our code.

2. ERROR CLASSES

In the following, we define three different type of errors—symmetric, asymmetric and unidirectional:

Symmetric errors. If both 0-to-1 and 1-to-0 errors appear in a data word with equal probability then the errors are termed symmetric errors, and the channel is termed a symmetric channel.

Asymmetric errors. When only either 0-to-1 or 1-to-0 errors occur in any data word and the error type is known *a priori*, the errors are termed asymmetric errors, and the channel is termed an ideal asymmetric channel.

Unidirectional errors. If both 0-to-1 and 1-to-0 errors can occur in a data word, but in any particular word only one type of error occurs, then the errors are termed unidirectional errors.

A binary channel model [1–4], given in Fig. 1, can be used to illustrate both symmetric as well as asymmetric channels.

If $p = q$, it means that the probabilities of 1-to-0 and 0-to-1 transitions are the same. For such a case the channel would become a binary symmetric channel. But if $p \geqslant q$ or vice versa, the channel would become asymmetric. For the ideal asymmetric channel, either $p = 0$ or $q = 0$. When errors are unidirectional in nature, 0-to-1 and 1-to-0 errors occur with equal probability, but in any given word only one type of error can occur.

3. NECESSARY AND SUFFICIENT CONDITIONS

We will now present some necessary and sufficient conditions for symmetric, asymmetric and unidirectional error detection. We start with the following definitions:

Definition 1

A word $X = (x_1, x_2, \ldots, x_n)$ is said to cover another word $Y = (y_1, y_2, \ldots y_n)$ if $\forall\, i$, $y_i = 1$ implies $x_i = 1$. We write $X \geqslant Y$.

If neither covers the other, the words are said to be unordered. Else if $X \geqslant Y$ or $Y \geqslant X$, X and Y are said to form an "ordered pair". For example, if $X_1 = (0011)$ and $Y_1 = (0010)$ then $X_1 \geqslant Y_1$, and X_1 and Y_1 form an ordered pair. But if $X_2 = (1010)$ and $Y_2 = (1001)$ then neither covers the other, and they are said to be unordered. Note that a word always covers itself. The symbol $\geqslant$ will also be used to compare decimal numbers. It will be clear from the context as to what sense it is used in.

Definition 2

The Hamming distance $d(X, Y)$ between two words X and Y is the number of bit positions they differ in.

For example, if $X = (1011)$ and $Y = (0101)$, then $d(\mathrm{X}, \mathrm{Y}) = 3$.

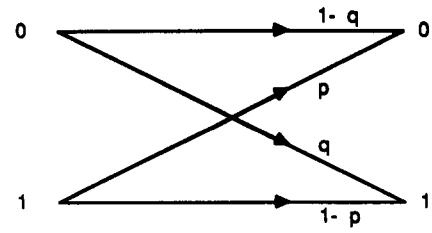

Fig. 1. A binary channel.

The following theorem by Hamming [13] gives the symmetric error-detecting capabilities of binary block codes.

Theorem 1

A code C is capable of detecting t or fewer symmetric errors iff the minimum Hamming distance of the code is at least $t + 1$.

From the definition of the error classes it is obvious that a code C satisfying the above condition will also detect t-unidirectional or t-asymmetric errors.

The next theorem [14] gives the necessary and sufficient condition for the detection of unidirectional errors.

Theorem 2

A code C is capable of detecting all unidirectional errors if every pair of codewords is unordered.

However, if we want to detect only t-unidirectional or t-asymmetric errors this condition needs to be modified, as given in Theorem 3 below [10]:

Theorem 3

A code C is capable of detecting t-asymmetric errors if and only if the following condition is true. For all $X, Y \in C$, either X and Y are unordered, or $d(X, Y) \geqslant t + 1$ when one covers the other. Further, a code capable of detecting t-asymmetric errors is also capable of detecting t-unidirectional errors.

We will use this theorem later to establish the t-UED capability of our code.

4. A t-UED CODE

It was mentioned earlier that Berger codes [7] are optimal systematic codes. For these codes $r = \lceil \log_2(k + 1) \rceil$. When $k < 2^r$ they are superior to any systematic t-UED code. Hence, for the purpose of this paper, we assume that $k \geqslant 2^r$.

As mentioned before, in Ref. [10] optimal codes to detect 2, 3 and 6 unidirectional errors using 2, 3 and 4 checkbits, respectively, were given. It is easy to see that single checkbit parity codes are optimal 1-UED codes. Furthermore, in Ref. [10], for $r \geqslant 5$, a systematic code capable of detecting up to $5 \times 2^{r-4} + r - 4$ unidirectional errors was given. We will show that our code detects a higher number of unidirectional errors for the same r for a given range of k. Typically, when k is between 2^r to $K \times 2^r$, where K is a constant roughly equal to 1.3 for odd r and 1.4 for even r, our code performs better. Beyond this range the Bose–Lin code [10] performs better.

A. Notations and Definitions

Definition 3

A number $M(r, x)$ is defined as follows:

$$M(r, x) = \sum_{j=1}^{x} \binom{r}{j}.$$

For our code we will need the number $M(r, \lfloor r/2 \rfloor)$. One can verify that the following is true:

$$M(r, \lfloor r/2 \rfloor) = \begin{cases} 2^{r-1} - 1, & r \text{ odd}, \\ 2^{r-1} + \frac{1}{2}\binom{r}{r/2} - 1, & r \text{ even}. \end{cases}$$

Definition 4

A set $S(r)$ is defined as follows:

$$S(r) = \{x \mid x \in (\lfloor r/2 \rfloor - y)\text{-out-of-}r \text{ codeword}, \ y = 0, 1, 2, \ldots, \lfloor r/2 \rfloor - 1\}.$$

We assume that in the set $S(r)$, the set of $(\lfloor r/2 \rfloor - y_1)$-out-of-$r$ codewords is placed before the set of $(\lfloor r/2 \rfloor - y_2)$-out-of-$r$ codewords if $y_1 < y_2$. Within any set of $(\lfloor r/2 \rfloor - y)$-out-of-$r$ codewords, the words are arranged in the order of decreasing value.

For example, $S(4) = \{1100, 1010, 1001, 0110, 0101, 0011, 1000, 0100, 0010, 0001\}$.

It is easy to see that $S(r)$ consists of $M(r, \lfloor r/2 \rfloor)$ words. We will denote the ith word in $S(r)$ as $C_i(r)$. For example, $C_1(4) = 1100$, $C_8(4) = 0100$ and so on.

Definition 5

A set $A_i(r)$, $i = 1, 2, \ldots, l$, for some $l \leqslant M(r, \lfloor r/2 \rfloor)$, is defined as follows:

$$A_i(r) = \{x \mid x \leqslant C_i(r) \text{ and } x \neq C_i(r)\},$$

$$A_{l+1}(r) = \text{null}.$$

The words in $A_i(r)$ are arranged in the order of decreasing value.

For example, let $r = 4$. We can see that $M(r, \lfloor r/2 \rfloor) = M(4, 2) = 10$. Let $l = 3$. From the earlier example we know that $C_1(4) = 1100$, $C_2(4) = 1010$ and $C_3(4) = 1001$. Hence,

$$A_1(4) = \{1000, 0100, 0000\}$$

$$A_2(4) = \{1000, 0010, 0000\}$$

$$A_3(4) = \{1000, 0001, 0000\}$$

and

$$A_4(4) = \text{null}.$$

Definition 6

$$B_i(r) = A_i(r) - A_i(r) \cap \left[\bigcup_{j=i+1}^{l} A_j(r) \right], \qquad i = 1, 2, \ldots, l-1,$$

$$B_l(r) = A_l(r).$$

For the above example,

$$B_1(4) = A_1(4) - A_1(4) \cap (A_2(4) \cup A_3(4)) = \{0100\}.$$

Similarly, $B_2(4) = \{0010\}$ and $B_3(4) = \{1000, 0001, 0000\}$.

Definition 7

A compaction operation on a set X, which consists of words of length r, is denoted as COMP, and defined as follows:

$$\text{COMP: } X \to Y \text{ where } Y = \left\{ y \mid y \in X \text{ and } y \notin \bigcup_{i=1}^{l} A_i(r) \right\}.$$

For example, if X is the set of 16 words of 4 bits, arranged in the order of decreasing value, and $l = 3$, then

$$\text{COMP: } X \to \{1111, 1110, 1101, 1100, 1011, 1010, 1001, 0111, 0110, 0101, 0011\}.$$

Definition 8

An append operation on any m sets $X_1, X_2, \ldots, X_m$ is denoted as APP, and defined as

$$\text{APP: } (X_1, X_2, \ldots, X_m) \rightarrow Z \text{ where } Z = X_1.X_2 \ldots X_m.$$

From the previous example, APP: $[C_3(4), B_3(4)] \rightarrow \{1001, 1000, 0001, 0000\}$.

Note that if $X_1 = \{1001, 1000\}$ and $X_2 = \{1001\}$, then APP: $(X_1, X_2) \rightarrow \{1001, 1000, 1001\}$. So if two or more of the sets have common members, the appended sequence includes each instance of the common members.

Given a sequence of words, in which a particular word Q occurs in at most two different positions, $p_1(Q)$ and $p_2(Q)$ will denote the numbers of the positions in which Q occurs.

A word W which has m zeros is said to have a group number m. This is denoted as $\text{Gn}(W) = m$. For example, $\text{Gn}(10000) = 4$ and $\text{Gn}(11111) = 0$.

The number of 1-to-0 transitions from a word W_1 to another word W_2 is denoted as $N(W_1, W_2)$. For example, $N(1000, 0111) = 1$ and $N(0111, 1000) = 3$.

B. Properties of the Different Sets

We will now look into the properties of some of the sets we have already defined.

Lemma 1

$$B_j(r) \cap B_k(r) = \text{null}, \qquad \text{for any } j, k = 1, 2, \ldots, l, j \neq k.$$

Proof. From Definition 6, $B_i(r)$, $i = 1, 2, \ldots, l-1$, contains those words from $A_i(r)$ which are not present in any $A_m(r)$, $i < m \leqslant l$. Also, $B_l(r) = A_l(r)$. Without loss of generality, let us assume $j < k$. If $B_j(r) \cap B_k(r) \neq$ null, then there exists at least one word (say W) which belongs to both $B_j(r)$ and $B_k(r)$. Since $B_j(r)$ is a subset of $A_j(r)$, and $B_k(r)$ is a subset of $A_k(r)$, it follows that $W \in A_j(r)$ and $W \in A_k(r)$. But this means that if $W \in B_j(r)$, then $W \notin B_k(r)$. This is a contradiction. Therefore, $B_j(r) \cap \text{B}_\text{k}(\text{r}) = \text{null}$. □

Lemma 2

$$\bigcup_{i=1}^{l} B_i(r) = \bigcup_{i=1}^{l} A_i(r).$$

Proof. From Definitions 5 and 6 it is clear that any word $V \in \{A_i(r) - B_i(r)\}$, $i = 1, 2, \ldots, l$, belongs to $B_j(r)$, where j is the least integer $> i$, such that V does not belong to any $A_k(r)$, $j < k \leqslant l + 1$. This, coupled with the fact that $B_i(r)$ is a subset of $A_i(r)$, implies that every word that belongs to any $A_i(r)$, $i = 1, 2, \ldots, l$, also belongs to some $B_m(r)$, $m = 1, 2, \ldots, l$. But, since $B_m(r)$ is a subset of $A_m(r)$, it follows that □

$$\bigcup_{i=1}^{l} B_i(r) = \bigcup_{i=1}^{l} A_i(r).$$

Lemma 3

Let $W_1 \in B_i(r)$, $i = 1, 2, \ldots, l-1$, and $W_2 \in B_j(r)$, $j = 2, 3, \ldots, l$, $j > i$. W_2 can never cover W_1.

Proof. Since $B_i(r)$ is a subset of $A_i(r)$ and $B_j(r)$ is a subset of $A_j(r)$, $W_1 \in A_i(r)$ and $W_2 \in A_j(r)$. If W_2 covers W_1 then W_1 must belong to $A_j(r)$ as well. Then, since W_1 belongs to both $A_i(r)$ and $A_j(r)$ and $j > i$, W_1 does not belong to $B_i(r)$. This is a contradiction. Hence, W_2 cannot cover W_1. □

C. Encoding Technique

Our code requires the number of checkbits r to be equal to $\lfloor \log_2 k \rfloor$. Note that a Berger code, which detects all unidirectional errors, requires $\lceil \log_2(k+1) \rceil$ checkbits. The price we pay for reducing the number of checkbits is that we can no longer detect all unidirectional errors. We assume that $k - 2^r < M(r, \lfloor r/2 \rfloor)$. We will see later that this assumption does not create any problems. Procedure 1 below gives our encoding technique.

Table 1. Check symbol assignment

Group No.	Check symbol
18	1111
17	1110
16	1101
15	1100
14	1011
13	1010
12	1001
11	0111
10	0110
9	0101
8	0011
7	1100
6	0100
5	1010
4	0010
3	1001
2	1000
1	0001
0	0000

Procedure 1

(1) Let $r = \lfloor \log_2 k \rfloor$. Set $l = k - 2^r + 1$.
(2) Obtain $S(r)$ and then get $C_i(r)$, $i = 1, 2, \ldots, l$.
(3) Obtain $A_i(r)$ and $B_i(r)$, $i = 1, 2, \ldots, l$.
(4) Let X be the set of r-bit words arranged in the order of decreasing value. Perform COMP: $X \rightarrow Y$.
(5) Perform APP: $[Y, C_1(r), B_1(r), C_2(r), B_2(r), \ldots, C_l(r), B_l(r)] \rightarrow Z$. Z constitutes a sequence of check symbols which are to be assigned successively to words with group numbers $k, k-1, \ldots, 0$.
(6) Find $p_1[C_i(r)]$ and $p_2[C_i(r)]$ for each $C_i(r) \in Z$, $i = 1, 2, \ldots, l$, and get

$$t = \min_i \{p_2[C_i(r)] - p_1[C_i(r)]\} - 1. \qquad \square$$

Procedure 1 has been implemented in a Pascal program in order to derive the check symbols and the value of t. The following example illustrates how Procedure 1 works. Let $k = 18$. Hence, $r = \lfloor \log_2 18 \rfloor = 4$ and $l = 3$. S(4), $C_i(4)$, $A_i(4)$, $B_i(4)$ for $i = 1, 2, 3$, and COMP: $X \rightarrow Y$ have been found in earlier examples. Now APP: $[Y, C_1(4), B_1(4), C_2(4), B_2(4), C_3(r), B_3(4)] \rightarrow \{1111, 1110, 1101, 1100, 1011, 1010, 1001, 0111, 0110, 0101, 0011, 1100, 0100, 1010, 0010, 1001, 1000, 0001, 0000\}$.

The 19 check symbols thus obtained are consecutively assigned to words with group numbers 18, 17, ..., 0, as listed in Table 1.

So if the information symbol has 18 zeros then its check symbol is 1111; if it has 17 zeros then its check symbol is 1110 and so on.

D. Proof of Validity of our Code

The validity of the above code is considered in the following theorem:

Theorem 4

The code derived by Procedure 1 is capable of detecting t or fewer unidirectional errors.

Proof. Let k be the number of bits in the information symbols that have to be encoded. Hence, $r = \lfloor \log_2 k \rfloor$ and $l = k - 2^r + 1$. Let $C_i(r)$, $A_i(r)$, $B_i(r)$, $i = 1, 2, \ldots, l$, be obtained according to Definitions 4–6, respectively.

In Procedure 1, we defined X to be a set of r-bit words arranged in the order of decreasing value. By doing COMP: $X \rightarrow Y$ we threw out all those words from X that are covered by any $C_i(r)$. In other words, Y does not contain any word from any $A_i(r)$. Hence, from Lemma 2, it follows that Y also does not contain any word from any $B_i(r)$.

From the above arguments and Lemma 1, we can deduce that $C_i(r)$'s are the only words which appear twice in the sequence Z in Step 5 of Procedure 1. It is also clear that the words that were

deleted from X are added back in Z, although at different locations. Since each $C_i(r)$ appears exactly twice in Z, and $|X| = 2^r$, it means that $|Z| = 2^r + l = k + 1$. This is exactly the number of check symbols required for encoding words of length k. Another thing to note from Definitions 4 and 6 is that any word from $B_i(r)$, $i = 1, 2, \ldots, l-1$, is not covered by any $C_j(r)$, $j = i+1$, $i+2, \ldots, l$.

When a check symbol is assigned to an information symbol a codeword is formed. All such codewords from the code space. Henceforth, we will refer to the information symbol and check symbol of a codeword D as E and F, respectively.

Any two codewords, whose information symbols have the same group number, have the same check symbol. But if the group number of the two information symbols is the same then they will be unordered, and, hence the two codewords will be unordered. So let us take any two codewords D_1 and D_2 such that $\mathrm{Gn}(E_1) \neq \mathrm{Gn}(E_2)$. Without loss of generality let us assume that $\mathrm{Gn}(E_1) > \mathrm{Gn}(E_2)$. This means that F_1 will occur before F_2 in Z. Two cases may arise.

Case 1: F_1 and F_2 are not the same. For this case there are seven subcases. For each of these subcases we will show that $N(F_1, F_2) \geqslant 1$. Since $\mathrm{Gn}(E_1) > \mathrm{Gn}(E_2)$, $N(E_2, E_1) \geqslant 1$. Hence

$$N(D_1, D_2) = N(E_1, E_2) + N(F_1, F_2) \geqslant 1$$

and

$$N(D_2, D_1) = N(E_2, E_1) + N(F_2, F_1) \geqslant 1.$$

Therefore, by proving $N(F_1, F_2) \geqslant 1$, we prove that D_1 and D_2 are unordered.

(1) $F_1, F_2 \in Y$.
Since $\mathrm{Gn}(E_1) > \mathrm{Gn}(E_2)$, the value of $F_1 >$ the value of F_2. Hence, $N(F_1, F_2) \geqslant 1$.

(2) $F_1 \in Y$, $F_2 = C_i(r)$, $i = 1, 2, \ldots, l$.
Since any word that F_2 covers cannot be in Y (from Definition 7), F_2 cannot cover F_1. This implies that either F_1 covers F_2, or F_1 and F_2 are unordered. Hence, $N(F_1, F_2) \geqslant 1$.

(3) $F_1 \in Y$, $F_2 \in B_i(r)$, $i = 1, 2, \ldots, l$.
F_2 cannot cover F_1 because a word in Y has weight $\lfloor r/2 \rfloor$ or higher, while a word in $B_i(r)$ can only have weight $\lfloor r/2 \rfloor - 1$ or lower. Hence, $N(F_1, F_2) \geqslant 1$.

(4) $F_1 = C_i(r)$, $i = 1, 2, \ldots, l$, $F_2 = C_j(r)$, $j = 2, 3, \ldots, l$, $i < j$.
Since the weight of $F_2 \leqslant$ the weight of F_1, either F_1 and F_2 are unordered, or F_1 covers F_2. In other words, $N(F_1, F_2) \geqslant 1$.

(5) $F_1 = C_i(r)$, $i = 1, 2, \ldots, l$, $F_2 \in B_j(r)$, $j = 1, 2, \ldots, l$.
If $i = j$, F_1 will cover F_2. If $i \neq j$, then $j > i$, from Step 5 of Procedure 1. In that case F_1 and F_2 are unordered, or F_1 covers F_2. Therefore, $N(F_1, F_2) \geqslant 1$.

(6) $F_1 \in B_i(r)$, $i = 1, 2, \ldots, l-1$, $F_2 = C_j(r)$, $j = 2, 3, \ldots, l$.
Note that $j > i$. From Definitions 4 and 6, we can see that F_2 cannot cover F_1. So, $N(F_1, F_2) \geqslant 1$.

(7) $F_1 \in B_i(r)$, $i = 1, 2, \ldots, l$, $F_2 \in B_j(r)$, $j = 1, 2, \ldots, l$.
If $i = j$, the value of $F_1 >$ the value of F_2. Hence, $N(F_1, F_2) \geqslant 1$. If $j > i$, then from Lemma 3, F_2 does not cover F_1. Hence, again, $N(F_1, F_2) \geqslant 1$.

Therefore, by taking into account all these subcases of Case 1, which exhaust all the possibilities, we find that D_1 and D_2 will be unordered.

Case 2: F_1 and F_2 are the same. This case can only arise when F_1, $F_2 = C_i(r)$, $i = 1, 2, \ldots, l$. From Step 6 of Procedure 1 we know that $d(D_1, D_2)$ is at least $t + 1$.

From the arguments given for Cases 1 and 2, and from Theorem 3, we see that the code derived by Procedure 1 is a t-UED code. □

E. A Note on the Bose–Lin Code

A method is given in Ref. [10] in which the r checkbits of a check symbol are divided into two parts of 4 bits and $r-4$ bits, respectively. The first 4 bits take any one of the 2-out-of-4 codewords, namely, 0011, 0101, 0110, 1001, 1010 or 1100, and the last $r-4$ bits take any one among the 2^{r-4} binary $(r-4)$-tuples. Therefore, there are $6 \times 2^{r-4}$ distinct check symbols in this code. It was shown

Table 2. Unidirectional errors detected by the Bose–Lin code and our code

r	e_1	k	e_2	K	r	e_1	k	e_2	K
		32	21				512	465	
		33	18				525	383	
5	11	35	16	1.28	9	165	550	310	1.31
		38	12				600	229	
		40	11				640	197	
		41	11				672	166	
		64	49				1024	961	
		70	35				1100	596	
6	22	75	28	1.42	10	326	1200	462	1.45
		85	26				1350	407	
		90	23				1425	342	
		91	22				1485	327	
		128	105				2048	1949	
		132	89				2200	1250	
7	43	140	72	1.31	11	647	2400	869	1.30
		150	56				2500	848	
		160	49				2600	737	
		168	44				2664	647	
		256	225				4096	3969	
		280	144				4400	2370	
8	84	300	113	1.42	12	1288	4800	1872	1.42
		331	104				5300	1581	
		360	92				5820	1303	
		364	84				5827	1289	

that this code is capable of detecting $5 \times 2^{r-4} + r - 4$ unidirectional errors. It was conjectured that this code is optimal or near-optimal.

It can be seen that if the r checkbits are divided into two parts of x bits and $r - x$ bits, respectively, then the number of unidirectional errors t detected by a similar code is given by

$$t = \left[\binom{x}{\lfloor x/2 \rfloor} - 1\right] \cdot 2^{r-x} + r - x.$$

This reduces to $5 \times 2^{r-4} + r - 4$ for $x = 4$. It can be verified by graphical means, or otherwise, that t is maximized for $x = 4$. So, indeed, of all the choices available for breaking up the r checkbits into two parts, Bose and Lin chose the best one, although they did not prove this fact in their paper.

However, the Bose–Lin code is not optimal or near-optimal for every value of k. In the next section we will show that our code performs better than the Bose–Lin code when k lies between 2^r to $K \times 2^r$, where K is a constant which is roughly equal to 1.3 for odd r and 1.4 for even r.

5. ERROR-DETECTING CAPABILITY OF OUR CODE

We present in Table 2 a comparison of the error-detecting capability of the Bose–Lin code [10] with our code. For the Bose–Lin code we will denote the number of unidirectional errors detected by e_1. For our code the number of unidirectional errors detected also depends on the number of information bits k for a given r. We will denote this number as e_2. K will denote the ratio $k_{\max}/2^r$, where $k_{\max}$ is the number of information bits for or less than which our code performs as well or better than the Bose–Lin code. The numbers in the table were obtained by implementing Procedure 1 in a program written in Pascal. A program was needed to get e_2 since we have not been able to derive an exact formula for e_2 in terms of k and r.

From the above table we can see that our code performs better than the Bose–Lin code in the range 2^r to $K \times 2^r$, where K is roughly equal to 1.3 for odd r and 1.4 for even r. In this range the number of unidirectional errors detected (e_2) decreases with increasing k. Let us suppose we want to encode 64 information bits in a systematic code. By choosing $r = 6$, we can detect 49 unidirectional errors by our code, whereas the Bose–Lin code can detect only 22 unidirectional errors.

Figure 2 presents a plot of $\log_{10} e_2$ vs $\log_{10} k$ for different values of r.

It is our conjecture that the empirical values of K derived above are valid $\forall\, r$. In any case, the maximum value of k that we have considered in Table 2 is 5827. Most practical applications require the encoding of much fewer information bits.

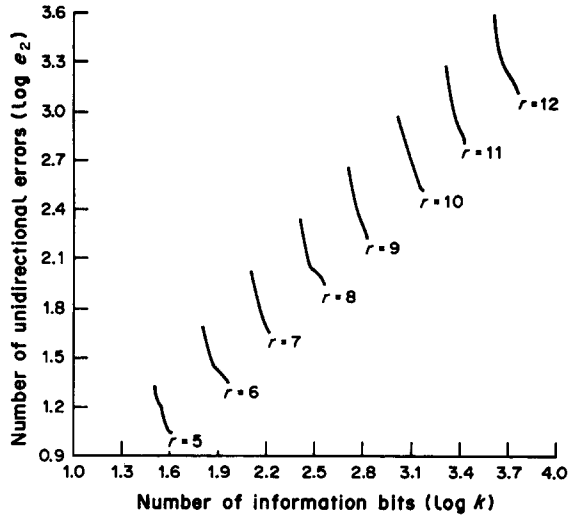

Fig. 2. $\log_{10} e_2$ vs $\log_{10} k$ for different r.

6. A CHECKER FOR OUR CODE

The concept of totally self-checking (TSC) circuits was introduced in Refs [15, 16] for functional circuits as well as checkers. These circuits can detect errors on-line. The checker is used to monitor the outputs of the functional circuit to catch non-codewords. The TSC checker concept was extended to strongly code disjoint (SCD) checkers in Ref. [17]. It was shown that SCD checkers are the largest class of checkers which meet the TSC goal. These checkers have the capability of performing properly in spite of the presence of undetectable faults, unlike TSC checkers. In Ref. [18] we have presented an SCD checker design applicable to any systematic code. If the outputs of the functional circuit are encoded using the systematic code presented in this paper, then we can use this design to obtain an SCD checker.

7. CONCLUSION

The issue of reliability is becoming increasingly important for VLSI circuits. Therefore, it is essential to find efficient codes to encode the data words. For VLSI, unidirectional errors have been found to be the most common type of errors. It may not always be necessary to detect unidirectional errors of all sizes. In this paper we have presented a new systematic code to detect t-unidirectional errors. This code performs better than the previously known codes for certain ranges of k, the number of information bits. Since an SCD checker design is known for this code, its applicability to real-life circuits is high.

Acknowledgement—This work was supported in part by the National Science Foundation under Grant MIP-8708728.

REFERENCES

1. W. W. Peterson and E. J. Weldon, *Error Correcting Codes*. MIT Press, Cambridge, Mass. (1972).
2. E. R. Berkelamp, *Algebraic Coding Theory*. McGraw-Hill, New York (1968).
3. S. Lin, *An Introduction to Error Correcting Codes*. Prentice-Hall, Englewood Cliffs, N.J. (1970).
4. N. J. A. Sloane and F. J. MacWilliams, *The Theory of Error Correcting Codes*. North-Holland Amsterdam, The Netherlands (1977).
5. R. W. Cook *et al.*, Design of self-checking microprogram control. *IEEE Trans. Comput.* **C-22**, 255–262 (1973).
6. D. K. Pradhan and J. J. Stiffler, Error correcting codes and self-checking circuits in fault-tolerant computers. *IEEE Comput.* **13**, 27–37 (1980).
7. J. M. Berger, A note on error detecting codes for asymmetric channels. *Inf. Control* **4**, 68–73 (1961).
8. J. E. Smith, On separable unordered codes. *IEEE Trans. Comput.* **C-33**, 741–743 (1984).
9. C. V. Frieman, Optimal error detecting codes for completely asymmetric binary channels. *Inf. Control* **5**, 64–71 (1962).
10. B. Bose and D. J. Lin, Systematic unidirectional error-detecting codes. *IEEE Trans. Comput.* **C-34**, 1026–1032 (1985).
11. D. Dong, Modified Berger codes for detection of unidirectional errors. In *Dig. Papers, 12th Int. Symp. Fault-Tolerant Comput.*, pp. 317–320 (1982).

12. J. M. Borden, Optimal asymmetric error detecting codes. *Inf. Control* **53,** 66–73 (1982).
13. R. W. Hamming, Error detecting and error correcting codes. *Bell Syst. tech. J.* **29,** 147–160 (1950).
14. B. Bose and T. R. N. Rao, Theory of unidirectional error correcting/detecting codes. *IEEE Trans. Comput.* **C-31,** 521–530 (1982).
15. W. C. Carter and P. R. Schneider, Design of dynamically checked computers. In *Proc. IFIP '68, Edinburgh, Scotland,* Vol. 2, pp. 878–883 (1968).
16. D. A. Anderson and G. Metze, Design of totally self-checking circuits for m-out-of-n coes. *IEEE Trans. Comput.* **C-22,** 263–269 (1973).
17. M. Nicolaidis, I. Jansch and B. Courtois, Strongly code disjoint checkers. In Proc. *14th Int. Symp. Fault-tolerant Comput.*, Orlando, Fla, pp. 16–21 (1984).
18. N. K. Jha, Strongly code disjoint checkers for systematic and separable codes. Submitted to *18th Int. Symp. Fault-Tolerant Comput.*, Tokyo (1988).