sends $ID(u)$ to all adjacent nodes. Thus, all the nodes will know the ID of the leader.

**MESSAGE COMPLEXITY**

The following lemma specifies the maximum number of live nodes that reach phase $i$.

**Lemma 10:** Let $i$ be an integer such that $2 \leq i \leq \lceil (n + 2)/2(r - 1)f \rceil + 1$. For every $i$, there are at most $\lceil n/(i - 1) - 1 \rceil f$ nodes $u$ such that $u$ is live when $u$ sets phase$(u)$ to $i$.

**Proof:** Suppose that there are at most $k$ nodes $u$ such that $u$ is live when $u$ sets phase$(u)$ to $i$. Trivially, $k \leq n$. Let $u_j$ denote the $j$th live node to set its phase to $i$, for every $1 \leq j \leq k$. By induction on $i$, we can show that $u_j$ must have eliminated at least $(i - 1) - (i - 1)f$ nodes to reach phase $i$. By Lemma 4, Victims$(u_j, i) \cap$ Victims$(u_i, i) = \emptyset$ for every $1 \leq j \neq p \leq k$. Hence, $k \leq \lceil n/(i - 1)f \rceil - 1$.

**Theorem 2:** The algorithm uses $O(nrf + (nr/(r - 1)f) \log (nr/(r - 1)f))$ messages in the worst case, where $n$ is the number of nodes in the network, and $f$ is the maximum number of faulty links.

**Proof:** Let $i \in \lceil (n + 2)/2(r - 1)f \rceil + 1$ be an integer. If a node $u$ is live when it sets phase$(u)$ to $i$ at some time $t$, then $u$ sends $(r - 1)f$ new messages during phase $i$. By Lemma 8, $u$ sends at most $f$ refresher messages during phase $i$. Hence, $u$ sends at most $rf$ eliminate-messages during phase $i$. Each eliminate-message generates at most three additional messages as follows.

1) Suppose that $u$ sends an eliminate-message to a node $v$. Node $v$ sends "Potential-Suppressor$(i, U, ID(u))"$ on Suppressor-Link$(u)$.
2) Node $v$ sends "Potential-Suppressor-Successful$(i, ID(u))"$ or "Potential-Suppressor-Unsuccessful$(i, ID(u))"$ on Suppressor-Link$(u)$.
3) Finally, $u$ sends a reply to $v$.

Thus, $u$ causes at most $4rf$ messages to be generated while phase$(u)$ is $i$. The total number of messages the algorithm uses is the sum of the number of messages generated to elect a leader plus the number of messages used to inform all the nodes of the ID of the leader. By Lemma 10, the number NUM of the messages generated to elect a leader is as follows:

$$\sum_{i=1}^{\lceil (n + 2)/2(r - 1)f \rceil} (\text{num. live nodes } u \text{ that reach phase } i)$$

$$= \sum_{i=2}^{\lceil (n + 2)/2(r - 1)f \rceil} \lceil n/(i - 1) - 1 \rceil f \cdot 4rf + 4nrf$$

$$\leq 4nr \sum_{i=1}^{\lceil (n + 2)/2(r - 1)f \rceil} \frac{1}{i} + 4nr \cdot \frac{1}{r - 1}$$

$$\text{NUM} = O\left(\frac{nr}{r - 1} \log \left(\frac{n}{(r - 1)f}\right) + nrf\right).$$

The algorithm uses $(2f + 1)n = O(nrf)$ messages to inform all the nodes of the ID of the leader.

A detailed analysis, omitted here, shows that the value of $r$ that minimizes the total number of messages is $r = \min \{1 + C \ln(n + 2)/f) + (n - 4)/2f, C = 1 + O(\log f/\log n)\}$. For every value of $n$ and $f$ subject to $f \leq \lceil \sqrt{2n}/2 \rceil$, $7/8 \leq C \leq 1$. Thus, the minimum number of messages that our algorithm uses is $O(nrf + \log n)$ messages.

**TIME COMPLEXITY**

**Theorem 3:** The algorithm takes at most $O(n/(r - 1)f)$ time units to complete, where $n$ is the number of nodes, and $f$ is the maximum number of faulty links.

**Proof:** The maximum running time of the algorithm is the maximum time spent to elect a leader in any execution. Consider any execution $E$, and let $u$ be the leader when $E$ terminates. Assume the delay on each reliable link to be at most one time unit. Let $M$ be any eliminate-message that $u$ sends. As in the proof of Theorem 2, $M$ generates at most three additional messages. Each message reaches its destination in at most one time unit. Hence, $u$ receives a reply for $M$ in at most four time units. Thus, by induction on $i$, $u$ spends at most $4(i - 1) - (r - 1)f$ time units to elect itself as the leader. By the algorithm, all the nodes will know the ID of the leader in at most two more time units. Since all the nodes start executing the algorithm simultaneously, the algorithm will terminate in at most $4(i + 2)/2(r - 1)f + 2 = O(nrf)$ time units.

**ACKNOWLEDGMENT**

I wish to thank M. C. Loui for his critique of my paper.

**REFERENCES**

new codes can detect longer bursts than previously known codes. Encoding and decoding procedures are indicated.

Index Terms—Bursts, decoder, encoder, information bits, redundancy, systematic code, unidirectional errors.

I. INTRODUCTION

Consider a binary block code, i.e., each codeword has a certain length \( n \). We say that a codeword has suffered a unidirectional error when all errors are either transitions 0\( \rightarrow 1 \) or 1\( \rightarrow 0 \) but not both of them at the same time. It has been reported that in some VLSI circuits the errors are of unidirectional type \([1],[10],[14]\). The theoretical and practical aspects of unidirectional error correcting/detecting codes have aroused considerable interest in recent literature \([2],[4]-[9],[11]-[16]\).

In certain semiconductor computer memory architectures, the unidirectional errors tend to be confined in a burst, i.e., a cluster of adjacent bits up to a certain length is affected (assuming that the rightmost bit is adjacent to the leftmost bit). In the rest of this paper, a burst of unidirectional errors will be called a burst. In applications, it is important to implement codes that are capable of detecting bursts of a certain length using a minimum of redundancy. Conversely, given a certain number of redundant bits, we want to detect bursts as long as possible. We want the codes to have very easy encoding and decoding procedures. To that end, they have to be systematic (i.e., if \( ? \) denotes any burst of length up to \( 2' \). In Section IV, encoding and decoding procedures are given.

II. GENERAL CONSTRUCTION AND COMBINATORIAL PROPERTIES

In this section, we present a general method to construct a burst detecting code. In the next one, we give a specific construction.

Let \( r \approx 1 \). Denote by \( \mathcal{Z}_r \) the integers mod \( 2' \). Let \( f: \mathcal{Z}_r \rightarrow \{0, 1\}^r \) be \( 1 \cdot 1 \). From now on, \( \{0, 1\}^r \) is considered as a directed cycle with the cyclic structure induced by \( f \), i.e.,

\[
\begin{align*}
\text{wt}(f(i+j \mod 2')) = \text{wt}(f(i)) & \leq 1, \quad i \in \mathcal{Z}_r. 
\end{align*}
\]

We say that a binary \( m \)-tuple \( \mathbf{x} = (x_1, x_2, \ldots, x_m) \) covers another binary \( m \)-tuple \( \mathbf{y} = (y_1, y_2, \ldots, y_m) \) if \( x_j = 1 \) whenever \( y_j = 1 \) (notation: \( \mathbf{x} \supseteq \mathbf{y} \)). We call the weight of \( \mathbf{x} = (x_1, x_2, \ldots, x_m) \) the number of 1's in \( \mathbf{x} \), i.e., \( \text{wt}(\mathbf{x}) = \sum_{j=1}^{m} x_j \).

Let \( i \in \mathcal{Z}_r \). We define

\[
\begin{align*}
\text{c}(i; f; r) & = \min \{ j : 1 \leq j \leq 2', f(i) \neq f(i+j \mod 2'), \\
\text{wt}(f(i+j \mod 2')) & = \text{wt}(f(i)) \leq 1 \} \quad (2.2)
\end{align*}
\]

and

\[
\begin{align*}
\text{c}(f; r) & = \min \text{c}(i; f; r). 
\end{align*}
\]

Example 2.1: Let \( r = 3 \) and \( f: \mathcal{Z}_3 \rightarrow \{0, 1\}^3 \) given by Table I. In the third column, we write the values of \( \text{c}(i; f; 3) \) according to definition (2.2). Taking the minimum on the values of this column, we obtain \( \text{c}(f; 3) \), which in this case is 4.

For simplicity, call \( c = c(f; r) \). The bits in a codeword \( \overrightarrow{v} \) are arranged as follows:

\[
\overrightarrow{v} = a_1 a_2 \cdots a_r \cdots a_{2r-1} d_1 d_2 \cdots d_{2r-1} \equiv (a_1 \oplus d_1) \oplus (a_2 \oplus d_2) \oplus \cdots \oplus (a_{2r-1} \oplus d_{2r-1}) \oplus a_{2r} = b.
\]

We denote \( C(f; r) \) the code obtained by this construction.

Example 2.2: Let \( k = 9 \). Consider \( C(f; 3) \) when \( f \) is the function defined by Table I. Some codewords are listed below.

<table>
<thead>
<tr>
<th>( t )</th>
<th>( \text{c}(f; t) )</th>
<th>( \text{c}(f; t)\text{ mod } 2' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 1 1</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>0 1 1</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>1 0 1</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>1 1 0</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>0 0 1</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>1 0 0</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>0 0 0</td>
<td>2</td>
</tr>
</tbody>
</table>

The first two information vectors have weight 5. According to Table I and (2.4), the parity vector is \( 0 \cdot 0 \). Similarly, the third information vector has weight 7, so the parity vector is \( 0 \cdot 0 \cdot 0 \). The parity and information bits are distributed according to (2.5).

We can now prove our main result.

Theorem 2.4: The code \( C(f; r) \) can detect any burst of length up to \( c(f; r) \).

Proof: Assume a burst of length \( l \), \( 1 \leq l \leq c(f; r) \). Without loss, the burst may be considered of type \( 0 \rightarrow 1 \), so the weight of a codeword increases \( l' \), where \( 1 \leq l' \leq l \). According to (2.5), a burst of length up to \( c(f; r) \) either does not affect the parity bits or affects exactly one parity bit. We treat the two cases separately.

Case I: No parity bits are affected. Let \( \overrightarrow{b} \) be the transmitted codeword, where \( \overrightarrow{u} \) denotes the information bits and \( \overrightarrow{b} \) denotes the parity bits. Since no parity bits are affected, the received word has the form \( \overrightarrow{u} \overrightarrow{b} \overrightarrow{u} \overrightarrow{b} \overrightarrow{u} \) being a corrupted version of \( \overrightarrow{u} \). The burst will not be detected only if \( \overrightarrow{b} \overrightarrow{b} \) is a codeword in \( C(f; r) \). If this is the case, by (2.4),

\[
\overrightarrow{b} = f(\text{wt}(\overrightarrow{u} \overrightarrow{b} \overrightarrow{u}) \mod 2'),
\]

but also

\[
\overrightarrow{b} = f(\text{wt}(\overrightarrow{u} \overrightarrow{b} \overrightarrow{u} \overrightarrow{b} \overrightarrow{u}) \mod 2'),
\]

Since \( f = 1 \cdot 1, \) we must have \( l' = 0 \mod 2' \), so \( l' \neq 2' \), a contradiction (notice that \( c(f; r) < 2' \).)

Case II: Exactly one parity bit is affected.
Let $\tilde{\mathbf{b}}$ be the transmitted codeword and $\tilde{\mathbf{b}}^*$ be the received word. Now we have $\text{wt}(\tilde{\mathbf{b}}') = \text{wt}(\tilde{\mathbf{b}}) + l' - 1$ and $\text{wt}(\tilde{\mathbf{b}}^*) = \text{wt}(\tilde{\mathbf{b}}) + 1$. As before, assume that the burst is undetected so $\tilde{\mathbf{b}}^*$ is a codeword. By (2.4), we have

$$\tilde{\mathbf{b}}^* = f(\text{wt}(\tilde{\mathbf{b}}^*) \mod 2^r)$$

(2.6)

$$= f(\text{wt}(\tilde{\mathbf{b}}) + l' - 1 \mod 2^r)$$

Also $\tilde{\mathbf{b}} = f(\text{wt}(\tilde{\mathbf{b}}) \mod 2^r)$. Since $\text{wt}(\tilde{\mathbf{b}}') = 1$ and $\tilde{\mathbf{b}} \leq \tilde{\mathbf{b}}^*$, (2.2) and (2.6) imply

$$l' - 1 \leq c(\text{wt}(\tilde{\mathbf{b}}), f; r) \geq c(f; r)$$

Since $l' \leq c(f; r)$, this is a contradiction. \qed

Example 2.5: Consider Tables I-IV. When $r = 3$ (Table I), according to Theorem 2.4, the code obtained from Construction 2.2 can detect bursts of length up to 4. This ties the performance of the code described in [5]. When $r = 4$ (Table II), however, the code $C(f; 4)$ can detect any burst of length up to 9 while the code in [5] can detect any burst of length up to 8.

The difference is more dramatic in Tables III and IV. $C(f; 5)$ and $C(f; 6)$ have burst-detecting capability 19 and 41, respectively, while the codes in [5], for the same redundancy, have burst-detecting capability 16 and 32, respectively.

Of course, the functions $f$ described in Tables I-IV have not been taken at random (there is a slight abuse of notation by calling $f$ these four functions, but this should not lead to confusion). We describe them in the next section together with an estimate for $c(f; r)$.

III. A PARTICULAR CONSTRUCTION

In this section, we generalize the functions $f$ given in Tables I-IV. Given a binary r-tuple $\tilde{\mathbf{b}} = (b_1, b_2, \cdots, b_r)$, we call the support of $\tilde{\mathbf{b}}$ (denoted $\text{supp}(\tilde{\mathbf{b}})$) the set of coordinates where $\tilde{\mathbf{b}}$ is nonzero, i.e.,

$$\text{supp}(\tilde{\mathbf{b}}) = \{k : 0 \leq k \leq r - 1, \ b_k = 1\}$$

Clearly $|\text{supp}(\tilde{\mathbf{b}})| = \text{wt}(\tilde{\mathbf{b}})$.

Assume $\text{supp}(\tilde{\mathbf{b}}) = \{(j_1, j_2, \cdots, j_l) \subseteq \{0, 1, \cdots, r - 1\}, j_l \leq r - 1$. Since a binary r-tuple is uniquely determined by its support, we denote $\tilde{\mathbf{b}} = [j_1, j_2, \cdots, j_l]$ (if $\tilde{\mathbf{b}} = 0$, we consider $l = 0$).

Define the following 1-1 function

$$f : \mathbb{Z}_r \rightarrow \{0, 1\}^l$$

$$[j_1, j_2, \cdots, j_l] \mapsto \sum_{k=1}^{l} \binom{r}{k} \sum_{j=1}^{k} \binom{j}{k}$$

(3.1)

Several remarks have to be made regarding the definition of $f$. We are assuming the following conventions: $\binom{n}{0} = 0$ when $n > m$ and $\sum_{j=1}^{k} \binom{j}{k} = 0$.

It is not difficult to prove that $f$ is 1-1, but we omit the proof. The
reader can verify that the function $f$ defined by (3.1) corresponds to the function defined in Tables I–IV when $r = 3, 4, 5$ and 6, respectively.

Another way of describing the function $f$ is as follows: observe that in Tables I–IV we are ordering the $2^r$-binary $r$-tuples in a certain way. First we write the only vector of weight $r$; then the $r$ vectors of weight $r - 1$ in increasing order when considered as binary numbers; then the $(r - 1)$ vectors of weight $r - 2$. In general, if we have written the $(r - k)$ vectors of weight $k$, then we write the $(r - k - 1)$ vectors of weight $k - 1$ in increasing order as binary numbers. The last vector is the all zero vector.

Given an $r$-tuple, say $(i)$, we want to find the $r$-tuple $f(i + j \text{mod } 2^r)$, $0 \leq j \leq 2^r$ such that $f(i) \leq f(i + j \text{mod } 2^r)$, $w(f(i + j \text{mod } 2^r)) - wt(f(i)) \leq 1$ and $j$ is minimum with these properties [see (2.2)]. If $i = 0$, $f(0) = (1 \cdots 1)$, $j = 2^r$ and $f(0 + j) = f(j)$. If $i = 2^r - 1$, $f(2^r - 1) = (00 \cdots 0)$, and $f(2^r - 1 + j) = (00 \cdots 0 1)$.

In general, if $f(i) = (j_1, j_2, \ldots, j_k)$, $1 \leq k \leq r - 1$, we want to find the first $r$-tuple $f(i + j)$ of weight $k + 1$ covering $f(i)$. Let $l$ be the first index such that $j_i > l - 1$ (i.e., $j_l = \alpha - 1$ for $1 \leq \alpha < l$). Then

$$f(i) = [0, 1, \ldots, l - 2, j_l, j_{l+1}, \ldots, j_k]$$

and

$$f(i + j \text{mod } 2^r) = [0, 1, \ldots, l - 2, l - 1, j_l, j_{l+1}, \ldots, j_k].$$

Applying (3.1) to (3.2) and (3.3), we obtain

$$j = 2^r - \left( \frac{r}{k + 1} \right) + \sum_{\alpha = 0}^{k} \frac{j_\alpha}{\alpha + 1} - \frac{j_\alpha}{\alpha}.$$ 

(3.6)

Since this $j$ is the minimum with this property, $j = c(i, f; r)$. Also, the sum in (3.6) can start at $\alpha = 1$. We summarize our results in the following lemma.

Lemma 3.3: Consider $f: \mathbb{Z}_{2^r} \rightarrow \{0, 1\}^r$ defined by (3.1) and $c(i, f; r)$ defined by (2.2). Then if $f(i) = (j_1, j_2, \ldots, j_k)$,

$$c(i, f; r) = 2^r - \left( \frac{r}{k + 1} \right) + \sum_{\alpha = 0}^{k} \frac{j_\alpha}{\alpha + 1} - \frac{j_\alpha}{\alpha}.$$ 

(3.7)

Example 3.2: Consider Table IV. Here, $r = 6$. Since $f(63) = (0 0 0 0 0 1)$, $k = 0$, and according to (3.7), $c(63, f; 6) = 2^6 - 6 = 58$, confirming the result in the table. Also $f(0) = (1 1 1 1 1 1) = [0, 1, 2, 3, 4, 5]$. Here, $k = 6$ so (3.7) gives

$$c(0, f; 6) = 2^6 - \left( \frac{6}{7} \right) + \sum_{\alpha = 0}^{5} \frac{\alpha - 1}{\alpha + 1} - \frac{\alpha - 1}{\alpha} = 54.$$ 

For a less pathological example, consider $f(26) = (0 1 0 0 0 1)$. The first $r$-tuple (as a binary number) of weight 4 covering $(0 0 0 0 0 1)$ is $(0 1 0 1 1 1) = f(6)$. Hence, $c(26, f; 6) = 64 + 8 - 26 = 46$. We also have $f(26) = [0, 1, 4, 5]$. Applying (3.7), we obtain

$$c(26, f; 6) = 64 - \left( \frac{6}{4} + \frac{0}{2} + \frac{0}{1} + \frac{1}{3} - \frac{1}{2} \right) + \left( \frac{4}{4} - \frac{4}{3} \right) = 64 - 15 + 1 - 4 = 46,$$

confirming formula (3.7).

We want now to find

$$c(f; r) = \min_i c(i, f; r).$$

From an observation of Tables I–IV, we conclude that this minimum is obtained at several values. However, it is enough to identify one of those values.

In Table I, the minimum is achieved at $(0 1 0)$, in Table II at $(0 1 0 0)$, in Table III at $(0 1 0 0 1 0)$, and in Table IV at $(0 1 0 0 1 0 0)$. This tends to suggest that in general, the minimum is achieved at $(i_x)$, where $x$ denotes the integer part of $i$.

$$\left( \cdots 1 0 1 0 \right) = \left[ 1, 3, 5, \cdots, 2 \left( \frac{r}{2} - 1 \right) \right].$$

For $f(i) = (\cdots 1 0 1 0)$, according to (3.7), we have

$$c(i, f; r) = 2^r - \left( \frac{r}{2} \right) + \sum_{\alpha = 0}^{\lceil r/2 \rceil} \left( \frac{2\alpha - 1}{\alpha + 1} \right) - \left( \frac{2\alpha - 1}{\alpha} \right).$$

Notice that for $r = 3, 4, 5$ and 6, this formula gives the values of $c(f; 3)$, $c(f; 4)$, $c(f; 5)$, and $c(f; 6)$, respectively. We want to prove the result in general.

Theorem 3.3: Consider $f: \mathbb{Z}_{2^r} \rightarrow \{0, 1\}^r$ defined by (3.1) and $c(f; r)$ defined by (2.3). Then

$$c(f; r) = 2^r - \left( \frac{r}{2} \right) + \sum_{\alpha = 0}^{\lceil r/2 \rceil} \left( \frac{2\alpha - 1}{\alpha + 1} \right) - \left( \frac{2\alpha - 1}{\alpha} \right).$$

(3.8)

The proof of (3.8) is given in [3]. The next lemma, also proved in [3], gives an easier expression for (3.8).

Lemma 3.4: Consider $f: \mathbb{Z}_{2^r} \rightarrow \{0, 1\}^r$ defined by (3.1) and $c(f; r)$ defined by (2.3), then

$$c(f; r) = 2^r - \sum_{\alpha = 0}^{\lceil r/2 \rceil} \left( \frac{\alpha}{\alpha + 1} \right).$$

(3.9)

In Table V, we compare $c(f; r)$ to $2^r - 1$, the burst detecting capability of the codes described in [5]. We have made $r$ range from 3
the r-tuple \( b_2 \), Similarly, if \( f(\text{wt}(B)) = B' \), the decoder decides that no errors have occurred. If \( f(\text{wt}(B)) \neq B' \), the decoder declares that a burst has occurred.

### Table VI

<table>
<thead>
<tr>
<th>( r )</th>
<th>( c(f; r) )</th>
<th>( \frac{c(r)}{2^r} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>.594</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>.641</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>.732</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>.784</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>.817</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>.892</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>.896</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>.904</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>.910</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>.920</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>.927</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>.935</td>
<td></td>
</tr>
<tr>
<td>170</td>
<td>.939</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>.943</td>
<td></td>
</tr>
<tr>
<td>210</td>
<td>.948</td>
<td></td>
</tr>
</tbody>
</table>

The decoder uses the same circuits as the encoder. A comparator is added at the end. Say that \( B \) was transmitted and \( B' \) is received. The comparator compares \( f(\text{wt}(B')) \) to \( B' \). If \( f(\text{wt}(B')) = B' \), the decoder decides that no errors have occurred. If \( f(\text{wt}(B')) \neq B' \), the decoder declares that a burst has occurred.

### V. Conclusions

We have presented a family of systematic codes \( C(f; r) \), where \( r \geq 3 \) is the number of redundant bits and \( f: \{0, 1\}^r \rightarrow \{0, 1\}^1 \) is 1-1. If the information bits are given by a vector \( B = (a_1, a_2, \ldots, a_t) \), the redundant bits \( B' = (b_{r-1}, b_r, \ldots, b_t) \) are obtained as \( B' = f(\text{wt}(B) \bmod 2^r) \). These redundant bits are then distributed in a particular way. We gave a specific family of functions \( f \). The burst detecting capability of a code \( C(f; r) \) is \( c(f; r) \), where an explicit formula for \( c(f; r) \) was given. If we call \( b(r) \) the maximum burst detecting capability of a systematic code with \( r \) redundant bits, it was known that \( 2^{r-1} \leq b(r) \leq 2^r - 1 \). We showed that \( 2^{r-1} < c(f; r) \leq b(r) \leq 2^r - 1 \) when \( r \geq 4 \) and that \( c(f; r) \) and \( 2^r \) are asymptotically equal. Finally, simple encoding and decoding procedures were indicated. Note that the codes normally used for burst error detection, like CRC, can only detect bursts of length up to \( r \), being the number of redundant bits. So, knowing that the errors are of unidirectional type allows the design of codes a lot more powerful for burst error detection.

### Acknowledgment

The author wants to thank T. Howell and P. Siegel for useful discussions. Lemma 3.4 was pointed out by T. Howell.

### References


