Unordered Error-Correcting Codes and their Applications

Mario Blaum and Jehoshua Bruck
IBM Research Division
Almaden Research Center
San Jose, CA 95120

Abstract

We give efficient constructions for error correcting unordered (ECU) codes, i.e., codes such that any pair of codewords are at a certain minimal distance apart and at the same time they are unordered. These codes are used for detecting a predetermined number of (symmetric) errors and for detecting all unidirectional errors. We also give an application in parallel asynchronous communications.

1 Introduction

Given a binary vector of length n, we say that the *support* of a vector is the set of non-zero coordinates. We say that two vectors are unordered if their supports are unordered as sets, i.e., none of them contains the other. For instance, let $\underline{u} = 11001$, $\underline{v} = 01100$ and $\underline{w} = 11000$. The supports of $\underline{u}$, $\underline{v}$ and $\underline{w}$ are respectively $\{1,2,5\}$, $\{2,3\}$ and $\{1,2\}$. We easily see that $\underline{u}$ and $\underline{v}$ and $\underline{v}$ and $\underline{w}$ are unordered, while $\underline{u}$ and $\underline{w}$ are not unordered. In fact, since the support of $\underline{w}$ is contained in the support of $\underline{v}$, we say that $\underline{w}$ is *contained* in $\underline{v}$.

We say that a code $\mathcal{C}$ is unordered if every pair of codewords in $\mathcal{C}$ is unordered. Unordered codes are important in several applications. Among them, let us point out protecting a WOM (write only memory) against hostile overwrites [9], parallel asynchronous communications [14, 2] (see also Section 4) and detection of asymmetric and unidirectional errors in computer memories [1].

A natural question is what is the maximal size of an unordered code of length n? This question was answered by Sperner [12] in 1928: the maximal size is $\binom{n}{\lceil n/2 \rceil}$ and the code is obtained by taking all the binary vectors of length n and weight (i.e., number of 1's) $\lceil n/2 \rceil$ (by $\lceil x \rceil$ we denote the smallest integer larger or equal than x and by $\lfloor x \rfloor$ the largest integer smaller or equal than x).

Sperner codes, though, are difficult to implement when n is large. Except for look-up tables, there are no efficient encoders. Most encoders used in practice are systematic, i.e., given any k information bits, the encoder adds a tail of r redundant bits. In our case, we want to add a tail such that the resulting code is unordered. An optimal solution to this problem was found by Berger [1] in 1962. Berger's encoder works as follows: given an information vector of length k and weight w, add a tail to this information vector that consists of the binary representation of $k - w$. For instance, if $k = 5$ and we want to encode the information vector 10110, since $k - w = 2$ whose binary representation is 010, we have to append this tail to 10110. It is not difficult to show that the resulting code is unordered. Moreover, the code is optimal in the sense that r is the length of the shortest tail that needs to be appended to the information part in order to obtain an unordered code [7].

In this paper, we study unordered codes having error correcting capability. Namely, any two codewords are unordered and are at distance at least $t + 1$ apart, where t is a prescribed number. Among the applications of error correcting unordered (ECU) codes are codes that can detect up to t symmetrical errors and detect all unidirectional errors [10, 17]. Such codes are useful in semiconductor computer memories, in which faults affecting a large number of bits tend to be of unidirectional type.

Another interesting application is in parallel asynchronous communications. In effect, consider a communication channel that consists of several sub-channels transmitting simultaneously and asynchronously [14]. As an example of this scheme, we can consider a board with several chips. The sub-channels represent wires connecting between the chips where differences in the lengths of the wires might result in asynchronous reception. In current technology, the receiver acknowledges reception of the message before the transmitter sends the following message. Namely, pipelined utilization of the channel is not possible.

We developed a scheme that enables to transmit without an acknowledgement of the message allowing for pipelined communication and providing a higher bandwidth [2]. Moreover, our scheme allows for a certain number of transitions from a second message to arrive before reception of the current message has been completed, a condition that we call skew. We have derived necessary and sufficient conditions for codes that can detect skew as well as they can correct the skew and allow continuous operation. It turns out that ECU codes satisfy the necessary and sufficient conditions.

Our main contribution in this paper is the construc-

0731-3071/92 $03.00 © 1992 IEEE

tion of efficient ECU codes. We present 3 possible constructions of error-correcting unordered (ECU) codes and compare between them. Our first construction, (also the most general one), is as follows: we first encode the information bits into an error-correcting code, and then we add a tail in such a way that the resulting code is unordered. Although such a construction is not globally optimal, it has the advantage that the resulting code is systematic when the error-correcting code is systematic. Our construction generalizes the Berger construction. In fact, Berger codes are a particular case of our construction when the underlying error-correcting codes have minimum distance 1. We also prove that our codes are optimal when the error-correcting code is an extended Hamming code, and also for certain BCH codes. Our second construction works for the case in which the minimum distance is either 3 or 4 and has the advantage that it is systematic and easy to implement. Our third construction is based on Sperner's result. We first encode the information into a balanced vector and then encode the balanced vector into an error correcting code. While this construction is more efficient for large n it has the disadvantages that it is not systematic and that encoding is difficult.

The paper is organized as follows: in the next section we present our three constructions together with tables that compare their efficiencies. In Section 3 we prove that our first construction adds the optimal tail for certain important families of codes. Finally, in Section 4 we review the important application of ECU codes to parallel asynchronous communications.

2 Construction of ECU codes

We start this Section giving a generalization of the Berger construction.

Construction 2.1 Assume that we want to construct an ECU code $\mathcal{C}$ with minimum distance d and dimension k. Choose an $[n', k, d]$ error correcting (EC) code $\mathcal{C}'$. Let $\underline{u}$ be an information vector of length k. Then proceed as follows:

1. Encode $\underline{u}$ into a vector $\underline{v} \in \mathcal{C}'$.
2. Let j be the (Hamming) weight of $\underline{v}$. Then append to $\underline{v}$ the complement of the binary representation of $\lfloor j/d \rfloor$.

The code obtained with this encoding procedure is ECU with minimum distance d.

Before proving that the code is ECU, we observe the following:

1. The code $\mathcal{C}$ has length $n' + \lceil \log_2 \lceil (n'+1)/d \rceil \rceil$.
2. The Berger construction corresponds to the special case in which $\mathcal{C}'$ is the $[k, k, 1]$ code.
3. The code $\mathcal{C}$ is systematic if the code $\mathcal{C}'$ is systematic.
4. We may sometimes make the construction more efficient when the all-1 vector is in $\mathcal{C}'$ by taking a coset of this code. The construction is analogous but we have less than $n'+1$ different weights in the coset [5].
5. For a table with the best error-correcting codes, see [15, 16].

Lemma 2.1 The code $\mathcal{C}$ obtained in Construction 2.1 is ECU with minimum distance d.

Proof: It is clear that the minimum distance is d. Assume that we have two codewords $\underline{u}$ and $\underline{v}$ in $\mathcal{C}'$ with weights i and j respectively, where $i \leq j$. Notice that $N(\underline{v}, \underline{u}) > 0$. Let $\underline{t_u}$ and $\underline{t_v}$ be the tails when we encode using Construction 2.1. We will prove that $N\left((\underline{u}, \underline{t_u}), (\underline{v}, \underline{t_v})\right) > 0$.

We have two possibilities: either $\lfloor i/d \rfloor = \lfloor j/d \rfloor$ or $\lfloor i/d \rfloor \neq \lfloor j/d \rfloor$.

If $\lfloor i/d \rfloor = \lfloor j/d \rfloor$, then $j - i \leq d - 1$. If $\underline{u} \subseteq \underline{v}$, then $d_H(\underline{u}, \underline{v}) = j - i \leq d - 1$, a contradiction. So, in particular, $(\underline{u}, \underline{t_u})$ and $(\underline{v}, \underline{t_v})$ are unordered.

If $\lfloor i/d \rfloor \neq \lfloor j/d \rfloor$, then, in particular, $\lfloor i/d \rfloor < \lfloor j/d \rfloor$. According to Construction 2.1, $\underline{t_u}$ as a binary number is larger than $\underline{t_v}$ as a binary number. This means, $N(\underline{t_u}, \underline{t_v}) > 0$. Since we had that $N(\underline{v}, \underline{u}) > 0$, it follows that $(\underline{u}, \underline{t_u})$ and $(\underline{v}, \underline{t_v})$ are unordered. □

Example 2.1 Assume that we want to construct an ECU code with minimum distance 4 and dimension 57 using Construction 2.1. We first encode the 57 information bits into a $[64, 57, 4]$ extended Hamming code. Then we add a tail of length $\lceil \log_2 \lceil 65/4 \rceil \rceil = 5$ bits. This gives a total of 12 redundant bits. If we take a coset of the $[64, 57, 4]$ code, then the weight distribution goes from 1 to 63, so we have 63 different weights. Now, $\lceil \log_2 \lceil 63/4 \rceil \rceil = 4$ bits, so we save one bit in the total redundancy.

The next construction gives a very simple method to obtain ECU codes with minimum distance $d = 3$.

Construction 2.2 Let $(u_0, u_1, \ldots, u_{k-1})$ be an information vector. Then proceed as follows:

1. Add a parity bit $u_k = \bigoplus_{i=0}^{k-1} u_i$.
2. Let $\{i_1, i_2, \ldots, i_j\}$ be the set where $(u_0, u_1, \ldots, u_k)$ is zero (i.e., the complement of the support). Let $\underline{r}$ be the binary representation of $\sum_{l=1}^{j} i_l$. Appending $\underline{r}$ to $(u_0, u_1, \ldots, u_k)$ completes the encoding.

The code obtained with this encoding procedure is ECU with minimum distance 3.

Before proving that Construction 2.2 gives an ECU code with minimum distance 3, let us illustrate it with an example.

Example 2.2 Assume that we want to construct an ECU code with minimum distance 3 and dimension 5 using Constructions 2.1 and 2.2. Let us start with Construction 2.1. Assume that we want to encode (1 0 0 1 1). By using the parity check matrix

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix},$$

the vector is encoded into

$$(1\,0\,0\,1\,1\ \ 1\,1\,1\,1).$$

We finally have to add a tail that is the complement of the binary representation of $\lceil 7/3 \rceil = 3$. Since this is the possible maximum number, it is enough with two bits to represent this tail, i.e., we have to add 0 0. The final encoded vector is

$$(1\,0\,0\,1\,1\ \ 1\,1\,1\,1\ \ 0\,0),$$

so we need 6 redundant bits with Construction 2.1.

Let us encode the same vector using Construction 2.2. We first add a parity bit, so we obtain

$$(1\,0\,0\,1\,1\ \ 1).$$

The set of zeros is $\{1, 2\}$. Since the possible maximum is 15, we need 4 extra redundant bits. Now, 1+2=3, which represented in binary with 4 bits is 0 0 1 1. The final encoded vector is

$$(1\,0\,0\,1\,1\ \ 1\ \ 0\,0\,1\,1).$$

As we can see, we need now 5 redundant bits, so we have saved a bit with respect to Construction 2.1. However, this is not always the case. The choice of one construction over the other depends on the parameter k being considered. Moreover, by looking at Table 1, this appears to be the only case in which Construction 2.2 performs better than Construction 2.1.

Observe that, given k information bits, Construction 2.2 adds $\lceil \log_2 k + \log_2(k+1) \rceil$ redundant bits.

Decoding ECU codes when used for detection of t (symmetric) errors and detection of all unidirectional errors is very simple: we reencode the information bits, and we check if the obtained redundancy coincides with the received redundancy. If they coincide, then no errors are detected, otherwise we conclude that errors have occurred.

Let us prove now that Construction 2.2 gives in fact an ECU code with minimum distance 3.

Lemma 2.2 The code $\mathcal{C}$ obtained in Construction 2.2 is ECU with minimum distance 3.

Proof: Consider two codewords $\underline{v}_1 = (\underline{u}_1, \underline{r}_1)$ and $\underline{v}_2 = (\underline{u}_2, \underline{r}_2)$, where $\underline{u}_1$ and $\underline{u}_2$ are the information vectors plus the extra parity check bits. In particular, $d_H(\underline{u}_1, \underline{u}_2) \geq 2$.

Assume first that $\underline{u}_1$ and $\underline{u}_2$ are unordered. If $d_H(\underline{u}_1, \underline{u}_2) > 2$, we are done, so let $d_H(\underline{u}_1, \underline{u}_2) = 2$. Since, in particular, $\underline{v}_1$ and $\underline{v}_2$ are also unordered, we have to show that $\underline{r}_1 \neq \underline{r}_2$, which will make $d_H(\underline{v}_1, \underline{v}_2) \geq 3$.

Since $d_H(\underline{u}_1, \underline{u}_2) = 2$ and $\underline{u}_1$ and $\underline{u}_2$ are unordered, let j be the location where $\underline{u}_1$ is 0 and $\underline{u}_2$ is 1 and l the location where $\underline{u}_1$ is 1 and $\underline{u}_2$ is 0. Without loss of generality, assume that $j > l$. Therefore, $\underline{r}_1$ will be larger than $\underline{r}_2$ as a binary number. In particular, they are different, so $d(\underline{v}_1, \underline{v}_2) \geq 3$.

Assume next that $\underline{u}_1$ and $\underline{u}_2$ are not unordered. Without loss of generality, assume that $\underline{u}_1 \leq \underline{u}_2$. In particular, the set of zeros of $\underline{u}_2$ is contained into the set of zeros of $\underline{u}_1$, thus, $\underline{r}_1$ is larger than $\underline{r}_2$ as a binary number. Therefore $\underline{r}_1$ is not contained in $\underline{r}_2$, proving that $\underline{v}_1$ and $\underline{v}_2$ are unordered. We can also easily see that $d_H(\underline{v}_1, \underline{v}_2) \geq 3$. In effect, since $\underline{u}_1 \leq \underline{u}_2$, $d_H(\underline{u}_1, \underline{u}_2) \geq 2$, and since, in particular, $\underline{r}_1 \neq \underline{r}_2$, this completes the proof. □

We give a third construction, that is quite natural but it is not systematic. The encoding is giving by a look-up table, so this construction is not very practical when k is large. Essentially, we reverse the order of Construction 2.1: we first encode the k information symbols into m symbols such that $2^k \leq \binom{m}{\lceil m/2 \rceil}$ and m is as small as possible. We then add r redundant bits is such a way that we obtain an error-correcting code with minimum distance d. It is clear that the resulting construction gives an ECU code with minimum distance d and length $m + r$. Formally:

Construction 2.3 Let $\underline{u} = (u_0, u_1, \ldots, u_{k-1})$ be an information vector. Let f be a 1-1 assignment from the 2^k information vectors to balanced vectors of length m and weight $\lceil m/2 \rceil$, where m is the minimum such that $2^k \leq \binom{m}{\lceil m/2 \rceil}$. Let $\mathcal{C}'$ be an $[n, m, d]$ error-correcting code. Then proceed as follows:

1. Obtain $\underline{v} = f(\underline{u})$.
2. Encode $\underline{v}$ into a vector $\underline{w} \in \mathcal{C}'$.

The code obtained with this encoding procedure is ECU with minimum distance d.

Example 2.3 Assume that we want to encode $\underline{u} = (1\ 0\ 0\ 1\ 1)$ as in Example 2.2 to obtain an ECU code with minimum distance 3, but this time using Construction 2.3. The smallest m such that $2^5 = 32 \leq \binom{m}{\lceil m/2 \rceil}$ is 7, so we have to encode $\underline{u}$ into a vector of length 7 and weight 3 (or 4). We can use lexicographic order. The vector $\underline{u}$ taken as a binary number corresponds to 19. The 19th vector of length 7 and weight 3 is $\underline{v} = (0\ 1\ 1\ 1\ 0\ 0\ 0)$. Now, we can use an $[11, 7, 3]$ Hamming code, say, the one whose parity check matrix is

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Vector $\underline{u}$ is then encoded into

$$(0\,1\,1\,1\,0\,0\,0\ \ 0\,1\,0\,1).$$

As we can see, we need 6 redundant bits with Construction 2.3.

In order to obtain an ECU code with minimum distance $d = 4$ using Construction 2.2 or Construction 2.3, we simply add a parity bit to the a code with $d = 3$. When Construction 2.1 is used, sometimes codes with minimum distance 4 tie the redundancy of codes with minimum distance 3.

Table 1 gives the total redundancy we have to add to k information bits (for some values of k) to obtain ECU codes with $d = 3$ and $d = 4$ using the different constructions.

Table 2 gives the total redundancy we have to add to k information bits (for some values of k) to obtain ECU codes with $d = 5$, 6, 7, 8, 9 and 10 using Construction 2.1. For the first 4 rows of the table, we used the tables given in [15, 16]. For $k = 128$ and $k = 256$, since they are beyond the scope of the table, we used BCH codes [18].

In Table 3, we do the same thing with respect to Construction 2.3. Not surprisingly, the reader can observe that Construction 2.1 performs better than Construction 2.3 for relatively small values of k and large values of d.

In the next section, we deal with the issues of optimality of ECU codes.

3 Optimality of ECU Codes

In this section, we prove the optimality of Construction 2.1 for extended Hamming codes and for certain BCH codes in the following sense: the tail added to the error correcting code has minimal length, i.e., it is impossible to find a shorter tail making the code unordered.

We begin by defining the concept of a *chain* of vectors.

Definition 3.1 A set of binary vectors $\{V_1, V_2, \ldots, V_m\}$ is a chain of length m if any two vectors in the set are ordered.

The idea in proving the optimality of our constructions is to exhibit a long enough chain of codewords in the error correcting code. The following lemma gives the key:

Lemma 3.1 Let $\{C_1, C_2, \ldots, C_m\}$ be a chain of vectors, each being a codeword in a given code $\mathcal{C}$. Then the length of the a tail that we have to add to $\mathcal{C}$ to make it unordered is at least $\lceil \log_2 m \rceil$ bits.

Proof: Since all the codewords in the chain are ordered, we need to have a different tail for every one of them to make them unordered. Hence, we need at least m different tails. □

We prove the optimality of some of our constructions by exhibiting chains of length $\lceil n/d \rceil + 1$ in an $[n, k, d]$ code. First we prove the optimality of our construction for the extended Hamming code by exhibiting a chain of $2^{m-2} + 1$ codewords in a code of length 2^m.

Proposition 3.1 The $[2^m, 2^m - m - 1, 4]$ extended Hamming code contains a chain of $2^{m-2}+1$ codewords.

Proof: The columns of the parity check matrix of a $(2^m, 2^m - m - 1, 4)$ extended Hamming code are:

$$\{(v_1, v_2, \ldots, v_m, 1)^T : (v_1, v_2, \ldots, v_m) \in \{0,1\}^m\}.$$

Note that we can arrange the columns in the parity check matrix in pairs such that the first m bits are complementary. Namely, column $(v_1, v_2, \ldots, v_m, 1)^T$ is paired with $(\bar{v}_1, \bar{v}_2, \ldots, \bar{v}_m, 1)^T$. Hence, the sum of a pair of columns in this arrangement gives the vector $(1, 1, \ldots, 1, 0)^T$ and the sum of 2 pairs (4 columns) is the all-0 vector. We call this matrix H_m. For example,

$$H_3 = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Consider the extended Hamming code that corresponds to the matrix H_m. It follows from the construction of H_m that the following set of $2^{m-2} + 1$ codewords is a chain:

$$\{(1111)^i 0^{2^m - 4i} : 0 \leq i \leq 2^{m-2}\}$$

where S^i, S a binary vector, is a vector obtained by concatenating S i times. □

The second result is related to BCH codes. We prove that in many cases we can exhibit chains of codewords that show the optimality of our construction. The key to exhibiting long chains is the following lemma [6]:

Lemma 3.2 Consider a binary t-error-correcting BCH code defined in a standard way, i.e., as a cyclic code of length 2^m-1. Let a and b be two integers such that

$$a \cdot b = 2^m - 1$$

and

$$a \geq 2t+1.$$

Then the following b polynomials correspond to codewords:

$$z_1(X) = 1 + X^b + X^{2b} + \cdots + X^{(a-1)b}$$

and for $2 \leq i \leq b$,

$$z_i(X) = X^{i-1} z_1(X).$$

Using this lemma we can prove the following:

Proposition 3.2 Given a t-error correcting BCH code of length $2^m-1 = a \cdot b$ where a and b are integers, and $a \geq 2t+1$, we can exhibit a chain of length $b+1$.

Proof: The proof follows from lemma 3.2. The chain consists of the all-0 vector and the set of b vectors that correspond to partial sums of the polynomials from Lemma 3.2 as follows:

$$\left\{ \sum_{i=1}^{j} z_i \ : \ 1 \leq j \leq b \right\}.$$

□

Example 3.1 Consider the case $t = 2$, namely $2t+1 = 5$. We can exhibit a chain of $((2^m-1)/5)+1$ codewords in all the cases in which $2^m - 1 \equiv 0 (\mathrm{mod}\ 5)$. For example, for $m = 4$ we can exhibit a chain of length 4. In general, we can exhibit a long chain whenever $m \equiv 0 (\mathrm{mod}\ 4)$ (by Fermat's Theorem). Similarly, for $2t+1 = 7$, we can exhibit a long chain for all all the cases in which $m \equiv 0 (\mathrm{mod}\ 6)$.

4 Application to Parallel Asynchronous Communications

In this section, we give an application of ECU unordered codes to parallel asynchronous communications.

Consider a communication channel that consists of several sub-channels transmitting simultaneously. Namely, we would like to transmit a binary vector of length n using n parallel channels/wires. Every wire can carry only one bit of information. Each wire represents a coordinate of the vector to be transmitted. The propagation delay in the wires varies. The problem is to find an efficient communication scheme that will be delay-insensitive.

Since the signals do not arrive at the same time, a natural question is: how does the receiver know that the reception is complete? This problem is studied in [14]. There, the forgoing physical model is described as a scheme in which the sender communicates with the receiver via parallel tracks by rolling marbles (that correspond to a logical 1) in the tracks. Although the marbles are sent in parallel, the channels are asynchronous. This means that marbles are received randomly and at different instants.

Let us introduce some notation. The tracks are represented with the numbers $1, 2, \ldots, n$. After the m-th marble has arrived, the receiver obtains a sequence $X_m = x_1, x_2, \ldots, x_m$, where $1 \leq x_i \leq n$, the number x_i meaning that the i-th marble was received at the x_i-th track. The set $\{x_1, x_2, \ldots, x_m\}$ is the support (i.e., the set of non-zero coordinates) of a vector and determines uniquely a binary vector. From now on, the sequence $X_m = x_1, x_2, \ldots, x_m$ will denote either the sequence as defined above, or a binary vector as defined by its support. Also, X may denote either a vector or its support. This abuse of notation should be clear from the context.

For example, let a vector $X = 0110$ and a vector $Y = 0100$. In the language of sets we have $X = \{2, 3\}$ and $Y = \{2\}$. Clearly, when the receiver gets a marble in track number 2, it is not clear whether he just received Y or he should wait to get a marble in track number 3 (this will correspond to receiving X).

So, we have the following situation: assuming that a vector X is transmitted, once reception has been completed, the receiver acknowledges receipt of the message. The next message is sent by the sender only after the receipt of the acknowledgement. The problem is finding a code $\mathcal{C}$ whose elements are messages such that the receiver can identify when transmission has been completed. It is easy to see, as proved in [14], that the codes having the right property are unordered codes.

One of the disadvantages of using the asynchronous type of communication is the fact that the channel is not fully utilized. Namely, there is at most one vector in the wires at any given time. This becomes very critical when the transmission rates are getting higher and lines are getting longer, so it is desirable to have a scheme that enables a pipelined utilization of the channel. In addition, our scheme has the important feature of not using a handshake (acknowledgement) mechanism. Hence, there is no need of communication between receiver and sender.

We note that if one is ready to pay in performance, then a possible strategy, if acknowledgment of messages is not allowed, is that the sender will wait long enough between messages. So, if the sender sends a codeword X followed by a codeword Y, it will be very unlikely that a marble from Y will arrive before the

reception of X has been completed. With this scheme, we can again use unordered codes as in [14].

So, we would like to study parallel asynchronous pipelined communication without acknowledgement. The main difficulty in this scheme is that a certain number of marbles from the second message might arrive before reception of the current message has been completed, a condition that we call *skew*.

It turns out that skew should be defined using two parameters. Assume that we transmit a vector X followed by a vector Y. In general, since X is transmitted first, the marbles from X will tend to arrive before the marbles from Y. Also, if a marble in Y arrives before the transmission of X has been completed, it is very likely that few marbles remain in X. Let us call t_1 the maximum number of marbles that may remain in X when a marble from Y arrives. Also, we do not expect too many marbles from Y to arrive before the transmission of X has been completed. Let us call t_2 the maximum number of marbles from Y that may arrive before the completion of X.

Our approach to dealing with skew is to use coding theory methodology and to try to identify the properties of a family of vectors (a code) that can handle the skew. In some applications, we might merely want to detect that skew has occurred, and then invoke a protocol that will halt transmission and allow for retransmission. Codes detecting skew are called *skew-detecting* codes. Formally:

Definition 4.1 We say that a code $\mathcal{C}$ is (t_1, t_2)-skew-detecting if, for any pair of codewords $X, Y \in \mathcal{C}$ such that codeword X is transmitted followed by codeword Y, and the skew between X and Y is limited by the following two conditions:

1. at most t_1 marbles may still be missing from X when a marble from Y arrives; and
2. at most t_2 marbles from Y may arrive before all the marbles from X have arrived;

then $\mathcal{C}$ will correctly decode X when there is no skew between X and Y, and will detect at a certain point the presence of skew provided it does not exceed the t_1 and t_2 constraints.

In other applications, we might want to go further and correct the skew, since this will allow for continuous operation. Codes capable of correcting skew are called *skew-tolerant* codes. Formally,

Definition 4.2 We say that a code $\mathcal{C}$ is (t_1, t_2)-skew-tolerant if, for any pair of codewords $X, Y \in \mathcal{C}$ such that codeword X is transmitted followed by codeword Y, and the skew between X and Y is limited by the following two conditions:

1. at most t_1 marbles may still be missing from X when a marble from Y arrives; and
2. at most t_2 marbles from Y may arrive before all the marbles from X have arrived;

then $\mathcal{C}$ will correctly decode X when the skew between X and Y does not exceed the t_1 and t_2 constraints.

Before stating the necessary and sufficient conditions for skew tolerant and skew detecting codes, we give some notation. Given two binary vectors X and Y of length n, we denote by $N(X, Y)$ the number of coordinates in which X is 1 and Y is 0. For example, if $X = 10110$ and $Y = 00101$, we have $N(X, Y) = 2$ and $N(Y, X) = 1$. Notice that $N(X, Y) + N(Y, X) = d_H(X, Y)$, where d_H denotes Hamming distance.

The following theorem gives necessary and sufficient conditions for a code to be (t_1, t_2)-skew-detecting.

Theorem 4.1 Let $\mathcal{C}$ be a code and, given two positive integers t_1 and t_2, let $t = \min\{t_1, t_2\}$ and $T = \max\{t_1, t_2\}$. Then, code $\mathcal{C}$ is (t_1, t_2)-skew-detecting if and only if, for any pair of codewords X and Y in $\mathcal{C}$, at least one of the following two conditions occurs:

(a) $\min\{N(X, Y), N(Y, X)\} \geq t + 1$

or

(b) $\min\{N(X, Y), N(Y, X)\} \geq 1$ and $\max\{N(X, Y), N(Y, X)\} \geq T + 1$.

The following corollary is clear from the necessary and sufficient conditions.

Corollary 4.1 A code $\mathcal{C}$ is (t, t)-skew-detecting if and only if, for any $X, Y \in \mathcal{C}$, $\min\{N(X, Y), N(Y, X)\} \geq 1$ and $\max\{N(X, Y), N(Y, X)\} \geq t + 1$.

The following theorem gives necessary and sufficient conditions for a code to be (t_1, t_2)-skew-tolerant.

Theorem 4.2 Let $\mathcal{C}$ be a code, t_1 and t_2 two positive integers and $t = \min\{t_1, t_2\}$. Then, code $\mathcal{C}$ is (t_1, t_2)-skew-tolerant if and only if, for any pair of codewords X and Y in $\mathcal{C}$, at least one of the following two conditions occurs:

(a) $\min\{N(X, Y), N(Y, X)\} \geq t + 1$

or

(b) $\min\{N(X, Y), N(Y, X)\} \geq 1$ and $\max\{N(X, Y), N(Y, X)\} \geq t_1 + t_2 + 1$.

For a proof of Theorems 4.1 and 4.2 together with decoding algorithms, the reader is referred to [2].

The connection between ECU codes and (t_1, t_2)-skew-detecting and tolerant codes is given by the following lemma:

Lemma 4.1 Let t_1 and t_2 be positive integers and $t = \min\{t_1, t_2\}$. Then:

1. Let $\mathcal{C}$ be an ECU with minimum distance $\geq t_1 + t_2 + 1$. Then $\mathcal{C}$ is (t_1, t_2)-skew-detecting.

2. Let $\mathcal{C}$ be an ECU with minimum distance $\geq t_1 + t_2 + t + 1$. Then $\mathcal{C}$ is (t_1, t_2)-skew-tolerant.

Proof:

1. Let $t = \min\{t_1, t_2\}$ and $T = \max\{t_1, t_2\}$. Let $X, Y \in \mathcal{C}$, and assume that condition (a) is violated, say, $N(X,Y) \leq t$. The codewords are unordered, and also, $N(Y,X) = d_H(X,Y) - N(X,Y) \geq t_1 + t_2 + 1 - t = T + 1$. Hence, X and Y satisfy condition (b), proving that the code is (t_1, t_2)-skew-detecting.

2. Let $X, Y \in \mathcal{C}$, and assume that condition (a) is violated, say, $N(X,Y) \leq t$. The codewords are unordered, and also $N(Y,X) = d_H(X,Y) - N(X,Y) \geq t_1 + t_2 + 1$. Hence, X and Y satisfy condition (b), proving that the code is (t_1, t_2)-skew-tolerant.

□

In the particular case in which $t_1 = t_2 = t$, an ECU code with minimum distance $2t + 1$ gives a (t, t)-skew-detecting code, while an ECU with minimum distance $3t + 1$ gives a (t, t)-skew-tolerant code.

We also notice that, given t_1 and t_2, $t = \min\{t_1, t_2\}$, a t-error correcting/all unidirectional error detecting (EC/AUED) code [11] is (t_1, t_2)-skew-detecting and (t_1, t_2)-skew-tolerant, since it satisfies the first of the necessary and sufficient conditions in Theorems 4.1 and 4.2. For efficient constructions of t-EC/AUED codes, the reader is referred to [3, 4, 5, 13].

In general, constructions using ECU codes have less redundancy than constructions using EC/AUED codes, unless there is a large imbalance between t_1 and t_2 [2].

Let us point out that some of the constructions of (t_1, t_2) skew-tolerant codes were improved in a recent paper [8].

References

[1] J. M. Berger, "A note on error detecting codes for asymmetric channels," Information and Control, Vol. 4, pp. 68-73, March 1961.

[2] M. Blaum and J. Bruck, "Coding for Skew-Tolerant Parallel Asynchronous Communications," IBM Research Report, RJ 8268 (75629), July 1991.

[3] M. Blaum and H. van Tilborg, "On t-Error Correcting/All Unidirectional Error Detecting Codes," IEEE Trans. on Computers, vol. C-38, pp. 1493-1501, November 1989.

[4] F. J. H. Boinck and H. van Tilborg, "Constructions and bounds for systematic tEC/AUED codes," IEEE Trans. on Information Theory, vol. IT-36, No. 6, pp. 1381-1390, November 1990.

[5] J. Bruck and M. Blaum, "New Techniques for Constructing EC/AUED Codes," IBM Research Report, RJ 6818 (65352), May 1989, to appear in IEEE Trans. on Computers.

[6] R. H. Deng and M. A. Herro, "DC-Free Coset Codes," IEEE Trans. on Information Theory, vol. IT-34, pp. 786-792, July 1988.

[7] C. V. Freiman, "Optimal Error Detecting Codes for Completely Asymmetric Binary Channels," Information and Control, Vol. 5, pp. 64-71, March 1962.

[8] L. H. Khachatrian, "Construction of (t_1, t_2)-tolerant Codes," to appear in Proceedings of Dilijan Conference, Sept. 1991.

[9] E. L. Leiss, "Data Integrity in Digital Optical Disks," IEEE Trans. on Computers, Vol. C-33, No. 9, pp. 818-827, Sept. 1984.

[10] D. Nikolos, "Theory and Design of t-Error Correcting/d-Error Detecting ($d > t$) and All Unidirectional Error Detecting Codes," IEEE Trans. on Computers, Vol. C-40, No. 2, pp. 132-142, Feb. 1991.

[11] D. K. Pradhan, "A new class of error-correcting detecting codes for fault-tolerant computer application," IEEE Trans. on Computers, vol. C-29, pp. 471-481, June 1980.

[12] E. Sperner, "Ein Satz über Untermengen einer endlichen Menge," Math. Z. **27**, 544-548, 1928.

[13] D. L. Tao, C. R. P. Hartmann and P. K. Lala, "An Efficient Class of Unidirectional Error Detecting/Correcting Codes," IEEE Trans. on Computers, vol. C-37, pp. 879-882, July 1988.

[14] T. Verhoeff, "Delay-insensitive codes - an overview," Distributed Computing, 3:1-8, 1988.

[15] T. Verhoeff, "An Updated Table of Minimum Distance Bounds for Binary Linear Codes," IEEE Trans. on Information Theory, vol. IT-33, pp. 665-680, Sept. 1987.

[16] T. Verhoeff, "An Updated Table of Minimum Distance Bounds for Binary Linear Codes," updated January 1989, preprint.

[17] J. H. Weber, C. de Vroedt and D. E. Boekee, "Necessary and Sufficient Conditions on Block Codes Correcting/Detecting Errors of Various Types," Proceedings of the Tenth Symposium on Information Theory in the Benelux, Houthalen, Belgium, May 25-26, 1989, pp. 31-36, to appear in IEEE Trans. on Computers.

[18] W. Wesley Peterson and E. J. Weldon, "Error-Correcting Codes," Second Edition, MIT Press, 1984.

	Construction 2.1		Construction 2.2		Construction 2.3	
k	$d=3$	$d=4$	$d=3$	$d=4$	$d=3$	$d=4$
4	5	6	5	6	6	7
5	6	7	5	6	6	7
6	6	7	6	7	6	7
7	6	7	6	7	7	8
8	7	7	7	8	7	8
9	7	7	7	8	8	9
10	7	7	7	8	8	9
11	7	8	8	9	8	9
12	8	9	8	9	8	9
13	8	9	8	9	8	9
14	8	9	8	9	8	9
15	8	9	8	9	8	9
16	8	9	9	10	8	9
22	9	9	9	10	8	9
23	9	9	10	11	8	9
26	9	10	10	11	9	10
32	10	11	11	12	9	10
64	12	13	13	14	11	12
128	14	15	15	16	12	13
256	16	17	17	18	14	15

Table 1: Parameters of some codes using Constructions 2.1, 2.2 and 2.3

k	$d=5$	$d=6$	$d=7$	$d=8$	$d=9$	$d=10$
4	5	6	5	6	6	7
10	11	12	13	14	19	20
20	13	14	18	19	23	24
32	15	15	20	21	27	28
64	17	18	23	24	31	32
128	20	21	29	30	37	38
256	24	25	33	34	42	42

Table 2: Parameters of some codes using Construction 2.1.

k	$d=5$	$d=6$	$d=7$	$d=8$	$d=9$	$d=10$
4	5	6	5	6	6	7
10	12	13	16	17	21	22
20	13	14	19	20	24	25
32	14	15	20	21	27	28
64	17	18	23	24	32	33
128	20	21	28	29	36	37
256	23	24	32	33	41	42

Table 3: Parameters of some codes using Construction 2.3.