ACKNOWLEDGMENT

The author gratefully acknowledges the thorough critique of the referees. In particular, one referee cited an omission in the proof of Lemma 1 and suggested the argument by contradiction that completed the proof.

Codi ng Techniques for Failure-Tolerant Counters

IRVING S. REED, SENIOR MEMBER, IEEE, AND ALBERT C. L. CHIANG, MEMBER, IEEE

Abstract—This paper delineates an application of two classes of parity-check codes to the design for failure-tolerant counters. They are 1) a modified first-order Reed–Muller code and 2) the perfect Hamming code. The first code employs a majority element for implementing the error-correcting scheme while the second one makes use of a variable $2^{n-k-1} \times 1$-out-of-$2^n+1$ majority element. These coding techniques can be applied in principle to other logic hardware to increase its reliability.

Index Terms—Failure-tolerant counter, Hamming code, majority element, parity-check code, Reed–Muller code.

I. INTRODUCTION

The need for reliable digital computers has been rising rapidly. Many attempts have been made to improve the computer reliability. For instance, in early 1950, Hamming devised his code primarily for solving computer reliability problems [5]; subsequently, Armstrong [3], Pierce [4], and others made contributions in this area.

More recently, Reed applied coding to the development of a systematic method for increasing the reliability of sequential circuits [1].

In this paper, coding techniques for parity-check codes are applied to a study for the design of failure-tolerant counters. By a failure-tolerant counter we mean an encoded counter that is able to operate without malfunction even if a logic component failure occurs within it. A counter is probably the most elementary sequential circuit in a digital computer. The coding techniques for the design of the reliable counter can be extended in principle to the design of other logic hardware for improving its reliability.

A parity-check code is characterized by its parity-check matrix. A parity-check matrix $H$ of $n$ columns and $n-k$ rows for any binary error-correcting code can be expressed in general in a reduced-echelon form [7]:

$$H = \begin{bmatrix} q_{11} & q_{12} & q_{13} & \cdots & q_{1k} & 1 & 0 & \cdots & 0 \\ q_{21} & q_{22} & q_{23} & \cdots & q_{2k} & 0 & 1 & \cdots & 0 \\ q_{31} & q_{32} & q_{33} & \cdots & q_{3k} & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ q_{n-k1} & q_{n-k2} & q_{n-k3} & \cdots & q_{n-kk} & 0 & 0 & 0 & \cdots & 1 \end{bmatrix} = [Q, I_{n-k}]$$

where $I_{n-k}$ is an $n-k$ identity matrix and $Q$ is an $n-k$ by $k$ matrix with binary element $q_{ij}$.

The corresponding code space $V$ consists of all elements $v$ such that $vH^T = 0$, where $H^T$ is the transpose of matrix $H$. More specifically, if $v = (A_1, A_2, \cdots, A_k, B_1, B_2, \cdots, B_{n-k})$, then $v$ is a code word if and only if

$$A_1q_{11} \oplus A_2q_{12} \oplus A_3q_{13} \oplus \cdots \oplus A_kq_{1k} \oplus B_1 = 0$$

or

$$B_i = A_1q_{i1} \oplus A_2q_{i2} \oplus A_3q_{i3} \oplus \cdots \oplus A_kq_{ik}$$

for $i = 1, 2, \cdots, n-k$, where $\oplus$ denotes the modulo 2 sum. This code $V$ is called an $(n, k)$ code, where $n$ denotes the block length and $k$ the bit length for the information symbols. The bit length for the check symbols is given by $n - k$.

The classes of parity-check codes discussed in this article are 1) a certain modified first-order Reed–Muller code whose parity-check matrix has exactly three 1’s in each row and 2) the perfect Hamming code. The first code is a low-density parity-check code in the sense that its parity-check matrix contains mostly 0’s and relatively few 1’s. Its coding is simple to implement. A 2-out-of-3 majority element is used for purpose of error correcting. The second code needs relatively fewer check bits than the first one for the same block length, but a relatively more complicated implementation problem is involved. Further, for purposes of error correcting, a variable $2^{n-k-1} + 1$-out-of-$2^n+1$ majority element is used.
The error-correcting ability of a parity-check code can be specified by its parity-check matrix. In general, a parity-check code is $e$-error correcting if and only if every set of $2e$ columns of its parity-check matrix is linearly independent [8]. Hence, a parity-check code is single-error correcting if its parity-check matrix contains at least one column with exactly two 1’s. From the viewpoint of applications this paper treats only the special case of a single-failure-tolerant counter.

II. MODIFIED FIRST-ORDER REED–MULLER CODES

The modified first-order Reed–Muller code considered herewith has exactly three 1’s in each row of its parity-check matrix. For example, consider a 3 by 6 parity-check matrix $H$ as follows:

$$
H = \begin{bmatrix}
1 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 1
\end{bmatrix}
$$

Let $v=(A_1, A_2, A_3, B_1, B_2, B_3)$ be a code word, where $A_1, A_2, A_3$ are information bits and $B_1, B_2, B_3$ are check bits. Then $vH^T=0$. From the given matrix $H$ a set of parity-check equations can be derived as follows.

$$
B_1 = A_1 \oplus A_2 \\
B_2 = A_2 \oplus A_3 \\
B_3 = A_3 \oplus A_1
$$

or

$$
A_1 = A_2 \oplus B_1 = A_3 \oplus B_3 \\
A_2 = A_1 \oplus B_1 = A_3 \oplus B_3 \\
A_3 = A_1 \oplus B_3 = A_2 \oplus B_2. \tag{1}
$$

Note that each $A_i$, for $i=1, 2, 3$ in the set of (1), can be determined by exactly three independent relationships.

Define $A_i(k) = \operatorname{Maj}_k (A_1, A_2 \oplus B_1, A_3 \oplus B_3)$

$$
A_2(k) = \operatorname{Maj}_k (A_2, A_1 \oplus B_1, A_3 \oplus B_2) \\
A_3(k) = \operatorname{Maj}_k (A_3, A_1 \oplus B_3, A_2 \oplus B_2) \tag{2}
$$

where

$$
\operatorname{Maj}_k (x_1, x_2, x_3) = 1 \quad \text{if} \quad x_1 + x_2 + x_3 \geq 2, \\
\operatorname{Maj}_k (x_1, x_2, x_3) = 0 \quad \text{if} \quad x_1 + x_2 + x_3 < 2,
$$

and where the subscript $k$ denotes the $k$th physical realization of the particular majority element. These majority elements of (2) give the correct output if at most one of the terms $A_1, A_2, A_3, B_1, B_2, B_3$ has a component failure.

From the set of (1), since

$$
\bar{A}_1 = \bar{A}_2 \oplus B_1 = \bar{A}_3 \oplus B_3 \\
\bar{A}_2 = \bar{A}_1 \oplus B_1 = \bar{A}_3 \oplus B_2 \\
\bar{A}_3 = \bar{A}_1 \oplus B_3 = \bar{A}_2 \oplus B_2,
$$

we can also define

$$
\bar{A}_1(k) = \operatorname{Maj}_k (\bar{A}_1, \bar{A}_2 \oplus B_1, \bar{A}_3 \oplus B_3) \\
\bar{A}_2(k) = \operatorname{Maj}_k (\bar{A}_2, \bar{A}_1 \oplus B_1, \bar{A}_3 \oplus B_2) \\
\bar{A}_3(k) = \operatorname{Maj}_k (\bar{A}_3, \bar{A}_1 \oplus B_3, \bar{A}_2 \oplus B_2)
$$

to be the $k$th physically distinct majority element for $\bar{A}_1, \bar{A}_2, \bar{A}_3$, respectively.

These results can be applied to the design of a single-failure-tolerant three-stage counter. As an illustration, consider the design of an ordinary three stage binary counter $A_1, A_2, A_3$ with three auxiliary check stages $B_1, B_2, B_3$ and with the following state sequence.

<table>
<thead>
<tr>
<th>State</th>
<th>$A_3$</th>
<th>$A_2$</th>
<th>$A_1$</th>
<th>$B_3$</th>
<th>$B_2$</th>
<th>$B_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_0$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$S_1$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$S_2$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$S_3$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$S_4$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$S_5$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$S_6$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$S_7$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$S_8$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Let $\Delta x$ denote the change in the variable $x$. The change equations for $A_i$ and $B_i$ can be derived from the given state sequence [6].

$$
\Delta A_1 = 1, \quad \Delta B_1 = \bar{A}_1 \\
\Delta A_2 = A_1, \quad \Delta B_2 = A_1\bar{A}_2 \\
\Delta A_3 = A_1A_2, \quad \Delta B_3 = \bar{A}_1 + \bar{A}_2.
$$

Let $C$ be an equally spaced clock pulse and the counter be intended to count $C$, using toggle flip-flops (or $J$-$K$ flip-flops with $J$ and $K$ tied together). Six $J$-$K$ flip-flops ($A_1, A_2, A_3, B_1, B_2, B_3$) and eight majority elements (three for $A_1$, two for $A_2$, one for $A_3$, two for $\bar{A}_1$, two for $\bar{A}_2$) are required. The detailed logic gate diagram for the designed counter is shown in Fig. 1. Since a different majority element is used each time $A_i$ is required, any majority element in the network of Fig. 1 can fail without affecting the counter mechanism. This approach is useful because the encoded counter does not depend upon the reliability of any single element.

In general, for the design of an $n$-stage single-failure-tolerant binary counter we can choose a parity-check matrix $H$ of $2n$ columns and $n$ rows in such a manner that each row has exactly three 1’s and each column for the first $n$ columns has exactly two 1’s as follows.

$$
H = \begin{bmatrix}
1 & 1 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & 0 \\
0 & 1 & 1 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 & \cdots & 0 \\
0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 1 \\
1 & 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & 1
\end{bmatrix}
$$

Let $v=(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_n)$ be a code word, where $A_1, A_2, \ldots, A_n$ are information bits and $B_1, B_2, \ldots, B_n$ are check bits.

The set of parity-check equations derived from the given matrix $H$ is
Then \(B\) at most and of these given and \(A_1(k)\)
remained, and \(A_2(k)\) \(A_3()\) \(A_4()\) \(A_n,(k)\) \(A_{n-1}(k)\). The matrix \(H\) is shown below.

\[
H = \begin{bmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 
\end{bmatrix}
\]

The code space \(V\) consists of all \(v\) such that \(vH^T = 0\). Now let \(V_\perp\) be the set of all nonzero elements generated by \(H\). Then \(V_\perp\) together with the zero element is the dual space of \(V\). \(V_\perp\) is represented by a \(2^j - 1\) by \(2^j - 1\) matrix of which each row and column contains exactly \(2^j - 1\) ones. Let \(v = (A_1, A_2, \cdots, A_{2^j - 1}, B_1, B_2, \cdots, B_j)\) be a code word. We can derive from \(V_\perp\) a set of \(2^j - 1\) parity-check equations of which \(2^j - 1\) equations contain the term \(A_i\) for each \(i\). In other words, there are exactly \(2^j - 1 + 1\) independent ways for determining each \(A_i\). Thus \(A_i(k)\) can be established by defining a \(2^j - 2 + 1\)-out-of-\(2^j - 1 + 1\) majority element for each \(i\). It is well known that a variable majority element can always be implemented by using 2-out-of-3 majority elements.

As an example, consider \(j = 3\) and \(n = 7\). We have

\[
V_\perp = \begin{bmatrix}
1 & 1 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 
\end{bmatrix}
\]

The set of parity-check equations is

\[
A_1 = A_2 \oplus A_4 \oplus B_1 = A_3 \oplus A_4 \oplus B_2 = A_3 \oplus B_1 \oplus B_3 \\
A_2 = A_1 \oplus A_4 \oplus B_1 = A_3 \oplus B_1 \oplus B_2 \\
A_3 = A_1 \oplus B_1 \oplus B_2 \\
A_4 = A_1 \oplus A_2 \oplus B_1 = A_1 \oplus A_3 \oplus B_2 = A_2 \oplus A_3 \oplus B_3
\]

Thus we can define

\[
A_1(k) = \text{Maj}_k(A_1, A_2 \oplus A_4 \oplus B_1, A_3 \oplus A_4 \oplus B_2, A_3 \oplus B_1 \oplus B_3) \\
A_2(k) = \text{Maj}_k(A_2, A_1 \oplus A_4 \oplus B_1, A_3 \oplus B_1 \oplus B_2, A_3 \oplus A_4 \oplus B_3, A_1 \oplus B_2 \oplus B_3)
\]
\[ A_3(k) = \text{Maj}_k(A_3, A_1 \oplus A_2 \oplus B_1, A_4 \oplus B_2, A_2 \oplus A_4 \oplus B_3, A_1 \oplus B_1 \oplus B_2) \]

\[ A_4(k) = \text{Maj}_k(A_4, A_1 \oplus A_2 \oplus B_1, A_3 \oplus A_4 \oplus B_2, A_2 \oplus A_3 \oplus B_3, A_1 \oplus B_2 \oplus B_3) \]

Each \( A_i(k) \) is determined by a 3-out-of-5 majority element. These majority elements still give the correct output if at most one of the terms \( A_1, A_2, A_3, A_4, B_1, B_2, B_3 \) has a component failure. One can apply these majority elements to the design of a four-stage single-failure-tolerant counter in a manner similar to that for the previous three-stage counter.

REFERENCES


Functional Partitioning and Simulation of Digital Circuits

MELVIN A. BREUER, MEMBER, IEEE

Abstract—In this paper we present a method for obtaining a functional partitioning of the logic of a computer. It is shown that given a basic function to be performed, such as addition, the computer logic can be partitioned into four disjoint sets, namely the active information logic \( \mathcal{J} \), the semiactive flip-flops \( \mathcal{S} \), the activated control logic \( \mathcal{C} \), and the dormant logic \( \mathcal{D} \). Techniques involved in implementing the partitioning algorithm such as an event-directed simulator and three-value simulation are discussed. An application of this partitioning scheme as part of a large logic simulation system is described.

Index Terms—Background simulation, design automation, event-directed simulation, functional logic partitioning, simulation, three-value simulation.

I. INTRODUCTION

PROCEDURES for the automated partitioning of computer logic have previously been discussed [1]–[5]. Most of these techniques carry out the partitioning process based upon the number of wires between the various components and ignore the functional aspects of the design. It has long been a desirable goal to develop a procedure for automatically carrying out functional partitioning. A number of applications for this type of partitioning exists, such as in simulation systems, in specifying the contents of the circuit cards in a computer, and in the retrieval of specific subcircuits of a large design whose description is stored in a mass memory.

In this paper we will present an algorithmic procedure for obtaining a functional partitioning of a circuit with respect to a given partial state, referred to as a partially specified initial condition, and a given subset of the flip-flops and primary outputs of the circuit. In particular, the logic is partitioned into four disjoint sets, called the active information logic \( \mathcal{J} \), the activated control logic \( \mathcal{C} \), the semiactive flip-flops \( \mathcal{S} \), and the dormant logic \( \mathcal{D} \).

To illustrate and informally define these various partitions, consider the logical circuit of a digital computer. Assume the current state of the computer is such that when the next clock pulse occurs, the sum of two registers, for example \( A \) and \( B \), will be stored in register \( A \). Then the \( A \) register, and that logic that actually implements the addition, is referred to as the active information logic. For example, this may consist of a 32-bit parallel adder. Note that the actual information being added does not influence the selection of what constitutes this active information logic.

The sum is placed into register \( A \) at this clock time because the operation code and phase counter are in the appropriate state to allow this information to be gated into this register. That control logic that decodes the operation code and phase counter and allows this transfer of information to take place is referred to as the active control logic.

Manuscript received April 8, 1969; revised April 2, 1970. This work was supported by the Joint Services Electronics Program AF-AFOSR-496-68 and by the National Science Foundation under Grant GK 4386.

The author is with the Department of Electrical Engineering, University of Southern California, Los Angeles, Calif. 90007.