Weight-Based Codes and Their Application to Concurrent Error Detection of Multilevel Circuits

Debaleena Das and Nur A. Touba

Computer Engineering Research Center
Department of Electrical and Computer Engineering
University of Texas, Austin, TX 78712-1084
E-mail: \{ddas, touba\}@cat.ece.utexas.edu

Abstract

This paper proposes a new class of codes termed “weight-based codes” where each output bit is assigned a weight and the check bits represent the sum of the weights of the output bits which have value ‘1’. A Berger code is a special member of this proposed class of codes where each output bit is assigned a weight of one. This paper describes the application of these codes for the efficient on-line error detection of arbitrary multilevel circuits. The overall probability of detecting any number of erroneous bits at the output caused by a single internal fault is shown to be higher for weight-based codes than standard error detecting codes. Further, a very efficient design exists for the checker. The checker is area and speed efficient, has low power consumption, and can be tested by a small set of incoming code words. There is always a tradeoff between the fault detection capability and area overhead requirement of an error detecting code. Weight-based codes present a controlled way of increasing the number of check bits to achieve a desired fault detection capability.

1. Introduction

The development of high-density low-cost integrated circuits has made on-chip error detection techniques a necessity. In applications where dependability and data integrity are important, on-line (or concurrent) error detection circuitry is used to detect transient and intermittent errors. Early detection of errors is crucial for preserving the state of the system and preventing data corruption. The move towards deep-submicron technologies with lower voltage levels and smaller noise margins is increasing the susceptibility of systems to transient and intermittent faults thereby making on-line error detection increasingly important.

This paper focuses on on-line error detection based on error detecting codes. The outputs of a circuit are encoded with an error detecting code. A checker monitors the outputs and gives an error indication if a non-codeword occurs (as illustrated in Fig. 1). Such circuits, which monitor their own faults during normal operation, are termed self-checking circuits [Anderson 84].

![Figure 1. General structure of On-Line Error Detection Based on Systematic Codes](image)

The operation of a self-checking circuit is critically dependent on the correct functioning of the checker. Correct functioning can be achieved by making the checker totally self-checking (TSC) [Anderson 84]. A TSC checker guarantees the detection of the first error to appear at the output bits under the single fault assumption. The single fault assumption for self-checking circuits is: 1) faults occur one at a time, and 2) there is a sufficient time interval between the occurrence of any two faults; the first fault will get detected before the occurrence of the next fault. Thus, the checker should be capable of detecting all errors at the output resulting from single faults in the circuit.

The origin of error detecting codes is in communications engineering. Error detecting codes were first developed to monitor data transfer on channels. These codes provided the tool to transmit messages reliably over a noisy channel. Hence these codes were optimally adapted to the error models prevalent in channels which are independent single bit errors or error bursts of certain maximum length. For arbitrary multilevel circuits, however, a single fault in the circuit can lead to any number of errors at the output depending on the circuit structure. Thus, though the coding methods developed in communications...
Engineering can be used for concurrent error detection in arbitrary multilevel circuits, they are effective only for circuits with a large degree of regularity such as PLA’s [Mak 82], [Nicolaidis 91] and ALU’s [Pradhan 86], [Lo 92], [Gorshe 96] where the types of error are restricted.

In this paper, we propose a new class of codes termed “weight-based codes” where each output bit is assigned a weight and the check bits represent the sum of the weights of the output bits which have value ‘1’. Note that a Berger code [Berger 61] is a special member of this proposed class of codes where each output bit is assigned a weight of one. We describe the application of these codes for the efficient on-line error detection of arbitrary multilevel circuits. Weight-based codes are shown to be superior to other standard codes used for on-line error detection. Weight assignment has been used to reduce aliasing in a compression technique called accumulator compression testing (ACT) in [Saxena 86].

The highlights of weight-based codes that make it an efficient solution for on-line error detection are:

1. They have high fault detection capability, i.e., the overall probability of detecting any number of erroneous bits at the output caused by a single internal fault in the circuit is high.

2. They present a controlled way of increasing the number of check bits to achieve a desired fault detection capability. The weights assigned to the outputs can be chosen to achieve a desired percentage of fault detection. The fault detection capability will depend on the number of different weights used and the relationship between the weights.

3. Weight-based codes also present a way of exploiting the circuit structure to increase fault detection capability. The fault detection capability of weight-based codes can be enhanced by assigning weights to the output bits based on the circuit structure.

4. A very efficient design exists for the checker. The checker is area and speed efficient, has low power consumption, and can be tested by a small set of incoming code words.

The concurrent error detection of circuits requires the design of a TSC checker. The reliability of the self-checking circuit depends upon the ability of its checker to function correctly despite the occurrence of faults. Further, the checker size has to be kept at a minimum and the speed of operation high. These checker requirements are usually difficult to satisfy.

Recently a novel method for designing highly efficient Berger Code checkers was proposed by Kavousianos and Nikolos in [Kavousianos 98]. These checkers are testable by a small set of code words, are near optimal with respect to the number of transistors required for their implementation, are speed efficient, and have low power consumption. A major advantage of the codes proposed in this paper is that the checker can be designed along the same lines as in [Kavousianos 98] as will be described later. Further, we present a general design for the checker where any existent Berger code checker can be used as a sub-module.

Special synthesis procedures have been proposed to modify the original circuit so as to have only unidirectional faults [Jha 93], [De94], [Saposhnikov 98] or to control the number of erroneous bits at the output by restricting fanout [Touba 94, 97], [Das 98]. While these synthesis procedures are very efficient for medium size circuits, they break down for large circuits. Moreover, in most cases it may be highly undesirable to modify the synthesis of the original circuit as this will affect timing and other crucial parameters. In this paper, we propose a method of on-line error detection that does not require any modification of the circuit to be monitored.

One way to increase the fault detection capability for standard error detection codes (e.g. Berger Codes) is to partition the outputs. Partitioning of outputs and computing check bits per partition will increase the number of faults detected but will also tremendously increase the number of check bits. For example, a circuit with 120 output bits will require 18 check bits if 3 partitions of 40 bits each are made as compared with 7 bits if no partitions are made. Also the area overhead of three checkers with 46 inputs each will be higher than one checker of 127 inputs. The weight-based codes proposed here can be used to increase fault detection capability without partitioning.

The paper is organized as follows: Section 2 explains the proposed weight-based codes. Section 3 compares the fault detection capabilities and size of check bits of different codes. Section 4 explains the totally self-checking checker design. Section 5 concludes the paper.

2. Weight-Based Codes

We propose the concept of weight-based codes where each output bit is assigned a weight and the check bits represent the sum of the weights of the output bits which have value ‘1’. Consider the simplest case of weight-based codes where each output is assigned a weight according to its position, the first output is assigned a weight of 1, the second a weight of 2 and so on. This is an error detecting code that can detect all unidirectional errors as well as any single or double error. By choosing a specific set of weights instead of assigning consecutive weights, the code can be made to guarantee detection of
more than two errors. Note that no aliasing can occur
due to a combination of errors in both the information
bits and check bits since the logic that generates them is
synthesized separately. Any single fault can affect either
the information bits or the check bits but not both.

Weight-based codes have a practical application in
on-line testing. For on-line error detection of multilevel
circuits, we need a code that detects output errors with a
high probability. Two properties are desirable for this:

1. The code should distinguish between 1→0 and 0→1
type errors. Thus aliasing will occur for a certain set
of erroneous bits only if the error in each bit is of a
specific type (1→0 or 0→1).

2. The code should be a “positional” code, i.e., the
check bits are a function of the erroneous bits as well
as their position in the output. For “non-positional”
codes, the check bits are a function of the erroneous
bits alone and not their position in the output. Thus
Hamming code is a “positional” code whereas Berger
and parity codes are “non-positional”. For random
errors, aliasing will be lower for “positional” codes.

Besides high fault detection capability, the error
detecting code should have a low number of check bits
and an efficient TSC checker.

Weight-based codes are both “positional” and
distinguish between 1→0 and 0→1 type errors. Thus they
can be expected to have high fault detection capabilities.
They can always detect all faults that cause only
unidirectional errors at the output irrespective of the
weights assigned.

Aliasing depends upon the weight assignment. An
erroneous output bit (with assigned weight w) will
increase or decrease the total sum of the weights by w,
depending on whether the error type is 0→1 or 1→0
respectively. Aliasing occurs for multiple erroneous bits
where the net effect on the total sum is zero. For
example, consider a fault that causes 1→0 errors in
output bits o1 and o2 and an 0→1 error in the output bit
o3. Let the weights assigned to o1, o2, and o3 be w1, w2, and w3. Aliasing will occur iff (-w1+w2+w3) = 0.

Now, let us compare the fault detection capability of
the proposed weight-based codes with Berger code. The
advantage over Berger code is that being a “positional”
code, it can detect all faults that cause an equal number
of 1→0 and 0→1 errors at the output where the two kinds
of errors do not have the same weights assigned to them.
The disadvantage is that some faults that cause an odd
number of errors at the output may get aliased. This
aliasing can be controlled by the weight assignment.

Fault detection capability is discussed further in
Section 3. Since Berger codes are a special case of
weight-based codes, where each output bit is assigned a
weight of one, the TSC checker for weight-based codes
can be designed as a modification of Berger code TSC
checkers. The TSC checker design is discussed in
Section 4.

3. Fault Detection Capability

In this section, the relationship between the fault
detection capability and the weight assignment scheme is
studied. Further, the fault detection capability of the
proposed weight-based codes is compared with that of
standard systematic error detecting codes that have been
mentioned in self-checking circuit literature. The codes
considered are parity[Goessel93], [De 94], [Touba 94, 97]
and Berger [Jha 93], [Goessel 93], [De 94].

Experiments have been performed to assess the fault
detection capability of the codes. There are two
assumptions. It is assumed that the original circuit is
intact, no modification has been made during synthesis
to increase fault detection capability. Second, the check
bits are synthesized separately from the information bits.
Under the single fault assumption, any fault can affect
either the information bits or the check bits but not both.
Since there is a many-to-one mapping between
information bits and check bits, errors in the check bits
will always get detected whereas errors in the
information bits can get aliased. We shall henceforth
concentrate on faults that affect the information bits, i.e.
the bits at the circuit output.

There are two degrees of freedom in the weight
assignment scheme: the choice of weights, and the
partitioning of the output bits for the weight assignment.
Presented below is a detailed study of the effect of
weight assignment on the fault detection capability. Let
the weight-set denote the set of unique weights used in
the weight assignment scheme. For example, if the
output bits are only assigned weights 2, 3, or 4, then the
weight-set is {2,3,4}.

An important issue is what choice of weights will
have low aliasing without greatly increasing the number
of check bits. There are two kinds of aliasing in weight-
based codes. The first is due to repetition of weights.
Output bits having the same weight assigned to them will
cause aliasing if they have errors in opposite directions.
Increasing the weight-set can reduce this aliasing.
Partitioning of the outputs based on circuit structure also
helps to control this aliasing. We have partitioned
the output bits into clusters such that there is minimal
sharing of logic between output bits in the same cluster.
Output bits in a cluster are assigned the same weight.

The second kind of aliasing is due to the sum of a
group of weights being zero. This aliasing increases
with increase in the weight-set as more such groups are
possible. As an example, consider the weight-set \(\{2,3,4,5\}\). Groups of weights involving at least two different weights, with number of elements less than five and sum of weights equal to zero are \(\{2,2,-4\}\), \(\{2,3,-5\}\), \(\{2,-3,-3,4\}\), \(\{2,-3,-4,5\}\) and \(\{3,-4,-4,5\}\). Aliasing can be controlled by the choice of weights. Note that the weights used should be small to keep the number of check bits low. Thus there is a tradeoff. Increasing the number of different weights used decreases the first kind of aliasing and increases the second.

Experiments have been done on the ISCAS benchmark circuits to compare different weight assignments. These results supplement the deductions about the fault detection capabilities. A set of 10,000 pseudo random test vectors was applied to the benchmark circuits and the fault detection capabilities of the codes measured. For each test vector, all the faults in the circuit were simulated. Faults were simulated one at a time since self-checking circuits have a single fault assumption. If any fault was excited and propagated to the circuit output by a test vector, it was checked whether the error at the output was detectable by the proposed weight-based codes. The fourth column in Table 1 gives the total number of times that faults in the circuit were excited and propagated to the output.

### Table 1. Benchmark Circuits

<table>
<thead>
<tr>
<th>Circuits</th>
<th>Primary Inputs</th>
<th>Primary Outputs</th>
<th>Total Faults Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>36</td>
<td>7</td>
<td>282582</td>
</tr>
<tr>
<td>C499</td>
<td>41</td>
<td>32</td>
<td>1260334</td>
</tr>
<tr>
<td>C880</td>
<td>60</td>
<td>26</td>
<td>167976</td>
</tr>
<tr>
<td>C1355</td>
<td>41</td>
<td>32</td>
<td>1547371</td>
</tr>
<tr>
<td>C1908</td>
<td>33</td>
<td>25</td>
<td>1402446</td>
</tr>
<tr>
<td>C2670</td>
<td>233</td>
<td>140</td>
<td>2934479</td>
</tr>
<tr>
<td>C3540</td>
<td>50</td>
<td>22</td>
<td>2121782</td>
</tr>
<tr>
<td>C5315</td>
<td>178</td>
<td>123</td>
<td>3773836</td>
</tr>
<tr>
<td>C6288</td>
<td>32</td>
<td>32</td>
<td>15745993</td>
</tr>
<tr>
<td>C7522</td>
<td>207</td>
<td>108</td>
<td>5934156</td>
</tr>
</tbody>
</table>

Cases involving different choices of weights, keeping the cardinality of the weight-set constant, are analyzed first. Tables 2, 3 and 4 present experimental results for different choice of weights for weight-sets with cardinality two, three and four respectively.

The variation in fault detection capability in these tables will be entirely due to variations in aliasing of the second kind. Aliasing of the first kind occurs due to repetition of weights and involves output bits having the same weight assigned to them (irrespective of the magnitude of the weights). Thus, keeping the number of different unique weights constant and changing the magnitude of the weights will not vary aliasing of the first kind.

Consider the case where a set of two weights \(\{w_1, w_2\}\) is repeated for the output bits. This will have very low aliasing of the second kind if \(w_1\) and \(w_2\) are chosen to be mutually prime. The lowest number of erroneous bits that can cause aliasing of the second type will be \(w_1+w_2\). Bits of assigned weight \(w_2\) having errors in the opposite direction. As an example, if the weight set \(\{5,6\}\) is repeated, aliasing can occur when six bits of weight 5 have \(1\rightarrow 0\) type error and five bits of weight 6 have \(0\rightarrow 1\) type error or vice versa. Since the multiplicity of errors falls after a certain point, there is a ceiling to the increase in fault detection capability that can be obtained through choosing the weights.

### Table 2. Fault Detection with a Set of Two Weights

<table>
<thead>
<tr>
<th>Circuits</th>
<th>Percentage of Faults Detected with Weights Used</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{1,2}</td>
</tr>
<tr>
<td>C432</td>
<td>95.8</td>
</tr>
<tr>
<td>C499</td>
<td>98.1</td>
</tr>
<tr>
<td>C880</td>
<td>99.0</td>
</tr>
<tr>
<td>C1355</td>
<td>98.4</td>
</tr>
<tr>
<td>C1908</td>
<td>96.8</td>
</tr>
<tr>
<td>C2670</td>
<td>99.2</td>
</tr>
<tr>
<td>C3540</td>
<td>95.9</td>
</tr>
<tr>
<td>C5315</td>
<td>95.6</td>
</tr>
<tr>
<td>C6288</td>
<td>99.6</td>
</tr>
<tr>
<td>C7522</td>
<td>97.2</td>
</tr>
</tbody>
</table>

The results presented in Table 2 are in sync with the analytic deductions. There is a considerable increase in the percentage of faults detected by the weight-set \(\{2,3\}\) as compared to \(\{1,2\}\). However, the increase is significantly lower from \(\{2,3\}\) to \(\{3,4\}\). This indicates that the multiplicity of errors beyond 5 drops sharply for these circuits.

Next consider the case where three weights \(\{w_1, w_2, w_3\}\) are repeated for the output bits. This case will have higher aliasing of the second kind. Aliasing can be kept at a minimum by the following guidelines for the weight assignment: The weights should be mutually prime, and the sum of two weights should not be a multiple of the third. The lowest number of erroneous bits required to cause aliasing for the weight set \(\{3,7,10\}\) is just three since \(3+7=10\). No one weight should be much larger than the other two, example \(\{3,5,11\}\), since the largest weight can easily alias with the smaller two \(\{3+3+5=11\}\).
Increasing the number of weights used increases the groups of weights that alias out. The choice of weights can keep a ceiling on such groups. Thus the weight-set \( \{3,4,5\} \) does not have any group of three weights which alias out, whereas \( \{2,3,4\} \) has. Table 3 shows the resulting increase in the percentage of faults.

### Table 3. Fault Detection with a Set of Three Weights

<table>
<thead>
<tr>
<th>Circuits</th>
<th>Percentage of Faults Detected with Weights Used</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( {1,2,3} )</td>
</tr>
<tr>
<td>C432</td>
<td>96.4</td>
</tr>
<tr>
<td>C499</td>
<td>98.9</td>
</tr>
<tr>
<td>C880</td>
<td>98.7</td>
</tr>
<tr>
<td>C1355</td>
<td>99.0</td>
</tr>
<tr>
<td>C1908</td>
<td>97.8</td>
</tr>
<tr>
<td>C2670</td>
<td>99.7</td>
</tr>
<tr>
<td>C3540</td>
<td>97.0</td>
</tr>
<tr>
<td>C5315</td>
<td>98.1</td>
</tr>
<tr>
<td>C6288</td>
<td>96.3</td>
</tr>
<tr>
<td>C7552</td>
<td>98.0</td>
</tr>
</tbody>
</table>

Table 4 presents the percentage of faults detected with four weights. As there can be considerable aliasing within four weights, some of the circuits show significant increase in fault detection capability from \( \{1,2,3,4\} \) to \( \{2,3,4,5\} \) to \( \{3,4,5,6\} \).

### Table 4. Fault Detection with a Set of Four Weights

<table>
<thead>
<tr>
<th>Circuits</th>
<th>Percentage of Faults Detected with Weights Used</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( {1,2,3,4} )</td>
</tr>
<tr>
<td>C432</td>
<td>97.4</td>
</tr>
<tr>
<td>C499</td>
<td>99.2</td>
</tr>
<tr>
<td>C880</td>
<td>99.6</td>
</tr>
<tr>
<td>C1355</td>
<td>99.4</td>
</tr>
<tr>
<td>C1908</td>
<td>98.8</td>
</tr>
<tr>
<td>C2670</td>
<td>99.6</td>
</tr>
<tr>
<td>C3540</td>
<td>95.3</td>
</tr>
<tr>
<td>C5315</td>
<td>98.6</td>
</tr>
<tr>
<td>C6288</td>
<td>97.2</td>
</tr>
<tr>
<td>C7552</td>
<td>98.6</td>
</tr>
</tbody>
</table>

Table 5 summarizes the results of fault detection. Columns two to four give the percentage of these faults detected by parity, Berger and weight-based codes respectively. Assume that the required fault detection capability is \( \geq 99\% \). An efficient weight-based code was chosen for each circuit based on the results presented in Table 2 through 4. Columns five and six give the particulars of the weight-based code used. Column five gives the weight assignment. Column six gives the number of check bits required; the number of check bits required by Berger Code is given along side in parenthesis.

### Table 5. Comparison of Fault Detection Capability of Error Detecting Codes

<table>
<thead>
<tr>
<th>Circuits</th>
<th>Percentage of Faults Detected by Parity</th>
<th>Berger</th>
<th>Weight Codes</th>
<th>Particulars of Weight Codes</th>
<th>Check Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>69.6</td>
<td>91.3</td>
<td>99.0</td>
<td>( {3,4,5,6} )</td>
<td>5 (3)</td>
</tr>
<tr>
<td>C499</td>
<td>92.4</td>
<td>96.1</td>
<td>99.1</td>
<td>( {2,3,4} )</td>
<td>7 (6)</td>
</tr>
<tr>
<td>C880</td>
<td>91.3</td>
<td>96.9</td>
<td>99.0</td>
<td>( {1,2} )</td>
<td>6 (5)</td>
</tr>
<tr>
<td>C1355</td>
<td>93.3</td>
<td>96.7</td>
<td>99.0</td>
<td>( {1,2,3} )</td>
<td>6 (6)</td>
</tr>
<tr>
<td>C1908</td>
<td>86.0</td>
<td>93.6</td>
<td>99.2</td>
<td>( {2,3,4,5} )</td>
<td>7 (5)</td>
</tr>
<tr>
<td>C2670</td>
<td>85.0</td>
<td>95.4</td>
<td>99.2</td>
<td>( {1,2} )</td>
<td>8 (8)</td>
</tr>
<tr>
<td>C3540</td>
<td>78.6</td>
<td>91.7</td>
<td>99.0</td>
<td>( {3,4,5,6} )</td>
<td>7 (5)</td>
</tr>
<tr>
<td>C5315</td>
<td>81.0</td>
<td>88.5</td>
<td>99.2</td>
<td>( {2,3,4} )</td>
<td>9 (7)</td>
</tr>
<tr>
<td>C6288</td>
<td>69.0</td>
<td>77.0</td>
<td>99.6</td>
<td>( {1,2} )</td>
<td>6 (6)</td>
</tr>
<tr>
<td>C7552</td>
<td>84.5</td>
<td>92.4</td>
<td>99.1</td>
<td>( {3,4,5} )</td>
<td>9 (7)</td>
</tr>
</tbody>
</table>

The results presented in Table 5 show that the fault detection capability of weight-based codes is much higher than that of the other codes. It can be made significantly higher than that of a Berger code (special member of weight-based codes) by a slight increase in the number of check bits. Thus we have presented a class of codes where the fault detection capability can be increased by a controlled increase in the number of check bits.

Recent literature on on-line error detection of multilevel circuits [Jha 93], [De 94], [Das 98], [Touba 94, 97], [Saposhnikov 98] has been focused on modifying the circuit structure based on the code chosen for on-line error detection. In this paper, we have presented a method of choosing an efficient error detecting code from the proposed class of weight-based codes based on the circuit structure.

Currently, the area overhead of the proposed weight-based codes is slightly higher than a Berger code. However, the area overhead can be reduced by taking a modulus of the total sum of weights. For example, the number of check bits will be 3 if we take modulo 8 of the total sum of weights. The fault detection capability in this case will be lower than the values presented in this paper. However, this can be offset by refining the output clusters such that the probability of non-unidirectional faults occurring in a group is minimal.

### 4. TSC Checker

In this section we present two designs for the TSC checker for weight-based codes. The first design is a transistor-level design and is based on the efficient Berger code checker design proposed by Kavousianos and Nikolos in [Kavousianos 98]. The second is a more general design comprising of two or more Berger code
checkers followed by a small adder circuit. Any design for Berger code checker can be used in this design.

4.1 TSC Checker: Transistor Level Design

The design of a totally self-checking checker is crucial to the working of a self-checking circuit. Experimental results in [De 94], [Das 98] have shown that the area overhead of the TSC checker is usually very large. However, a novel method for designing highly efficient Berger code checkers has been proposed in very recent literature [Kavousianos 98]. The highlights of these checkers are:

1. They are near optimal with respect to the number of transistors required for their implementation.
2. All faults, including stuck-at, transistor stuck-open, transistor stuck-on, resistive bridging faults and breaks, are testable by a very small set of code words. The number of code words does not increase with the number of information bits (C-testable).
3. They have efficient speed of operation and low power consumption.

The design is based on a circuit called “\((r,n)\) aggregate-weight threshold circuit” [Kavousianos 98], which have also been used to design \(k\)-order comparators in [Kavousianos 97]. The \((r,n)\) aggregate-weight threshold circuit is a ratioed circuit where the W/L ratios of the transistors (Fig. 2) are chosen such that:

Number of ones in \(X \geq C_1 \cdot 2^{i-1} + C_2 \cdot 2^{i-2} + \ldots + C_0\) gives \(\text{OUT}=1\) else \(\text{OUT}=0\).

![Figure 2. Single Output TSC Checker [Kavousianos 98]](image)

For the Berger code checker, based on the \((r,n)\) aggregate-weight threshold circuit, \(X\) corresponds to the information bits and \(C_{i-1}, C_{i-2}, \ldots, C_0\) correspond to the check bits. When the input vector is a Berger code word the output \(\text{OUT}\) gets the value \((0,1)\) during a period of the signal \(I\) otherwise it gets the value \((1,1)\) or \((0,0)\). The nmos transistors are chosen to have the same width and length, the W/L ratio of the pmos transistor corresponding to \(C_i\) is chosen to give it an equivalent weight of \(2^i\). Thus the aggregate weight of the conductive pmos transistors is: \(C_r \cdot 2^{i-1} + C_0 \cdot 2^{-2} + \ldots + C_r\).

A Berger code can be considered a special case of weight-based codes where each output bit is assigned a weight of one. Thus, the modification of the checker for the general weight-based code would be to ratio the nmos transistors corresponding to each output bit according to its assigned weight. Consider the weight-based code where the weight set used is \{2,3\} i.e. each odd output bit is assigned a weight of two and each even output bit a weight of three. Here, the W/L ratio of the nmos transistors have to be such that the equivalent weight for each odd transistor is two and that for each even transistor is three.

4.2 TSC Checker: General Design

We present an efficient solution for reducing the problem of checker design for the general weight-based code to the Berger code checker design. The output bits are partitioned in a manner such that the problem reduces to computing the number of ones for a group of bits.

The design is illustrated with an example where the weight set used is \{2,3\}, i.e., each odd output bit is assigned a weight of two and each even output bit a weight of three. The binary representation of these bits will require \(\lceil \log_2(3+1) \rceil = 2\) bits. Form partitions corresponding to each bit in the binary representation of the weights. The output bits are distributed in the partitions as follows: an output bit is included in all partitions where its assigned weight’s binary representation is ‘1’ for that bit. There will be two partitions corresponding to the least significant binary bit of weight 2\(^0\) and the higher binary bit of weight 2\(^1\). The partitions formed for this example will be (all output bits with weight 3) and (all output bits with weights 2 and 3) respectively.

Compute the number of ones for each partition using the ones counter’s module in any Berger code checker. For the partition corresponding to the binary bit of weight 2\(^1\), assign a weight of 2\(^1\) to the number of ones computed. Add the number of ones after the weighting to get the final result. Assigning a weight of 2\(^1\), simply means shifting \(i\) bits to the left with respect to the number of ones corresponding to 2\(^1\). It can be trivially verified that the sum so obtained is the sum of the weights whose corresponding information bit is one. The example is illustrated in Figure 3. Outputs with assigned weight \(w_i\) contribute \(1\) to \(\text{bit}_i\); outputs with assigned weight \(w_j\) contribute \(1\) to \(\text{bit}_j\). The ones counter of any Berger code checker design can be used to compute \(\text{sum}_i\) and \(\text{sum}_j\).
The design can easily be generalized for any set of weights. Thus we have effectively reduced the problem of designing TSC checkers for weight-based codes to the problem of designing a Berger code checker. The modules in the proposed weight-based code checker are Berger code checkers followed by a small adder block. The TSC property of the checker follows from the TSC property of the Berger code checker.

![Diagram of Berger code checker]

Figure 3. Reduction of the Problem of Summing Weights to Counting Number of Ones

5. Conclusion

This paper has presented the concept of weight-based codes, which can be considered a generalization of Berger codes. Existing checker designs for Berger codes can be modified to design checkers for weight-based codes. Weight-based codes present a controlled method of increasing the number of check bits over the number required by Berger codes to achieve a desired fault detection capability. Experimental results have been presented which show the superior fault detection capability of weight-based codes as compared to standard error detecting codes.

In this paper, we have presented an approach of choosing an efficient error detecting code from the proposed class of weight-based codes based on the circuit structure. In several scenarios the original circuit structure cannot be modified. For example, reuse of legacy designs having optimized layout, or custom design of circuits with little or no automation of synthesis. In such cases, the approach presented in this paper is the effective solution for on-line error detection.

Acknowledgments

This work is part of the ROAR project and is based upon work supported in part by the Defense Advanced Research Projects Agency (DARPA) under Contract Number DABT63-97-C-0024.

References


