ERROR DETECTION AND CORRECTION FOR ADDITION AND SUBTRACTION, THROUGH USE OF HIGHER RADIX EXTENSIONS OF HAMMING CODES

by
James E. Robertson

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

Abstract

The properties of Hamming codes for error detection and correction can be extended from the binary parity check to addition, modulo 2r. Malfunctions in hardware during addition, modulo 2r, can be detected and corrected. Since carry-save and signed-digit addition, radix r, are included in addition, modulo 2r, this extension of Hamming codes makes possible new techniques for detection and correction of hardware malfunctions during signed-digit and carry-save addition.

1. Introduction

Binary Hamming codes are currently limited almost exclusively to detection and correction of errors during transmission or storage of information. Little or no use is made of the fact that binary Hamming codes can be used for detection and correction of errors resulting from hardware malfunctions for the Boolean operations of EXCLUSIVE OR and complementation.

It is the purpose of this paper to indicate that errors resulting from malfunctions in hardware for performing addition and subtraction in radix r, can be detected and corrected through use of Hamming codes extended to addition, modulo 2r. The first observation is the obvious one that the parity check, the Boolean EXCLUSIVE OR, addition modulo 2, and subtraction modulo 2, are identical operations. The generalization used here is therefore from addition, modulo 2 to addition, modulo 2r. The second observation is that the fact that binary Hamming codes can be used for detection and correction of hardware malfunctions during addition modulo 2 can be generalized to extended Hamming codes being used for detection and correction of hardware malfunctions during addition, modulo 2r. The third observation is that the operations of carry-save and signed-digit arithmetic, radix r, are subsets of addition, modulo 2r.

The details of the generalization for Hamming codes are given in Section 2. In particular, it is shown that the generalization depends only on the associativity of addition, modulo k.

In section 3, the discussion is particularized in several ways. The Hamming codes are extended to addition, modulo 4, for checking binary arithmetic. The eight digit single error correcting, double error detecting code is described in detail. In section 4, a specific numerical example is discussed for the radix 4 single error correcting, double error detecting extended Hamming code.

In section 5, the considerations determining the representation of a radix 4 digit by two binary digits are given. This lends to a specific logical design. Section 6 summarizes the problems encountered when negative digital values are employed.

2. Extension to Addition modulo k.

Let X represent an n-tuple of digits X_i (i=0,1,2,...,n-1) in radix k. One or more of the digits X_i, say X_j, is not independent, but is related to the other digits by the expression:

$X_j=\sum_{s_j} X_i$ (modulo k), where s_j designates a subset of all possible values of i, excluding j. The n-tuple Y and Z are similarly defined, with digits Y_j and Z_j related by addition (modulo k) of similarly designated subsets of Y and Z, respectively,

Consider the transformation Z = X + Y modulo k, which means $Z_i = X_i+Y_i$ modulo k for every i. In particular, Z_j is formed as X_j+Y_j, modulo k. Therefore $Z_j=X_j+Y_j$ (modulo k) $= \sum_{s_j} X_i+\sum_{s_j} Y_i$ (modulo k). By associativity of addition modulo k, $Z_j=\sum_{s_j} (X_i+Y_i)$ (modulo k) $= \sum_{s_j} Z_i$ (modulo k). Therefore, in the absence of error, the error protection properties associated with X_j and Y_j apply also to Z_j.

A check digit C_j is generated as $C_j=-Z_j+\sum_{s_j} Z_i$. In the absence of an error in the digits Z_i (i=j, or $i\epsilon s_j$), the check digit C_j is zero.

All of the equations in this section reduce to the binary Hamming codes if k = 2 and if it is recognized that addition modulo 2 and subtraction modulo 2 are the same.

3. Extension to Addition (modulo 4) of The Single Error Correcting, Double Error Detecting Hamming Code, With n = 8.

For the single error correcting, double error detecting Hamming code, the equations of the previous section for the extension to addition (modulo 4) become: n=8, i=0,1,2,3,4,5,6,7

CH2419-0/87/0000/0226$01.00 © 1987 IEEE

j=0,1,2,4 $\qquad s_0 = 1,2,3,4,5,6,7$

$s_1 = 3,5,7 \qquad s_2 = 3,6,7 \qquad s_4 = 5,6,7$

$$X_0=\sum_{i=1}^{7} X_i \text{ modulo } 4 \qquad \text{Eqn 3.1a}$$

$$X_1=X_3+X_5+X_7 \text{ modulo } 4 \qquad \text{Eqn 3.1b}$$

$$X_2=X_3+X_6+X_7 \text{ modulo } 4 \qquad \text{Eqn 3.1c}$$

$$X_4=X_5+X_6+X_7 \text{ modulo } 4. \qquad \text{Eqn 3.1d}$$

Similar expressions are applicable for the Y_i

If each Z_i (i=0,1,2,...,7) is defined as $Z_i=X_i+Y_i$ (modulo 4), then, for example,

$Z_1=X_1+Y_1$ (modulo 4)

$= X_3+X_5+X_7+Y_3+Y_5+Y_7$ (modulo 4)

$= (X_3+Y_3) + (X_5+Y_5) + (X_7+Y_7)$ (modulo 4)

$= Z_3+Z_5+Z_7$ (modulo 4)

The associated check digit C_1 is

$C_1=-Z_1+Z_3+Z_5+Z_7$ (modulo 4)

For error correction and detection, assume that the n-tuples X and Y have been checked and are free of error. An error E_i ($E_i \neq 0$, modulo 4) is introduced for one value of i during the addition $Z_i=X_i+Y_i$ modulo 4, so that $Z_i=X_i+Y_i+E_i$ (modulo 4). When the check digits C_j are generated from the Z_i, there are two cases:

1) i=3,5,6, or 7

$C_0 = E_i$				Eqn 3.2.1a
$C_1 = E_i$	(i = 3,5,7)	$C_1 = 0$	(i=6)	Eqn 3.2.2a
$C_2 = E_i$	(i=3,6,7)	$C_2 = 0$	(i=5)	Eqn 3.2.3a
$C_4 = E_i$	(i = 5,6,7)	$C_4 = 0$	(i = 3)	Eqn 3.2.4a

2) i= 0,1,2,4

$C_0 = E_i$	(i = 1,2,4)	$C_0 = -E_0$	Eqn 3.2.1b
$C_1 = 0$	(i = 0,2,4)	$C_1 = -E_1$	Eqn 3.2.2.b
$C_2 = 0$	(i = 0,1,4)	$C_2 = -E_2$	Eqn 3.2.3b
$C_4 = 0$	(i = 0,1,2)	$C_4 = -E_4$	Eqn 3.2.4b

In the first case (i = 3,5,6,7), a single error is indicated by the value of the error E_i in C_0, and by the same value E_i in two or more of C_1, C_2, and C_4 in a pattern indicative of the value of i, and hence the location of the error. In the second case (i=0,1,2,4) a single error is indicated by the value $-E_i$ in C_i. For i=0, the only error indication is $C_0=-E_0$; for i=1,2 or 4, the error is indicated by $C_0=E_i$ and $C_i=-E_i$.

4. A numerical example.

A numerical example of the binary addition 7+5+3 is given in table 4.1. For addition purposes, column 7 is assigned binary weight 8, column 6 is weight 4, column 5 is weight 2, and column 3 is weight 1. Row 1 is therefore the binary operand 7, row 2 is the operand 5, and row 9 is the operand 3. For these rows, the values in columns 4,2,1, and 0 are obtained by addition, modulo 4, of appropriate subsets of columns 7,6,5, and 3, in accordance with equations 3.1. It is assumed that all digits in rows 1,2,4 and 0 have been checked and are free of error, within the limitations of the error correction procedure in use.

A single binary addition requires the use of two simpler binary operations; the half-adder $b^0 \leftarrow a^0+a^0$, followed by the carry generator $2a^0+a^0 \leftarrow b^0+a^0$ (reference 9.2). Theoretically, each digit in row 3 could be obtained from the corresponding digits of rows 1 and 2 by addition modulo 4; in practice, the arithmetic digits of columns 7,6,5, and 3 would be obtained using the simple hardware of the half adder $b^0 \leftarrow a^0+a^0$. Malfunctions in the hardware used to generate the digits Z_i, of row 3, should be detectable and correctable (equations 3.2), within the limitations of the error correction method in use.

The X_i of row 4 are the Z_i of row 3, and the Y_i of row 5 are zeroes. The addition of the operands 7 and 5 is completed by combining rows 4 and 5 to form row 6, using carry generators for the arithmetic digits of columns 7,6,5, and 3, and using addition, modulo 4, for the check digits of columns 4,2,1, and 0. The sum, 12, is represented by the digits in row 6, as 8x0+4x2+2x1+1x2.

It is imperative that the digits in row 6 be checked and made free of error. Each of the radix 4 digits in columns 7,6,5, and 3 of row 6 are split into two binary digits, one a carry which is shifted left to appear in row 7, and the other a binary sum which appears in row 8. For example, the 2 in row 6, column 6, becomes the carry of 1 in row 7, column 7, and the binary sum digit 0 in row 8, column 6. Thus the value 12, represented as binary-weighted radix 4 digits in row 6, becomes two conventional binary numbers whose values are 10 in row 7, from the shifted carries, and 2 in row 8, the binary sum digits. Because of the shift of the carries, it is necessary at this point to calculate, using equations 3.1, the digits of columns 4,2,1, and 0, of rows 7 and 8.

Row / i	7	6	5	3	4	2	1	0
1 X_i	0	1	1	1	2	2	2	1
2 Y_i	0	1	0	1	1	2	1	2
	b^0 <-- a^0 + a^0				c^0 = c^0 + c^0 (mod. 4)			
3 Z_i	0	2	1	2	3	0	3	3
4 X_i	0	2	1	2	3	0	3	3
5 Y_i	0	0	0	0	0	0	0	0
	$2a^0$ + a^0 <-- a^0 + b^0				c^0 = c^0 + c^0 (mod. 4)			
6 Z_i	0	2	1	2	3	0	3	3
7 K_i	1	0	1	0	2	1	2	3
8 X_i	0	0	1	0	1	0	1	3
9 Y_i	0	0	1	1	1	1	2	2
	b^0 <-- a^0 + a^0				c^0 = c^0 + c^0 (mod. 4)			
10 Z_i	0	0	2	1	2	1	3	1
11 X_i	0	0	2	1	2	1	3	1
12 Y_i	1	0	1	0	2	1	2	3
	$2a^0$ + a^0 <-- a^0 + b^0				c^0 = c^0 + c^0 (mod. 4)			
13 Z_i	1	0	3	1	0	2	1	0
14 K_i	0	1	0	0	1	1	0	3
15 X_i	1	0	1	1	2	2	3	2

Table 4.1 A numerical example of binary addition.

Thereafter, the addition of the third operand, of value 3, is performed in a manner similar to that of rows 1 through 8. The sum digits in row 8, of numerical value 2, are added to the third operand of value 3 in row 9, using a half adder, to generate the sum of value 5 in row 10. This sum, copied without change to become row 11, is added to the carry digits of value 10 in row 12, copied from row 7. The sum, whose value is 15 is represented as binary-weighted radix 4 digits in row 13, and collectively by the two binary numbers, the shifted carries of value 4 in row 14, and the binary sum digits of value 11 in row 15.

5. Representation of digits and logical design.

The problem of representation is one of establishing a correspondence between the four values of the radix 4 digit set $c^0 = \{0,1,2,3\}$ and the four states 00,01,10 and 11, of two binary variables η and y. Clearly, there are $4! = 24$ possible correspondences, which fall into 3 groups of 8 each under permutation and negation of η and y. For addition, it is reasonable to impose the condition that the value 0 of digit set c^0 shall be represented by state 00. This reduces the number of correspondences to the six shown in Table 5.1, which form three pairs of correspondences under permutation of η and y.

η	y	c_a^0	c_b^0	c_c^0	c_d^0	c_e^0	c_f^0
0	0	0	0	0	0	0	0
0	1	1	3	1	2	2	3
1	0	2	2	3	1	3	1
1	1	3	1	2	3	1	2

Table 5.1
Correspondencies between four states and four values.

Under permutation of η and y, $c_a^0 = c_d^0, c_b^0 = c_e^0$, and $c_e^0 = c_f^0$. Since permutation reduces to a matter of nomenclature, further consideration is given only to c_a^0, c_b^0, and c_c^0.

The final consideration which determines the correspondence is that of separability of the radix 4 digit c^0 into two binary digits, a carry and a sum, as exemplified by the generation of rows 7 and 8 from row 6, (and of rows 14 and 15 from row 13) in Table 4.1 Separability requires the relationship $c^0 = 2\eta + y$, which indicates that the correspondence indicated by column c_a^0 in Table 5.1 should be used.

The other correspondences c_b^0 and c_c^0 can be made separable with very little hardware (an EXCLUSIVE OR gate in each case), but this hardware would have to be used at a particularly critical point. With reference to Table 4.1, the digits c^0 in row 6 (or row 13) can be checked and corrected, implying that the individual binary digits are also correct. If the correspondence is separable, it is reasonable to assume that the binary digits in rows 7 and 8 (or rows 14 and 15) are correct, and that it is safe to compute the check digits of columns 4,2,1, and 0 for these rows.

The logical design for a radix four adder (mod. 4) is completely determined once the correspondence c_a^0 of Table 5.1 is chosen. In terms of states, the addition table is that of Table 5.2.

		y_i (η_i, y_i)				
		00	01	10	11	
	00	00	01	10	11	
X_i	01	01	10	11	00	sum z_i
(ξ_i, x_i)	10	10	11	00	01	(ζ_i, z_i)
	11	11	00	01	10	

Table 5.2
Addition table for $c_a^0 = C_a^0 + c_a^0$ (mod.4)

Table 5.2, in turn, can be converted directly into the truth tables necessary for logical design, and leads to the design equations:

$$\zeta_i = \xi \oplus \eta_i \oplus x_i y_i \qquad \text{Eqn. 5.1}$$

$$z_i = x_i \oplus y_i \qquad \text{Eqn. 5.2}$$

The notation for these equations is given by $Z_i = X_i + Y_i$, modulo 4, $Z_i = 2\zeta_i + z_i$, $X_i = 2\xi_i + x_i$, and $Y_i = 2\eta_i + y_i$.

All of the additions (modulo 4) implicit in the operations of Table 4.1 are of the type given by Equations 5.1 and 5.2.

6. Extension to negative values and subtraction.

If all variants of half adders and carry generators are to be used, consideration must be given to digit sets with negative values, as exemplified by those in Tables 6.1 and 6.2.

η	y	c_a^0	c_a^1	c_a^2	c_a^3
0	0	0	0	0	0
0	1	1	1	1	$\bar{3}$
1	0	2	2	$\bar{2}$	$\bar{2}$
1	1	3	$\bar{1}$	$\bar{1}$	$\bar{1}$

Table 6.1. Format a for four-valued digit sets.

η	y	c_b^0	c_b^1	c_b^2	c_b^3
0	0	0	0	0	0
0	1	3	$\bar{1}$	$\bar{1}$	$\bar{1}$
1	0	2	2	$\bar{2}$	$\bar{2}$
1	1	1	1	1	$\bar{3}$

Table 6.2. Format b for four-valued digit sets.

In the previous section, it was noted that separability requires that the format choice $c_a^0 = 2\eta + y$ be made. Similarly, for digit sets with negative values, separability necessitates the unique choices $c_b^1 = 2\eta - y$ $c_a^2 = -2\eta + y$, and $c_b^3 = -2\eta - y$. To summarize, format a must be used for even values of offset and format b must be used when the offset is odd.

Consider Table 5.2, the addition table for $z_i = x_i + y_i$, modulo 4, when the digit sets for z_i, x_i, and y_i, are all of type c_a^0. It can be shown that when z_i, x_i and y_i are all of format a, regardless of offset, then table 5.2 applies, and therefore the same logical design of equations 5.1 and 5.2 should be used. Similiarly, table 5.2 and equations 5.1 and 5.2 are valid if z_i, x_i, and y_i are all represented using format b. If the formats of z_i, x_i, and y_i are mixed, then a different addition table, and hence a different logical design, will result.

It is proposed that all adders, modulo 4, used shall be os the same logical design. This choice necessitates that the inputs to a carry generator must be of the same format as

b^0	α	a	c_a^0	η	y	c_b^1	η	y
0	0	0	0	0	0	0	0	0
1	0	1	1	0	1	1	1	1
2	1	0	2	1	0	$\underline{2}$	1	0
			3	1	1	1	0	1

Table 6.3 Two interpretations of digit set b^0.

that of imposed by the output. Table 6.3 is an example to illustrate that this is possible. For format a, $b^0 \epsilon c_a^0$, and the adder inputs are $\eta = \alpha$ and y = a. For format b, $b^0 \epsilon c_b^1$, and the adder inputs are $\eta = \alpha$va and y = a. The fact that all inputs to half adders and carry generators are either two-valued or three-valued indicates that an interpretation for either format choice can always be made.

7. Comments on the Radix 4 Example

Of the two bits which comprise a radix 4 digit, the least significant, or sum bit, is also a parity bit in the sense of the binary Hamming code, for the particular representations chosen in the example. Thus the properties of the binary Hamming code apply, independent of the most significant, or carry, bit.

With proper account taken of the negative values in equations 3.2, it seems possible to identify both the amount and the location of any single error E_i, and hence to correct it. It also seems likely that a single sum error $E_i = 1$ and a single carry error $E_j = 2$, can both be corrected, whether or not i=j. When $E_i = 1$ and $E_j = 1$, with i≠j, the consequent carry indications provide additional information not available for the binary Hamming code. This case warrants further investigation.

The hardware cost of generating the carry digits in the adders modulo 4 is roughly twice as great as the binary parity check. Another disadvantage is that, for generality, two different interpretations of the same digit set are necessary, as exemplified by table 6.3.

8. Conclusions

The previous sections represent a preliminary investigation into higher radix extensions of Hamming codes. Many directions for further investigation suggest themselves.

It is also evident that all the Boolean operations, when performed in parallel, can be checked and corrected, since the AND operation is implicit in the half adder, and the NOT operation is a special case of the EXCLUSIVE OR.

The approach here also suggests that the binary parity check be considered a count modulo 2 of the errors that have occurred, and suggests that the effect of counting the errors to a greater value than two be investigated.

The extensive research inspired by Hamming's original discoveries should be reviewed with higher radix applications to arithmetic in mind.

9. Bibliography

9.1 Hamming, R. W.; Error Detecting and Error Correcting Codes, Bell System Technical Journal, Volume XXIX (April, 1950) pp. 147-160.

9.2 Robertson, J.E. A Systematic Approach to the Design of Structures of Arithmetic, Proc. 5th Symposium on Computer Arithmetic, May 1981.

Acknolwedgement:

This research was supported in part by the National Science Foundation under grant NSF MCS 83-08576.