International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

Checkbit Prediction for Logic Functions By Using Dong's Code Method

Dinesh Babu .N[1], Ramani .G[2]

[1]PG Scholar, Department of EEE, Nandha Engineering College, Erode, India

[2]Associate Professor, Department of EEE, Nandha Engineering College, Erode, India

Abstract: ***Detecting errors in arithmetic and logic operations is very important, and the simplest way is by using Duplication. However, the use of duplication is very expensive. In this arithmetic and logic function the errors are detected by using check bit prediction schemes. The only available technique that can be used to detect the three types of errors (transfer errors, arithmetic errors[12], and logic errors) where only one non-arithmetic code for example Parity code, Berger code, Dong's code. In this paper the mathematical equations for the check bit prediction of the logical functions using Dong's code are outlined.***

Keywords: Duplication, Parity code, Berger code, Dong's code, Check bit prediction

1. Introduction

The arithmetic and logic functions are performed by the general purpose processors. This processor is also having a self checker, using the Concurrent Error Detection (CED) techniques; it can detect the errors in arithmetic and logic units. In the earlier stage they are using the duplication process. It is one of the easiest ways of detecting errors. The duplication is very expensive. If the check symbol is same size as the information bit. The number of information bit is large so this process not is not widely used. In this arithmetic function the errors are detected by using arithmetic code [1][2][3]. Because of the inherent difference between arithmetic errors and logical errors, arithmetic codes are inefficient in detecting logical errors [4].

The Two-rail Code method is used only for check the logical operation[5]. This process is used for self testing and fault secure for all faults affecting in this logical and arithmetic unit. But the fault can occur in both unit are undetectable. The redundancy is 100%, plus the hardware overheads for the checker and synchronization. It is preferable to use the same code for checking both arithmetic and logic operations. The only available technique that can be used to detect the three types of errors (transfer errors, arithmetic errors, and logic errors) is the Check Symbol Prediction (CSP) technique, where only one non-arithmetic code for example Parity code[6], Berger code[7], Dong's code[8], Residue codes [10], and Berger codes [11 is used for error detection throughout the entire system.

2. Code Composition

For constructing Dong's code we want to set the maximum weight of unidirectional errors. The check symbols have two parts that are C1, C2. The number of bits in C1 is i, where i = [$\log_2$ (m + 1)].C1 will count the number of zeros in the information bit and using the j formula,C2 will counting the number of zeros in C1 result. If the C2 will saving the bits.

Here we are using a 32 bit information word. So we assume
a- Maximum weight (m) of unidirectional errors needed to be detected is 7.
b- Number of bits in C1 will be i = [$\log_2$ (m+1)] = 3 from the above the code can be constructed as follows:

1) Count the number of zeros in the information word.
2) Take modulo (m+1) of the number of zeros in the information word.
3) Represent the result in binary form using i - bits (j=3) , to obtain C1.
4) Count the number of zeros in C1 and represent it in binary form, which represents C2.

Table 1: Encoding Example

Information bits	Number of zeros	0s mod (m+1)	CK1	CK2
0000……0000	32	0	000	11
0000……0001	31	7	111	00
0000……0011	30	6	110	01
0111……1111	1	1	001	10
1111……1111	0	0	000	11

3. Error Detection Capability

The code detects all unidirectional errors except those which affect only the information bits and have weight equal to (m+1) or its multiples [8] . In other words if m is set to 7, the number of information bits is 32 , and the errors affect only the information bits, then the code can detect any unidirectional error of weight not equal to 8,16 24 and 32, but all other weight can be detected .

The code can also detect some other types of errors. For example, if the check bits are affected by any number of unidirectional errors then the code can detect all types of errors (unidirectional or bi-directional errors which affect the information bits) this comes from the fact that the check symbols of the code form a set of unordered words in which no check symbol can be changed into another by any unidirectional errors; this is an advantage over the Berger code itself.

Licensed Under Creative Commons Attribution CC BY

4. Check Symbol Prediction

Check symbol prediction is one of the schemes used to perform concurrent error detection in arithmetic functions, and it has been considered for some codes, but from literature survey to date, however, the only prediction method designed for Dong's code is for arithmetic functions [12]. In this section the mathematical equations for the check symbol prediction for Logic functions using Dong's code, will be outlined. The equations are based on the mathematical foundation for the prediction of Berger codes described in [11].

Given a logic operation S = X op Y , where the operands X and Y are coded into Dong's code . Let X_{Ck1} and Y_{Ck1} be the Ck1 check symbols of the operands. The predicted Ck1 of the result (S_{Ck1}) can be computed form X, Y, X_{Ck1}, and Y_{Ck1}, i.e.S_{Ck1} = f (X, Y, X_{Ck1}, Y_{Ck1}) only Ck1 of the check symbol of the result needs to be predicted, Ck2 is then generated from Ck1.

5. CSP for Logic Operation

This section describes the CSP technique for Logic functions for Dong's code, the CSP equations used are based on the equations for the Logic functions for Berger Code presented in [11].

The three basic logical operations are OR (∨), AND(∧), and the Exclusive OR (⊕), the function of the three operations are shown in Tables (2-4). The last column of each table is used for the number of ones lost; a one is said to be lost if it appears at the input of the gate, but it does not propagate to the output of the gate. For the OR gate when two ones appear at the inputs, only one of them propagates to the output, and the other is lost . If the lost column in the truth table of the OR gate is compared with the output column (S_i) of the AND gate, the two columns are exactly the same. In other words the number of ones in the result of the OR operation is equal to the number of ones in the inputs (X_i,Y_i) minus the number of ones in the result of AND operation .

For the AND gate, if there are two ones at the inputs of the gate, then only of them propagates to the output, and the other is lost, and if there is only one 1 at the inputs, that one will be lost too . The last column in the truth table of the AND Gate is exactly the same as the output column in the truth table of the OR gate (S_i).

The XOR gate is different, when there are two ones at the inputs of the XOR gate, then the two ones cannot be propagated, and both are lost. The number of ones lost is exactly equal to the double he ones propagated by the AND gate (S_i column of AND gate) .

Form above, we can verify the following:

$X_i \vee Y_i = X_i + Y_i - (X_i \wedge Y_i)$

$X_i \wedge Y_i = X_i + Y_i - (X_i \vee Y_i)$

$X_i \oplus Y_i = X_i + Y_i - (X_i \wedge Y_i)$

In general, if the two operands X and Y have n-bits each, and number of ones in X is N (X), the number of ones in Y is N(Y), then the number of ones in the result of the three Logical operations can be predicted as follows:

$N(X \vee Y) = N(X) + N(Y) - N(X \wedge Y)$ (1)

$N(X \wedge Y) = N(X) + N(Y) - N(X \vee Y)$ (2)

$N(X \oplus Y) = N(X) + N(Y) - 2xN(X \wedge Y)$ (3)

The above equations are the basic equations used for check symbol prediction for Berger Code [11], but in Dong's Code, only modulo 2^r of the number of Zeros are used to generate the Ck1 of the check symbol, therefore, the above equation can be modified as shown below :

$N(X \vee Y) \bmod 2^r = N(X) \bmod 2^r + N(Y) \bmod 2^r - N(X \wedge Y) \bmod 2^r$ (4)

$N(X \wedge Y) \bmod 2^r = N(X) \bmod 2^r + N(Y) \bmod 2^r - N(X \vee Y) \bmod 2^r$ (5)

$N(X \oplus Y) \bmod 2^r = N(X) \bmod 2^r + N(Y) \bmod 2^r - 2_X N(X \wedge Y) \bmod 2^r$ (6)

Table 2: Truth table for OR Operation

X_i	Y_i	S_i	1's in X_i and Y_i	1's in S_i	1's Lost
0	0	0	0	0	0
0	1	1	1	1	0
1	0	1	1	1	0
1	1	1	2	1	1

Table 3: Truth table for AND Operation

X_i	Y_i	S_i	1's in X_i and Y_i	1's in S_i	1's Lost
0	0	0	0	0	0
0	1	0	1	0	1
1	0	0	1	0	1
1	1	1	2	1	1

Table 4: Truth table for XOR Operation

X_i	Y_i	S_i	1's in X_i and Y_i	1's in S_i	1's Lost
0	0	0	0	0	0
0	1	0	1	1	0
1	0	0	1	1	0
1	1	1	2	0	2

5.1 CSP for OR Operation

The check symbol of the result (S), where:

S = X ∨ Y, can be predicted from the check symbol of the operands (X, Y) and (X ∧ Y) as shown in equation (4).

In other words the number of Zeros in the result of the OR operation depends on the number of ones in the two operands (X and Y) , and the number of ones in X ∧ Y. However, only the first part of the check symbol Ck1, needs to be predicted, and then the second part of the check symbol Ck2 is obtained by simply counting the number of zeros in Ck1. Check symbol of the result Ck1 of the OR operation can be predicted by the following equation:

$Ck1 = (2^r + X_{Ck1} + Y_{Ck1} - (X \wedge Y)_{Ck1}) \bmod 2^r$ (7)

Example : Table 5 shows the OR operation of X and Y , where X and Y are two 32 bit numbers, if these two numbers are encoded into Dong's Code, and r = 3 , then to predict the result of the OR operation, an AND operation has to be

Licensed Under Creative Commons Attribution CC BY

performed on X and Y, the number of Zeros mod 2^r of the AND operation will be used with the Ck1 of the two operands (X, Y) to predict Ck1 of the check symbol of the result of the OR operation Ck2 can be then obtained by counting the number of Zeros in the predicted Ck1 .

X_{Ck1} = number of Zeros in X mod 2^3
= 17 mod 8 = 1
Y_{Ck1} = number of Zeros in Y mod 2^3
= 13 mod 8 = 5
$(X \wedge Y)_{Ck1}$ = number of Zeros in (X and Y) mod 2^3
= 25 mod 8 = 1
Ck1 of the check symbol of the result can be predicted by substituting in equation (7),
$Ck1 = (2^r + X_{Ck1} + Y_{Ck1} - (X \wedge Y)_{Ck1}) \bmod 2^r$
= (8 = 1 = 5 – 1) mod 8
= 5, in binary = 101

The last row in Table 2 shows the result of the OR operation, the result contains exactly 5 Zeros, which means Ck1 of the result is 101 in binary, and it is equal to the predicted Ck1 of the result of OR operation. The second part of the check symbol Ck2 is equal to the number of Zeros in the predicted Ck1=01, and the complete check symbol = 01 101.

5.2 CSP for AND Operation

The first part Ck1 of the check symbol of the AND operation (X ∧ Y), can be predicted based on equation (2), using the check symbol of the two operands (X, Y) and (X ∨ Y) .

$$Ck1 = (2^r + X_{Ck1} + Y_{Ck1} - (X \vee Y)_{Ck1}) \bmod 2^r \quad (8)$$

Example : if the AND operation, needs to be performed on X and Y, where the values of X and Y are shown in Table 6 then to predict Ck1 of the check symbol of the result of the OR operation (X ∨ Y) should be evaluated, the number of Zeros in the result of (X ∨ Y) mod 2^r is used with the Ck1 of the operands X and Y, to predict Ck1 of the result of (X ∧ Y) the equation given below can be used :

X_{Ck1} = number of Zeros in operand X mod 2^3
= 18 mod 8 = 2
Y_{Ck1} = number of Zeros in operand Y mod 2^3 = 20 mod 8 = 4
$(X \wedge Y)_{Ck1}$ = number of Zeros in (X ∨ Y) mod 2^3 = 14 mod 8 = 6
Then, $Ck1 = (2^r + X_{Ck1} + Y_{Ck1} - (X \vee Y)_{Ck1}) \bmod 2^r$
= (8 + 2 + 4 – 6) mod 8 = 0 , in binary = 000
The check bits of $(X \wedge Y)_{Ck1}$ = 11 000 .

5.3 CSP for Complement Operation

The complement operation X is performed by complementing all the bits of X bit by bit, which means the number of zeros in X is equal to the number of ones in X, and the number of ones in X is equal to n-X_{Ck1}; where n is the number of bits in X. Therefore Ck1 of the check symbols of the result of the complement operation can be predicted as:

$$Ck1 = X_{Ck1} = (2^{j+1} - X_{Ck1}) \bmod (m+1) \quad (9)$$

5.4 CSP for XOR Operation

To predict the check symbol of the Exclusive OR (XOR) operation, an AND operation should be performed. Thereafter Ck1 of the two operands, X and Y, twice the number of Zeros in (X ∧ Y) and the number of bits (n) in the information word, are used to predict Ck1 of the operation (X⊕ Y) by substituting in the equation (3) . It should be noted that the number of bits n is used to ensure that the Ck1 is always positive.

$$Ck1 = (2^r + X_{Ck1} + Y_{Ck1} - 2(X \wedge Y)_{Ck1} + n) \bmod 2^r \quad (10)$$

Table 5: Example of Check symbol prediction of OR operation

X	1100 0101 0000 0011 1111 0011 1010 0001	Zeros in X=17
Y	0111 1010 1100 1010 0101 1111 0101 0011	Zeros in Y= 13
X^Y	0100 0000 0000 0010 0101 0011 0000 0001	Zeros in X^Y=25
XVY	1111 1111 1100 1011 1111 1111 1111 0011	Zeros in XVY=5

Table 6: Example of Check symbol prediction of AND operation

X	1100 1000 0101 0010 1101 0011 0010 1001	Zeros in X=18
Y	0100 1010 0100 0110 0101 1001 0100 0001	Zeros in Y= 20
XVY	1100 1010 0101 0110 1101 1011 0110 1001	Zeros in XVY=14
X^Y	0100 1000 0100 0010 0101 0001 0000 0001	Zeros in X^Y=24

5.5 CSP for Logical Shift

Two logic shift operation can be performed on an operand X, during the logical shift all the bits of X will be shifted to the left or to the right one place, one bit of X will be moved out to the output carry C_{out}, and the input carry C_{in} shifted in, if the C_{in} is set then a 1 is loaded in, but if C_{in} is reset then a 0 is shifted in .

5.5.1 Logic Shift Left

If $X = (x_n, x_{n-1}, \ldots, x_1)$ is to be shifted to the left, then all the bits of X are shifted to the left one place, and X becomes, $X=(x_{n-1},x_{n-2},\ldots,x_1,C_{in})$ and $C_{out} = x_n$.

Ck1 of the result of Logic Shift Left operation can be predicted as follows:

$$Ck1 = (X_{Ck1} + C_{out} - C_{in}) \bmod 2^r \quad (11)$$

5.5.2 Logic Shift Right

When $X = (x_n,\ldots,x_2,x_1)$ is shifted to the right, then the least significant bit is shifted out to the output carry and C_{in} is loaded into the most significant bit, the operand becomes $X=(C_{in},x_n,\ldots,x_2)$ $C_{out} = x_1$.

The check symbol of the operation can be predicted using the equation below:

$$Ck1 = (X_{Ck1} + C_{out} - C_{in}) \bmod 2^r \quad (12)$$

Licensed Under Creative Commons Attribution CC BY

In summary the above Equations are the Check Symbol Prediction equations used to predict the check symbol of the most common logic operations.

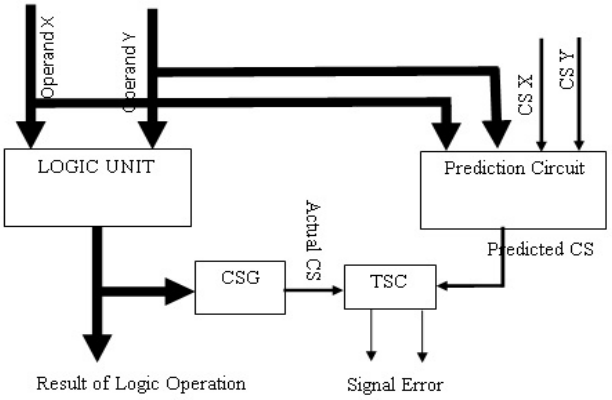
Other operations for example, NOR, NAND, NOR… can be obtained by using the prescribed operations. These equations with equations presented in [12] will be used in the check symbol prediction hardware for a 32-bit Self-Checking RISC (SC-RISC) Processor using Dong's Code.

6. Check Symbol Prediction Circuit

Fig(1) shows block diagram of the circuitry which generates the predicted check symbols Ck1 and Ck2, the circuit also generates the actual check symbols Ck1 and Ck2 from the result. When the result becomes available from the output of the Logic unit, the actual check symbol of the result is generated by check symbol generator, following the same steps described in section 2, Then the predicted and the actual check symbols are compared using Totally Self-checking Checkers (TSC); if they match then the result is error free, otherwise an error signal will be generated, and the execution sequence is halted. It should be noted that the error can be either from the Logic unit or from the prediction circuit itself. In both cases the error will be detected, and an error signal will be generated, since there will be a mismatch between the actual and the predicted check symbols of the result.

7. Conclusions

The capabilities of the Dong's code to predict the check symbol for some common logical operations have been demonstrated, and the block diagram of the check symbol prediction circuit is given.



References

[1] H. L garner, "Error codes for Arithmetic Operations," IEEE trans. Elec. Computer, vol. EC-15, Oct.1966, pp763-770.
[2] I. L. Sayers, and D. J. Kinniment," Application to Self-Checking VLSI Systems," IEE Proceedings, vol. 132, pt. E, July 1985, pp197-202.
[3] Russell, G. and Elliot, I.D., "Design of Highly Reliable VLSI Processors Incorporating Concurrent Error Detection and Correction", Proceedings EURO ASIC91, May 1991 Paris.
[4] J. F. Wakerly, "Error Detecting Codes, Self-Checking Circuits and applications", Elsevier North-Holland, 1978.
[5] T. Nanya and T. Kawamura, "Error Secure/Propagating Concept and its Application to the design of Strongly Fault-Secure Processors, "IEEE Trans Computer, vol. C-37, Jan. 1988, pp 14-24.
[6] H. L. Garner "Generalized Parity Checking, "IRE Trans. Electron. Computer, vol. EC-7, 1958, pp 207-213.
[7] J.M. Berger, "A note on error detection codes for Asymmetric Channels", Information and Control, vol. 4, March 1961, pp68 – 73.
[8] H. Dong, "Modified Berger codes for detection of unidirectional errors", IEEE Trans. Computer, vol. C-33, June 1984, pp. 572-575.
[9] T. R. N. Rao and E. Fujiwara, "Error-Control Coding for Computer Systems", Englewood Cliffs, NJ: Prentice-hall, 1989.
[10] I. L. Sayers, D. J. Kinniment, and E. G. Chester, " Design of a Reliable and Self-Testing VLSI Datapath Using Residue Coding Techniques, " IEE Proc., vol-133, Part E, May 1986, pp 167-179.
[11] J.Lo, S. Thanawastien, T. R. Rao, and, M. Nicolaidis", An SFS Berger Check Prediction ALU and its Application to Self Checking Processor Designs", IEEE Tran on CAD, vol. 11, no 4, April 1992, pp 525 – 540.
[12] A. Maamar, G. Russell, "Checkbit prediction Using Dong's Code for Arithmetic Functions", Proc. Of 3rd IEEE Int. On-Line Testing Workshop, Greece, July 1997, pp254-258.

Licensed Under Creative Commons Attribution CC BY