Concurrent Error Detection in Arithmetic and Logical Operations Using Berger Codes

Jien-Chung Lo
Department of Electrical Engineering
University of Rhode Island
Kingston, RI 02881

Suchai Thanawastien and T. R. N. Rao
The Center for Advanced Computer Studies
University of Southwestern Louisiana
Lafayette, LA 70504-4330

Abstract

In this paper, we propose a novel approach to designing concurrent-error-detecting arithmetic and logic units using Berger code. Several theorems are developed on Berger check predictions for arithmetic and logical operations. Specifically, the Berger check prediction is proposed for additions and subtractions with unsigned numbers as well as signed numbers. Berger check prediction for sizteen logical operations and shift operations, multiplication and division are given here. The proposed scheme may provide a considerable saving in the hardware logic (or chip area) in implementing a self-checking ALU, and may ultimately make feasible a single-chip self-checking microprocessor or RISC design.

1. Introduction

Arithmetic and logic unit (ALU) is the heart of a processor. In designing a concurrent-error-detecting (CED) processor, the design of an ALU is unique in the sense that its error type is different from that of the rest of the processor. Historically, arithmetic codes [RAO74, RAO89] were developed to handle the arithmetic errors due to the differences in the nature of arithmetic errors. Unfortunately, the differences in the nature of arithmetic errors also make the arithmetic codes inefficient in handling the logical operations [WAK78]. For the logical operations, two-rail code [WAK78] and Reed-Muller code [PRA72] were studied. Since the two-rail code is also useful in the arithmetic operations, in practice, it is widely used in the CED ALU's [HAL84, NIC85, NAN88]. But, the hardware required by a two-rail encoded ALU is at least twice of that required by a non-encoded ALU [NAN88] and therefore is considered to be inefficient.

Parity codes were proven to be useful in designing a CED ALU through a design technique called check prediction. In the past, a single parity code was applied to the design of CED adders [SEL68, LAN70] and the design of CED ALU [SEL68, RAO72]. Also, the residue code encoded logical operations [GAR68], checksum code encoded adders [WAK78] and the error correcting linear code encoded adder [FUJ81] were proposed. Moreover, the arithmetic units for the AN code with A=15 and for the reverse residue code modulo 15 were built and used in JPL STAR computer [AVI73]. The major drawbacks of these designs are: (1) These designs cannot cover both arithmetic and logical operations; and (2) these designs cannot achieve the totally self-checking goal [WAK78, NAN88]. If the arithmetic and logical operations cannot be checked by the same code, then obviously additional error control codes are required, and consequently the hardware cost is significantly increased. Further, if a scheme cannot achieve the TSC goal then it is unsuitable for a strongly fault-secure or totally self-checking design. Presently, only the two-rail encoded ALU's are proven to be useful in the design of self-checking processors [HAL84, NIC85, NAN88]. One of the main reasons that the two-rail code encoded ALU is suitable in the design of self-checking processors is that the code is a systematic all unidirectional error detecting (AUED) code [LO89a].

In this paper, we propose a new design approach in which a more efficient AUED code is used in implementing the CED capability of ALU. Specifically, we consider the Berger code encoded ALU's, since not only is the Berger code [BER61] a systematic AUED code but it is also an optimal one in terms of the check-bit length [FRE62]. It is clear that a Berger code encoded ALU requires less hardware than a two-rail code encoded ALU in most cases due to the difference in the numbers of check bits required by the two codes. However, the Berger code has never been studied previously for either arithmetic or logical operations. Hence, the Berger check prediction scheme propose in this paper for both arithmetic and logical operations is a significant advance in the design of CED ALU's.

† This research was supported by a grant from the Board of Regents, State of Louisiana, LaSER program under contract number 86-USL(2)-127-03.

the design of CED ALU's.

In Section 2, we shall present the Berger check predictions for add and subtract operations for different signed-number representations - namely, signed-2's complement, signed-1's complement and signed-magnitude representations. In Section 3, the Berger check predictions for all sixteen logical operations on two operands will be described. The rotate and shift operations are also considered in this section. Further, we shall extend the Berger check prediction to array multipliers and dividers in Sections 4 and 5, respectively. The conclusions are given in Section 6.

2. Berger Check Prediction in Arithmetic Operations

2.1. Berger Check Prediction in Addition

Consider the addition of two n-bit numbers, $X=(x_n, \cdots, x_2, x_1)$ and $Y=(y_n, \cdots, y_2, y_1)$ to obtain the sum $S=(s_n, \cdots, s_2, s_1)$, with internal carries $C=(c_n, \cdots, c_2, c_1)$, where $x_i, y_i, s_i, c_i \in \{0,1\}$, the operation for the i^{th} bit of the two operands can be described as follows.

$$x_i + y_i + c_{i-1} = 2c_i + s_i \qquad (1)$$

$$= (s_i + c_i) + c_i$$

Let $N(X)$ denote the number of 1's in the binary representation of X. Then, trivially $N(x_i) = x_i$. Using this notation, we give the following Lemma.

Lemma 1

$$N(X) + N(Y) + c_{in} = N(S) + c_{out} + N(C) \qquad (2)$$

where c_{in} is the carry input and $c_{out} = c_n$.

A similar result were presented by Garner [GAR58] in deriving the check prediction equations for a number of arithmetic operations for single parity code. However, in [GAR58], the modulo-2 additions were used for single parity code, whereas the additions are used in Lemma 1.

The check symbol of a Berger code is the binary representation of the number of 0's (or the complement of the binary representation of the number of 1's) in the information bits [BER61]. In this paper, we consider only the first encoding scheme, i.e., counting the number of 0's in the information bits. For an n-bit number X whose Berger check symbol is X_c, $X_c = n - N(X)$.

For the two operands X and Y, whose Berger check symbols are X_c and Y_c respectively, the check symbol of the sum of X and Y is given as follows.

Theorem 1

The Berger check symbol, S_c, of the sum $S = X+Y$ can be predicted as

$$S_c = X_c + Y_c - c_{in} - C_c + c_{out} \qquad (4)$$

where $C_c = n - N(C)$ which is the number of 0's of the internal carries.

2.2. Berger Check Prediction in Subtractions

Similar to Theorem 1, we can establish the prediction for subtractions. Here, we use $b_i \in \{0,1\}$ to denote a borrow, and $B = (b_n, \cdots, b_1)$ to represent the concatenation of all the internal borrow bits.

Lemma 2

$$N(X) + N(B) + b_{out} = N(Y) + N(S) + b_{in} \qquad (5)$$

As shown in Lemma 2, we can find the relationship between the numbers of 1's in both operands and the result and internal borrow bits. The Berger check of the subtraction can be predicted as given in Theorem 2.

Theorem 2

The Berger check symbol of the result, S_c, can be predicted as

$$S_c = X_c - Y_c + B_c + b_{in} - b_{out} \qquad (6)$$

Equations (4) and (6) play a central role in implementing the Berger check prediction circuit for the adder and subtractor, respectively. The implementation of (4) and (6) needs only 1's counter and adders. However, it should be noted that Lemma 2 and Theorem 2 are derived for unsigned numbers. In practice, subtraction is often performed on an adder with redundant number representation, such as 2's complement, 1's complement and sign-magnitude representations. In the following, we shall address the Berger check predictions for different number representations.

2.3. Berger Check Prediction for 2's Complement Subtraction

The subtraction operation $S = X - Y$, in two's complement ALU design, is handled by taking the bitwise complement of Y, $\bar{Y}$, and performing the addition $S = X + \bar{Y} + 1$. However, if a carry input is required for the subtraction, the carry input to the adder must be complemented to obtain the result $S = X - Y - c_{in}$. Thus, for the general case, we assume that the inputs to ALU are X, Y and c_{in}, but the inputs to adder during the two's complement subtraction are X, $\bar{Y}$ and $\bar{c}_{in}$. With these inputs, we can have a result similar to Lemma 1 as follows.

Lemma 3

$$N(X) + N(\bar{Y}) + \bar{c}_{in} = N(S) + c_{out} + N(C) \qquad (7)$$

In Equation (7), $N(\bar{Y}) = Y_c$, since the number of 1's in the complemented information bits must be the number of 0's in the original information bits.

Theorem 3

For the operation $S = X - Y - c_{in}$ in two's complement arithmetic, the Berger check symbol of the result of subtraction, S_c, obtained from the 2's complement sub-

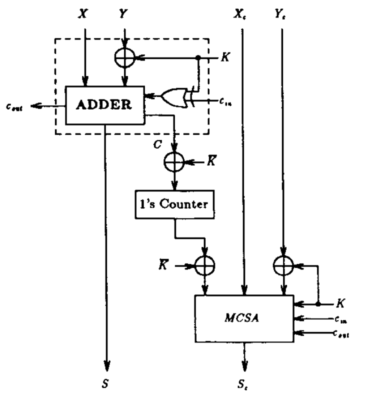

Figure 1. Berger Check Prediction Two's Complement Adder/Subtractor.

traction can be predicted as

$$S_c = X_c - Y_c - \bar{c}_{in} + N(C) + c_{out} \tag{8}$$

By implementing Equations (4) and (8) simultaneously, we have a Berger check prediction for 2's complement adder/subtractor, as shown in Figure 1. The circuit enclosed in the dashed box in Figure 1 is a typical 2's complement adder/subtractor. The signal "K" is use to select the add or subtract operation corresponding to K=0 or 1, respectively. During the add operation, the internal carries, represented by C, are complemented and fed into the 1's counter. Therefore, in this case, the output of the 1's counter is $N(\bar{C}) = C_c$. The 1's counter is implemented as that in a Berger code checker [MAR78, LO88]. The computation of the check symbol, which performs Equations (4) or (8), is accomplished by a multioperand carry save adder (MCSA).

Since C_c is negative in Equation (4), we complement C_c before it enters the MCSA. However, to have a 2's complement form of $-C_c$ as well as to include both c_{in} and c_{out} in the computation, we add a pseudo variable δ so that the output of the MCSA is the S_c as described by Equation (4). The output of the MCSA is now $S_c = X_c + Y_c - C_c - 1 + \delta$. It is obvious that δ may range from 0 to 2. Thus, we use two single bit variables α and β, where $\alpha, \beta \in \{0,1\}$, to represent δ, such that $\delta = 2\alpha + \beta$. The relationship between K, c_{in} and c_{out}, and α and β is shown in Table 1. Figure 2 shows an example of a three inputs 4-bit MCSA with modified inputs, α and β.

As for the subtraction, the internal carries are not complemented, the output of the 1's counter is thus $N(C)$. Since Y_c is negative in Equation (8), we must complement Y_c before it enters the MCSA. The output of the MCSA also has a similar form as that of the add operation. However, the definition of δ differs from that of the add operation. The values of δ, α and β for 2's

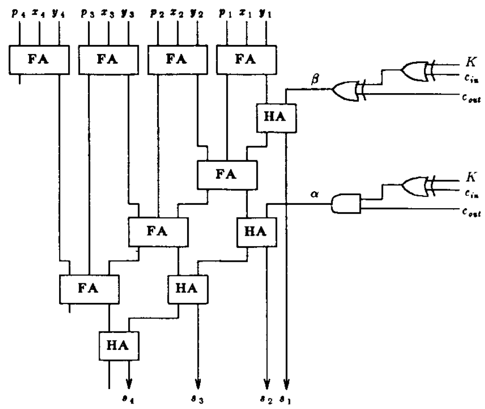

Figure 2. Three Inputs 4-bit Multioperand Carry Save Adder (MCSA) with Modified Inputs α and β.

Table 1. The Function and the Values of α and β in Figure 2.

K	c_{in}	c_{out}	output of MCSA	δ	α	β
0	0	0	$S_c = X_c + Y_c - C_c - c_{in} + c_{out}$	1	0	1
0	0	1		2	1	0
0	1	0		0	0	0
0	1	1		1	0	1
1	0	0	$S_c = X_c - Y_c + N(C) - \bar{c}_{in} + c_{out}$	0	0	0
1	0	1		1	0	1
1	1	0		1	0	1
1	1	1		2	1	0

complement subtraction are also shown in Table 1.

2.4. Berger Check Prediction for 1's Complement Subtraction

The negative number can also be represented in diminished radix complement form, or 1's complement in binary system. A 1's complement adder/subtractor uses end-around carry such that the carry output generated at the first cycle is used at the second cycle as the carry input. The correct sum is then generated at the end of the second cycle. Since we already have the Berger check prediction equation for the unsign number, we can modified the circuit in Figure 1 to be used in 1's complement operations. Figure 3 shows the BCP 1's complement adder/subtractor. The add or subtract operation takes two cycles to establish the desired result. At the beginning of the first cycle, the two operands are loaded into latches and the carry input is clear to "0". At the end of the first cycle, the carry output is latched to the carry input latch. The BCP circuit will then start its evaluation at the end of the second cycle, while the desired sum or difference has been generated. Note that, the α and β in Figure 3 are defined as in Table 1, too.

2.5. Berger Check Prediction for Signed-Magnitude Subtraction

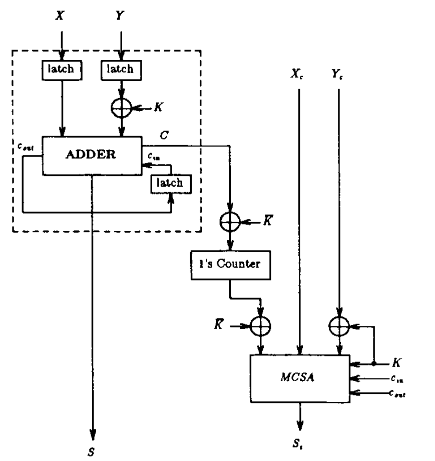

Figure 3. Berger Check Prediction One's Complement Adder/Subtractor.

For the signed-magnitude representation, an n-bit number is actually the concatenation of an (n-1)-bit unsigned number and a sign bit. A typical sign-magnitude adder/subtractor [HWA79] is shown in the dashed box in Figure 4. We use the notation X_L to represent the magnitude of the X such that $X_L = (x_{n-1}, \cdots, x_2, x_1)$, and x_n to denote the sign bit of X. The (n-1)-bit adder in the dashed box in Figure 4 is connected as a 1's complement adder/subtractor. Thus, the Berger check for the adder's output, S_L^* can be predicted as described in the last section. The sign bit of the sum/difference is dependent on the control signals K, x_n, y_n and c_{out}. Hence, the Berger check prediction equations for the signed-magnitude adder/subtractor can be derived directly from the results presented in the previous sections and are summarized in Table 2.

The Berger check prediction circuit shown in Figure 4 is similar to that shown in Figures 1 and 3. However, the *MCSA* in Figure 4 is a four-input multi-operand carry save adder. Since, as shown in Table 2, the value of δ^* may vary from $n+3$ to -2, it may take $\log_2 n + 1$ bits to represent δ^*. Further, a control PLA is used to initiate the proper operation of the Berger check prediction circuit. The functional table of this control PLA is exactly the same as Table 2 except the generation of δ^*.

3. Berger Check Prediction in Logical Operations

The check symbol prediction for the logical operations were considered for residue codes [GAR68] and single parity codes [SEL68, RAO72]. Also, in the work on the design of self-checking processors [HAL84, NAN88], the logical operations are handled by the two-rail code. In this section, we shall introduce the Berger check predictions for all the two-operand logical operations. In

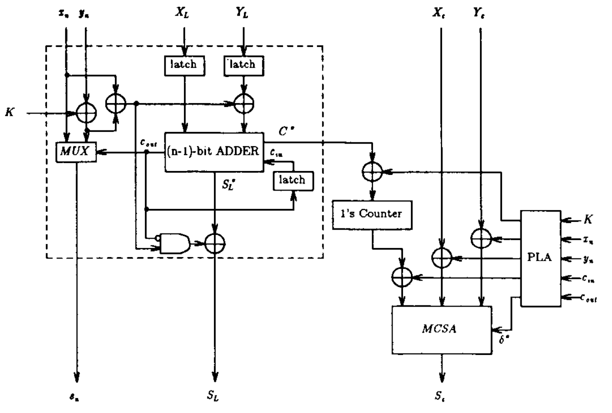

Figure 4. Berger Check Prediction Sign-Magnitude Adder/Subtractor.

Table 2. The Begrer Check Prediction for Sign-Magnitude Adder/Subtractor.

K	x_n	y_n	c_{out}	$S_c =$	δ^*
0	0	0	0	$X_c + Y_c - C_c^* + x_n - c_{in} + c_{out}$	$1 - c_{in}$
0	0	0	1	$X_c + Y_c - C_c^* + y_n - c_{in} + c_{out}$	$1 - c_{in}$
0	0	1	0	$-X_c - Y_c + C_c^* + n + 1 - 2y_n + c_{in} - c_{out}$	$n + 1 + c_{in}$
0	0	1	1	$X_c - Y_c + N(C^*) - y_n - c_{in} + c_{out}$	$-c_{in}$
0	1	0	0	$X_c + Y_c - C_c^* - x_n + c_{in} - c_{out} + 1$	$1 + c_{in}$
0	1	0	1	$X_c - Y_c + N(C^*) - y_n - c_{in} + c_{out}$	$1 - c_{in}$
0	1	1	0	$X_c + Y_c - C_c^* + y_n - c_{in} + c_{out}$	$2 - c_{in}$
0	1	1	1	$X_c + Y_c - C_c^* + x_n - c_{in} + c_{out}$	$2 - c_{in}$
1	0	0	0	$-X_c - Y_c + C_c^* + n + 1 - y_n + \overline{y}_n + c_{in} - c_{out}$	$n + 4 + c_{in}$
1	0	0	1	$X_c - Y_c + N(C^*) - y_n - c_{in} + c_{out}$	$1 - c_{in}$
1	0	1	0	$X_c + Y_c - C_c^* + x_n + y_n - \overline{y}_n - c_{in} + c_{out}$	$2 - c_{in}$
1	0	1	1	$X_c - Y_c + N(C^*) - y_n - c_{in} + c_{out}$	$-c_{in}$
1	1	0	0	$X_c + Y_c - C_c^* + x_n + y_n - \overline{y}_n - c_{in} + c_{out}$	$1 - c_{in}$
1	1	0	1	$X_c - Y_c + N(C^*) - y_n - c_{in} + c_{out}$	$1 - c_{in}$
1	1	1	0	$-X_c - Y_c + C_c^* + n + 1 - y_n + \overline{y}_n + c_{in} - c_{out}$	$n + 2 + c_{in} + n$
1	1	1	1	$X_c - Y_c + N(C^*) - y_n - c_{in} + c_{out}$	$-c_{in}$

addition, we also present the Berger check prediction for rotate and shift operations.

3.1. Berger Check Prediction in Two-Operand Logical Operations

There are 16 possible logical operations on two operands including six trivial operations, 0, 1, X, Y, $\overline{X}$, $\overline{Y}$, and ten nontrivial operations. First, we examine the three basic logical operations AND ($\wedge$), OR ($\vee$), XOR ($\oplus$). From observing their truth tables, as shown in Table 3, one can easily verify the following

$$x_i \wedge y_i \equiv x_i + y_i - (x_i \vee y_i) \ , \tag{9}$$

$$x_i \vee y_i \equiv x_i + y_i - (x_i \wedge y_i) \ , \text{ and} \tag{10}$$

$$x_i \oplus y_i \equiv x_i + y_i - 2(x_i \wedge y_i) \ . \tag{11}$$

Equations (9)-(11) were also given in [RAO72]. Here, we apply these equations to determine the relationships in terms of the number of 1's. For the AND operation, the number of 1's in $X \wedge Y$ can be evaluated as

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

Table 3. Truth Tables for AND, OR and XOR operations.

AND				OR				XOR			
x_i	y_i	s_i	1's lost	x_i	y_i	s_i	1's lost	x_i	y_i	s_i	1's lost
0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	1	1	0	0	1	1	0
1	0	0	1	1	0	1	0	1	0	1	0
1	1	1	1	1	1	1	1	1	1	0	2

Based on the above result, we can derive the relationships for the rest of the logical operations on the two operands, X and Y. The Berger check prediction for logical operations can then be readily formulated as Theorem 4.

Theorem 4

The Berger check symbols of the results of logical operations can be predicted as follows.

$$S = X\wedge Y \quad \Rightarrow \quad S_c = X_c + Y_c - (X\vee Y)_c \tag{12}$$

$$S = \overline{X}\wedge Y \quad \Rightarrow \quad S_c = -X_c + Y_c + (\overline{X}\vee Y)_c \tag{13}$$

$$S = X\wedge \overline{Y} \quad \Rightarrow \quad S_c = X_c - Y_c + (X\vee \overline{Y})_c \tag{14}$$

$$S = \overline{X}\wedge \overline{Y} \quad \Rightarrow \quad S_c = -X_c - Y_c + (X\wedge Y)_c + n \tag{15}$$

$$S = X\vee Y \quad \Rightarrow \quad S_c = X_c + Y_c - (X\wedge Y)_c \tag{16}$$

$$S = \overline{X}\vee Y \quad \Rightarrow \quad S_c = -X_c + Y_c + N(\overline{X}\wedge Y) \tag{17}$$

$$S = X\vee \overline{Y} \quad \Rightarrow \quad S_c = X_c - Y_c + N(X\wedge \overline{Y}) \tag{18}$$

$$S = \overline{X}\vee \overline{Y} \quad \Rightarrow \quad S_c = -X_c - Y_c + (X\vee Y)_c + n \tag{19}$$

$$S = X\oplus Y \quad \Rightarrow \quad S_c = X_c + Y_c - 2(X\wedge Y)_c + n \tag{20}$$

$$S = \overline{X\oplus Y} \quad \Rightarrow \quad S_c = -X_c - Y_c + 2N(\overline{X}\wedge Y) \tag{21}$$

$$S = X \quad \Rightarrow \quad S_c = X_c \tag{22}$$

$$S = Y \quad \Rightarrow \quad S_c = Y_c \tag{23}$$

$$S = \overline{X} \quad \Rightarrow \quad S_c = n - X_c \tag{24}$$

$$S = \overline{Y} \quad \Rightarrow \quad S_c = n - Y_c \tag{25}$$

$$S = 0 \quad \Rightarrow \quad S_c = n \tag{26}$$

$$S = 1 \quad \Rightarrow \quad S_c = 0 \tag{37}$$

3.2. Berger Check Prediction for Rotate and Shift Operations

Besides the logical operations described, an ALU must also capable of performing the rotate and shift operations. In general, there are three basic types of such operation: rotate, logical shift (rotate through carry) and arithmetic shift. The rotate operations are defined as follows.

Rotate Left

$(x_{n-1}, x_{n-2}, \cdots, x_1, x_n) \leftarrow (x_n, x_{n-1}, \cdots, x_2, x_1)$

Rotate Right

$(x_1, x_n, \cdots, x_3, x_2) \leftarrow (x_n, x_{n-1}, \cdots, x_2, x_1)$

The Berger check of the result of a rotate operation is simply the Berger check of the operand, since no information bit is discarded except for their position.

The logical shift operation involves the carry bit in the operation. It is assumed that c_{in} is used as the input, while c_{out} represents the shift-out bit after the operation.

Logical Left Shift

$(x_{n-1}, x_{n-2}, \cdots, x_1, c_{in}) \leftarrow (x_n, x_{n-1}, \cdots, x_2, x_1)$

Logical Right Shift

$(c_{in}, x_n, \cdots, x_3, x_2) \leftarrow (x_n, x_{n-1}, \cdots, x_2, x_1)$

Thus, $c_{out} = x_n$ for logical left shift operation and $c_{out} = x_1$ for logical right shift operation. It is obvious that the Berger check of the result S can be predicted as

$$S_c = X_c - c_{in} + c_{out} \tag{28}$$

The arithmetic shift operation is used for the signed numbers. Here, we consider the arithmetic shift operations for 2's complement operands. During the right shift operation, the most significant bit is extended to the right (sign extension). For the arithmetic left shift, a "0" is inserted at the least significant bit.

Arithmetic Left Shift

$(x_{n-1}, x_{n-2}, \cdots, x_1, 0) \leftarrow (x_n, x_{n-1}, \cdots, x_2, x_1)$

Arithmetic Right Shift

$(x_n, x_n, x_{n-1}, \cdots, x_3, x_2) \leftarrow (x_n, x_{n-1}, \cdots, x_2, x_1)$

Here, we assume that $c_{out} = x_n$ for the arithmetic left shift operation and $c_{out} = x_1$ for the arithmetic right shift operation. Thus, the Berger check of the result of an arithmetic left shift can be predicted as

$$S_c = X_c + c_{out} \tag{29}$$

and the Berger check of the result of an arithmetic right shift can be predicted as

$$S_c = X_c - x_n + c_{out} \tag{30}$$

3.3. Berger Check Prediction Circuit for Logical Operations

The effective implementation of Theorem 4 is a nontrivial problem. Fortunately, according to Equations (12)-(21), we can find the following general form for the Berger check prediction of all the nontrivial logical operations.

$$S_c = [\, p_1 \times X_c + p_2 \times Y_c + p_3 \times p_4 \,] + p_6 \times n \tag{31}$$

where, p_1 and p_2 are either +1 or -1, p_3 is either 1 or 2, p_4 can be one of the following: $(X\vee Y)_c$, $(\overline{X}\vee Y)_c$, $(X\vee \overline{Y})_c$, $(X\wedge Y)_c$, $N(\overline{X}\wedge Y)$ and $N(X\wedge \overline{Y})$, and p_6 is either 0 or 1.

Figure 5 shows the Berger check prediction circuit implemented based on Equation (31) for the sixteen logical operations defined in Theorem 4. There are 12 internal control signals, t_1-t_{12}, to adapt the circuit for all the 16 logical operations as well as the rotate and shift

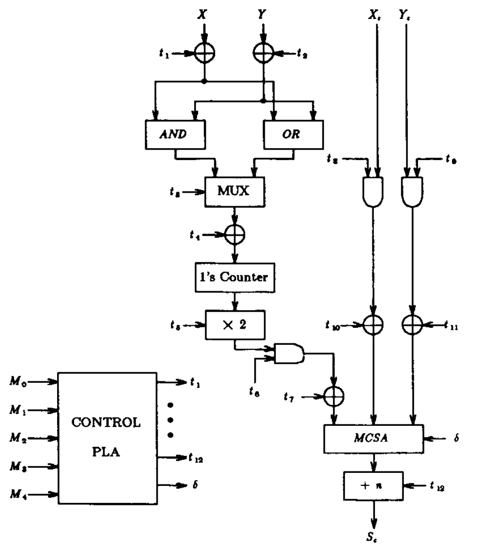

Figure 5. Berger Check Prediction Circuit for All Two-Operand Logical, Shift and Rotate Operations.

operations. The 12 control signals and δ for the MCSA are generated by a PLA, whose function is defined in Table 4.

Among the six trivial logical operations, $S = X$ and $S = Y$ are handled by passing the check symbol X_c or Y_c directly to the output of the prediction circuit. For the operations $S = \overline{X}$ and $S = \overline{Y}$, the output of the prediction circuit is produced by negating the check symbol X_c or Y_c, respectively, and adding a constant "n", the information length. The operations $S = 0$ is handled by generating the logic "0", accomplished by setting the controls t_6, t_8 and t_9 to "0". Whereas for the operation $S = 1$, besides generating a "0", a constant "n" is added.

The BCP circuit shown in Figure 5 can also handle the rotate and shift operations, as described by Equations (28)-(30). For example, the logical shift operations described by Equation (28) is accomplished by setting t_6, t_7, t_8 and t_9 to "1", and t_{10}, t_{11} and t_{12} to "0". Also, t_1-t_5 are ignored, since they will not affect the result of evaluation. Therefore, when $\delta = 0$, the output of $MCSA$ is equal to $X_c - 1$. The variable δ is then generated by examine the value of c_{in} and c_{out}, as shown in Table 4. For the arithmetic shift operations, t_6-t_{11} are set as in logical shift operations, the only difference is the generation of δ.

From Figures 1 and 6, we find that it is possible to merge these two circuits to form a Berger check prediction circuit for an ALU that performs additions, 2's complement subtractions, all 16 logical operations on two operands, and rotate and shift operations. In fact, the only modification required is to add an extra input port

Table 4. Functions of Control PLA in Figure 5.

Controls					t_i's $i =$												Function	δ
M_0	M_1	M_2	M_3	M_4	1	2	3	4	5	6	7	8	9	10	11	12		
0	0	0	0	X	X	X	X	X	X	0	0	1	0	0	0	0	$S = RR\ (X)$	0
0	0	0	1	X	X	X	X	X	X	0	0	1	0	0	0	0	$S = RL\ (X)$	0
0	0	1	0	X	X	X	X	X	X	1	1	1	1	0	0	0	$S = LRS\ (X)$	$1-c_{in}+c_{out}$
0	0	1	1	X	X	X	X	X	X	1	1	1	1	0	0	0	$S = LLS\ (X)$	$1-c_{in}+c_{out}$
0	1	0	0	X	X	X	X	X	X	1	1	1	1	0	0	0	$S = ARS\ (X)$	$1-x_n+c_{out}$
0	1	0	1	X	X	X	X	X	X	1	1	1	1	0	0	0	$S = ALS\ (X)$	$1+c_{out}$
1	0	0	0	0	X	X	X	X	X	0	0	1	0	1	0	1	$S = \overline{X}$	1
1	0	0	0	1	0	0	0	1	0	1	0	1	1	1	1	1	$S = \overline{X}\wedge\overline{Y}$	2
1	0	0	1	0	1	0	1	1	0	1	0	1	1	1	0	0	$S = \overline{X}\wedge Y$	1
1	0	0	1	1	X	X	X	X	X	0	0	0	0	0	0	1	$S = Logic\ 0$	0
1	0	1	0	0	0	0	1	1	0	1	0	1	1	1	1	1	$S = \overline{X}\vee\overline{Y}$	2
1	0	1	0	1	X	X	X	X	X	0	0	0	1	0	1	1	$S = \overline{Y}$	1
1	0	1	1	0	0	0	0	1	1	1	1	1	1	0	0	1	$S = X\oplus Y$	1
1	0	1	1	1	0	1	1	1	0	1	0	1	1	0	1	0	$S = X\wedge\overline{Y}$	1
1	1	0	0	0	1	0	0	0	0	1	0	1	1	1	0	0	$S = \overline{X}\vee Y$	1
1	1	0	0	1	1	0	0	0	1	1	0	1	1	1	1	0	$S = \overline{\overline{X}\oplus\overline{Y}}$	2
1	1	0	1	0	X	X	X	X	X	0	0	0	1	0	0	0	$S = Y$	0
1	1	0	1	1	0	0	1	1	0	1	1	1	1	0	0	0	$S = X\wedge Y$	1
1	1	1	0	0	X	X	X	X	X	0	0	0	0	0	0	0	$S = Logic\ 1$	0
1	1	1	0	1	0	1	0	0	0	1	0	1	1	0	1	0	$S = X\vee\overline{Y}$	1
1	1	1	1	0	0	0	0	1	0	1	1	1	1	0	0	0	$S = X\vee Y$	1
1	1	1	1	1	X	X	X	X	X	0	0	1	0	0	0	0	$S = X$	0

X : don't care
RR (RL) : Rotate Right (Left)
LRS (LLS) : Logical Right (Left) Shift
ARS (ALS) : Arithmetic Right (Left) Shift

at "MUX" in Figure 6, for the internal carries, C, from the ALU itself [LO89a]. Thus, although the BCP circuit shown in Figure 6 is not cost effective if only logical and shift operations need to be handled, the modified circuit of Figure 6 is very cost effective for a typical ALU design [LO89a].

4. Berger Check Prediction for Array Multipliers

One of the most common multipliers is the Braun's unsigned array multiplier [HWA79]. The single parity check prediction for such array multiplier was given in [PRA86]. Here, we extend the results obtained in Section 2 to the Berger check prediction for the array multiplier. A 4-bit by 4-bit Braun's array multiplier is shown in Figure 6(a), where eight full adders and four half adders are used. Each individual adder is labeled for proper identification. The temporary sums and carries generated by each adder is then labeled accordingly. For instance, the full adder labeled (3,2) receives $x_4/\backslash y_2$, $x_3/\backslash y_3$ and $c_{3,1}$ as its inputs and generates the sum $s_{3,2}$ and the carry $c_{3,2}$.

The input/output relationship of each adder can be properly described by Equation (1). Then, by summing all the equations, we can obtain

$$N(X) \cdot N(Y) = N(S) + N(C^*) \qquad (32)$$

where $N(C^*) = \sum_{i=1}^{3} \sum_{j=1}^{4} c_{i,j}$. From Equation (35), the Berger check symbol of the product generated by an array multiplier with input operands X and Y, whose Berger check symbol is X_c and Y_c, respectively, can be predicted as

$$S_c = 4X_c + 4Y_c - X_c Y_c - C_c^* \qquad (33)$$

where $C_c^* = 12 - N(C^*)$.

Based on Equation (33), we can implement a Berger check prediction circuit for the array multiplier, as

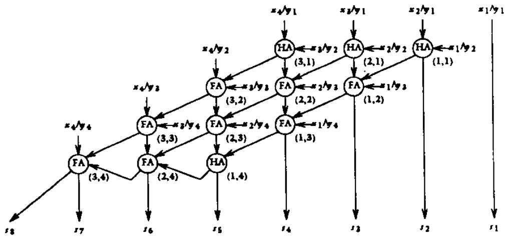
Figure 6(a). Braun's Array Multiplier.

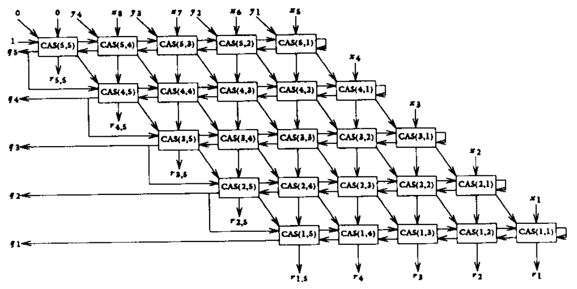
Figure 7(a). Nonrestoring Array Divider.

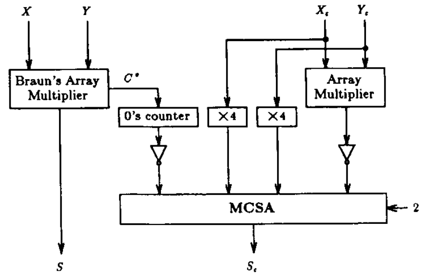

Figure 6(b). Berger Check Prediction Array Multiplier.

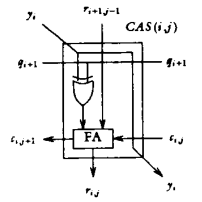

Figure 7(b). The Controlled Adder/Subtractor.

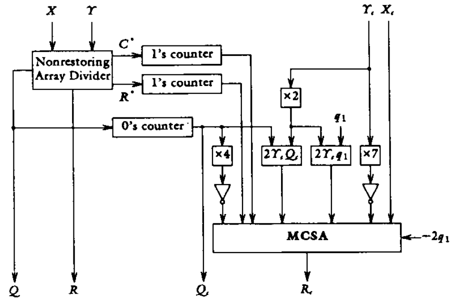

Figure 7(c). Berger Check Prediction Nonrestoring Array Divider.

shown in Figure 6(b). Further, it is conjectured that Equation (33) can be extended to the n-bit case.

5. Berger Check Prediction for Array Dividers

As for the array dividers, the check prediction for the quotient is impossible, since the quotient is in the form of carry bits, as shown in Figure 7(a). In the past, the single parity prediction array divider was proposed in [FUR83] that predicts only the check symbol of the remainder. To simplify our discussion, we consider only the nonrestoring array divider [HWA79]. A nonrestoring array divider with an 8-bit dividend, $X=\{x_8, x_7, \cdots, x_1\}$, and a 4-bit divisor, $Y=\{y_4, y_3, y_2, y_1\}$, generating a 5-bit quotient, $Q=\{q_5, q_4, q_3, q_2, q_1\}$, and a 4-bit remainder, $R=\{r_4, r_3, r_2, r_1\}$, is depicted in Figure 7(a). This array divider is constructed from controlled adder/subtractors (CAS's) shown in Figure 7(b). Based on the characteristics of a CAS, we obtain the following equations:

$$x_5 + (y_1 \oplus 1) + 1 = 2c_{5,2} + r_{5,1}$$

$$x_6 + (y_2 \oplus 1) + c_{5,2} = 2c_{5,3} + r_{5,2}$$

$$\cdots$$

$$r_{2,4} + (0 \oplus q_2) + c_{1,5} = 2q_1 + r_{1,5}$$

By summing the above equations, we have

$$\sum_{i=1}^{8} x_i + \sum_{i=1}^{5}\sum_{j=2}^{6} (y_i \oplus q_j) + 1 =$$

$$\sum_{i=1}^{5}\sum_{j=2}^{5} c_{i,j} + \sum_{i=1}^{5} q_i + q_1 + \sum_{i=1}^{5} r_{i,5} + \sum_{i=1}^{4} r_i \qquad (34)$$

where $y_5 = 0$ and $q_6 = 1$. Let $\psi_{i,j} = y_i \oplus q_j$ then $\psi_{i,j} = y_i$ if $q_j = 0$, and $\psi_{i,j} = \bar{y}_i$ if $q_j = 1$. Thus,

$$\sum_{i=1}^{5}\sum_{j=2}^{6} \psi_{i,j} = 5(\sum_{i=1}^{5} y_i + \sum_{j=2}^{6} q_j) - 2\sum_{i=1}^{5} y_i \sum_{j=2}^{6} q_j$$

Therefore, Equation (34) can be rewritten as

$$N(R) = N(X) + 3N(Y) + 4N(Q) - 2N(Y)N(Q)$$

$$+ 2q_1N(Y) - 6q_1 - N(C^*) - N(R^*) + 6$$

where $N(C^*) = \sum_{i=1}^{5}\sum_{j=2}^{5} c_{i,j}$ and $N(R^*) = \sum_{i=1}^{5} r_{i,5}$. Since the Berger check symbol of the remainder $R_c = 4 - N(R)$, we have

$$R_c = X_c - 7Y_c - 4Q_c + 2Y_cQ_c$$

$$+ 2Y_cq_1 + N(C^*) + N(R^*) - 2q_1 - 2 \qquad (35)$$

Figure 7(c) show an implementation of the Equation (35).

6. Conclusions

We have derived the formulas for the Berger check predictions for the arithmetic and logical operations in this paper. Specifically, we have derived the equations for predicting the Berger check symbols for adder/subtractor for the three signed-number representations, all sixteen two-operand logical operations, array multiplication and array division. The only codes that are previously known to handle all these are single parity code and two-rail code. However, with single parity code, the code itself limits the error detecting capability to only odd number of unidirectional errors. Also, when the two-rail code is applied the hardware requirement is doubled.

Although Berger code is an AUED code, a BCP circuit that handles addition, 2's complement subtraction, logical operations, and rotate/shift operations, is proved to be capable of handling single and double arithmetic errors as well as multiple unidirectional errors [LO89a]. In other words, the error handling capabilities of BCP ALU's are comparable to two-rail encoded ALU's. Thus, the BCP design is superior to the single parity encoded ALU's in terms of error handling capability and more cost effective than the two-rail encoded ALU's [LO89a].

One disadvantage of the check prediction scheme is the delay penalty. In the single parity check prediction, the evaluation of the parity of the sum is started after the duplicated carry circuit generated the duplicated carries. Although in the BCP circuit, the carries are from the ALU itself, the Berger check evaluation cannot be started until the ALU generated all the carries. This means that the total delay time of the ALU is the sum of delay of ALU and that of the BCP circuit. A two-rail encoded ALU will not have such impact on the delay time. However, when the entire data path is considered, the time penalty induced by the proposed BCP scheme is about the same as that by a two-rail encoded ALU. For instance, a typical critical path in a data path is to load operand from the register file, process by ALU and then store it back to the register file. Since it is impractical to encode the register file in a two-rail code, due to the high hardware requirement. Let us assume that the register file is Berger encoded similar to that used in [NAN88]. A transcoder is therefore required to translate the two-rail code words to Berger code words, and vice versas. The delay time introduced by the transcoder is about the same as the BCP circuit, because it is basically a 0's counter, as in the BCP circuit. Moreover, in designing a 32-bit RISC processor [LO89b], a four-stage instruction pipeline is used such that the evaluation of the Berger check symbol is overlapped with ALU's or shifter's function. In that case, the delay penalty induced by the BCP circuit is omitted, and the processor's processing speed is basically the same as a non-Berger-encoded ALU.

In sum, we can see that the ability to predict the Berger check is significant since it leads to a reduced hardware implementation of a processor with concurrent error detection capability.

REFERENCES

[AVI73] A. Avizienis, "Arithmetic Algorithms for Error-Coded Operands," *IEEE Trans. on Comput.*, Vol. C-22, No. 6, pp. 567-572, June 1973.

[BER61] J. M. Berger, "A Note on Error Detection Codes for Asymmetric Channels," *Inform. Control*, Vol. 4, pp. 68-73, March 1961.

[FRE62] C. V. Freiman, "Optimal Error Detecting Codes for Completely Asymmetric Binary Channels," *Inform. Control*, Vol. 5, pp. 64-71, March 1962.

[FUJ81] E. Fujiwara and K. Haruta, "Fault-Tolerant Arithmetic Logic Unit Using Parity-Based Codes," *Trans. Inst. Electron. Commun. Eng. Japan*, pp. 653-660, October 1981.

[FUR83] K. Furuya, Y. Akita and Y. Tohma, "Logic Design of Fault-Tolerant Dividers Based on Data Complementation Strategy," *Proc. FTCS-13*, pp. 306-313, June 1983.

[GAR58] H. L. Garner, "Generalized Parity Checking," *IRE Trans. on Elec. Comput.*, pp. 207-213, September 1958.

[GAR68] O. N. Garcia and T. R. N. Rao, "On the Methods of Checking Logical Operations," *Proc. 2nd Princeton Conf. Inform. Sci. Sys.*, pp. 89-95.

[HAL84] M. P. Halbert and S. M. Bose, "Design Approach for a VLSI Self-Checking MIL-STD-1750A Microprocessor," *Proc. FTCS-14th*, pp. 254-259, June 1984.

[HWA79] K. Hwang, *Computer Arithmetic: Principles, Architecture, and Design*, John Wiley and Sons, New York, 1979.

[LAN70] G. G. Langdon, Jr. and C. K. Tang, "Concurrent Error Detection for Group Look-ahead Binary Adders," *IBM J. Res. Develop.*, pp. 563-573, September 1970.

[LO88] J. C. Lo and S. Thanawastien, "The Design of Fast Totally Self-Checking Berger Code Checkers Based on Berger Code Partitioning," *Proc. FTCS-18*, pp. 226-231, June 1988.

[LO89a] J. C. Lo, S. Thanawastien and T. R. N. Rao, "A TSC Berger Check Prediction ALU and Its Application to Self-Checking Processor Designs," Technical Report TR-89-3-5, The Center for Advanced Computer Studies, University of Southwestern Louisiana, LA 70504-4330, 1989.

[LO89b] J. C. Lo, "Self-Checking VLSI Reduced Instruction Set Computers," Ph.D. Dissertation, University of Southwestern Louisiana, 1989.

[MAR78] M. A. Marouf and A. D. Friedman, "Design of Self-Checking Checkers for Berger Codes," *Proc. FTCS-8*, pp. 179-184, June 1978.

[NAN88] T. Nanya and T. Kawamura, "Error Secure/Propagating Concept and its Application to the Design of Strongly Fault-Secure Processors," *IEEE Trans. on Comput.*, Vol. 37, No. 1, pp. 14-24, January 1988.

[NIC85] M. Nicolaidis, "Evaluation of a Self-Checking Version of the MC68000 Microprocessor," *Proc. FTCS-15*, pp. 350-356, June 1985.

[PRA72] D. K. Pradhan and S. M. Reddy, "Error-Control Techniques for Logic Processors," *IEEE Trans. on Comput.*, Vol. C-21, No. 12, pp. 1331-1336, December 1972.

[PRA86] D. K. Pradhan, editor, *Fault-Tolerant Computing, Theory and Techniques*, Volume I, Prentice-Hall, New Jersey, 1986.

[RAO72] T. R. N. Rao and P. Monteiro, "A Residue Checker for Arithmetic and Logical Operations," *Proc. FTCS-2*, pp. 8-13, June 1972.

[RAO74] T. R. N. Rao, *Error Coding for Arithmetic Processors*, Academic Press, New York and London, 1974.

[RAO89] T. R. N. Rao and E. Fujiwara, *Error-Control Coding for Computer Systems*, Prentice-Hall, New Jersey, 1989.

[SEL68] F. F. Sellers, M. -Y. Hsiao and L. W. Bearnson, *Error Detecting Logic for Digital Computers*, New York : McGraw-Hill, 1968.

[WAK78] J. F. Wakerly, *Error Detecting Codes, Self-Checking Circuits and Applications*, North-Holland, 1978.