Abstract

As a consequence of studies concerning a fault-tolerant microprogrammed microprocessor an error-detection and error-correction scheme for logical operations has been developed. Implementing inverse residue coding for error-detection and inverse biresidue coding for error-correction the hardware is considered. The theoretical basis of residue coding is briefly summarized and the error-detection/error-correction of logical operations is presented. Concurrent error-detection capability is discussed as well as the estimation of hardware overhead.

Introduction

In the past there were not many studies concerning the logical part of an ALU. In most cases the logical unit (LU) is duplicated or tripled /GAR69/,/AVI71/. This is reasonable because in general the hardware for logical operations is smaller than that for arithmetic operations. But errors in the input operands go undetected. Even if there are additional circuits using paritycheck codes the bit-error detection ability is achieved for some special logic operations only /WAK78/.

Patel describes a method to check operations using time redundancy /PAT82/. The error model seems to be the same assumption given in /TRA83/ and used in this paper. That means a more functional level. The method is convenient to check and correct logical operations, but the correction of arithmetical operations is not easy to implement. Values for propagation delay and hardware overhead are not given and the input operands themselves are not checked by the RESO-method.

Some ED-implementations are based on the fact, that every logical operation can be developed into one elementary logical function (NAND) and an appropriate number of steps using the error-detecting arithmetic unit of the ALU /RAM72/. The design presented here avoids the amount of processing time caused by these iterative calculations. Checking is achieved by implementation of an inverse residue code and the appropriate handling of residues as checkparts.

Here a residue-coding scheme is discussed which uses the checkhardware that is already necessary to check arithmetic operations /TRA82/,
This implies that the same code set is used for arithmetic and logical operations. Since the input operands are presumed to be encoded for ED/EC, errors in these input operands will be detected too. Without dependency to any technology error modes are seen as functional errors. This includes the consideration of effects of stuck-at-faults as well as arithmetic faults depending on storage or transfer of data.

\[
A' = A \text{ AND } E_{SB,MB} \quad (s-a-0) \\
A' = A \text{ OR } E_{SB,MB} \quad (s-a-1)
\]

with \( E_{SB} = 2^i \) and \( E_{MB} = \sum_{i=0}^{\infty} a_i 2^i \) and \( a_i \in \{0,1\} \) and \( i=1,\ldots,n \)

and modification or operation of operands

\[
A' = A + E_{S,M}
\]

with \( E_S = \pm 2^i \) and \( E_M = \pm \sum_{i=0}^{\infty} a_i 2^i \) and \( a_i \in \{0,1\} \) and \( i=1,\ldots,n \).

(Equations (1) are the corrected representation of Eq.(1) in /TRA82/).

**Coding**

The method is based on the residue coding introduced by Rao /RA074/ and the residue arithmetic /SZT67/. The operands \( Op \) are encoded by appending an inverse residue \( IOP_{a} \) (\( Op_{a} = <Op, IOP_{a}> = <\text{datapart}, \text{checkpart}> \) ) according to

\[
IOP_{a} = (ma - Op \mod ma) \mod ma
\]

The advantage of this inverse coding is better error-detection capability in case of unidirectional errors /WAK78/. In this separate code \( Op \) is the original data and \( ma \) is a modulus that is chosen to be

\[
ma = 2^a - 1 \quad \text{with } a>1.
\]

In binary number systems \( a \) is the number of bits representing the modulus \( ma \). If equation (4) is valid the code is called a low-cost-code according to Avizienis /AVI71/. That is why the \( (\text{mod } ma) \)-operation can be realized by a carry-end-around adder-tree or other combinatorial regular circuits. In this case the one's-complement representation of numbers is assumed. An effective and unique error-detection is achieved if the binary representation of the modulus \( ma \) divides \( n \) without remainder, where \( n \) is the binary representation of the original operand.

Supposing there are 20 bit operands, the number of bits of a could be \( a \in \{2,4,5,10,20\} \). The choice of \( a \) establishes the degree of hardware overhead and the error-detection capability. Here \( a=4 \) is suggested as a middle course. Corresponding to residue algebra the operands and the residues are processed separately and a residue of the result is generated. In the error-free state this residue is equal to the result of the association of the two operand residues.

For error-correction the code set is expanded by appending another residue to the encoded operand. Thus the minimum distance of the code