RSRE

MEMORANDUM No. 4214

ROYAL SIGNALS & RADAR ESTABLISHMENT

ALGORITHMIC FAULT TOLERANCE

Author: I K Proudler

[illegible]

PROCUREMENT EXECUTIVE,
MINISTRY OF DEFENCE,
R S R E MALVERN,
WORCS.

A

ROYAL SIGNALS AND RADAR ESTABLISHMENT

Memorandum 4214

TITLE: ALGORITHMIC FAULT TOLERANCE

AUTHOR: I K Proudler

DATE: September 1988

SUMMARY

A reduction in the minimum attainable feature size in integrated circuits has lead to the possibility of more and more complex circuits being built on a single chip (VLSI). This technological advance brings with it the need to make these circuits fault tolerant: to increase yield and reliability and to reduce testing times. This Memorandum briefly reviews current techniques for designing fault tolerant circuits before concentrating on a new, high-level fault tolerance technique: algorithmic fault tolerance.

The concept of algorithmic fault tolerance is explained and various techniques are reviewed with regard to their suitability for providing fault tolerance for signal processing algorithms. Suggestions are made for the direction for further research. [illegible]

S80/59

CONTENTS

1 Introduction

The use of high-density integrated circuit manufacturing methods (VLSI and above) whilst bringing many advantages to integrated circuit design also create several new problems. The increase in complexity of the devices being produced, coupled with limited access to internal circuit nodes, means that testing the device is increasingly more difficult, both in the factory and in the field. The reduction in geometry size, that allows this increased packing density, means that defects in the silicon, or any of the numerous layers involved in modern IC construction, are more likely to result in the malfunction of a transistor [1]. The gate that contains this transistor, and ultimately the device itself, will thus exhibit a fault. The increased likelihood of a defective transistor is therefore reflected in a poorer process yield.

A third problem is that of soft failures. These are faults that exist intermittently, affecting the device for a finite period before disappearing, possibly to return at some later time. Possible causes of such a phenomenon include the alteration of the charge on a transistor's gate by cosmic radiation or α-particle emission, the latter usually being due to the case material (this is particularly common in solid state memory devices); electron tunnelling through thin barriers; electromagnetic pick-up within the device and capacitive coupling between adjacent circuit elements. The occurrence of such a fault renders the device strictly useless since no confidence can be placed in its operation.

These soft hardware-faults can be augmented by a similar set of software faults. The complexity of VLSI devices is such that the components of an equipment can now include microprocessors. These will be under software control and thus may make mistakes due to short comings in the program design. It is probably fair to assume that the program has, to some extent, been tested and so works under most conditions. It is possible, however, that certain (rare?) conditions exist under which the program will produce an erroneous result, returning to normal operation once these conditions have been removed.

To the class of soft hardware-failures, the more usual one of hard or permanent faults may be added. These include the well known problems of circuit failure due to ageing and shock and vibration. The distinction between soft and hard failures is that in the latter case the fault, once manifested, remains in evidence. Again the high densities involved in VLSI mean that such faults are more likely to occur, thus shortening the life of the device.

The net effect of all of the above points is that to gain the full benefit of these high-density fabrication methods, or to make software controlled equipment more reliable, it is becoming increasingly important to be able to design "fault tolerant" systems. One possible solution to these problems is some form of error detection coupled with an in-built redundancy. The error detection capability effectively provides a self-test capacity for the device, even after it has left the factory, whilst the redundancy in the system can be used mask-out any fault thus enabling the device to continue to function correctly. A "static" fault tolerant device that can mask-out faults under non-operational conditions will clearly improve yield and device lifetime (if the self-repair process is used intermittently). To protect against the soft failure, however, requires concurrent self-repair i.e. the ability to detect and mask faults in parallel with normal operations.

2 Fault Tolerant Techniques

There are many different approaches [2][3] to the task of making a system fault tolerant. A significant factor is the type of system being considered and the literature can accordingly be subdivided on the basis of the category into which the system being considered falls. A list of categories and some examples of both the category and the fault tolerant technique can be found in table 1. It should be realised that in some cases the distinction is artificial, and that the edges between categories are often blurred.

The majority of the work in the field of fault tolerant design has been concerned with the design of special purpose hardware or switching algorithms. The former category includes such circuits as AN-coded adders (see below). The idea here is to create fault tolerant systems by constructing them from fault tolerant components. In the case of the switching algorithms the interest is usually in adaptive configuration/routing in computer-based communication networks (meaning collections of complex, usually software controlled, processing elements). The object of the network may be anything from inter-mainframe-computer communication to dedicated hardware for systolic algorithms.

Category	Typical Members	Technique
Low Level Circuits	General circuits Modulo Reduction Adders/Multipliers Memories/data buses	Triple Modular Redundacy Self-checking Checkers Arithmetic Codes BCH Codes
High Level Circuits	FIR Filters Matrix Maths. Accelerators	BCH Codes Matrix Checksums Invariance Properties
Equipment	Systolic Array Multiprocessors	Hardware Controlled Reconfiguration Software Controlled Reconfiguration
Software	Computer Network Subroutine	Switched Packet Network Re-try Schemes

Table 1. Fault Tolerant Techniques.

Very little interest has been shown in the problem of algorithmic fault tolerance i.e. the design of hardware-independent fault tolerant mathematical algorithms. Such algorithms are concerned with the problem of achieving error-free results in mathematical calculations without having to specify the exact nature of the hardware. If a particular mathematical calculation were to be implemented in such an algorithm, either in software (e.g. general purpose computer) or hardware (e.g. systolic array), any fault with the underlying system (ALU or processing elements) will be detected and, ideally, corrected (see figure 1). Of the techniques mentioned, those based on arithmetic codes, BCH codes, matrix checksums and matrix invariance properties are most suitable for such an approach. These and several other techniques, which also appear to hold some promise, are discussed further in section 3.

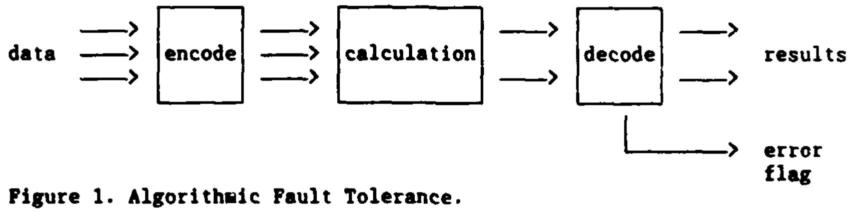

Figure 1. Algorithmic Fault Tolerance.

3 Algorithmic Fault Tolerance

3.1 Literature Survey

Abraham [4][5] and Luk [6][7] proposed the addition of row and column checksums to matrices in order to provide error protection in various matrix operations (multiplication, addition, LU and QR decomposition). The checksums are nothing more than the sum of the elements in a row or column but as Abraham noted these sums are invariant under the matrix operations mentioned. In a more recent paper [8] Abraham also briefly considered how to exploit any properties of the matrix that remain invariant under the given operation, in the detection of errors e.g. checking for a change in the norm of a row or column vector under a unitary transformation.

It has long been known that the convolution operation is equivalent to the multiplication of polynomials and this is exactly the frame work used in the theory of BCH error-correction codes. Redinbo [9] used these facts to incorporate error-correction in an FIR filter. As a result of this study he also noted that the so-called chord property[1] of the number theoretic transform effectively implies redundancy and so can also be used for error protection.

The AN and residue codes [10][11][12][14] have, up to now, been considered only in conjunction with special purpose hardware to construct fault tolerant arithmetic circuits. There is no reason, however, why these coding methods cannot be used at the algorithmic level in much the same way as the BCH codes were by Redinbo. In fact AN codes are entirely analogous to BCH codes, both codes being cyclic codes (i.e. ideals of a finite ring[2]: Z_{2^n-1} and $GF(q)[x]/\langle x^n-1\rangle$ respectively).

There are also several other areas of mathematics that contain results that may be of some use in the field of algorithmic fault tolerance. These and the above mentioned techniques are reviewed in the following.

1. necessary existence of conjugates.
2. Z_n is the ring of integers modulo n.
$GF(q)[x]/\langle x^n-1\rangle$ is the ring of polynomials with coefficients taken from the Galois field with q elements, modulo the polynomial x^n-1.

3.2 AN and BCH Codes

The basis of the use of cyclic codes for error protection is as follows. Let $\mathbf{R}$ be a "commutative ring with unity" (e.g. the ring of integers modulo a given integer) and consider the addition of $a' = g \cdot a \in \mathbf{R}$ and $x' = g \cdot x \in \mathbf{R}$, for arbitrary a and x and some given g (see figure 2).

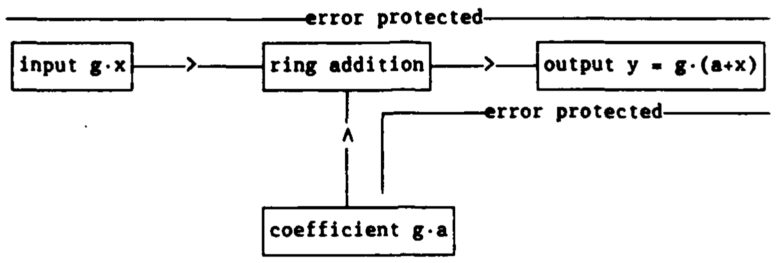

Figure 2. Cyclic Code for Addition

If an error should occur in the addition operation then the result y is given by

$$y = a' + x' + e = g \cdot a + g \cdot x + e = g \cdot (a + x) + e$$

Thus, provided g was chosen so that $e \not\equiv 0$ MOD g, for all $e \neq 0$, the error can be detected by the occurrence of a non-zero residue of y modulo g (i.e. by the fact that y is not a multiple of g). Indeed g may in fact be selected so that an error value e can be uniquely[3] recovered from the erroneous result and hence the correct result found. Consider the integers taken modulo 513:

$$Z_{513} = \{0,1,2,\ldots,510,511,512\}.$$

A single error correcting code results from the choice g = 19:

$$19*5 + 19*3 = 95 + 57 = 152 = 19*8.$$

3. the error value is unique provided that the least complex error is chosen.

However if an error occurs (a faulty carry circuit for bit 3, say):

```
      19*5 → 1011111
    + 19*3 → 0111001
             -------
             10010000 → 144.
```

Now 144/19 = 7 rem. 11, and the non-zero remainder indicates an error. In fact it can be shown that the only way to get a remainder of 11, with a single error, is if (-8) was added into the sum, thus the correct answer is

$$(144 -(-8))/19 = 8.$$

Note that by encoding the two inputs (a and x) in the above fashion both arms of the (dyadic) operation are protected (as indicated by the bars in figure 2).

The theories of AN and BCH codes are extensive ([10][11][12], [15][16][17]). The setting for AN codes is the ring of integers modulo 2^n-1, for some n, (cf. one's complement arithmetic) and so AN codes are clearly a good choice for attempting to develop a theory of fault tolerant (computer based) algebra. A fault tolerant algebra consists of redundant forms of multiplication and addition that can detect or even correct errors. In order to begin to look at this possibility, however, it is clearly necessary for the AN coding technique to be able to be applied to multiplication as well as addition. This is by no means straight forward: the AN code was originally designed to combat errors in addition only.

BCH codes on the other hand could be used in the field of signal processing by virtue of the equivalence between the convolution of finite sequences and the multiplication of polynomials[4] [18]. The regime for BCH codes is also that of polynomial algebra, however they also were originally designed for fault tolerant addition and the convolution operation is equivalent to multiplication.

One possible way to achieve error protection in the multiplication process is shown in figure 3. This is essentially the technique used by Redinbo [9] and Chien [13]. Notice that here only one of the two inputs is encoded (multiplied by g) and thus only this input is error-protected. The reason behind this is the

4. The coefficients of the polynomials are the elements of the sequences.

fact that product of two multiples of g is a multiple of g^2:

$$(g \cdot a) \cdot (g \cdot x) = g^2 \cdot (a \cdot x)$$

Thus the product would, in general, be the encoded form not of $a \cdot x$ but $g \cdot a \cdot x$ (see example at the end of section A.2.2). There are other short comings with the use of cyclic codes which are discussed further in Appendix A, along with some ideas on how to overcome them.

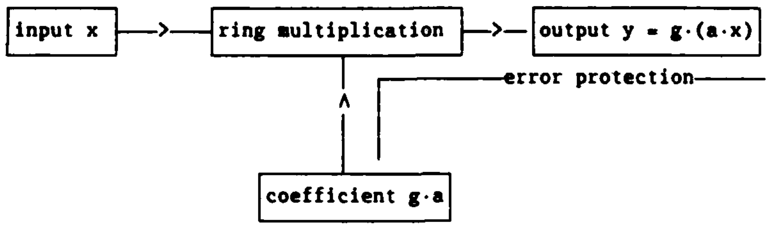

Figure 3. Cyclic Code for Mulitiplication

3.3 Number Theoretic Transforms

Redinbo's solution [9] to the above problem of protecting both operands of a multiplication (specifically in $GF(q)[x]/\langle x^n-1\rangle$ i.e. for polynomials), was to use the redundancy of the number theoretic transform (NTT) when the input is restricted to a proper subset of the domain. Redinbo considered the problem of how to error protect an FIR filter, i.e. the convolution of finite sequences. One way to perform this operation is by pointwise multiplication in a transform domain (figure 4. cf. convolution via DFT).

It is well known that the input space (the domain) of the DFT is the space of (finite) sequences of complex numbers. If the input is restricted so as to be a real sequence, the output sequence obeys certain relationships as a consequence: $V^*(k) = V(N-k)$, $0 \leq k < N$. The domain of a NTT is the space of finite sequences with coefficients from a certain[5] extension field of GF(q) ($GF(q^m)$ say[6]). If the input is then restricted to sequences over GF(q), the resulting "spectrum" again obeys certain relationships. These relationships are some times collectively referred to as "the chord property" of the sequence. The use of the chord property allows the input variable to be checked for errors - the coefficient

5. chosen so that it contains a primitive Nth root of unity
6. it may turn out that m=1.

being precomputed (as signified by the brackets in figure 4).

$$\begin{tikzcd}
\underline{x} \in GF(q)^n \arrow[r] & \boxed{NTT} \arrow[r, dash] & \underline{X} \in GF(q^m)^n \arrow[r] & \boxed{\cdot} \arrow[r] & \boxed{INTT} \arrow[r] & \underline{a}*\underline{x} \\
\left\{ \underline{a} \in GF(q)^n \right. \arrow[r, dotted] & \left. \boxed{NTT} \cdots \right\} & \underline{A} \in GF(q^m)^n \arrow[ur] & & &
\end{tikzcd}$$

Figure 4. Convolution via NTT

Specifically, the chord property of the NTT is

$$(V_j)^q = V_{((jq))_n} \qquad j = 1,..,n$$

where $\underline{V} \in GF(q^m)^n$ is the NTT of $\underline{v} \in GF(q)^n$. In effect this says that the transform coefficients that are indexed by the elements of a certain subset (a cyclotomic subset) can be determined from just one coefficient. Thus any error, not itself an allowable input, will cause the chord property to be violated and hence the error can be detected. The correct value can be found from the other coefficients and the above formula. If there are L_j members of the cyclotomic subset C(j) then the minimum distance of the "code" is L_j and hence it can detect L_j-1 errors.

As the NTT is linear, it commutes with the addition operator and so it is possible to protect the addition of sequences as well as their convolution. This then constitutes a fault tolerant algebra of finite sequences. The exact error-protection power of such an algebra is as yet unknown. It is also worth noting that the algebra of polynomials is also the setting for most Galois fields. It therefore seems likely that this technique can be used to construct a fault tolerant Galois field algebra.

3.4 Residue Codes

The main draw-back with AN codes is the fact that the minimum distance of the code tends to vary inversely with the size of code (number of codewords). Thus whilst it is possible to have single error-correcting codes with a reasonable number of codewords, the large distance codes tend to have very small sizes e.g.

the generator A = 13797 produces a distance 6 code (double error correcting and triple error detecting) but the arithmetic has to be done modulo $2^{18}-1$ so there are only $(2^{18}-1)/13797 = 19$ codewords.

A more attractive option is residue codes. These can be constructed, in amongst other ways, from AN codes. The minimum distance, and thus the error-protection capability, is preserved and the size of the code is that of the ring e.g. the residue code based on the AN code with A = 13797 (see above) has distance 6 and $2^{18}-1$ codewords. Residue codes also work just as well for multiplication as for addition. The main draw-back, if it is one, is that the coded form of a number is in the form of a vector and the arithmetic algorithms for the individual components are different from one another. This clearly adds to the complexity of the system.

A residue code works by keeping a check on the number by means of a separate "check number". The check number is the result of performing the same arithmetic operations on the check number as on the original number, except that the arithmetic is modulo a suitable base. The check arithmetic can thus be less complex than the main arithmetic e.g. a check modulo 2 would only require 1 bit arithmetic. Error detection is afforded by a comparison between the check number and the main result modulo the check base: a modulo 2 check clearly detects any error that turns an even number into an odd one, and vice versa. These modulo sums are, for obvious reasons, referred to as "checkers" (see figure 5).

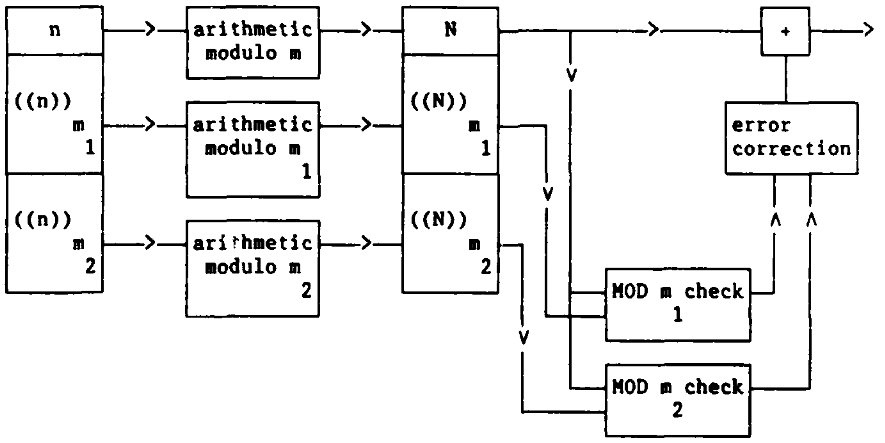

Figure 5. Residue Coded Arithmetic.

Clearly it is possible for one, or more, of the checkers to be in error as well as the main result. This makes the task of error protection slightly more difficult for the residue codes as compared to that for AN codes. The theory for this is however well established. It should be noted that as the full-range answer is available the problem of sign determination and magnitude comparison that besets RNS is not present.

3.5 Invariant properties

3.5.1 length, trace etc.

In certain mathematical operations various properties of the input variables are invariant e.g. length of vectors under unitary transformations, matrix trace under addition or multiplication. Such properties could be used to check on the final result i.e. for error detection. It is not clear as yet whether they can be used for error correction.

3.5.2 Operator Invariant Subspaces.

There appears to exist a certain amount of work, at least in finite linear algebra ([20] section 11.2), on subspaces that are invariant under a (linear) operator. Further investigation is called for.

3.5.3 Spectral Methods

Blahut [17] has shown that there exists a "frequency" domain representation of (BCH) error correcting codes. Such a representation appears to have several advantages from a coding point of view including the fact that the syndrome of a received word is just its DFT. The error correcting properties arise from the fact that the encoder can be considered to be a filter that has a certain distribution of zeros. When errors (noise?) are added to the filtered (encoded) sequence the result no longer has a spectrum with these zeros. The presence of errors can thus be detected by the non-zero spectral coefficients (via the DFT). Error correction results from a reconstruction of the error sequence based on a knowledge of some of its DFT coefficients (i.e. those at the zero positions).

There may be some possibility of exploiting the features of a convolution operator to obtain a degree of error protection without adding (further?) redundancy. Under suitable interpretations of "frequency" such an approach may well work for any linear operator, including matrix operators.

3.6 Codes over the Real Field.

Most of the techniques reviewed so far are based on finite algebraic structures and so are only suitable for integer problems. As most problems of interest are naturally represented in the Real/Complex field, it is desirable to consider error protection over the real field.

The most well known of the (data transmission) error correction codes, the block and the convolution codes, are based on the idea of convolving the data sequence with a fixed sequence. The result of this convolution has certain properties (e.g. zeros at certain frequency values) that enable any additive noise to be detected and then removed. The extension to real sequences should be considered. The similarity of the Berlekamp decoding algorithm and the Yule-Walker equations in Linear prediction/Kalman filters constitutes an appealing, if not conclusive, argument in favour the existence of Real valued codes.

3.7 Codes for Matrices

3.7.1 2D Codes

There exists a certain amount of work (e.g. [19]) on the subject of 2-D error correction codes. These coding schemes are based on the idea of appending check rows and columns to a data matrix. Each check digit is the result of a parity check over a row/column. Clearly such codes are the obvious extensions to Abraham's work. It is not known at present[7] what properties these codes have i.e. whether or not they are linear and hence invariant under addition and scalar multiplication.

3.7.2 Ideals in $M_{n \times m}(K)$.

As most of the successful error correcting codes to date are based on the use of an ideal as the code space, it should be fruitful to investigate the structure of ideals in the ring of n×m matrices over a field K (Mn×m(K)). The pure mathematics of this problem has, of course, already been solved [20].

7. by the author

4 Conclusion

The field of algorithmic fault tolerance appears to be wide open, with only a handful of people actively engaged in relevant work. There is probably still a lot of work in the area of (hardware) arithmetic codes/systems that could be pertinent to the algorithmic case. Further study of the research literature, as opposed to text books, is required.

Certainly there are still a lot of areas/techniques (see section 3) that, at first sight, look promising and warrant closer investigation. There are several problems with the use of AN codes for protecting complicated algebra. The advantages of such a technique are, however, such that more effort could be justified in this direction. It must however be remembered that (useful) AN codes are only single error correcting and that for higher error protection a switch to the use of residue codes would be required. The latter are at least as good as, and often better than, AN codes in all respects except for the complexity of the arithmetic (vector instead of scalar). Exactly how much of a penalty this is remains to be seen.

There is still a lot of work to be done, even for established methods like AN and residue codes and Redinbo's BCH-protected convolution, in establishing exactly how many and what sort of errors can be tolerated. Care must be taken in this question as to the definition of an error. The conventional approach, in AN and residue codes, is to assume a serial ripple-adder type circuit and so define an error as an additive difference of $\pm 2^i$. Increasingly in VLSI, arithmetic circuits are pipelined or based on redundant number representations for which the definition of $\pm 2^i$ as an single error may not be suitable. Also the question of determining, in terms of percentage overheads, how much the error protection costs must not be neglected. The underlying system that Redinbo proposed for a fault tolerant FIR filter has been known for a long time and has not been implemented in many real-time systems presumably due to the complexity involved.

The anticipated possibilities of the spectral approach to BCH codes as applied to the protection of FIR filters, and linear operators in general, seem to be great. This is the probably the next topic to which to devote the most effort. The use of an extension field (section 3.3) and the resultant redundancy in the arithmetic could prove fruitful depending on the cost of implementing finite field arithmetic, which has in the past proved to be great. The two other topics that appear to hold the most immediate potential are the 2-D codes (section 3.7) and the BCH/Yule-Walker connection (section 3.6).

5 Bibliography

[1] Physical Failures and Fault Models of CMOS Circuits.
S.A. Al-Arian, D.P. Agraval.
IEEE trans. CAS-34(3), Mar.1987, pp269/79

[2] Fault Tolerant Systems. A. Avizienis.
IEEE trans. C-25(12), Dec.1976, pp1304/12

[3] A Survey of Fault Tolerant Architectures & its Evaluation.
W.C. Carter, W.G. Bouricuis.
Computer, vol.4, No.1, Jan. 1971, pp9/16.

[4] Algorithm Based Fault Tolerance for Matrix Operations.
K-H. Huang, J.A. Abraham.
IEEE Trans. C-33(6), June 1984, pp518/528.

[5] Fault Tolerant Matrix Operations on Multiple Processor
Systems Using Weighted Checksums.
J-Y. Jou, J.A. Abraham.
SPIE Proc. vol.495, Real Time Signal Processing VII,
19-24 Aug. 1984, San Diego, CA., pp94/101.

[6] Fault Tolerant Matrix Triangularizations on Systolic Arrays.
F.T. Luk.
Tech. Rpt. EE-CEG-86-2, Computer Eng. Group, School of Electrical Eng.,
Cornell Univ., Ithica, NY, USA. Feb.86.

[7] An Analysis of Algorithm Based Fault Tolerant Techniques.
F.T. Luk, H. Park.
Proc. SPIE vol.696: Advanced Algorithms and Architectures for Sig.
Proc. 19-20 Aug. 1986, San Diego, Ca. pp 222/7

[8] Fault Tolerant Systems for the Computation of Eigenvalues
and Singular Values. C.Y. Chein, J.A. Abraham.
Proc. SPIE vol.696: Advanced Algorithms and Architectures for Sig.
Proc. 19-20 Aug. 1986, San Diego, Ca. pp228/37

[9] Fault Tolerant Digital Filtering Architectures using Fast
Finite Field Transforms.
G.R. Redinbo.
Signal Processing, vol.9, 1985, pp37/50.

[10] Error Coding for Arithmetic Processors
T.R.N. Rao
Academic Press, 1974, ISBN 0-112-580750-3

[11] Error Detecting Codes, Self Checking Circuits &
Applications. J. Wakerly.
Elsevier North-Holland, 1978. ISBN 0-444-00256-1

[12] Error Correcting Codes in Computer Arithmetic.
J.L. Massey, O.N. Garcia
Ch.5 of Advances in Information System Science, Vol. 4
J.T. Tou (Ed)
Plenum Press, 1972, ISBN 0-306-39404-9

[13] Error Correction in High Speed Arithmetic.
R.T. Chien, S.J. Hong.
IEEE trans. C-21(5), 1972, pp433/438.

[14] Digital Filters with Fault Tolerance.
M.H. Etzel, W.K. Jenkins.
Proc. 1979 Joint Automatic Control Conf., pp187/192.

[15] The Theory of Error Correcting Codes.
F.J. MacWilliams & N.J.A. Sloane.
North-Holland, 1977. ISBN 0-444-85009-0/0-444-850104-4

[16] Algebraic Coding Theory. E.R. Berlekamp.
McGraw-Hill, 1968.

[17] Theory and Practice of Error Control Codes.
R.E. Blahut.
Addison-Wesley, 1983. ISBN 0-201-10102-5

[18] Number Theory in Digital Signal Processing.
J.H. McClellen & C.M. Rader.
Prentice-Hall, 1979. ISBN 0-13-627349-1

[19] Multivariate Polynomials in Coding Theory.
H. Imai
Proc. 2nd Int. Conf. Applied Algebra, Algorithmics &
Error-Correcting Codes, Oct.84, Toulouse, Fr. pp36/60
Springer-Verlag, Lecture Notes in Computer Science, no.228.

[20] Rings, Modules & Linear Algebra.
B. Hartley & T.O. Hawkes.
Chapman-Hall, 1970.

APPENDIX A

Fault Tolerant Ring Algebra

A.1 Perceived Problems and Some Conjectures

As explained in section 3.2 in order to have a viable fault tolerant algebra using AN or BCH codes, a multiplication operator that commutes with the coding operation is required. Previous work [9][13] suggested a method whereby only one of the operands is encoded. This is because of the presence of an extra factor of g - the code generator - in the product if both operands are encoded. Although knowledge of the presence of this extra factor would allow a division by another factor of g and thus restore the correct value, the correction would have to be performed after each multiplication and not at the end of the calculation as desired. Further in a complex algorithm it is usually necessary to perform more than one multiplication, thus, even with only one input per multiplication encoded, factors of g build up with each multiplication (see figure A1). In calculations that contain products of sums the factors of g would be hopelessly caught-up in the rest of the expression with no hope of removing them.

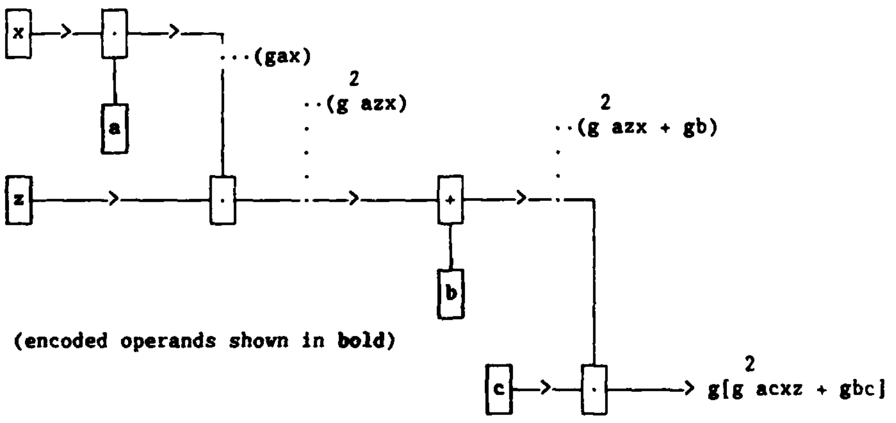

Figure A1. Arithmetic with AN Coded Operands

Even if a correction were applied after each multiplication there still are short-comings with this approach. Consider the basic mechanism of error protection for the product a·x. If the coefficient a is encoded as a′ = g·a then the result of a faulty calculation could be

$$(x + e_1)(ga + e_2) + e_3 = g(ax) + (xe_2 + e_1ga + e_1e_2 + e_3)$$

where e_1 is an error associated with the inputing and any preprocessing of the variable x,

e_2 is due to the erroneous storage of the coefficient a ,

and e_3 accounts for any errors on the main section of the multiplier.

As before the required result (ax) is a multiple of g but then so is the error factor (e_1ga). Thus, with a proper choice of generator g, it may be possible to protect the circuit against errors of the form e_2 and e_3 but it is never possible to provide protection against corruption of the variable x. It is also worth noting at this point that whereas the standard theory will work for the error e_3, the error e_2 appears as a multiple of x so that extra care must be taken that possible error values (e_2) and input values (x) are not such that the product e_2x is a multiple of g. If this were the case then the error will effectively be masked by the particular input value. The same can be said with reference to the term (e_1e_2). A possible solution may be to choose g such that it is prime (e.g. the Brown-Peterson AN codes - see [10] p101).

A solution to the problem of accruing factors of g is the use of idempotent generators. Under certain conditions it can be shown that there exists a generator for a cyclic code that is idempotent i.e. $\gamma^2 = \gamma$ (see section A.2). Using such a generator means that not only is the problem of accumulating factors of (in this case) γ solved but also both operators could now be encoded in the multiplication process (figure A2) or more significantly that the input may be left in coded form (from the previous operation).

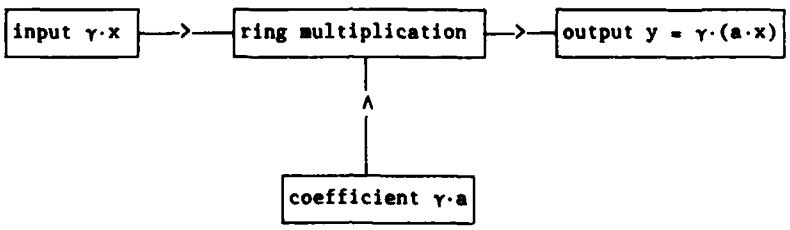

Figure A2. Idempotent Cyclic Code for Multiplication

This does not, however, mean that the input is protected by the code. On the contrary having the input as a multiple of γ can be considered to make matters worse. Consider

$$(\gamma x + e_1)(\gamma a + e_2) + e_3 = \gamma(ax) + (\gamma x e_2 + e_1 \gamma a + e_1 e_2 + e_3)$$

Here the output error terms $\gamma x e_2$, $e_1 \gamma a$ are both multiples of γ and hence are undetectable. Thus any error associated with the storage of the coefficient a (e_2) can only be detected in conjunction with an input error e_1 via the $e_1 e_2$ term.

One solution to this problem could be to use a mixture of codes. The input variable (x) could be encoded using one code, and the coefficient (a) using another:

$$(g_1 x + e_1)(g_2 a + e_2) + e_3 = g_1 g_2(ax) + (g_1 x e_2 + e_1 g_2 a + e_1 e_2 + e_3)$$

Such coding may then afford protection against errors of type e_1, e_2. What is then required is to be able to combat type e_3 errors in such a way that the output of the multiplier is encoded in the same code as x. This might be achieved by requiring that the generator for the coefficient code be an idempotent with respect to the generator of the input code i.e. such that $g_1 \cdot g_2 = g_1$ but this requires further research.

McWhirter[8] has pointed out that the problem of masking errors, particularly input errors (type e_1), in itself may not be that disastrous since the inputs to each multiplier could be checked. This is clearly not as satisfactory as a single check at the end of a (complex) calculation but is better than having to check the inputs to each adder as well as each multiplier.

Given the elegance of residue codes it may be argued that AN codes have no use in fault tolerant systems. The author believes, however, that this is not necessarily so. There are many low-distance AN codes that have quite a large number of codewords and hence have some practical value. The lack of error correcting power may not be all that prohibitive if the expected frequency of errors is low and detection, rather than correction, is acceptable. One application that springs to mind is in residue code systems. Here (see

8. private communication

section 3.4) it is possible for the checkers to be in error, however in this case it is not necessary to correct the result: knowing it is in error is sufficient. The checker circuits could then benefit from being coded for error detection.

AN codes also have the advantage that the arithmetic is "standard", unlike residue codes which have a parallel structure, thus AN codes could be used in on-line testing of conventional circuits. The basic fact that allows any coding scheme to work is that the codewords are recognisable as such. In the case of cyclic codes, they are all multiples of the generator. As AN codes are designed such that the codewords remain codewords under addition and, to a certain extent, multiplication, if the input to an algorithm are AN codewords then the output should be one also. Thus the inputs to a system could be periodically put in AN coded form and the output checked to see if it is also a codeword. It may not be possible to detect all errors (see above) but certainly some errors will be found.

A.2 Idempotents

Definition:

Let $\mathbf{R}$ be a ring, then $e \in \mathbf{R}$ is an idempotent if

$$e^2 = e$$

Theorem:

Let $\mathbf{K}$ be a Euclidean domain. An ideal[9] $\langle g \rangle$ in the Euclidean domain[10] $\mathbf{R} = \mathbf{K}/\langle j \rangle$, $j \in \mathbf{K}$ and $g \mid j$, can be generated by a unique idempotent γ, provided $(g, j/g) = 1$.

E.g. let $\mathbf{K}$ be the ring of integers, if $j = 65$ the ideal $\langle j \rangle$ is the set of integer multiples of 65 and the Euclidean domain $\mathbf{K}/\langle j \rangle$ is the ring of integers modulo 65 (i.e. Z_{65}). Now consider the ideal of Z_{65} generated by $g = 13$ (i.e. all integers that are multiples of 13 modulo 65):

$$\langle g \rangle = \{13, 26, 39, 52, 65 \equiv 0\}$$

On the other hand

$$\langle 26 \rangle = \{26, 52, 78 \equiv 13, 104 \equiv 39, 130 \equiv 0\} = \langle g \rangle,$$

and

$$(26)^2 = 676 \equiv 26 \text{ MOD } 65.$$

Proof: (cf. [15] ch.8)

Define $h = j/g$. If $(h,g) = 1$ then, by the Euclidean division algorithm, there exists p, $q \in \mathbf{R}$ such that

$$pg + qh = 1$$

Consider $\gamma = pg \in \mathbf{R}$,

$$\gamma(\gamma + qh) = \gamma$$

i.e.

$$\gamma^2 + pgqh = \gamma$$

9. $\langle g \rangle$ is the principal ideal generated by the element g.
10. $\mathbf{K}/\langle j \rangle$ is the ring of residue classes of $\mathbf{K}$ modulo $\langle j \rangle$

Multiplication is commutative in an Euclidean domain, hence

$$pgqh = (pq)(gh) = (pq)j = 0$$

Thus

$$\gamma^2 = \gamma$$

and γ is an idempotent.

In the case where $R = Z_{65}$ and $g = 13$, we have $h = 65/13 = 5$. As 13 and 5 are relatively prime we can find integers p, q such that

$$13p + 5q \equiv 1 \text{ MOD } 65.$$

In fact $p = 2$, $q = 8$ and hence $\gamma = 2*13 = 26$.

Now as

$$\gamma = pg$$

then

$$\langle\gamma\rangle \subseteq \langle g\rangle,$$

but

$$\begin{aligned} g\gamma &= g(1 - qh) \\ &= g - qj \\ &= g \end{aligned}$$

i.e.

$$\langle g\rangle \subseteq \langle\gamma\rangle$$

Thus

$$\langle g\rangle = \langle\gamma\rangle.$$

and γ generates the ideal.

Clearly the ideal generated by the element 13 must contain the element 26, so that <26> will be a subset of <13>.
But as $13*26 = 338 \equiv 13$ MOD 65, the element 13 can be considered to be a multiple of 26 and hence $\langle 13\rangle \subseteq \langle 26\rangle$

The idempotent γ is unique, for if $f \in R$ is another idempotent that generates $<g>$ then

$$\gamma \in <f>$$

i.e.

$$\gamma = af \qquad \text{for some } a \in R$$

Thus

$$\gamma f = af^2 = af = \gamma.$$

Similarly

$$f \in <\gamma>$$

i.e.

$$f = b\gamma \qquad \text{for some } b \in R$$

thus

$$\gamma f = b\gamma^2 = b\gamma = f.$$

Hence

$$\gamma = f.$$

If the element f generated the ideal then the element 26 must be a multiple of f: $26 \equiv af$ MOD 65.
Hence

$$26f \equiv (af)f \equiv af^2 \text{ MOD } 65.$$

If f is also an idempotent (i.e. $f^2 = f$) then

$$26f \equiv af \equiv 26 \text{ MOD } 65.$$

We already know that the element 26 generates the ideal, so that f must be some multiple of 26: 26b say.
Then

$$26f \equiv 26(26b) \equiv 26b \equiv f \text{ MOD } 65.$$

Hence f = 26 and the element 26 is the only idempotent generator.

∎

A.2.1 Idempotent Generators for BCH Codes

A BCH code is an ideal in $GF(q)[x]/\langle x^n-1\rangle$, for $(q,n) = 1$, generated by $g(x)$ a factor of x^n-1. It is well known that the roots of $g(x)$ consist of conjugate sets ([15] p199) and hence that $(g(x),(x^n-1)/g(x)) = 1$. Thus the conditions of the above theorem hold and the code can be generated by the idempotent

$$\gamma(x) = p(x)g(x)$$

$$\text{where } p(x) \equiv (g(x))^{-1} \text{ MOD } (x^n-1)/g(x).$$

A.2.2 Idempotent Generators for AN Codes

A cyclic AN code is an ideal in Z_{2^n-1}, generated by A, a divisor of 2^n-1. Thus provided that $(A,2^n-1/A) = 1$, the code can be generated by the idempotent

$$\alpha = pA$$

$$\text{where } p = A^{-1} \text{ MOD } 2^n-1/A.$$

It is not known[11] if the condition that A and $(2^n-1)/a$ be relatively prime can always be met. The condition certainly holds for some values of A. Consider the single error-correcting AN code in the ring Z_{65}, with $A = 13$ and $0 \leq N \leq 5$. Here we have

$$\alpha = 26.$$

To see the code at work in a multiplication consider the calculation

$$(a + b)\cdot c \text{ MOD } 5$$

where $a = 2$, $b = 4$, $c = 3$. Encoding the values a, b, c as AN codewords we get

$$a' = 52,$$
$$b' = 92 \equiv 39 \text{ MOD } 65,$$
$$c' = 78 \equiv 13 \text{ MOD } 65.$$

With perfect arithmetic we get

$$(52 + 39)*13 \equiv 26*13 \equiv 3*26 \text{ MOD } 65$$

Which is the coded form of 3, the correct answer. Repeating the calculation with the original generator $A = 13$, we find:

$$a' = 26$$
$$b' = 52$$
$$c' = 39$$

and hence

$$(26 + 52)39 = 78*39 \equiv 52 \text{ MOD } 65.$$

11. by the author

But 52 = 13*4 i.e. the coded form of the number 4, which is not the correct answer.

DOCUMENT CONTROL SHEET

Overall security classification of sheet UNCLASSIFIED

(As far as possible this sheet should contain only unclassified information. If it is necessary to enter classified information, the box concerned must be marked to indicate the classification eg (R) (C) or (S))

1. DRIC Reference (if known)	2. Originator's Reference Memorandum 4214	3. Agency Reference	4. Report Security Classification Unclassified
5. Originator's Code (if known) 7784000	6. Originator (Corporate Author) Name and Location Royal Signals and Radar Establishment St Andrews Road, Malvern, Worcestershire WR14 3PS		
5a. Sponsoring Agency's Code (if known)	6a. Sponsoring Agency (Contract Authority) Name and Location		
7. Title ALGORITHMIC FAULT TOLERANCE			
7a. Title in Foreign Language (in the case of translations)			
7b. Presented at (for conference papers) Title, place and date of conference			

8. Author 1 Surname, initials Proudler I K	9(a) Author 2	9(b) Authors 3,4...	10. Date 9.88	pp. ref. 26
11. Contract Number	12. Period	13. Project	14. Other Reference	
15. Distribution statement Unlimited				

Descriptors (or keywords)

continue on separate piece of paper

Abstract

A reduction in the minimum attainable feature size in integrated circuits has lead to the possibility of more and more complex circuits being built on a single chip (VLSI). This technological advance brings with it the need to make these circuits fault tolerant: to increase yield and reliability and to reduce testing times. This Memorandum briefly reviews current techniques for designing fault tolerant circuit before concentrating on a new, high-level fault tolerance technique: algorithmic fault tolerance.

The concept of algorithmic fault tolerance is explained and various techniques are reviewed with regard to their suitability for providing fault tolerance for signal processing algorithms. Suggestions are made for the direction for further research.