Towards Hamming Processor

Shlomi Dolev

Department of Computer Science,
Ben-Gurion University, Beer-Sheva, Israel
E-mail: dolev@cs.bgu.ac.il

Sergey Frenkel

Institute of Informatics Problems,
Russian Acad of Sc., Moscow, Russia
E-mail: slf-ipiran@mtu-net.ru

Abstract. Given Hamming coded or systematic Bose–Chaudhuri-Hocquenghem (BCH) or Reed-Solomon correctable operands x and y and an operation op, we present arithmetic to compute the operation Correct(x) op Correct(y) while preserving the correct-ability of the result. If the Hamming distance of x(y) from Correct(x) (Correct(y)) is d_x (d_y, respectively) and the maximal correctable distance is d_c then our implementation can withstand $d_c - (d_x + d_y)$ faulty gates. We also present a slightly more expensive implementation which corrects the inputs when no faulty gates exist. In contrast to traditional Hamming code-based approaches to operational block protection, we protect the operation themselves rather than only the registers of input/output data.

Keywords: Soft Errors, Hamming codes, BCH codes, Reed-Solomon code, fault tolerance.

1. Introduction

Fault tolerance is one of the most fundamental issues in computer science (e.g., [1]). Recently, soft-errors (Soft Error Upset –SEU) makes the issue of error correction during processing (and not only in memory) critical (e.g.,[2]), especially for nanoelectronic components. Redundancies in memory (e.g., error correcting codes, RAID) or time (repeated computations to re-validate the results) are the means used for coping with faults. As for processing the TMR (Triple Modular Redundancy), where the majority of over replicates of computing devices is used to cope with one third of the faulty computing devices [3]. The TMR approach is simple, but seems to be too expensive with regards to classical error correcting codes (e.g. Hamming codes) that do not duplicate bits but use the Hamming distance between codewords to correct errors. Besides, TMR does not mask soft-errors since, for example, the soft-errors may change the result of the last gate of the TMR circuit.

Other transient and SEU error protection techniques are based on Error Correcting Codes (ECC), known in channel coding. Presently error correction circuitry is included in various digital circuits, VLSI and microprocessors in particular, to achieve the fault tolerancy by making integrated circuit (IC) implementations [4]. The ECC techniques in the circuit design can be considered from the viewpoint of their use for different SEU types, as well as for their use for specific parts of circuits designed. Among the ECC, the Hamming code is most popular. The Hamming coding is used to correct one bit errors. However, a single defect on the chip can cause multiple output errors in case they use common intermediate results. Therefore, the Hamming code can be non-effective if the probability of the multiple bit errors is rather high. In this case other types of linear codes (Reed-Solomon, BCH) can be more effective. The use of Hamming coding is different in the scope of memory block of a circuit protection and operational block protection. Numerous publications on error correcting for

circuits [4,5,6] show that in fact, the Hamming protection is used only for memory elements of the circuits, although it is sometimes claimed necessity to use this approach for arithmetic blocks [7]. Note that protection of the enable signals in arithmetic blocks is also an important problem [7].

In this work we suggest ways for using Hamming, BCH and Reed-Solomon codes not only for memories but also for processing, as if the entire universe of computing memory and arithmetic operations is in linear codes, Hamming codes, for example, taking into account also the enable signal protection. We will consider the logic level of digital circuits' representation. The core idea of the paper is to provide proven reliability of an operational (arithmetic-logical (ALU)) block with respect to permanent and transient faults with given Hamming distance. This is different from both replication-based (TMR) and Error Correction Codes based (Hamming protection, self-checking etc) traditional and advanced methods, when there are some unreliable blocks (voting schemes, comparator etc.).We present techniques for designing Hamming, BCH and Reed-Solomon codes based processors, or in short "Hamming processor", as the Hamming distances is based on the errors correcting linear codes. In particular, we present an approach to designing ALU and opcodes. The inputs and outputs, the operands and microprograms are all represented by Hamming/BCH codes and the processing preserves (or even corrects when needed) the Hamming distance of the results. In the sequel we first refer to the case of the Hamming code (later we address the BCH and Reed-Solomon codes). We apply a micro operation *op* to two Hamming-encoded binary represented operand s x, y including both operands bit vectors d_x, d_y, and Hamming parity bit vectors of the operands h_x, h_y. Assume that both *x* and *y* are correctable, namely Correct(*x*) will output the desired operand bit vector for *x*, and Correct(*y*) will output the desired operand bit vector for *y*. The goal is to find a way to perform the operation *op* so as the result *r* of the operation would be a correctable Hamming code of Correct(*x*) *op* Correct(*y*) in spite of faulty gates or faulty inputs that sum-up to less than the correctable distance of the Hamming code used.

Note that a faulty gate may be the one just before the output thus, it is impossible to ensure that all the output bits $r=\{d_r,h_r\}$ are correct. However, as we show, we can ensure that *r* is correctable. Note further that an attempt to compute *r* by assigning d_r to be Correct(*x*) *op* Correct(*y*) and then using the resulting d_r for computing h_r will not solve the problem, since a single faulty gate which changes the value of a bit in d_r, just before computing h_r will result in a wrong uncorrectable *r*.

Thus, in this paper we propose a design methodology for building reliable computational (both logic and arithmetic) elements, using the fundamental strategy of fault-tolerance providing operations with ECC-protected operands, preserving Hamming distance during these operations, not providing protection of outputs of individual functions through the use of there coding, as it is made in traditional using of ECC [8]. This approach guarantees the obtaining of a correctable result of any bit-wise logical and arithmetic operations, while the traditional approaches try only to minimize the error probability of the blocks mentioned above.

Paper organization. Section 2 describes related works. Section 3 presents the ALU architecture for the Hamming processor. Opcode selection protection issues appear in Section 4. The BCH and Reed-Solomon extensions for covering more than one error are presented in Section 5. Conclusions and discussions of future work appear in Section 6.

2. Related Work

As it mentioned above, the circuitries used to protect the VLSI and microprocessors are usually based on the TMR or ECC (Hamming code), or a combination of them [9]. Next we summarized state-of-the-art techniques for SEU protection, especially for operational blocks, which are mostly combinational circuits.

Traditional approaches to TMR. In the TMR-based techniques for a memory block, all memory cells are triplicated and a voter is added to a circuit's cell outputs. For a system with *n* outputs, TMR systems use *n* single-bit voters to correct a single error. Two redundant circuits have to be added. If one of the three circuits produces an erroneous output, and the two other circuits produce the correct result, then using majority voting provides a correct output. The voter can be implemented by a few logic gates, for each bit [9,10]. The difference between SEU protection in combinational and sequential circuits is due to a difference of impact of transient faults in these two classes of circuits. Traditional TMR technique assumes that combinational structure of a circuit is not subject to bit flip due to some transient errors, making no necessary protection schemes for a data path, as a transient fault in the combinational logic may or may not be latched by a storage cell. As a result, for logic/arithmetic combinatorial blocks the voter circuit does not need to be inserted right after the module outputs, but voting can be accomplished in the next sequential logic block (e.g., in the result registers). However, in modern nanotechnology designs, higher clock frequencies increase the chance of glitches being captured [8,11]. The chance is to be equal to the transient fault rate of unprotected memory elements. As note by some authors, in presence of protection of sequential elements, combinational logic will quickly become the dominant source of the transient faults [12].

Advanced TMR techniques. A cause of an inability of the conventional TMR to mask glitches from the combinational circuit is that the redundant registers of conventional TMR are controlled by the same clock. When the glitch from the combinational circuit propagates to the sequential circuit at the rising edge of the clock, all the three registers will capture the glitch. Similarly, when a soft error occurs on the clock signal or the reset signal, all the redundant storage cells will execute incorrectly [12]. Therefore arose some new approaches to protecting combinational logic from the transient faults. These redundancy techniques called Space-Time TMR (ST-TMR) and Enhanced ST-TMR (EST-TMR), which spaces the redundancy of traditional TMR, add time redundancy with double edge triggered registers [12]. Note, that the common drawback of the TMR still exists, namely, a possibility of SEU to hit the last gate before a primary output. However, even these improvements do not resolve the mentioned above problem concerning the unreliability of the last gate in a voter.

Traditional Hamming codes for circuits protection. Hamming coding is considered as an alternative to TMR because they have a low-complexity decoding circuit and is most efficient for correcting a single error. In addition, since Hamming code is a systematic code there is no need to recode the original outputs. It is still used mostly for memory and register file protection [10]. In [10] a protection of a microprocessor Data Path is considered, but, in fact, they consider the protection of the output registers of the ALU. Moreover, the architecture

suggested in [8] supposes one encoding and one decoding for each operation. However, both encoding and decoding circuits are not protected. Note, that as it stressed in [6] the decoding circuit that performs the final error correction is not completely protected (although errors in its operation can be detected by making the circuit self checking). For this reason the decoding circuit must be as small as possible in comparison with the circuit to be protected.

TMR and Hamming codes trade-off. Numerous publications explore trade-offs between TMR and Hamming codes for micro-architecture level soft error mitigation solutions. For example, according to [3], TMR is the best solution if the latency of the register file is more important. [3] shows that, for memory-rich systems like microprocessors, the Hamming coding has smaller area overhead (less than double) in comparison with TMR [10].

As the TMR is subject to voting unreliability, and the traditional approach to Hamming coding is focused mostly in memory/registers (and also have a weak chain because they leave unprotected bits after Hamming decoding), we suggest new approaches to ECC – based operational blocks protection, based on preservation of Hamming distance during operations.

3. Hamming Distance Preserving Arithmetic Logic Unit

We first introduce an ALU design that preserves the correct ability of the Hamming coded operands in the result of the operation. Given Hamming coded correctable operands *x* and *y* and an operation *op*, we present arithmetic to compute the operation Correct(*x*) *op* Correct(*y*), while preserving the correctability of the Hamming coded result. If the Hamming distance of *x* (*y*) from Correct(*x*) (Correct(*y*)) is d_x (d_y, respectively) and the maximal correctable distance is d_c then our implementation can withstand $d_c-(d_x+d_y)$ faulty gates [4].

Later we will present a slightly more expensive implementation that corrects the inputs when no faulty gates exist, and produces a correctable result when less than d_c gate faults occur during the processing.

We will consider both logical and arithmetic operations, namely:

- *bit-wise XOR*. Bit-wise *xor* is the easiest operation to implement since the Hamming code additional bits are based on *xor* computations. In fact a circuit that consists of *d+h xor* gates will serve us to fulfill the Hamming distance preserving requirement. Each of these *xor* gate will compute one bit of *r*. An example for (7,4) Hamming code appears in Figure 1, the example corresponds to the hypotethesis of a single transient error.

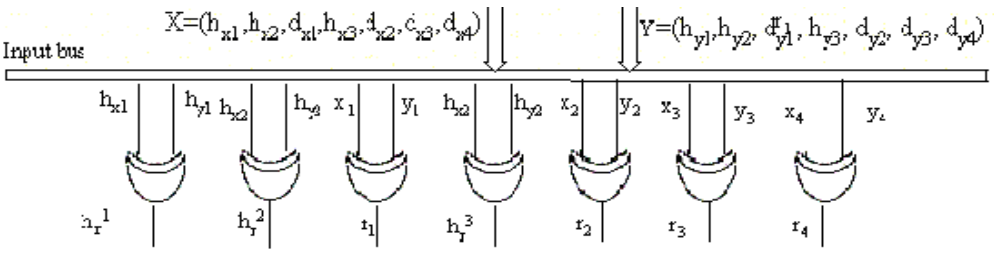

Figure 1 *Bit-Wise* XOR *Hamming Block*

The two inputs for such a *xor* gate will be the corresponding bits in *r*. The independent computation of bits and in particular the Hamming redundant bits will preserve the wrong bits location, which in turn will result in a correctable *r*.

• *bit-wise* NOT. This operation is in fact a bit-wise *xor* with one fixed operand that represents the Hamming code of *y* for which all the d_y bits are ones (Figure 2). Here $h_i(1_m)$ is the parity of the number of bits used to compute $h_{xi.}$ In particular, $h_1(1_m)$ =Parity (2)=0 since h_{x1} is computed by only two data bits (r_1 and r_3).

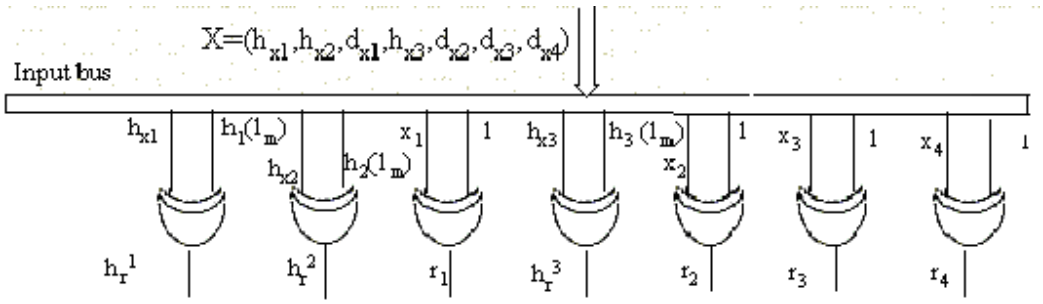

Figure 2. *The* XOR-Based Implementation Of *Bit-Wise Not Function*

• *bit-wise AND*. The circuit consists of an *AND* gate for each data bit of *r*, such that the corresponding inputs of the output bit are the data bits of the same index in *x* and *y*. However, we need a more sophisticated circuit for computing each of the h_r bits, with relation to the implementation of the bit-wise XOR. For instance, to compute the i^{th} bit of h_r, h^i_r we introduce (1) a correcting circuit for *x*, to try to obtain Correct(*x*), (2) a correcting circuit for *y*, to try to obtain Correct(*y*), (3) then compute the bitwise *and* on the result of Correct(*x*) and Correct(*y*) and (4) compute the h^i of the result to be h^i_r (Figure 3).

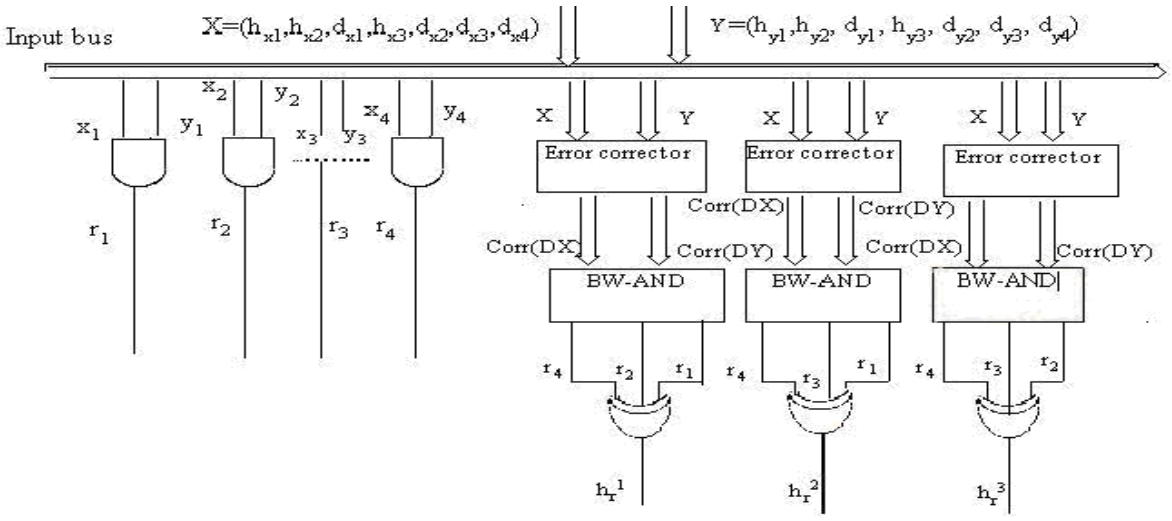

Figure 3. *Bit-Wise AND Hamming Block*

In this way, when the Hamming distance is three, if there is no error in the circuit which computes h^i_r then it is pointing to the incorrect bit at the output, otherwise, it is the only corrupted output and therefore correctable.

- *Arithmetic Binary Summation.* Here we use a (not necessarily full) copy of an adder for each data bit, over the corrected operands, thus an error in one gate propagates only to one output. In addition we use a circuit which corrects operands, computes the result over the corrected operands and uses the computed result to compute a single Hamming bit of r. In this way, in case there of an error in one gate, only the corresponding Hamming bit is corrupted.

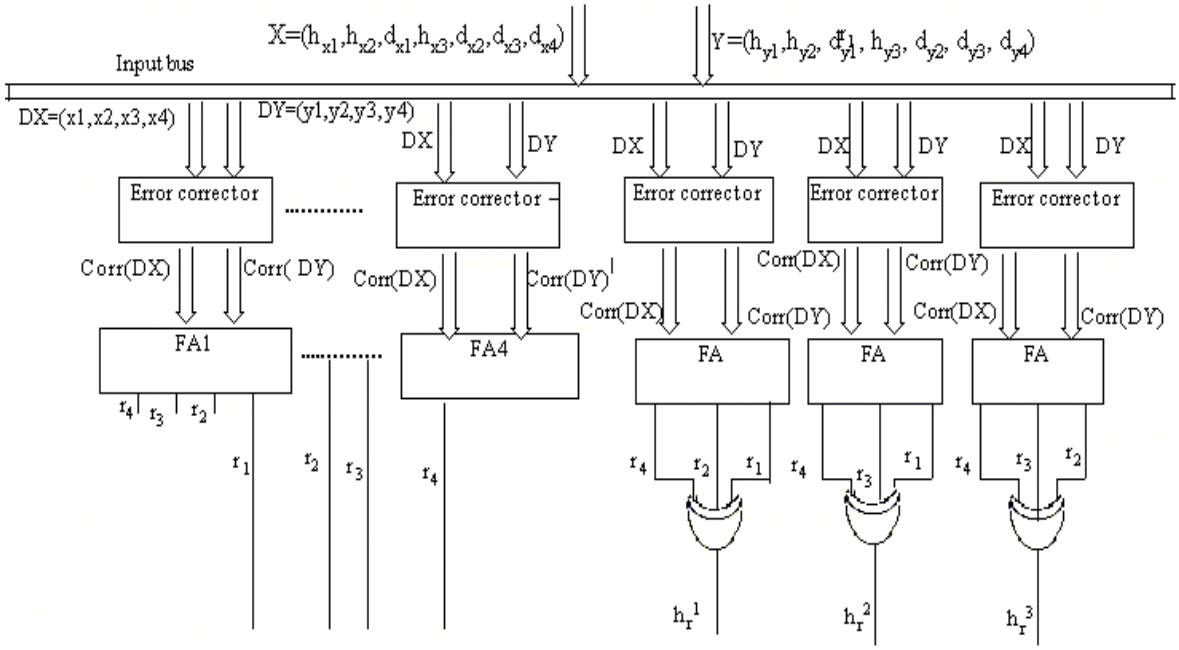

Figure 4. *The Architecture For Hamming Coding Full Adder*

Other logical operations, such as *bit-wise OR* and *shift* have analogous architecture to one of the *bit-wise AND*, as both allow the separation of the data computations and the Hamming bits computation.

In case one wants to correct the result when all gates operate correctly we may either integrate an independent correcting circuit before using the operands, or we may implement a {*correct-nop*} operation which corrects an operand upon a request. The implementation of this operation may be realized by using the add operation in figure 4, where we add all zero data, essentially computing the Hamming bits from the obtained corrected data bits.

Below we demonstrate an example of the protection proof for a bitwise-AND in the presence of any single bit error using our approach to Hamming coding (Figure 3).

We consider the following two cases: (1) an error exists in the data, or (2) an error occurs during the computation. In the first case the data is corrected by the error correcting circuit, since at most one bit of the data may be corrupted. Therefore, the code is correctable. In the second case the inputs are correct and since each output is computed by an independent

circuit at most one such circuit may be corrupted. In the same manner we may prove the protection properties for all other bit-wise operation implementations.

4. Opcode Selection Protection

In order to provide a correct choice of the logic and arithmetic operations (their opcodes) in the presence of transient faults (that leads to a single bit error in an opcode) we need to consider the results of all these operations at each step of a program execution. In the sequel, to provide clearer readability, we use the form $\{d_1,d_2,d_3,d_4,h_1,h_2,h_3\}$ of the Hamming code instead of the ordering $\{h_1,h_2,d_1,h_3,d_2,d_3,d_4\}$ used in Figures 1- 4.

Figure 5 demonstrates our design for the case of four opcodes.

We assume the existence of the following opcodes:

BW_XOR is 11,
BW_AND is 10
BW_Add is 01,
BW_OR is 00

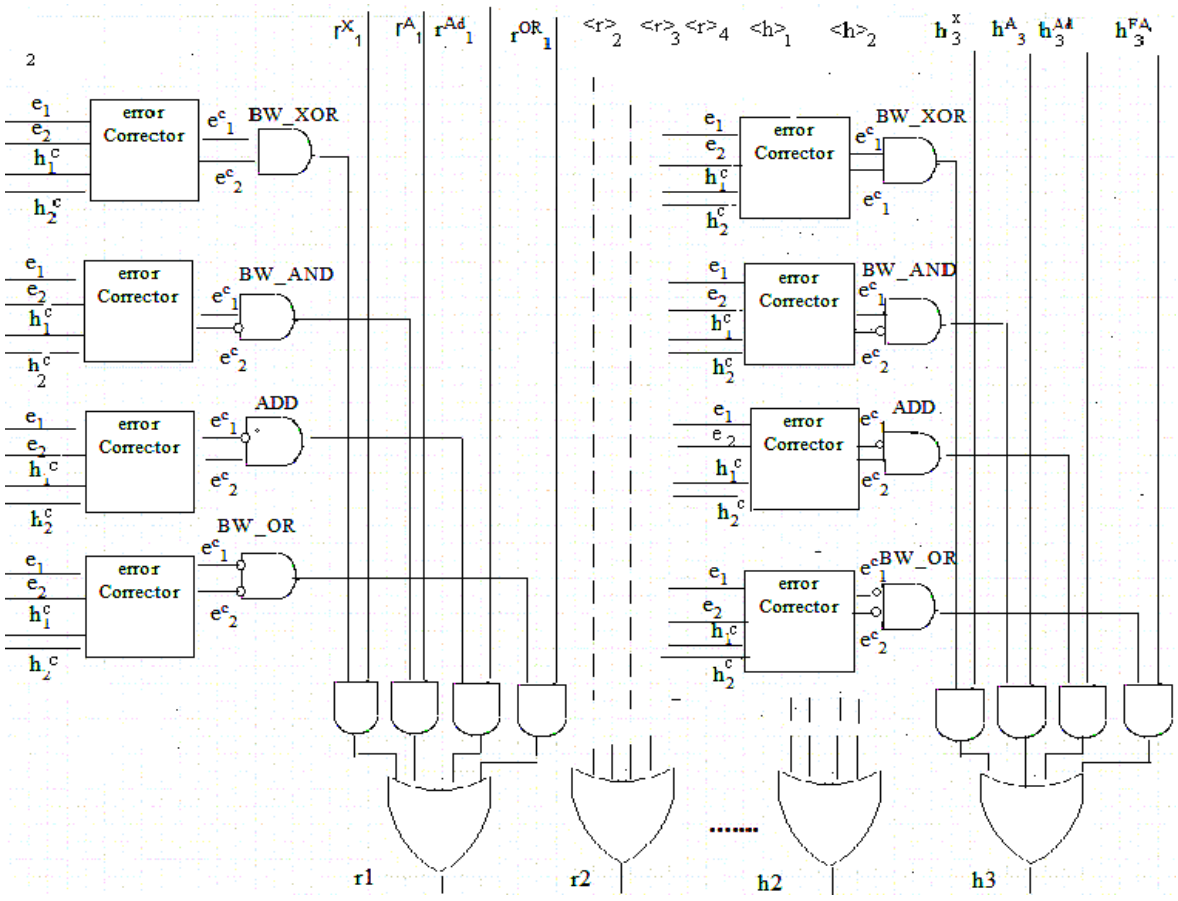

Figure 5. *The Opcode Selection Protection Circuit*

For such 2 bits opcodes we need 2 additional bits of Hamming code, h^c_1, h^c_2.
We also denote in this figure:

r^X_i, r^A_i, r^{Add}_i, r^{OR}_i, h^X_j h^A_j, h^{Add}_j, h^{OR}_j i=1,4 (number of data bits), j=1,3 (number of Hamming bits) are the outputs of the corresponding operational blocks, thereby $<r>_2$, corresponds to $< r^X_2, r^A_2, r^{Add}_2, r^{OR}_2 >$, as well as $<r>_3 = <r^X_3, r^A_3, r^{Add}_3, r^{OR}_3>$, $<r>_4 = <r^X_4, r^A_4, r^{Add}_4, r^{OR}_4>$, $<h>_2 = < h^X_2, h^A_2, h^{Add}_2, h^{OR}_2 >$, and the doted vertical lines corresponds to all these bits, similarly the lines $r^X_1, r^A_1, r^{Add}_1, r^{OR}_1, h^X_3, h^A_3, h^{Add}_3, h^{OR}_3$.

The following errors of opcode selection are possible:

- Error in one of the two input lines e_1, e_2 of the opcode selector. This error can be corrected by standard "error corrector" block, using the parity bits h^o_1, h^o_2 ("Hamming bits"),
- An error in a line of the error corrector. To correct this error we need to use the replication of the outputs of all four possible operations (which are the (n,k) Hamming codes). In fact, we perform all possible operations, and then select one of them according to the opcode.

Indeed, as it is easy to see, in the absence of any error, the output of the gate, corresponding to the operation selected, say BW_XOR, will be one, while others are zero. Otherwise, if one of any bits at the output of any error corrector is corrupted, all four outputs (BW_XOR, BW_AND, ADD, BW_OR) will be zero. For example, if the XOR operation has to be selected, but because of fault in one of inputs of BW_XOR gate, the input signal will be, say, (1,0) instead of (1,1), the output of the gate will be 0, while outputs of other gates are also zero. Therefore, the result of this error will be either masked, or because of independency of all the seven channels (and the assumption of at most a single error) will occur only at one of the output bits, resulting in a correctable output. In case of a corruption of one bit of the opcode error correctors in Figure 5, the circuit will correct the error and choose the right operation. In case an error occurs in one of the gates then by the independence of the circuit connected to each output gate the output will be corrected.

5. Multi-bits Error Correction, BCH Codes, Reed-Solomon Code

Unfortunately, for large circuits, the probability of multiple-bit errors can exceed the number of single-bit errors. Consequently, we need to enhance the code's error correcting capability by adding multiple-bit error correction. Since the idea is based on separation of the computations for data bits and Hamming bits, we may use some other linear systematic codes, for example, BCH (Bose–Chaudhuri-Hocquenghem) codes [13]. BCH are multiple-error correcting codes, which are considered in the literature as a generalization of Hamming codes. Moreover, it can be proven that a single error-correcting BCH code is isomorphic to a Hamming Code. In general, the length of BCH is $n=2^m-1$, the number of check bits is $n-k \leq mt$, $d \geq 2t+1$, where k is the number of data bits, m>0 is an integer, which characterizes the finite field $GF(2^m)$, over which we define the BCH code, t is the possible number of errors.

From the error correction complexity point of view, in order to detect and correct multiple errors, we need to extend the length of the syndrome vector of Hamming code in terms of BCH codes. The BCH codes handle randomly located errors in a data stream.

As for possible multiple error class, the only requirement is that errors are correctable on the operational block output, which means the number of error should be less than [D−1]/2, where D is the Hamming distance of the code. Therefore, we can use architecture analogous to the one used for the Hamming code. The circuit for each of the operations considered includes the BCH error correctors, corresponding to the coding used, while some parity bits may be computed by a corresponding linear circuit, of course, the circuit is more complex than a single XOR which is used for the Hamming codes.

Figure 6 demonstrates the circuit providing (n − k)/2 symbols error correctable BW_AND computation, where 2t = n−k parity bits allows to correct up to *t* symbols that contain errors in a codeword of length n with k data bit. The vector of data symbols result $R=(r_1, r_2,\ldots r_d)$ is the output of data bits of the bit-wise operations,

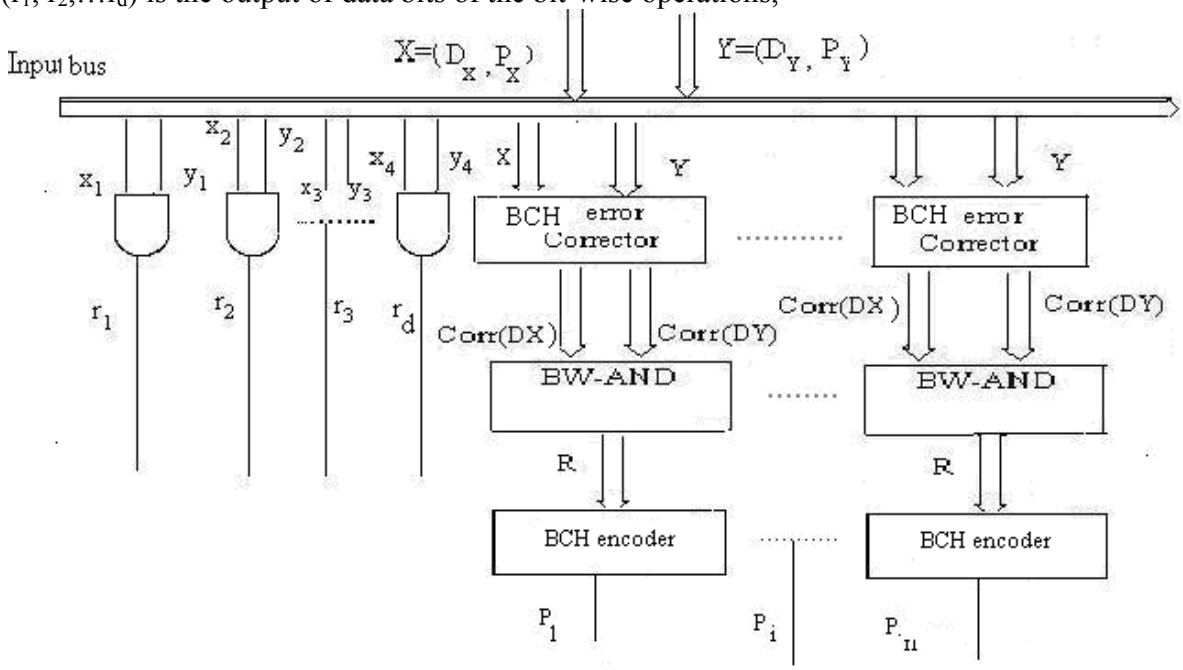

Figure 6. *Bit-Wise And Operation for The BCH Codewords*

The BCH encoding of a binary sequence c(x) is the computation $c(x)=x^{n-k}\ g(x)\ (\text{mod}\ p(x))$ where g(x) is a divider of the generating polynomial p(x). Therefore, the encoding circuit has to shift data bits (represented by polynomial g(x)) and add the remainder or the shifted polynomial divided by p(x). This gives us a systematic encoding.

The most popular subset of BCH codes are the Reed Solomon codes. However, the code deals with symbols of length s rather than a single bit [13]. Reed-Solomon (RS) codes are used for a symbol length equal to or large than 3, thus we deal with codewords of at least 8 symbols. Formally, the parameter of s "number of bits in a symbol" defines the RS code polynomial. So, it is impossible to consider the RS coderwords with 1-bit symbols. RS-codes are optimized to correct bursts of errors.

6. Discussion, Future Work and Conclusions

This is the first step into incorporating error correcting schemes in logic/arithmetic processing. Our approach to operational blocks SEU protection has some essential

differences from traditional Hamming coding approaches. This is an execution of all logic/arithmetic operations in a corresponding code form (Hamming, BSH, Reed-Solomon), while the traditional approaches deal, in fact, with a protection of the result of the operations (their storing and transmission, as well) via Hamming coding.

We believe that this approach opens new paths for investigation and design of robust implementations. Although some replication is used, the solution is more efficient than straightforward replication of processors. Comparing with TMR, the TMR approach is simple, but seems to be too expensive with regards to classical error correcting codes (e.g. Hamming codes, Reed-Solomon codes) that do not duplicate bits but use the Hamming distance between codewords to correct errors. Besides, TMR does not mask soft-errors since the soft-errors may change the result of the last gate of the TMR circuit. In other words, in contrast to the Hamming processor approach, TMR implies a probability for incorrect output even when a single corruption takes place.

Note, that Reed Solomon codes are typically used in memory systems with long memory words. The long words are broken into symbols and the code allows for all of the bits in a symbol to be corrected in the presence of an error. This way, a multi-bit burst error can be corrected. In contrast, we are considering the possibility to use the codes in operational blocks.

The implementation complexity of these circuits is a very important issue. As for Hamming coding for one-bits error, the Hamming microoperations complexity implies a logarithmic growth in the circuit size with the number of code bits if we deal with perfect codes, that is codes which satisfy the relation $2^{n-k} - 1 \geq n$ with equality. In general, for linear codes, the Hamming distance for correction t bits of error respects the inequality:

$$t \geq \log_2 V(n,t),$$

$$V(n,t) = \Sigma_{i=1,t} C_n^i$$

For example, if we consider a two bits error, in some n-bits vector, n single-bit and $n(n-1)/2$ double-bit errors can occur. The Hamming distance inequality in this case is:

$$2^{n-k} - 1 \geq n + n\,(n-1)/2, \text{ or}$$
$$2^{n-k} \geq (n^2+n+2)/2$$

The first two solutions which obey the equality are (5,1) and (90,78) codes. However, except for the single-bit error correcting Hamming code [14] no perfect code exists.

BCH codes, and the Reed-Solomon in particular, are special and widely implemented because they are "almost perfect", and the redundant bits added on by the encoder are the minimum for any level of error correction, so that no bits are wasted [15,16]. So, we may consider this growth also as an “almost logarithmic” one. As for hardware overhead, the encoding requires bitwise XOR of two symbols, and multiplication of two elements of two polynomials, representing the operands. These operations can be implemented as an AND-XOR network, as explained in [17,18,19].

Note, however, that since we have proved that our approach provides correction for a predefined number of operational block errors with given multi-bits errors (say, three, using BCH coding), an achievement that other approaches fail to achieve, we have a qualified

rather than only quantified advantage. Still, if we consider the entire design of a process and would like to qualify overhead, we need to compute and predict a trade-off between reliability and size-performance-power overhead. It may require investigates the following issues:

Taking into account that all latches influenced by possible errors in the combinational parts (operational block) which affect the result of the operations,

- Analyze the reliability relatively to errors corrupting more bits than the correcting system (say, Hamming system) can correct;
- Estimation of the fraction of latches (gates) used in correction of the circuit, affecting an output corruption;
- Taking into account the sensitivity area of a circuit (for example, in accordance to [21] memory occupies 88%, datapath (including registers occupies 9.5%) ;
- Taking into account power consumption issue, influence of the protection solution on the factor should be investigated in comparison with other approaches. For example, although the size of each Hamming code/decode block grows logarithmically in relation to the increase of the protected data bits, and the number of voters in TMR increases linearly with the number of bits in the protected word, TMR may also imply expensive cost in other important system characteristics, e.g. cost of peripherals, such as power supply and connections in the mother board for replicas.

A resolution of these problems may be helpful for a choice of various modifications of traditional SEU protection techniques also. For comparison, in TMR systems with bit-wise voters, *n* output systems use *n* single-bit voters. But if we consider the word-voter systems, the complexity is even higher [20], that is, if a TMR system has n outputs, the number of equivalent 2-input gates required by the word-voter is 3n. Therefore, such ambiguity with the size overhead of various approaches to the high-reliability implementation of the arithmetic blocks calls for the investigation of energy–reliability and performance-reliability tradeoff. This is left for future works.

We should also take into account that the power consumption issues depend on a signal switching activity (frequency) as well, which in turn, depends on the protection scheme implementation [12]. As described in [21]**,** extra parity bits of an error correction/detection code increase the signal transitions on the bus in the general case.

Bibliography

[1] J.Von Neumann, “Probabilistic logics and synthesis of reliable organisms from unreliable components", *Automata Studies,* pp. 43-98, 1956.

[2] S. Dolev and Y. A. Haviv, “Self-stabilizing microprocessor: analyzing and overcoming soft errors”, *IEEE Transactions on Computers*,Vol. 55(4), pp. 385-399, 2006.

[3] V.P. Nelson, “Fault-tolerant computing: fundamental concepts”, *IEEE Computer*, Vol. 23, (7), pp. 19–25, 1990.

[4] P. K. Lala. *Self-checking and fault-tolerant digital design,* Morgan Kaufmann, 2001.

[5] F.G. De Lima**, “**Single event upset (SEU) mitigation techniques, in Fault-Tolerance Techniques for SRAM-based FPGAs”*, DOI 10.1007/978-0-387-31069-5*, *SpringerLink: http://www.springerlink.com/content/tv5m73xp221n462q/fulltext.pdf.*

[6] N. F. Benschop, "Associative Digital Network Theory An Associative Algebra Approach to Logic, Arithmetic and State Machines", *Amspade Research*, The Netherlands, 2003.

[7] R. Jasinski, V.A. Pedroni, "Evaluating Logic Resources Utilization in an FPGA-Based TMR CPU", *MAPLD 2004.*
[8] K.J. Hass, J.W. Gambles, B. Walker, M. Zampaglione, “Mitigating single event upsets from combinational logic”,*7th NASA, Symposium on VLSI Design*, 1998.
[9] R. Hentschke, F. Marques, F. Lima, L. Carro, A. Susin, R. Reis, ″Analyzing Area and Performance Penalty of Protecting Different Digital Modules with Hamming Code and Triple Modular Redundancy", *Proceedings of the 15-th Symposium on Integrated Circuits and Systems Design (SBCCI’02).*
[10] F. Lima, S. Rezgui, E. Cota, L. Carro1, M. Lubaszewski, R. Velazco, R. Reis, "Designing a Radiation Hardened 8051-like Micro-controller", *Proceedings, XII Symposium on Integrated Circuits and Systems Design SBCCI2001,* Manaus, 2000. .
[11] D.G. Mavis, P.H. Eaton, “Soft error rate mitigation techniques for modern microcircuits”, *40th Annual Reliability Physics Symposium Proceedings*, 7-11 April 2002 Page(s):216 – 225.
[12] Wei Chen, Rui Gong, Fang Liu, Kui Dai, Zhiying Wang, "Improving the Fault Tolerance of a Computer System with Space-Time Triple Modular Redundancy", *Proceedings of International conference of Embedded Systems Applications, ESA06*, pp.183-190,2006.
[13] R.E. Blahut, *Theory and practice of error control codes*, Addison-Wesley Publishing Company, 1983.
[14] D. Johnon, “Error-correcting codes, Hamming Distance”, *http://cnx.org/content/m0097/latest/*, 2005.
[15] X. Chen, I. **S.** Reed, and T. K. Truong, “No binary quadratic residue code of length 8m-1 is quasi-perfect**",** *IEEE Transactions on Information theory* “*,* Vol. 40, 2, p.503, 1994.
[16] D. Bell and R. Laxton, “Some BCH codes are optimum“, *Electronics letters,*Vol.11, p.14, July 1975.
[17] H. Wu, “Bit-parallel finite field multiplier and squarer using polynomial basis”, *IEEE Transactions on Computers*, Vol. 51 , no. 7 , pp. 750 - 758, July 2002.
[18] A. Reyhani-Masoleh, M.A. Hasan, “Low complexity bit parallel architectures for polynomial basis multiplication over GF(2^m)”, *IEEE Transactions on Computers,* Vol. 53 , no 8, pp. 945 - 959, Aug. 2004.
[19] G.C. Cardarilli, S. Pontarelli, A. Salsano, “Design of a self -checking Reed Solomon encoder”, *Proceedings of the 11th IEEE International On-Line Testing Symposium (IOLTS’05),* 2005.
[20] S. Mitra and E. McCluskey, “Word-voter: A new voter design for triple modular redundant systems”, *Proceedings, IEEE VLSI Test Symp.,* May 2000, pp. 465–470.
[21] S.Komatsu *and* M. Fujta, ″Algorithms Low Power and Fault Tolerant Encoding Methods for On-Chip Data Transfer in Practical Applications"**,** *IEICE TRANS. FUNDAMENTALS, PAPER Special Section on VLSI Design and CAD,* vol.E88–A, no.12, 2005.