International Journal of Innovative
Computing, Information and Control
Volume **8**, Number **7(B)**, July **2012**

ICIC International ©2012 ISSN 1349-4198
pp. **5083–5094**

A FRAMEWORK FOR FAULT TOLERANCE TECHNIQUES IN THE ANALYSIS AND EVALUATION OF COMPUTING SYSTEMS

Hodjat Hamidi, Abbas Vafaei and Amirhassan Monadjemi

Department of Computer Engineering
University of Isfahan
Isfahan 81746-73441, Iran
hamidi@eng.ui.ac.ir; { vafaei; monadjemi }@ui.ac.ir

Received May 2011; revised October 2011

Abstract. *The Algorithm Based Fault Tolerance (ABFT) approach transforms a system that does not tolerate a specific type of faults, called the fault-intolerant system, to a system that provides a specific level of fault tolerance, namely safety and/or recovery. ABFT techniques are most effective when employing a systematic form. The error detection is employed based on a high-rate real convolution code. This paper addresses new methods for performing error correction when real number codes are involved. The parity values are determined according to a systematic real convolution code. Detection relies on two sets of parity values which are computed in two different ways, one set from the input data but with a simplified combined processing subsystem, and the other set directly from the output processed data, employing the parity definitions directly. The ABFT philosophy leads directly to a model from which error correction can be developed. By employing an ABFT scheme with effective real convolution code, the design allows high throughput as well as high fault coverage. The simulations show that the great difference between the round-off error and the computer-induced error is large enough to be distinguished.*

Keywords: Algorithm based fault tolerance (ABFT), Convolution code, Parity values, Round-off error, Redundancy

1. **Introduction.** In the case of fault tolerance, real convolution codes are primarily used for error detection, providing the vector space separations, and detected abnormal behavior leads to recomputation of the corrupted results. While the theory of real number coding is similar to codes over finite fields, the decoding for error-correcting purposes is more complicated. Algorithm based fault tolerance, proposed by Huang and Abraham [1], is a fault tolerance scheme that uses Concurrent Error Detection (techniques at a functional level). ABFT techniques are most effective when applied in a systematic form. The redundancy necessary for the ABFT method is commonly defined by real number codes, generally of the block type [2-8]. It has been used to reduce redundant hardware. ABFT methodologies used in [9,10] present parity values dictated by a real convolution code for protecting linear processing systems. A class of convolution codes called burst-correcting convolution codes is introduced in [10,11]. These codes provide error detection in a continuous mode using the same computational resources as the algorithm progresses.

The motivational model underlying ABFT as applied to linear processing of blocks of real data is shown in Figure 1. The ABFT error detection technique relies on the comparison of parity values computed in two ways. Number data processing errors are detected by comparing parity values associated with a convolution code. This article proposes a new computing paradigm in order to provide fault tolerance for numerical algorithms. The data processing system is protected through parity values defined by a

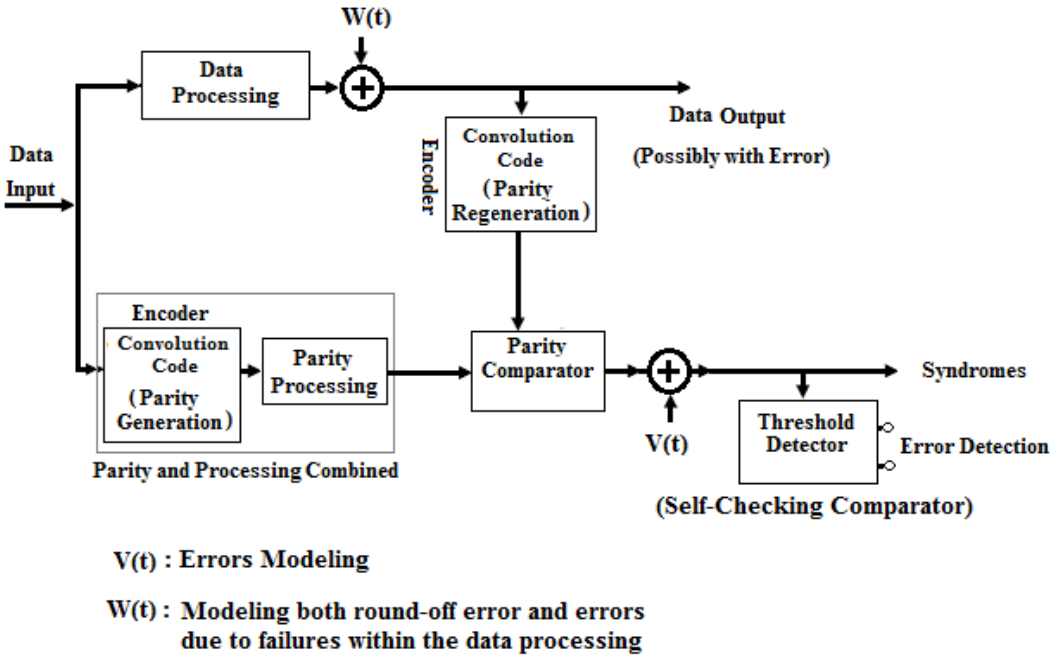


FIGURE 1. ABFT technique

high-rate real convolution code. Parity comparisons provide error detection, while output data correction is affected by a decoding method that includes both round-off error and computer-induced errors. In order to use ABFT techniques efficiently, a systematic form is desirable [11-13]. Performing error correction when infrequent intermittent errors appear in the protected output values is appealing in several settings. If corrupted output values are recomputed after error detection, the necessary control structure becomes very complicated and the overall processing speed throughput is degraded accordingly. In another situation, data and the related real-number parity values are located in storage and when they are required again, the occurrence of data errors is detected. Then correcting a few errors may be much simpler and faster than recomputing the original data, even if the same processes are still active. However, error correction would probably be employed if a viable error-correcting procedure was available.

The goal is to describe new protection techniques that are easily combined with normal data processing methods, leading to more effective fault tolerance. The error detection structures are developed and they not only detected subsystem errors but also corrected errors introduced in the data processing system. Concurrent parity values techniques are very useful in detecting numerical error in the data processing operations, where a single error can propagate to many output errors.

The following contributions are made in this article. In Section 2, the Convolution code, mean-square error (MSE) and majority logic decoding are discussed. In Section 3, the architecture of ABFT is proposed. In Section 4, the protected detector and corrector system is discussed. In Section 5, the simulations and experiment results are presented and finally in Section 6, conclusions are presented.

2. **Convolution Codes and Mean-Square Error (MSE).**

Convolution codes represent an efficient method for adding redundancy to the data symbols in a data processing system to combat additive error effects at the processed data and only systematic forms of convolution codes will be considered. Convolution codes differ from block codes in that the encoder contains memory and the encoder outputs at any given time unit depend

not only on the inputs at that time unit but also on some number of previous inputs [14-20]. With convolution codes, the incoming bit stream is applied to a K-bit long shift register. For each shift of the shift register, k new bits are inserted and n code bits are delivered, so the code rate is k/n. A rate $R = k/n$ convolution encoder with memory order m can be realized as a k-input, n-output linear sequential circuit with input memory m; that is, inputs remain in the encoder for an additional m time units after entering. The input digits are grouped together by segments of k digits, called symbols, because of the way the encoder introduces redundancy at the transmitting end. The encoding of a symbol produces a corresponding group of n output digits, each also an element of $GF(q)$. Symbolically the encoding process may be written as

$$u = \{u_0, u_1, u_2, \ldots, u_{i-1}, u_i, u_{i+1}, \ldots\}, \text{ input sequence} \tag{1}$$

$$u_i = (u_{i0}, u_{i1}, \ldots, u_{i(k-1)}), \text{ input symbol } i \tag{2}$$

where the k input digits $u_{ij} \in GF(q)$, and encoded sequence, codeword,

$$v = \{v_0, v_1, v_2, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots\} \tag{3}$$

$$v_i = (v_{i0}, v_{i1}, \ldots, v_{i(n-1)}), \text{ encoded symbol } i \tag{4}$$

where the n output digits $v_{ij} \in GF(q)$.

A new error criterion will be introduced by associating an integer with each input symbol u_i. In making this assignment, the k components in $u_i = (u_{i0}, u_{i1}, \ldots, u_{i(k-1)})$ are weighted so that the $(k-1)$th element, $u_{i(k-1)}$, has the largest numerical impact, while the zeroth one, u_{i0}, has the least. This fixes the numerical ordering for the k input positions, effectively indexing each input symbol u_i with an integer from zero through $(q^k - 1)$. A similar weighting is attached to the respective components in $\widehat{u}_i$, the decoder's output. A new performance criterion, the overall mean-square error, is employed under these conditions.

$$\Xi^2 = E\left\{\left[u_i - \widehat{u}_i\right]^2\right\} \tag{5}$$

where u_i and $\widehat{u}_i$ are viewed as real numbers, $u_i \leftrightarrow k$ input digits, and $\widehat{u}_i \leftrightarrow n$ decoded digits. The positions in the input and output symbols are treated differently by this criterion. The minimization process guarantees that the $(k-1)$th position will enjoy more error protection than the other positions because of the quadratic nature of the square error criterion. The goal is to minimize the overall mean-square individual symbols, Ξ^2, by simultaneously choosing the encoding and decoding functions.

Thus

$$\Xi_i^2 = E\left\{\left[u_i - \widehat{u}_i\right]^2\right\}, \quad i = 0, 1, 2, \ldots \tag{6}$$

The subspace of the convolution code that effects the encoding and decoding of a single input integer will be examined first. We restrict our attention to time-invariant convolution codes, as the extension to a time-varying code is a straightforward matter. The system is assumed to be in steady-state operation with $i > k$, and the decoding decision for output integer $\widehat{u}_i$ will be made based upon the outputs from the detectors for nm code digits, beginning with the output due to u_i.

The mean-square error criterion is interesting in its own right because many systems are designed on its basis. Nevertheless, we can relate the outputs of the mean-square error decoder to those of conventional decoders. When both systems use the same optimum encoding rule, the minimum mean-square error decoding rule produces the same outputs as the majority logic decoding (MLD). The mean-square error decoder always attempts to

confine any error during overload situations to the statistically least significant positions in the decoder's outputs.

2.1. **Majority logic decoding.** Viterbi [18] decoding is an optimum decoding method for convolution codes. Its performance depends on the quality of the channel, and the decoding effort is fixed and grows exponentially with code constraint length. Hence, it is only useful for short-constraint-length codes. The performance of sequential decoding is slightly suboptimum, but its decoding to be used long constraint lengths. The number of computations needed to decode a frame of data is a random variable. Although most frames are decoded very quickly, some undergo long searches, causing a few to be incompletely decoded or erased. This variable nature of the decoding effort has led sequential decoding to be referred to as a probabilistic decoding method. An algebraic approach can also be taken to the decoding of convolution codes. Majority-logic or threshold decoding differs from Viterbi decoding and sequential decoding in that the final decision made on a given information block is based on only one constraint length of received blocks rather than on the entire received sequence. The implementation of the decoder is much simpler [20]. This has led to the use of majority-logic decoding in applications such as fault tolerance and reliability, where a moderate amount of coding gain is desired at a relatively low cost. In the general case of an (n, k, m) systematic code with generator matrix

$$G = \begin{bmatrix} IP_0 & 0P_1 & 0P_2 & \cdots & 0P_m & & \\ & IP_0 & 0P_1 & \cdots & 0P_{m-1} & 0P_m & \\ & & IP_0 & \cdots & 0P_{m-2} & 0P_{m-1} & 0P_m \\ & & & \cdot & & \cdot & \\ & & & & \cdot & & \cdot \\ & & & & & \cdot & \end{bmatrix} \tag{7}$$

where I and 0 are identity and all zero $k \times k$ matrixes respectively and P_i with $i = 0$ to m is a $k \times (n-k)$ matrix whose entries are, [18]:

$$P_i = \begin{bmatrix} g_{1,i}^{(k+1)} & g_{1,i}^{(k+2)} & \cdots & g_{1,i}^{(n)} \\ g_{2,i}^{(k+1)} & g_{2,i}^{(k+2)} & \cdots & g_{2,i}^{(n)} \\ \vdots & \vdots & & \vdots \\ g_{k,i}^{(k+1)} & g_{k,i}^{(k+2)} & \cdots & g_{k,i}^{(n)} \end{bmatrix} \tag{8}$$

The parity-check matrix is given by:

$$H = \begin{bmatrix} P_0^T & I & & & & & & & & & & & \\ P_1^T & \mathbf{0} & P_0^T & I & & & & & & & & & \\ P_2^T & \mathbf{0} & P_1^T & \mathbf{0} & P_0^T & I & & & & & & & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & & & & & & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & & \cdot & & & & & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & & & \cdot & & & & \\ P_m^T & \mathbf{0} & P_{m-1}^T & \mathbf{0} & P_{m-2}^T & \mathbf{0} & \cdots & P_0^T & I & & & & \\ & & P_m^T & \mathbf{0} & P_{m-1}^T & \mathbf{0} & \cdots & P_1^T & \mathbf{0} & P_0^T & I & & \\ & & & & P_m^T & \mathbf{0} & \cdots & P_2^T & \mathbf{0} & P_1^T & \mathbf{0} & P_0^T & I \\ & & & & & \cdot & & & & & & & \cdot \\ & & & & & & \cdot & & & & & & \cdot \\ & & & & & & & \cdot & & & & & \cdot \end{bmatrix} \tag{9}$$

where in this case, I is $(n-k) \times (n-k)$ identity matrix and 0 is the $(n-k) \times (n-k)$ all-zero matrix (T stands for the transposed matrix). The syndrome equations, denoted

by vector S, are given by:

$$S = Hv^T = He^T \tag{10}$$

where v is the received sequence and e is error vector. The value of Hv^T is dependent only on the error vector and is independent of the codeword. Syndromes of the code words are 0. There are $(n-k)$ syndrome sequences, one corresponding to each parity error sequence, and $s = \left(s_0^{(k+1)}, \ldots, s_0^{(n)}, s_1^{(k+1)}, \ldots, s_1^{(n)}, s_2^{(k+1)}, \ldots, s_2^{(n)}, \ldots\right)$.

3. **ABFT Methodology.** The ABFT techniques that are applied to detect errors rely on the comparison of parity values computed in two ways. The parallel processing of input parity values produce output parity values comparable with parity values regenerated from the original processed outputs. The model basic ABFT as applied to linear processing of blocks of real data, Figure 1.

The parity values are determined according to a systematic real convolution code. Detection relies on two sets of parity values which are computed in two different ways, one set from the input data but with a simplified combined processing subsystem, and the other set directly from the output processed data, employing the parity definitions directly. These comparable sets will be very close numerically, although not identical because of round-off error differences between the two parity generation processes. The effects of internal failures and round-off error are modeled by additive error sources located at the output of the processing block and input at threshold detector. This model combines the aggregate effects of errors and failures and applies them to the respective outputs. Basic coded system model is presented in Figure 2, clearly indicating the computer-induced error (which may represent hardware failures in certain situations) and the round-off computational error introduced the syndrome calculations. The part of model accounting for computational effects is widely employed in signal processing systems [7-12]. This round-off error is usually considered white and uncorrelated with all other processes associated with the signal. The error corrector must operate on the syndromes as they are the only observable items which are data-independent. The input data are generally restricted to a large but finite alphabet, e.g., fixed-point integer values, so that round-off error in the processing, encoding, and parity generation operations does not cause intrinsic errors, even before other effects are included. The upper bound on the numerical values associated with the alphabet is affected by the dynamic range of the processing system whereas the precision determines round-off error tolerance levels.

4. **Protected Detector and Corrector System.** To achieve fault detection and correction properties of convolution code in data processing with the minimum additional computations, the block diagram is proposed in Figure 3. This figure summarizes an

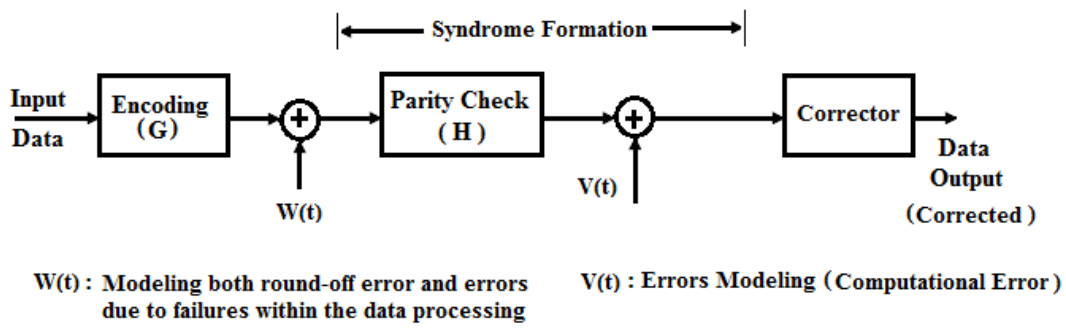


FIGURE 2. Basic coded system model

ABFT technique employing a systematic convolution code to define the parity values. The data processing operations are combined with the parity generating function to provide one set of parity values. The k is the basic block size of the input data, and n is block size of the output data, new data samples are accepted and $(n - k)$ new parity values are produced. Convolution codes are usually used over the transmission channels, through which both information and parity bits are sent. The upper way, Figure 3, is the processed data flow which passes through the process block (data processing block) and then feeds the convolution encoder (parity regeneration) in order to produce parity values. On the other hand, the comparable parity values are generated efficiently and directly from the inputs (parity and processing combined, see Figure 3), without producing the original outputs. The difference in the comparable two parity values, which are computed in different ways, is called the syndrome; the syndrome sequence is a stream of zero or near zero values.

The convolution code's structure is designed to produce distinct syndromes for a large class of errors appearing in the processing outputs. Figure 3 employs convolution code parity in detecting and correcting processing errors. One error correction method shown at the output of Figure 3, an error protection subsystem is included the decoding operation, as shown in Figure 3, guarding against overload situations that exceed the correcting capabilities of the code. Such final output checks are very common in fault-tolerant systems. The corrected outputs are tested by recomputing the syndromes and employing a threshold detector which is a checker containing small thresholds for round-off error tolerance. The basic dichotomy for a corrector of real number convolution codes is shown in Figure 3. However, errors may occur in the corrector itself and so additional error detection capabilities are included after the corrector by reforming the syndrome of the final output corrected. If there have been failures in the original syndrome formation or error-correction parts, the correction compute $v(t)$, see Figure 2, will be erroneous, introducing detectable errors in output corrected. Furthermore, when the corrector attempts correction of large computer-induced errors in excess of the code's capabilities, the stochastic estimate introduces additional large errors in the final compute. The syndrome reformation process detects this situation too. The fault tolerance detector parts of Figure 3 perform the final checking function. Such final output checks are very common in fault-tolerant systems. The corrected outputs are tested by recomputing the syndromes

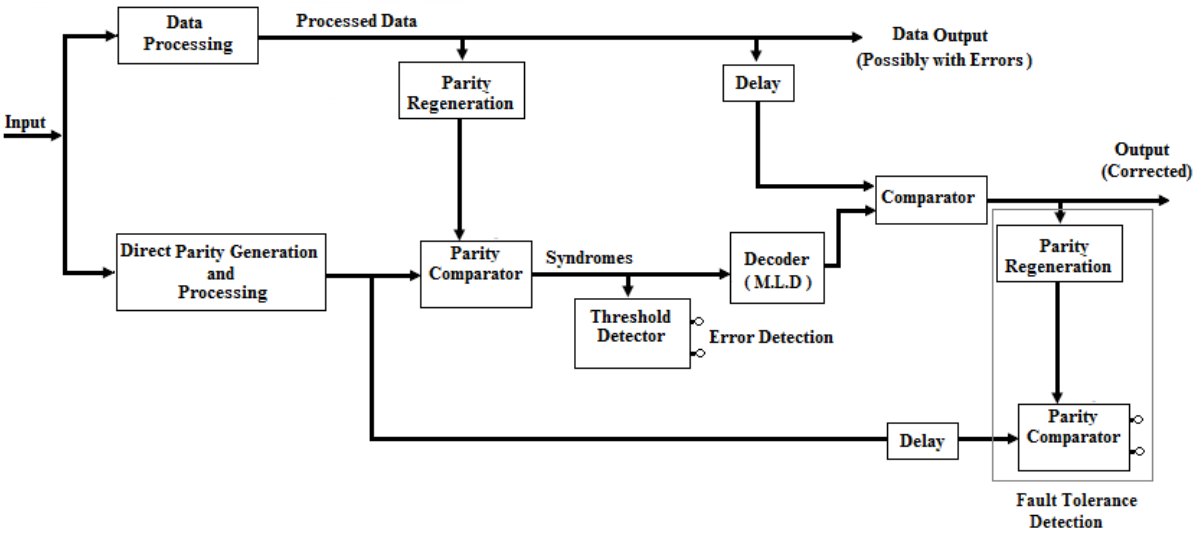


FIGURE 3. Protected detector and corrector system

and employing a threshold detector which is a totally self-checking checker containing small thresholds for round-off error tolerance [21,22] .

5. **Simulations and Experiment Results.** The basic operations needed for testing and simulating the new techniques explored in this approach are well-suited to MATLAB. Several simulation schemes modeling the ABFT method for detecting numerical level errors are described in MATLAB, version 2010a, where the modeling errors were assumed Gaussian with zero means and statistically independent from symbol to symbol. Errors are allowed in the parity values computed by the combined data parity generator, Figure 3, and in the processed data symbols. The MATLAB code forms the basis for a simulation program that explores the role of the threshold τ. If the threshold τ is set too low, even occasional round-off errors will exceed it, indicating failures leading to recomputation of parity values unnecessarily. It is generally permissible to accept a few small errors that are in the range of round-off levels. Nevertheless, the simulations examine how the threshold choice impacts undetected errors. The simulation program randomly selects the random magnitude error. The magnitude of each error is chosen from a Gaussian population with zero mean and fixed variance 2×10^{-4}. For small thresholds, large errors always lead to detection, whereas large thresholds increase the undetected error performance. The threshold is varied over a wide range so as to see the transition between low detected errors and high levels of missed errors. The input data size is 50 samples and each sample point is averaged over 500,000 random choices.

Figure 4 shows the simulation results after 500,000 iterations. In Figure 4, the syndrome sequence in the absence of any errors is shown and nonzero values are due to round-off errors. The syndrome in most cases is less than $\pm 3 \times 10^{-3}$. However, there are also a few errors as great as $\pm 3.5 \times 10^{-3}$. Therefore, we will select a threshold value equal to 17.5×10^{-3} for error detection purposes. The linear data processing system is then subjected to errors due to failures every 50 steps at data processing block outputs ($W(t)$ in Figure 1) and at comparator output ($V(t)$ in Figure 1). It is clear that the syndrome contains single nonzero values for injected errors. If there is only a single error in every constraint length the majority logic decoding can correct the error. The results

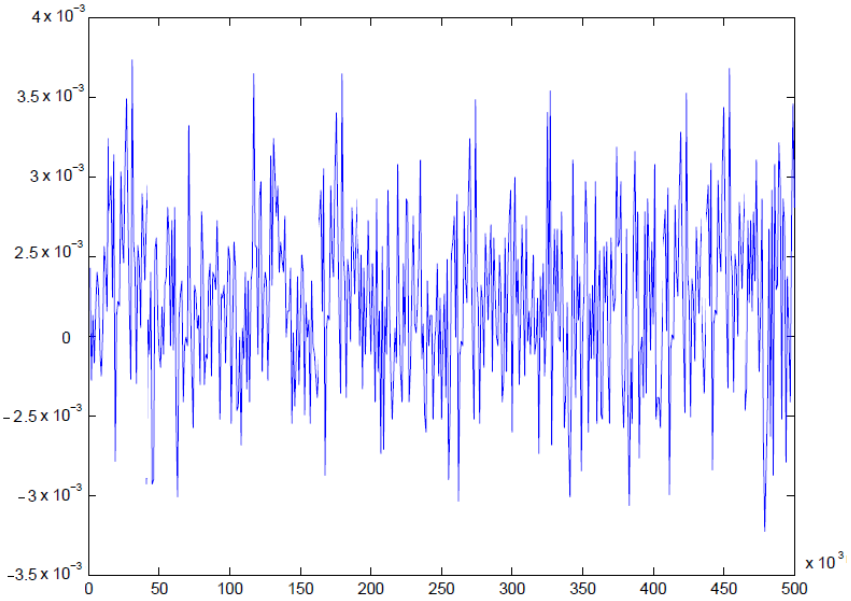
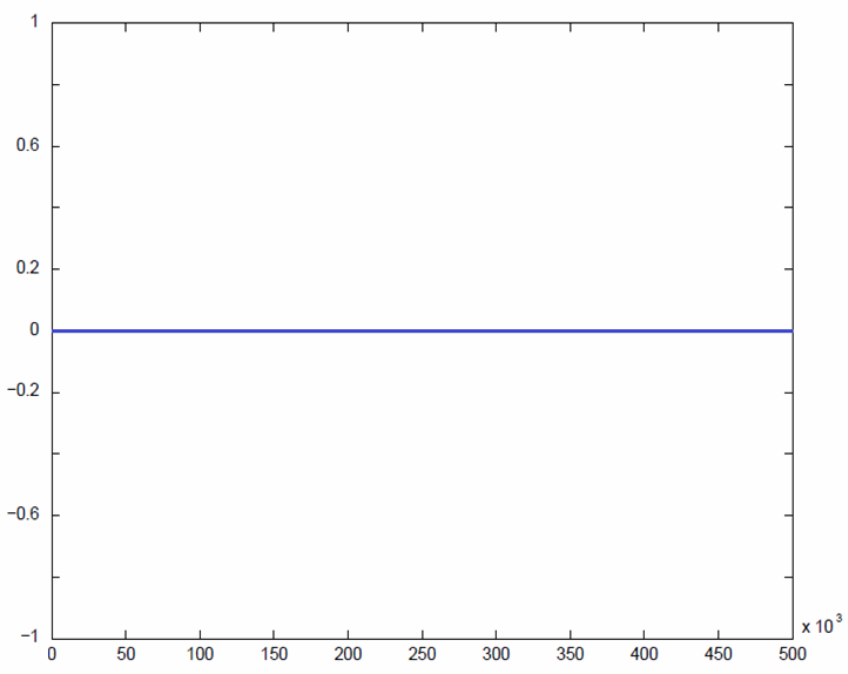
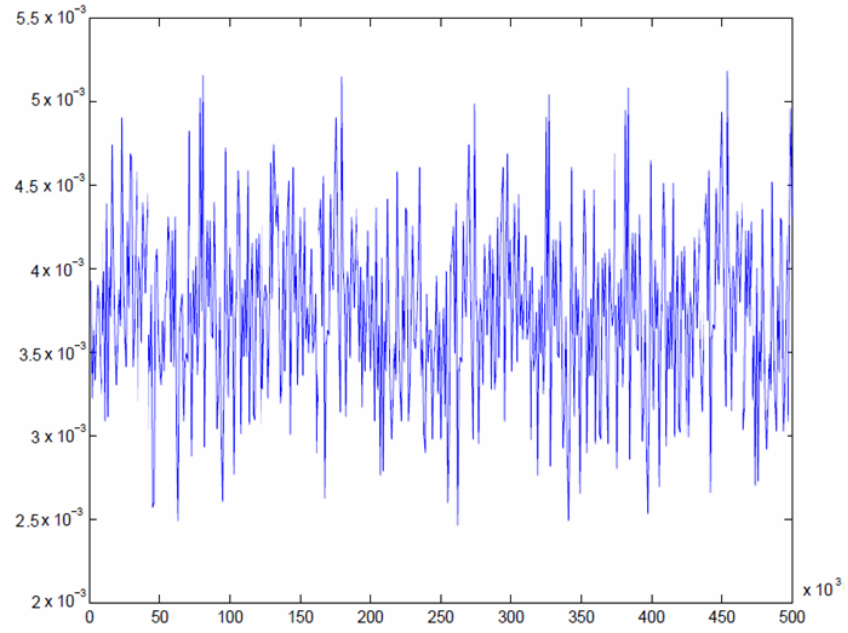


FIGURE 4. Syndrome sequence for no error condition (error-free condition)



(a)



(b)

FIGURE 5. (a) Plot probability that number of errors greater than 1 and (b) plot probability that number of errors greater than 0 from 500,000 iterations

are compared to determine the probability of errors. The experiment shows that there is positive probability of computer-induced errors greater than 0, and the probability of computer-induced errors greater than 1 is zero as shown in Figure 5. Beside the probability of error range, the MSE and the maximum MSE form 500,000 iterations are evaluated. The mean-square error performance for the fault-tolerant processing system protected by the $(6, 5, 2)$ code is displayed in Figure 6.

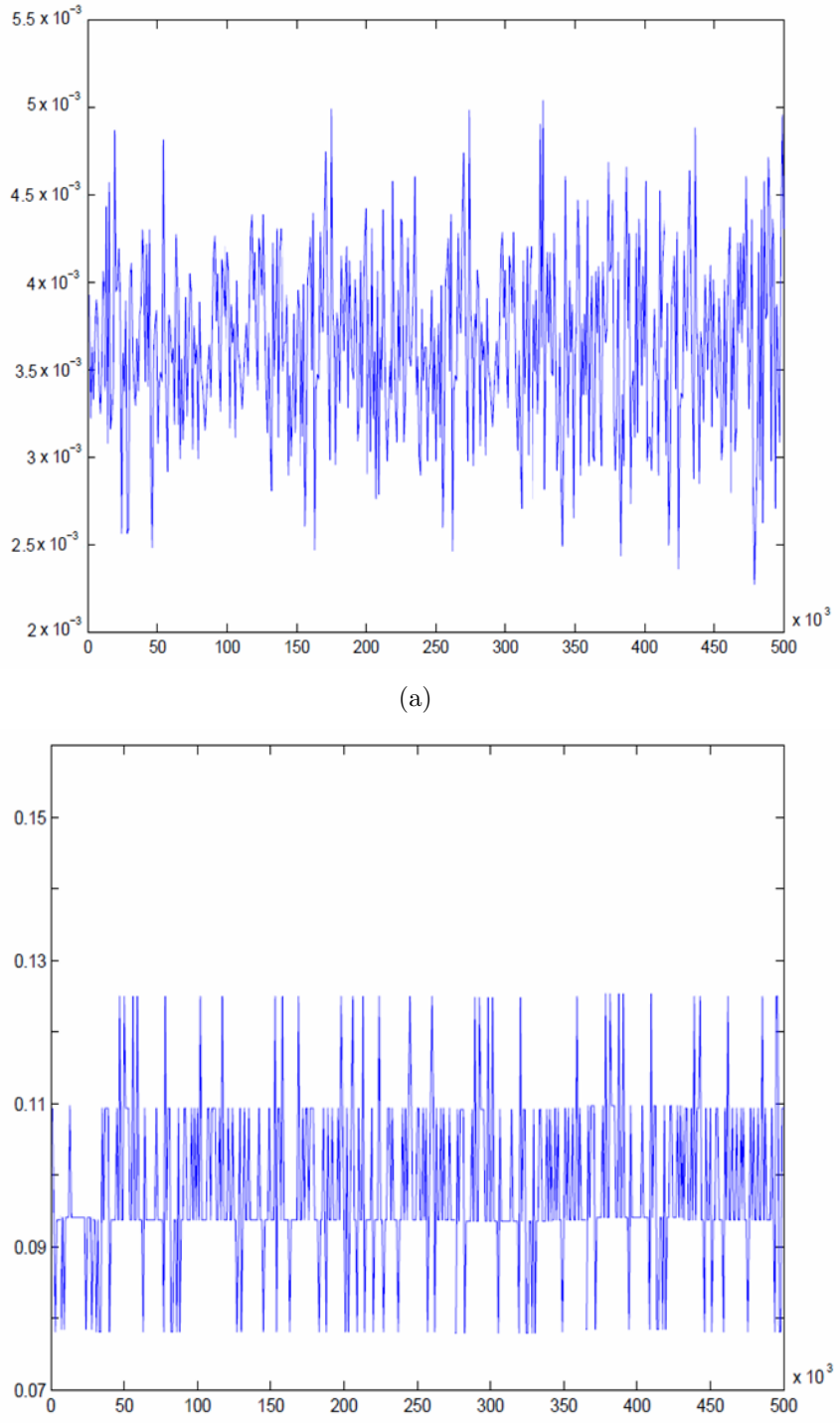
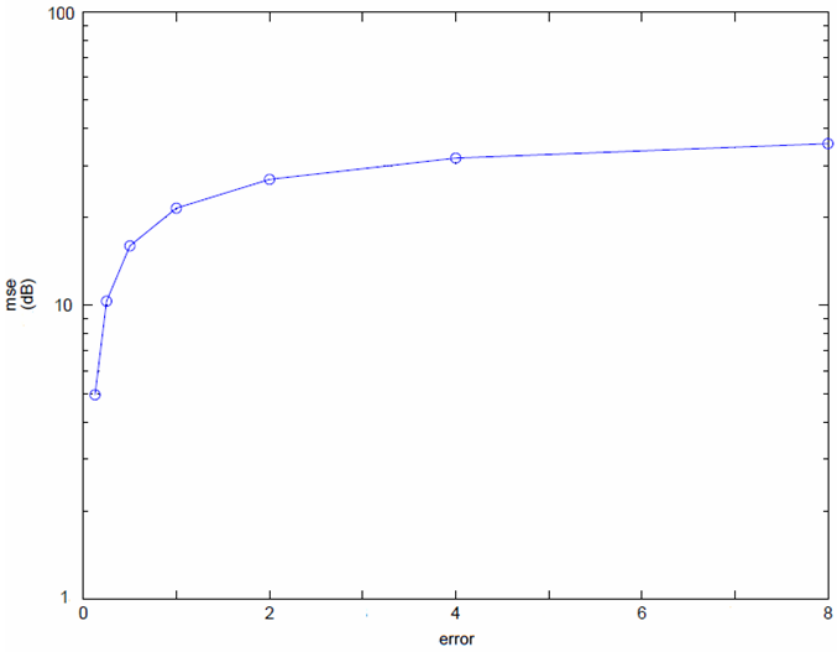
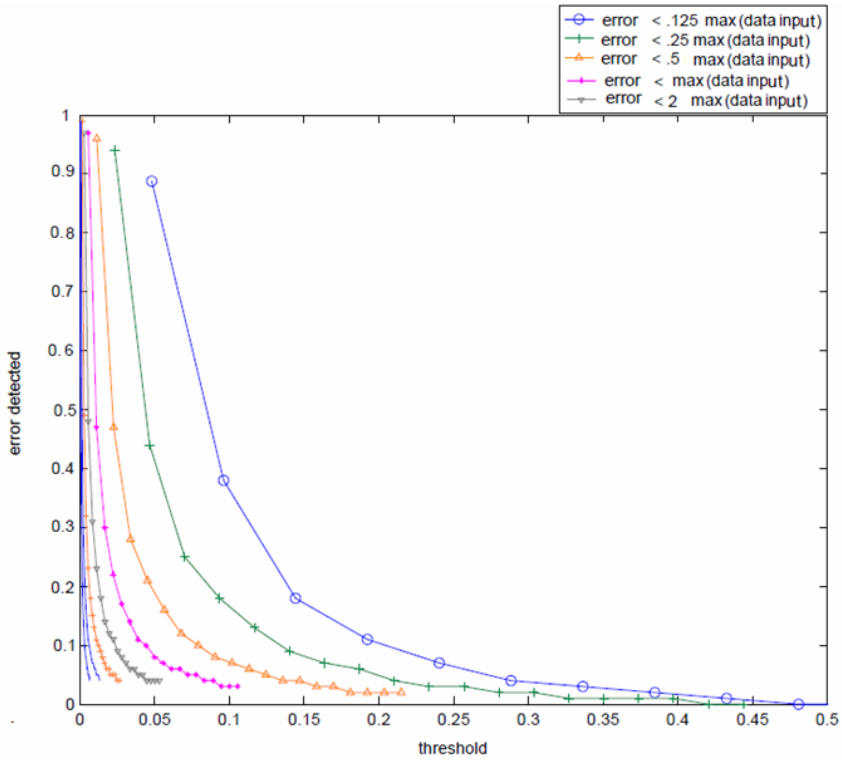


FIGURE 6. (a) The MSE and (b) maximum MSE form 500,000 iterations

The mean-square errors vs. injection error boundary and error detection performance vs. threshold, when error occurred in the linear data processing system, are shown in Figures 7(a) and 7(b). When the maximum error level is large, the error detection performance drops quickly versus the detection threshold as shown in Figure 8(a). Figure 8(h) shows the error detection performance curve corresponding to the error level below 0.125 the maximum of data input. The mean square error in this case is 0.9402 (or -0.2678dB).



(a)



(b)

FIGURE 7. (a) Mean square error vs. error injection level and (b) detection performance vs. threshold

The threshold can be selected based on the error condition and the detection performance curve. Error detection performance of the data processing system when the error injection level does not exceed, 16, 8, 4, 1, 0.5, 0.25 and 0.125 times the maximum data input value are shown in Figure 8.

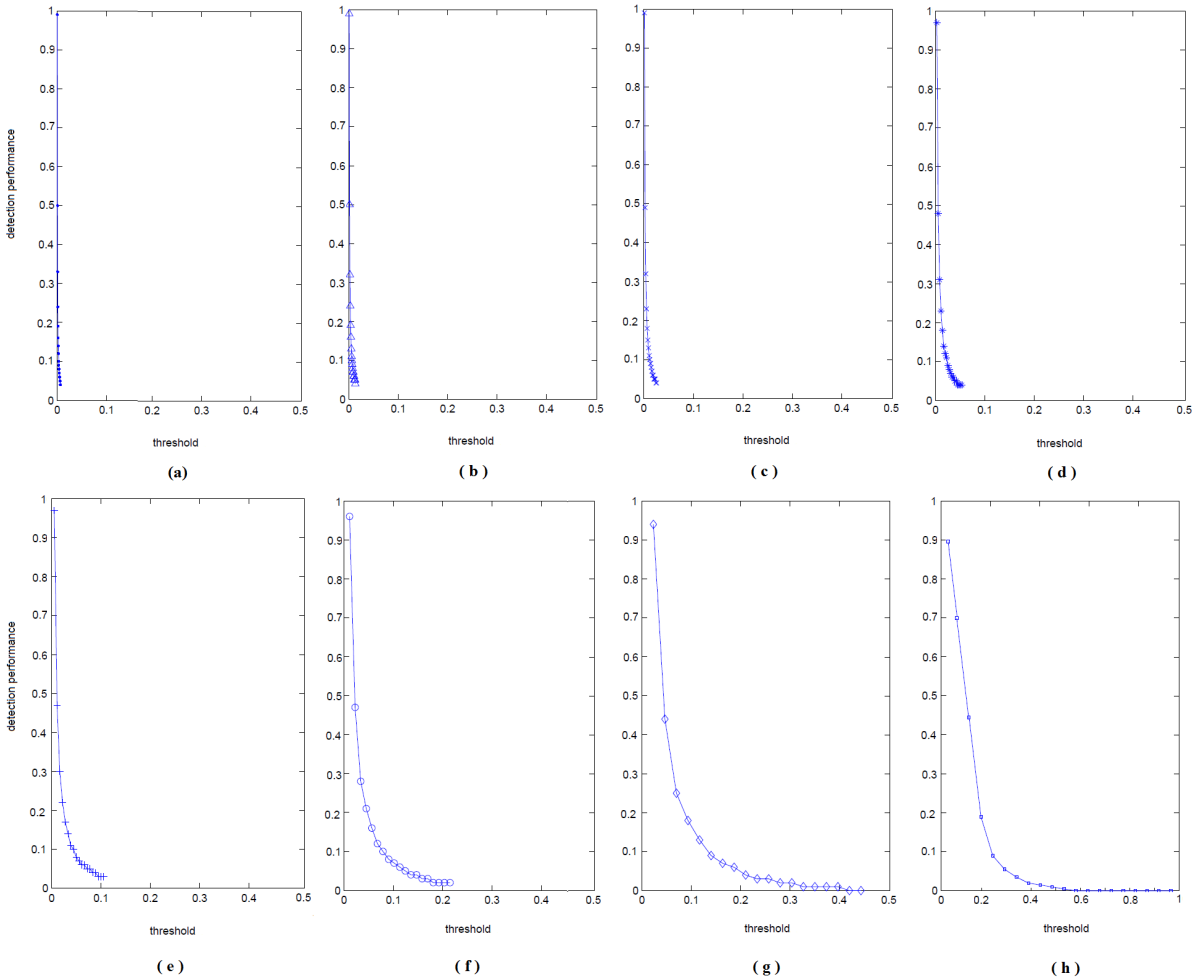


FIGURE 8. Error detection performance of the data processing system when the error injection level does not exceed, (a) 16, (b) 8, (c) 4, (d) 2, (e) 1, (f) 0.5, (g) 0.25 and (h) 0.125 times the maximum data input value

6. **Conclusions.** The ABFT technique employs real convolution error-correcting codes to encode the input data. In order to reduce the round-off error from the output decoding process, systematic real convolution encoding is employed. This paper proposes an efficient method to detect the arithmetic errors using convolution codes at the output compared with an equivalent parity value derived from the input data. Number data processing errors are detected by comparing parity values associated with a convolution code. These comparable sets will be very close numerically, although not identical because of round-off error differences between the two parity generation processes. The effects of internal failures and round-off error are modeled by additive error sources located at the output of the processing block and input at threshold detector. The detection performance in the data processing system depends on the detection threshold, which is determined by round-off tolerances. The simulations show that the great difference between the round-off error and the computer-induced error is large enough to be distinguished.

Acknowledgment. We are grateful to the comments from Mrs. Mahbobeh Meshkinfam that significantly improved the quality of this paper. The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] K. H. Huang and J. A. Abraham, Algorithm-based fault tolerance for matrix operations, *IEEE Transactions on Computers*, vol.33, pp.518-528, 1984.

[2] J. Y. Jou and J. A. Abraham, Fault tolerant matrix arithmetic and signal processing on highly concurrent computing structures, *Proc. of IEEE*, vol.74, no.5, pp.732-741, 1986.

[3] J. Y. Jou and J. A. Abraham, Fault-tolerant FFT networks, *IEEE Transactions on Computers*, vol.37, pp.548-561, 1988.

[4] P. Banerjee, J. T. Rahmeh, C. B. Stunkel, V. S. S. Nair, K. Roy and J. A. Abraham, Algorithm-based fault tolerance on a hypercube multiprocessor, *IEEE Transactions on Computers*, vol.39, pp.1132-1145, 1990.

[5] J. Rexford and N. K. Jha, Algorithm-based fault tolerance for floating-point operations in massively parallel systems, *Proc. of Int. Symp. on Circuits & Systems*, pp.649-652, 1992.

[6] V. S. S. Nair and J. A. Abraham, Real number codes for fault-tolerant matrix operations on processor arrays, *IEEE Transactions on Computers*, pp.426-435, 1990.

[7] G. Bosilca, R. Delmas, J. Dongarra and J. Langou, Algorithm-based fault tolerance applied to high performance computing, *Journal of Parallel and Distributed Computing*, vol.69, no.4, pp.410-416, 2009.

[8] T. Roche, M. Cunche and J.-L. Roch, Algorithm-based fault tolerance applied to P2P computing networks, *The 1st International Conference on Advances in P2P Systems*, pp.144-149, 2009.

[9] G. R. Redinbo, Generalized algorithm-based fault tolerance: Error correction via Kalman estimation, *IEEE Transactions on Computers*, vol.47, no.6, 1998.

[10] G. R. Redinbo, Failure-detecting arithmetic convolution codes and an iterative correcting strategy, *IEEE Transactions on Computers*, vol.52, no.11, pp.1434-1442, 2003.

[11] G. R. Redinbo, Wavelet codes for algorithm-based fault tolerance applications, *IEEE Transactions on Dependable and Secure Computing*, vol.7, no.3, pp.315-328, 2010.

[12] A. M. Nia and K. Mohammadi, A generalized ABFT technique using a fault tolerant neural network, *Journal of Circuits, Systems, and Computers*, vol.16, no.3, pp.337-356, 2007.

[13] P. Banerjee, Algorithm-based fault detection for signal processing applications, *IEEE Transactions on Computers*, 1990.

[14] J. Baylis, *Error Correcting Codes: A Mathematical Introduction*, Chapman and Hall LTD, 1998.

[15] V. S. Veeravalli, Fault tolerance for arithmetic and logic unit, *IEEE Southeastcon*, pp.329-334, 2009.

[16] D. Costello and S. Lin, *Error Control Coding Fundamentals and Applications*, 2nd Edition, Pearson Education Inc., NJ, USA, 2004.

[17] R. H. Morelos-Zaragoza, *The Art of Error Correcting Coding*, 2nd Edition, John Wiley & Sons, 2006.

[18] A. J. Viterbi and J. K. Omura, *Principles of Digital Communication and Coding*, Mc-Grawhill, 1985.

[19] E. R. Berlekamp, A class of convolution codes, *Information and Control*, vol.6, pp.1-13, 1962.

[20] J. L. Massey, Implementation of burst-correcting convolution codes, *IEEE Trans. Information Theory*, vol.11, pp.416-422, 1965.

[21] B. W. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems*, Addison-Wesley, MA, 1989.

[22] G. R. Redinbo, Optimum Kalman detector/corrector for fault-tolerant linear processing, *The 23rd Int. Symp. Fault-Tolerant Computing*, Toulouse, France, vol.23, pp.299-308, 1993.