Detecting Computer-Induced Errors in Remote-Sensing JPEG Compression Algorithms

Cung Nguyen, *Member, IEEE,* and G. Robert Redinbo, *Fellow, IEEE,*

***Abstract*—The JPEG image compression standard is very sensitive to errors. Even though it contains error resilience features, it cannot easily cope with induced errors from computer soft faults prevalent in remote-sensing applications. Hence, new fault tolerance detection methods are developed to sense the soft errors in major parts of the system while also protecting data across the boundaries where data flow from one subsystem to the other. The design goal is to guarantee no compressed or decompressed data contain computer-induced errors without detection. Detection methods are expressed at the algorithm level so that a wide range of hardware and software implementation techniques can be covered by the fault tolerance procedures while still maintaining the JPEG output format. The major subsystems to be addressed are the discrete cosine transform, quantizer, entropy coding, and packet assembly. Each error detection method is determined by the data representations within the subsystem or across the boundaries. They vary from real number parities in the DCT to bit-level residue codes in the quantizer, cyclic redundancy check parities for entropy coding, and packet assembly. The simulation results verify detection performances even across boundaries while also examining roundoff noise effects in detecting computer-induced errors in processing steps.**

***Index Terms*—Cyclic redundancy check (CRC) parity calculations, discrete cosine transform (DCT), fault-tolerant source coding, Huffman coding, JPEG data compression, quantization protection, soft errors.**

I. Introduction

COMPUTER technology has been extensively applied in modern signal and image processing. The advances of VLSI technologies have allowed processor arrays to be implemented in hardware [1]–[3] so that they can perform mathematical computations at high speed. However, fault tolerance has been a major issue for the high density memory and intensive computational systems, because the overall soft error rate is directly related to the number of circuit elements and the number of operators performed [4], [5]. Recent experiments [6], [7] show that soft errors pose significant problems in remote-sensing satellites. Furthermore, those errors are the main concern in commodity processors in general use, as studied by a consortium of researchers from leading technology companies [8]. The JPEG image compression systems can seriously be affected by soft errors because of their wide uses in remote sensing and medical imaging. In such applications, fault tolerance techniques are very important in detecting computer-induced errors within the JPEG compression system, thus guaranteeing the quality of image output while employing the compression data format of the standard.

Fault tolerance is different from error resilience in compression standards. Resilience generally refers to the capabilities of recovering from errors introduced by the transmission path of compressed data while fault tolerance protects against errors that are introduced by the computing resources executing the compressing and decompressing algorithms. Such computational errors appear before or after the transmission medium and therefore are not directly addressed in the compression standards although some features such as restart markers in JPEG [9] can be used to enhance fault tolerance when properly employed.

This paper investigates the implementation of the JPEG still image compression standard. Fault models for its functional stages will be developed based on the assumptions for error types and locations. The fault tolerance design strategies are developed to insure the detection of anomalies in the computational stages in the presence of errors. One straightforward but costly way to achieve this goal is recompute the error data section given the system has adequate available resources. Another way is to generate redundant information as a check code for that data, using this information and an error detection/correction algorithm to detect and possibly fix errors. However, error detection is the major goal of this research. For certain parts of the JPEG standard, earlier work employing algorithm-based fault-tolerance (ABFT), input and output weighted sums of the fixed point data lines can be used to distinguish the errors due to coefficient rounding from those due to computer-induced errors. For the other parts, parity check and coding techniques are used to detect errors. The checking of data across boundaries is mandatory for the multi-stage systems, guaranteeing the data are error-free when transferred between two adjacent stages, which can be accomplished by developing relationships between the quantities in the computations of two adjacent stages.

The outline of the paper is as follows. Section II reviews the major functional stages of the baseline JPEG, which includes discrete cosine transforms (DCTs), uniform Quantization, run-length coding, entropy Huffman coding, and JPEG data organization. Section III describes the fault models applicable to the functional stages, including for the across stages' boundaries. The implementations of concurrent error detection and parity checking codes based on proposed fault models are also explained in Section III, while Section IV examines the error detection performance of individual stages and overall system. Final conclusions are contained in Section V.

Manuscript received December 28, 2004; revised August 9, 2005. This work was supported in part by the National Science Foundation under Grant CCR-0104851. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Amir Said.

The authors are with the Department of Electrical and Computer Engineering, University of California, Davis, CA 95616 USA (e-mail: redinbo@ece.ucdavis.edu; cunguyen@ece.ucdavis.edu).

Digital Object Identifier 10.1109/TIP.2006.873425

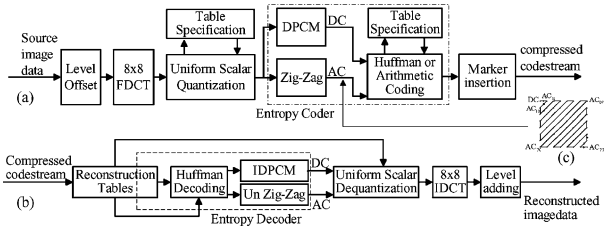


Fig. 1. (a) JPEG encoder, (b) JPEG decoder, and (c) zigzag sequence scanning.

II. Brief Review of JPEG Encoding/Decoding Systems

The JPEG standard is fairly complex because it defines a number of related image compression techniques with options for selecting ones suited for an image environment. For the sake of a simpler analysis, only the baseline JPEG model will be used to describe fault tolerance issues. More details about the JPEG standard can be found in [9]. The compression process, as shown in Fig. 1(a), starts with a level shift operator. For example, with an 8-bit input precision, the level shift is achieved by subtracting 128 from every sample of these patterns. The patterns are partitioned into nonoverlapping 8×8 blocks and each block is independently transformed by a forward DCT into a set of 64 DCT coefficients. One of these values is referred to as the DC coefficient while the remaining 63 are called AC coefficients. Each of the 64 coefficients is then quantized using 64 corresponding step size values from the quantization table. After quantization, the DCT coefficients are rearranged in a zigzag sequence order as shown in the elaboration of Fig. 1(c). Two coding procedures are used, one for the DC coefficients and another for all the AC coefficients. The DC coefficients are coded differentially using a one-dimensional (1-D) predictor, which is the quantized DC value from the most recently coded 8×8 block of the same component. At the beginning of the scan and the beginning of each restart interval, the prediction for the DC coefficient is initialized to 0. For the AC coefficients, runs of zero coefficients are identified and coded using intermediate symbols called descriptors. These descriptors are then encoded using Huffman code tables. If all the remaining coefficients in the zigzag sequence order are zero, this is coded explicitly as an end-of-block (EOB) which will be expressed later in this paper. These combined steps are called entropy coding.

Structurally, the formats of the compressed image consist of an ordered collection of parameters, markers, and entropy-coded data segments. Parameters in turn are often organized into marker segments. A marker segment is a combination of a marker and a set of parameters. Each marker is assigned two-byte codes (an FF byte followed by a byte which is not equal to 0 or FF) that allows a decoder to parse and locate various parts of the compressed data. The full descriptions of JPEG compressed data format can be found in [9].

The decompression is a reverse process of the compression as indicated in Fig. 1(b). Part of the received compressed image data stream is used to reconstruct the Huffman and quantization table specifications, whereas the remaining parts are fed into the entropy decoder. After the DC and AC coefficients of an 8×8 block are decoded by the entropy decoder, they are dequantized by multiplying each with the same quantization value as has been used at the encoder. The results are fed into the two-dimensional (2-D) inverse DCT (2-D IDCT). The quantization/dequantization steps are considered lossy. In addition, the forward and inverse DCT operations generate roundoff errors due to the finite precision in computer arithmetic. Fault tolerance models for the JPEG computational stages, are developed in the following sections along with techniques for separating roundoff and lossy errors.

For a JPEG image stream, errors in a DC or AC entropy codeword of compressed data stream may propagate and corrupt the rest of the data block containing these errors if one or more code lengths were incorrectly decoded. These errors may propagate to more data blocks and affect on a large image area. As discussed in [10], when an error propagates during the decompression process, it can have one of the following effects: 1) additional 8×8 blocks are detected, 2) 8×8 blocks are lost, and 3) the correct number of blocks are detected, even though with errors. The first two situations happen because of error propagation, an extra EOB character is detected, or one is missed. All of these situations can be the result of errors computer-induced errors in the JPEG compression system. These errors cannot be covered by channel coding because such errors occur before the channel coding is applied.

III. Error Models for JPEG Compression

A. Two-Dimensional DCT Processing

In this section, error detection capabilities for the 2-D FDCT using weighted sums over the input and output data are developed. The error detection for the 2-D IDCT is accomplished in a similar way. The 2-D FDCT is generally implemented by concatenating 1-D FDCT operations. The 2-D FDCT and 2-D IDCT for an $N \times N$ array are described formally by matrix equations

$$\begin{aligned} &\text{FDCT}: \mathbf{V} = \mathbf{C} \cdot \mathbf{U} \cdot \mathbf{C}^T; \quad \text{IDCT}: \mathbf{U} = \mathbf{C}^T \cdot \mathbf{V} \cdot \mathbf{C} \\ &\mathbf{C} = \sqrt{\frac{2}{N}}[c_{i,j}]; \quad c_{i,j} = \begin{cases} \frac{1}{\sqrt{2}}, & i = 0 \\ \cos\frac{i(2j+1)\pi}{2N}, & i \neq 0 \end{cases} \end{aligned} \tag{1}$$

where $\mathbf{U}$ and $\mathbf{V}$ are the $N \times N$ data input and DCT coefficients output matrices, respectively. The connection between 2-D and 1-D DCTs can be expressed by written (1) in another way

$$\mathbf{V} = \mathbf{A} \cdot \mathbf{C}^T = (\mathbf{C} \cdot \mathbf{A}^T)^T, \quad \text{where } \mathbf{A} \triangleq \mathbf{C} \cdot \mathbf{U}. \tag{2}$$

The input and output matrices $\mathbf{f}$ and $\mathbf{F}$, respectively, then can be represented by the following vectors:

$$\begin{aligned} &\mathbf{U} = [\mathbf{f}_0 \ \ldots \ \mathbf{f}_{N-1}]; \quad \mathbf{V} = [\mathbf{F}_0 \ \ldots \ \mathbf{F}_{N-1}]; \quad 0 \leq i < N \\ &\mathbf{f}_i \triangleq [f_{0,i} \ \ldots \ f_{N-1,i}]^T, \quad \mathbf{F}_i \triangleq [F_{0,i} \ \ldots \ F_{N-1,i}]^T. \end{aligned} \tag{3}$$

Matrix $\mathbf{A}$ can be expressed as a sequence of N, 1-D FDCTs

$$\mathbf{A} = \mathbf{C} \cdot \mathbf{U} = [\mathbf{C} \cdot \mathbf{f}_0 \quad \mathbf{C} \cdot \mathbf{f}_1 \quad \ldots \quad \mathbf{C} \cdot \mathbf{f}_{N-1}]. \tag{4}$$

The matrix-matrix product, $\mathbf{C} \cdot \mathbf{A}^T$, as shown in (2), can be computed using N 1-D DCTs on the columns of $\mathbf{A}^T$. The consequence is the computation of 2-D FDCT as prescribed by (1) involves $2N$ 1-D FDCTs, each performed on an N-point vector.

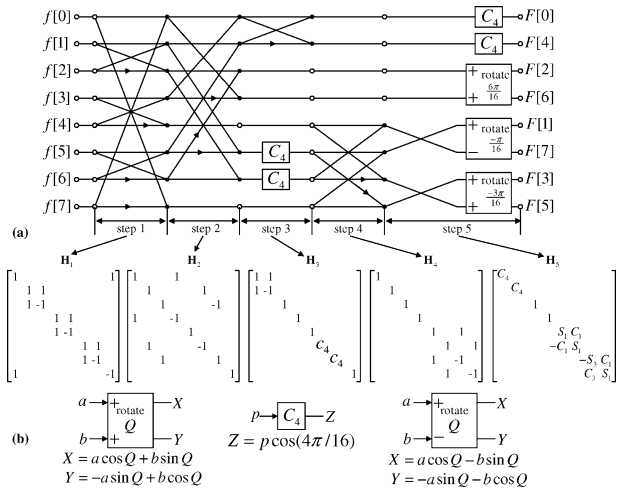


Fig. 2. Flowgraph for eight-point fast 1-D FDCT algorithm.

Furthermore, the N 1-D FDCTs as shown in (4) can be performed in parallel. Similarly, N 1-D FDCTs parallel computations can also be used to compute the $\mathbf{C} \cdot \mathbf{A}^T$ product. Hence, the overall the time overhead for a 2-D FDCT computation is only twice that of 1-D DCT.

To further improve the speed, fast DCT techniques and algorithms have been developed. Fig. 2 is a flowgraph of a fast 1-D DCT algorithm for $N = 8$ based on the results of Rao and Yip [11]. The overall transform is represented by a transform matrix, $\mathbf{C}$. As usual, the flow of operations in this graph is from left to right, and if a line contains an arrow, the signal is subtracted from the other signal fed to the node. The multiplication of a signal is indicated by C_4 in a box. Rotation boxes execute the indicated rotation through the given angles. If the input to the rotation is positive, the input line is labeled "+"; if the input is negative, the line is labeled "−". The processing of 8-point FDCT can be performed in five steps. Each processing step contains a number of arithmetic operations and the data transfers between nodes via the connection lines. For error modeling purposes, discussed in the next section, each of these processing steps can be represented by a transfer matrix. Let $\mathbf{H}_i$, $i = \{1, 2, 3, 4, 5\}$ be the transfer matrix for the processing step i. Then the transform matrix $\mathbf{C}$ can be represented by a matrix factorization

$$\mathbf{C} = \mathbf{H}_5\mathbf{H}_4\mathbf{H}_3\mathbf{H}_2\mathbf{H}_1 \Longrightarrow \mathbf{F} = \mathbf{H}_5\mathbf{H}_4\mathbf{H}_3\mathbf{H}_2\mathbf{H}_1\mathbf{f}. \quad (5)$$

$\mathbf{H}_1, \ldots, \mathbf{H}_5$ are shown in Fig. 2(a); C_i and S_i are defined by

$$C_i = \cos\left(\frac{i\pi}{16}\right), \quad S_i = \sin\left(\frac{i\pi}{16}\right), \quad i = 0, \ldots, 7. \quad (6)$$

B. DCT Error Model

It is generally assumed that transient errors can occur in the intermediate values at any time during the course of DCT processing as shown in Fig. 3. Furthermore, only one error is permitted during a sequence of operations to avoid complete overload. The data flow structure shows how a single error, caused by computer fault or any other failure, propagates to multiple

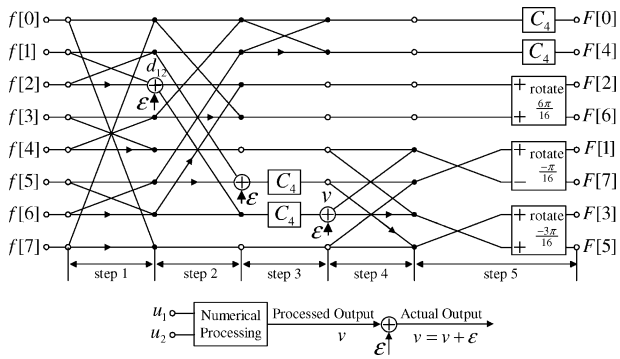


Fig. 3. Possible error modeling points in 8-point FDCT system.

outputs. Any error occurring in an arithmetic operation may accurately be described by adding a numerical value to the normal value at the output of the node. For example, consider the node d_{12}, where if the subtraction operator is error-free, its output value will be $d_{12} = f[1] - f[2]$. Otherwise, if an error occurs, the actual output $d_{12} = f[1] - f[2] + \varepsilon$, where ε is a numerical modeling error value. Any transient error in the DCT structure is assumed to emanate from a single numerical error value inserted at the input of a processing step. The single initiating error is assumed to be transient and only occurs at most once in each calculation cycle. Such an error propagates to the following steps with different magnitudes, and finally, emerges possibly at a number of output lines with different numerical impacts. This error effect can be determined by using transfer matrices describing the computation steps given in (5).

Consider error modeling applied to the 8-point FDCT. The computer-induced error in a location is described by adding an 8×1 unit vector $\vec{\varepsilon}_s^{(j)}$ to the intermediate data value

$$\vec{\varepsilon}_s^{(j)} = [0 \ \ldots \ 0 \ 1 \ 0 \ \ldots \ 0]^T \quad (\text{value } 1 \text{ in row } j)$$
$$\text{line}: \ j = 0, 1, \ldots, 7; \quad \text{step}: \ s = 1, 2, \ldots, 5. \quad (7)$$

The single error can have magnitude other than unity, say $\theta_s^{(j)}$, so that the modeling error can be presented by

$$\vec{\varepsilon} = \theta\vec{\varepsilon}_s^{(j)}. \quad (8)$$

When the error appears at the input to step s of the FDCT, the errors at the output of this step are added to the result obtained in the normal processing operation

$$\theta_s^{(j)}\mathbf{H}_s\vec{\varepsilon}_s^{(j)}. \quad (9)$$

The error effects at the FDCT output given a single error occurring at the input to step s, line j are determined by a product of transfer matrices representing steps following the error point. This matrix is labeled $\mathbf{G}_s$ and defined by

$$\mathbf{G}_s = \mathbf{H}_s\mathbf{H}_{s+1}\cdots\mathbf{H}_5; \quad s = 1, 2, \ldots, 5. \quad (10)$$

The errors at the FDCT outputs are represented by vector $\mathbf{e}_f$

$$\mathbf{e}_f = \theta_s^{(j)}\mathbf{G}_s\vec{\varepsilon}. \quad (11)$$

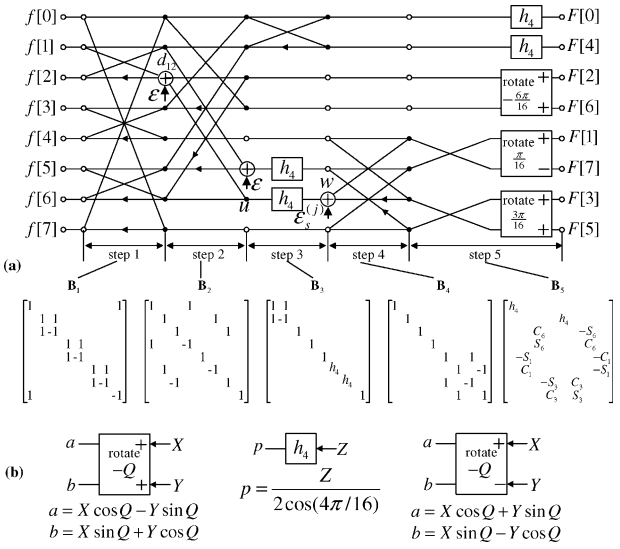


Fig. 4. Possible error modeling points in 8-point 1-D IDCT system.

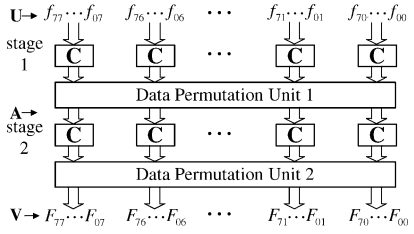


Fig. 5. The 8 × 8 2-D FDCT structure using 1-D FDCT arrays.

These errors are added into the DCT coefficients output according to the superposition principle

$$\mathbf{F}_e = \mathbf{F} + \mathbf{e}_f = \mathbf{C}\mathbf{f} + \theta_s^{(j)}\mathbf{G}_s\vec{\varepsilon}_s^{(j)}. \tag{12}$$

The structure for the 1-D IDCT is the same as that for the FDCT, except that the data flows are reversed as shown in Fig. 4(a). The rotations are also reverse in direction and their operations are shown in Fig. 4(b). The IDCT processing steps can be described using the inverse transfer matrices

$$\mathbf{f} = \mathbf{B}_1 \cdot \mathbf{B}_2 \cdot \mathbf{B}_3 \cdot \mathbf{B}_4 \cdot \mathbf{B}_5 \cdot \mathbf{F}$$

where $\mathbf{B}_1, \ldots, \mathbf{B}_5$ are shown in Fig. 2(b).

A similar error model can be developed for the eight-point 1-D IDCT. A single error $\vec{\varepsilon}_s^{(j)}$ is inserted to the step s input at line j of the transform structure. The error effects at the output of this section are described by the transfer matrix $\mathbf{B}_s$. For the general model, a scaling factor $\theta_s^{(j)}$ is required. The overall error at the IDCT output is represented by vector $\mathbf{e}_i$

$$\mathbf{e}_i = \theta_s^{(j)}\mathbf{B}_1 \cdots \mathbf{B}_s \cdot \vec{\varepsilon}_s^{(j)}. \tag{13}$$

Then, the overall error is added to the normal output

$$\mathbf{f}_e = \mathbf{f} + \mathbf{e}_i = \mathbf{B}_1 \cdots \mathbf{B}_s \cdot \left(\mathbf{B}_{s+1} \cdots \mathbf{B}_5 \cdot \mathbf{F} + \theta_s^{(j)}\vec{\varepsilon}_s^{(j)}\right). \tag{14}$$

The 2-D FDCT is divided into two transform stages, and for the eight-point 2-D transform example, each stage has eight 1-D FDCTs working in parallel as shown in Fig. 5. The first stage

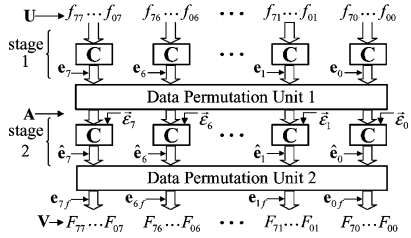


Fig. 6. Error model for an 8 × 8 2-D FDCT.

transforms the columns of the 8 × 8 data input array. The Data Permutation Unit 1 permutes eight DCT vector outputs so that the input vector to the ith 1-D FDCT on the second stage contains all the ith elements of the eight vectors from stage 1 output. The second DCT stage transforms these input vectors which is equivalent to the transforming eight rows of the 2-D array $\mathbf{A}$ as described in (2). The Data Permutation Unit 2 permutes the output data vector again so that $\mathbf{F}_i$ will be the ith column of the 2-D FDCT output. From a fault tolerance viewpoint, all these operations can experiences transient errors.

The fault model for the 2-D FDCT can be constructed from those of the arrays of 1-D FDCT as shown in Fig. 6, which again exemplifies the 8 × 8 case for exposition purposes. Let $\mathbf{e}_i = [e_{0i}, e_{1i}, \ldots, e_{7i}]^T$ for $i = 0, 1, \ldots, 7$, respectively, be the 8 × 1 error vector generated by the ith 1-D FDCT at stage 1. The elements of these eight error vectors are permuted in the Data Permutation Unit 1 so the error vector, $\vec{\varepsilon}_i$, entering the input lines of the ith 1-D FDCT in stage 2, is composed of the respective ith elements of the error vectors $\mathbf{e}_1, \ldots, \mathbf{e}_7$. Thus, $\vec{\varepsilon}_i$ can be represented by

$$\vec{\varepsilon}_i = [e_{i0}\ e_{i1}\ \ldots\ e_{i7}]^T. \tag{15}$$

The overall error effect at the output of the ith 1-D FDCT on stage 2 is a combination of the error vectors $\vec{\varepsilon}_i$, propagating from the stage 1 and the error $\hat{\mathbf{e}}_i$ caused by internal errors in stage 2. These errors can be consolidated into the vector $\mathbf{r}_i$

$$\begin{aligned}\mathbf{r}_i &= [r_{0i}\ r_{1i}\ \ldots\ r_{7i}]^T \\ &= \mathbf{H}_5 \cdot \mathbf{H}_4 \cdot \mathbf{H}_3 \cdot \mathbf{H}_2 \cdot \mathbf{H}_1 \cdot \vec{\varepsilon}_i + \hat{\mathbf{e}}_i.\end{aligned} \tag{16}$$

At the Data Permutation Unit 2 output, eight error vectors $\mathbf{r}_i$ can be collected into eight rows of an error matrix $\mathbf{E}_f$, where the subscript f means final output of the 2-D DCT

$$\begin{aligned}\mathbf{E}_f &= \begin{bmatrix} r_{00} & r_{01} & \ldots & r_{07} \\ \vdots & \vdots & \vdots & \vdots \\ r_{70} & r_{71} & \ldots & r_{77} \end{bmatrix} \\ &\triangleq \begin{bmatrix} \mathbf{e}_{0f} \\ \vdots \\ \mathbf{e}_{7f} \end{bmatrix}, \quad \text{where } \mathbf{e}_{if} = [r_{i0}\ r_{i1}\ \ldots\ r_{i7}].\end{aligned} \tag{17}$$

C. Error Detection Design for the 2-D DCT

The previous section developed a viable fault model of the 2-D FDCT based on the fault model of the 1-D FDCT. This

model is the basis for protecting the transforms in the JPEG compression system. This subsection proposes an efficient method for detecting arithmetic errors using weighted sums of the 1-D FDCT coefficients at the output as parity values compared with an equivalent parity value derived from the input data. From previous work concerning the discrete Fourier transform, it is possible to use a single parity value at the output for detecting computational errors within the transform stages [12]–[17]. A concurrent error detection scheme can effectively detect single errors introduced in any computation step even though the error propagates to many outputs.

The input and output weighting factors for computing comparable parities may be related from the definition of the N-point 1-D FDCT. The theoretical transform equations for outputs F_k are given $k = 0, 1, \ldots, N-1$

$$F_k = \alpha_k \sum_{n=0}^{N-1} f_n \cos\frac{(2n+1)k\pi}{2N}, \quad \alpha_k = \begin{cases} \sqrt{\frac{1}{N}}, & k=0 \\ \sqrt{\frac{2}{N}}, & 0<k<N. \end{cases}$$

The output parity number is defined as a weighted sum over the output values and it can be computed using the inner product between the DCT coefficient output vector $\mathbf{F}$ and the vector $\mathbf{w} = [w_0, w_1, \ldots, w_{N-1}]^T$

$$P_{\text{out}} = \mathbf{w}^T\mathbf{F} = \sum_{k=0}^{N-1}\left[\alpha_k \sum_{n=0}^{N-1} w_k f_n \cos\frac{(2n+1)k\pi}{2N}\right]. \quad (18)$$

Interchanging variables, (18) can be rewritten in a form

$$\begin{aligned} P_{\text{out}} &= \sum_{n=0}^{N-1} f_n \sum_{k=0}^{N-1} \alpha_k w_k \cos\frac{(2n+1)k\pi}{2N} \\ &= \sum_{n=0}^{N-1} f_n b_n, \\ &\text{where } b_n = \sum_{k=0}^{N-1} \alpha_k w_k \cos\frac{(2n+1)k\pi}{2N}. \end{aligned} \quad (19)$$

Note the values of b_ns can be determined and stored beforehand. If the input data $\{f_n\}$ are weighted by the factor $\{b_n\}$, then the input weighted sum is easily seen to be inner product between the input data vector $\mathbf{f}$ and the input weighting vector $\mathbf{b}$

$$P_{\text{in}} = \mathbf{b}^T\mathbf{f} = P_{\text{out}}, \quad \text{where } \mathbf{b} = [b_0\ b_1\ \ldots\ b_{N-1}]^T. \quad (20)$$

In the error-free condition, theoretically, $P_{\text{in}} = P_{\text{out}}$, so concurrent error detection principle can be applied here. Its computational overhead is in the order of N, since there are N multiplications and $2N$ additions.

The concurrent error detection using weighting parity values is shown in Fig. 7. The two parities P_{in} and P_{out} are equal if a computer-induced error does not occur, ignoring any roundoff

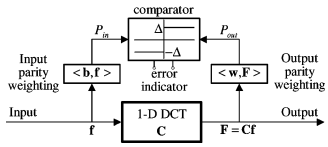


Fig. 7. Concurrent parity checking of 1-D FDCT.

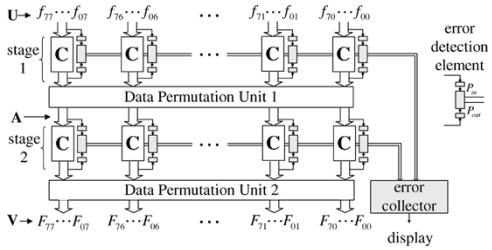


Fig. 8. Concurrent error detection designed for 8 × 8 2-D FDCT.

errors in the arithmetic computations. However, the difference between the parity values, $|P_{\text{in}} - P_{\text{out}}|$, considering a roundoff threshold, Δ, can be used to detect a computer-induced error

$$\text{if } |\delta| < \Delta, \text{ transform error} - \text{free, where } \delta = P_{\text{in}} - P_{\text{out}}. \quad (21)$$

The weighting factor in vector $\mathbf{w}$ should be carefully selected in order to prevent any overflow of the output weighted sum in (19) as well as being capable of detecting error effects in the output DCT values. Error detection can be achieved if certain error vector spaces are linearly independent which are determined during the design process. It can be shown that the factors $w_k = 1/(k+1)$ not only avoid overflow but allow the error effects to be detected at the outputs [15].

The error detection structure for the 2-D FDCT is proposed as shown in Fig. 8, where every 1-D FDCT is protected using the concurrent error detection scheme as described earlier in this section. The error indicators from individual 1-D FDCTs are consolidated into an error collector unit. The error collector unit analyzes the parity information issuing detection conditions when appropriate. The permutation units provide connections which can only cause errors on a single input line between stages.

There are many methods for implementing the DCT transforms required in the JPEG standard including using distributed arithmetic [18] which can have appealing features such as low power attributes [19], [20]. There was an early fault tolerance approach to DCT implementation using distributed arithmetic which relied on residue number codes internal to the computational structures [21]. Unfortunately, it was not very efficient and did not provide complete error coverage. However, the DCT protection approaches described above, which cover single errors, can be adapted easily for distributed arithmetic DCT implementations. The parity calculations (19), (20) and their comparison detector inequality (21) afford the same coverage as shown above. Furthermore, the two parity computations can be done using distributed arithmetic.

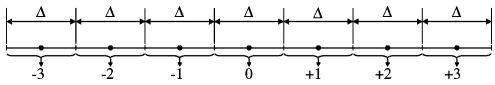


Fig. 9. Uniform scalar quantizer with quantization value Δ.

D. Computer-Induced Error Model for Quantization

As noted in Section II, quantization in JPEG is a non-invertible process because DCT coefficients are scaled and truncated to integer values. If the quantizer also experiences computer-induced errors, the output quantization error may be larger than normal. When these errors exceed the thresholds for visibility [22], the reconstructed image may show noticeable artifacts. Various techniques have been suggested to minimize the roundoff numerical errors associated with quantization. In the fault tolerance area, the data retry techniques suggested in [23] uses additional time to distinguish between roundoff errors and functional errors. The division operator plays a central role in the theoretical quantization process. The implementation for real number division is quite complicated and fault tolerance issues for it have never been addressed. A computer-induced error in the quantization is likely occur in the division operator because of its complexity. The objective of this section is to describe fault tolerance design algorithm for the quantization stage employed in the baseline JPEG.

Let y be a typical coefficient from an 8×8 DCT block. The quantization $Q : y \rightarrow q$ in baseline JPEG is defined by

$$q = \text{sign}(y) \left\lfloor \frac{|y| + 0.5\Delta}{\Delta} \right\rfloor, \text{ where } 1 \leq \Delta \leq 255. \quad (22)$$

The output values of a uniform quantizer is demonstrated in Fig. 9. The endpoints of each quantization interval are separated by the vertical lines and the quantized values are represented by the dots at the middle of these intervals. Each value y belonging to an interval will be mapped to a quantized value designated for that interval. On the other hand, given a quantized value q, the dequantizer estimates the value of y as $\hat{y} = Q^{-1}(q)$. For the parameters shown in Fig. 9, the inverse quantizer is simply determined by

$$\hat{y} = Q^{-1}(q) = q\Delta. \quad (23)$$

The difference between the original and reconstructed data in the quantization processes is

$$\begin{aligned} \varepsilon_q &= y - \hat{y} = y - \text{sign}(y) \left\lfloor \frac{|y|}{\Delta} + 0.5 \right\rfloor \Delta \\ &= \begin{cases} \left(\frac{y}{\Delta} - \left\lfloor \frac{y}{\Delta} + 0.5 \right\rfloor\right)\Delta, & \text{if } 0 \leq y \\ \left(\frac{y}{\Delta} + \left\lfloor 0.5 - \frac{y}{\Delta} \right\rfloor\right)\Delta, & \text{otherwise} \end{cases} \\ &\text{since } \left| \frac{y}{\Delta} \pm \left\lfloor 0.5 \mp \frac{y}{\Delta} \right\rfloor \right| \leq 0.5 \Longrightarrow \varepsilon_q \leq \frac{\Delta}{2}. \end{aligned} \quad (24)$$

If computer-induced errors occur during the quantization process, the quantized value may be changed from its correct output q to another integer value, say $\tilde{q}$. This error can be modeled as adding a numerical value to the quantized coefficient output as shown in Fig. 10. If σ is the modeled numerical error

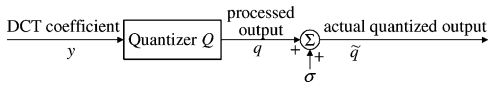


Fig. 10. Error model of the quantization stage.

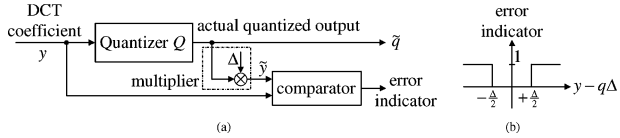


Fig. 11. Error detection structure for the scalar quantization.

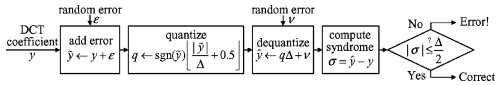


Fig. 12. Error detection algorithm for the proposed quantization structure.

added into the quantized DCT coefficient, the actual result of the quantization is modeled as follows:

$$\tilde{q} = q + \sigma, \quad \text{where } 1 \leq |\sigma|. \quad (25)$$

The actual quantization error in the presence of a computer-induced error is easily developed

$$\begin{aligned} \tilde{\varepsilon}_q &= y - \tilde{q}\Delta = y - (q - \sigma)\Delta = \varepsilon_q + \sigma\Delta \\ &\text{since } |\varepsilon_q| \leq \frac{\Delta}{2} \text{ and } \Delta \leq |\sigma|\Delta \Longrightarrow \frac{\Delta}{2} \leq |\tilde{\varepsilon}_q|. \end{aligned} \quad (26)$$

When a computer-induced error does not change the correct quantized coefficient output, it is as if this error has not occurred. This observation permits complete error modeling without analyzing the detailed operations inside the quantizer.

E. Proposed Error Detection Algorithm for the Quantization

The proposed error detection structure employing a comparator for the quantization is shown in Fig. 11(a). The input to the quantizer is a DCT coefficient with value y. The quantized value output, $\tilde{q}$, may be different from the correct value. To guarantee the quantized value output is correct, the actual output $\tilde{q}$ is multiplied by the corresponding quantization step value Δ. The product $q\Delta$ is then compared with the input value y via a comparator using the following conditions for detection purposes:

$$\varepsilon_q = y - \tilde{q}\Delta; \quad |\varepsilon_q| < \frac{\Delta}{2} \Longrightarrow \text{error} - \text{free}. \quad (27)$$

The resulting error indicator characteristic is shown in Fig. 11(b). If an error is detected, recomputation could be initiated for correction. The full verification of the detection process is shown in Fig. 12.

F. Entropy Encoding Error Model

From Section II, the quantized DC coefficient and the 63 AC coefficients associated with each 8×8 DCT block are compressed by the entropy encoder. The compressing efficiency of the encoding process is improved by exploiting the runs of zeros and the decreasing in magnitude tendency of DCT coefficients

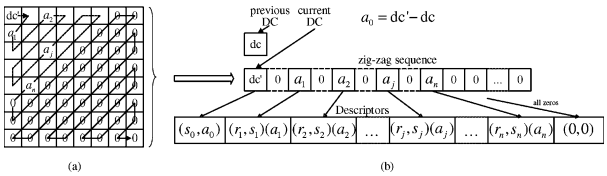


Fig. 13. Zigzag sequence and the equivalent run-length coding result for a sample quantized DCT coefficients block.

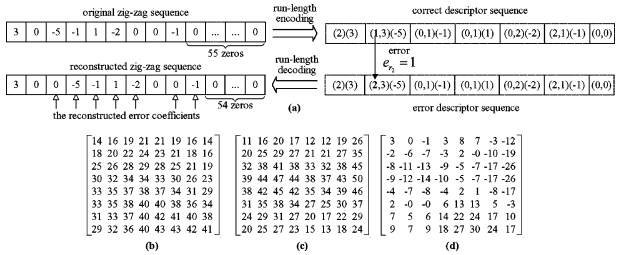


Fig. 14. Propagation of an error in the run-length encoding and its error effects to the reconstructed data block. (a) Run-length encoding/decoding processes. (b) Reconstructed 8 × 8 data image for run-length coding error-free. (c) Reconstructed 8 × 8 data image when error occurred as shown in (a). (d) Error pattern.

in a zigzag sequence and finally from the optimization of variable length codewords. Because of the high computation complexity, if an error occurs during encoding a DCT block the entire block is influenced by this error and cannot be used. On the other hand, the end-of-block (EOB) codeword acts as a synchronization marker, which helps prevent an error in one block from propagating to the others. However, if an EOB marker is in error, a single error can spread to the next data block. A fault model for the run-length and Huffman coding will be expressed in this section.

The run-length encoding process of an 8 × 8 quantized DCT block uses a zigzag sequence as illustrated in Fig. 13(a). The difference between the current DC value and that of the most recent coded block is represented by $(\mathrm{SIZE}, \mathrm{AMPLITUDE}) = (s_0, a_0)$ descriptor symbols. Each nonzero AC coefficient a_j is encoded by a $(\mathrm{RUNLENGTH}, \mathrm{SIZE})(\mathrm{AMPLITUDE}) = (r_j, s_j)(a_j)$ descriptor symbols, where $0 \leq r_j \leq 15$, represents the number of consecutive zero-valued AC coefficients in the zigzag sequence preceding the nonzero AC coefficient. If a zero-run length is greater than 15, an advantageous situation for compression, one or more extension descriptors (15,0) are used, where each (15,0) represents a set of 16 consecutive zeros. The SIZE symbol represents the number of bits associated with the nonzero AC coefficient as dictated by predetermined tables. Finally, the $\mathrm{EOB} = (0,0)$ descriptor terminates the block.

The relationships between the descriptors will be analyzed in order to develop an error detection strategy for the run-length encoder. Let I_K, $(0 \leq I_K \leq 63)$, be the index of the nonzero coefficient in a segment of the zigzag sequence. A sequence of descriptors representing a typical DCT block is shown in Fig. 13(b), where n is the number of nonzero AC coefficients present in the sequence. If a jth nonzero AC coefficient is preceded by $R_j > 15$ consecutive zeros, then the relationship between R_j and the descriptors applied to this coefficient is

$$\begin{aligned} &R_j = 16 \times i_j + r_j, \quad 0 \leq r_j \leq 15 \quad 1 \leq i_j \leq 3 \\ &j\text{th descriptor} = (15,0)\ldots(15,0)(r_j, s_j)(a_j) \end{aligned} \tag{28}$$

The (15,0) pattern is repeated i_j times. If the run-length of zeros preceding the jth nonzero AC coefficient is r_j (<16), the coefficient is simply represented by

$$(r_j, s_j)(a_j), \quad \text{which implies } i_j = 0. \tag{29}$$

Let n be the number of nonzero AC coefficient. The index of the jth nonzero AC coefficient, I_j, can be computed from descriptors of the first j nonzero AC coefficients

$$I_j = j + \sum_{k=1}^{j} (r_k + 16i_k) = I_{j-1} + (1 + r_j + 16i_j), \quad j \leq n. \tag{30}$$

This is a useful relationship between the nonzero AC coefficient indexes and the descriptors. Numerical errors in the run-length coding subsystem can be described as the unexpected changing value of a descriptor at the output of the run-length encoder. Particularly, when a descriptor $(r_j, s_j)(a_j)$ suffers an error in its the zero run-length symbol r_j, it can be represent by $(r_j + e_{r_j}, s_j)(a_j)$, where the numerical error e_{r_j} is an integer. This error will propagate to the rest of the AC coefficients in the zigzag sequence. Consequently, the index of the last nonzero AC coefficient in the zigzag sequence is

$$\begin{aligned} \tilde{I}_n &= n + \sum_{k=1}^{j-1} (r_k + 16i_k) + \left(r_j + e_{r_j} + 16i_j\right) \\ &\quad + \sum_{k=j+1}^{n} (r_k + 16i_k) \\ &= I_n + e_{r_j}. \end{aligned} \tag{31}$$

Similar error models are applicable for s_j and a_j symbols

$$\begin{aligned} &(r_j, s_j)(a_j) \rightarrow \left(r_j, s_j + e_{s_j}\right)(a_j) : \text{size error} \\ &(r_j, s_j)(a_j) \rightarrow (r_j, s_j)\left(a_j + e_{a_j}\right) : \text{value error}. \end{aligned} \tag{32}$$

The errors in s_j and a_j will not propagate to the rest of the sequence since these do not carry the information related to neighbor coefficients. Fig. 14 illustrates the propagation of an error $e_{r_2} = 1$ occurring in the zero-runlength symbol r_2. The locations of a group of the reconstructed coefficients are shifted right e_{r_2} positions that make a burst error with length less than or equal to $I_n - I_j + 1 + e_{r_2} = 9 - 3 + 1 + 1 = 8$. This location shifting generates significant error effects on the reconstructed image data as shown in Fig. 14(d). Errors are likely to occur anywhere in the data block.

In the Huffman encoder, each (s_0, a_0) (DC descriptor) or $(r_j, s_j)(a_j)$ (AC descriptor) is compressed into a variable length codeword using the combination of either DC or AC code table with the bits determined by the coefficient value. These codewords are concatenated to each other to form a compressed bitstream of an image data block. According to the JPEG Huffman encoding of DC coefficients guideline [9, pp. 443–444], the a_0 values are grouped into 12 categories based on their bit size, s_0, required to represent them. Each category is represent by a Huffman code and an extra codeword with s_0 bit length is appended to represent the actual a_0 value. Particularly, when a_0 is positive, the s_0 low order bits of a_0 are appended, otherwise, s_0

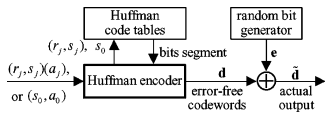


Fig. 15. General error model of the DC/AC Huffman encoder.

low order bits of $a_0 - 1$ are appended. The value s_0 plays as an index in accessing the DC Huffman code table.

For AC component, each (r_j, s_j) pair in the $(r_j, s_j)(a_j)$ descriptor is Huffman coded using a Huffman code table. Each Huffman code is followed by additional bits which specifying the sign and exact amplitude of the a_j value. The value a_j gives the number of bits required to specify the sign and the precise amplitude of a_j. The format for the additional bits is the same as in the coding of the DC coefficients. The additional bits are either the low-order bits of a_j when a_j is positive or the low-order bits of $a_j - 1$ when a_j is negative. In the coding process, the catastrophic effect of errors in the Huffman encoder or Huffman code tables can lead to the errors in the output codewords. Errors in the additional bits segment of the DC coefficient can propagate, since the DC coefficients are differentially encoded. Errors in the additional bits segment of the AC coefficient do not propagate and will affect only one AC coefficient in the 8×8 data block. An error in the bits segment of the DC or AC coefficient can propagate since the error may change the Huffman codeword into another Huffman codeword with different length. The error propagates until the end of the image bitstream or until the decoder cannot decode any further. Assume the errors in the Huffman coding process do not add or remove bits from the compressed data, then error can be modeled by XORing an error agent bit, say e_i, with the data bit d_i output from the Huffman encoder, which could lead to the flip of the d_i value. The fault tolerance model of the Huffman is shown in Fig. 15.

Let n be the length of a block compressed image output. Under the influence of intermittent errors, the actual image output $\tilde{\mathbf{d}} = [\tilde{d}_0, \ldots, \tilde{d}_n]$, can be presented by the correct compressed $\mathbf{d} = [d_0, d_1, \ldots, d_n]$ XORs with an equal length error sequence, $\mathbf{e} = [e_0, \ldots, e_n]$

$$\tilde{d}_i = d_i \oplus e_i; \quad d_i, e_i \in \{0,1\};\ i = 0, \ldots, n$$
$$\text{if } e_i = 1 \Longrightarrow i\text{th bit gets error.} \tag{33}$$

The error pattern is defined by vector, $\mathbf{e}$. Error detection method for the intermittent single error in the entropy encoder will be described and verified in the following subsection.

G. Error Detection for the Entropy Encoder

The error detection strategies are designed to enhance the reliability of the run-length and Huffman encoders. The run-length coding error can be detected by checking the indexes of nonzero AC coefficients as described in (30). The protected encoding algorithm is shown in Fig. 16, where each jth nonzero AC coefficient in the zigzag sequence input can be represented by its (index, value) pair. After the coefficient is encoded to the $(r_j, s_j)(a_j)$ descriptor, the index $\tilde{I}_j$ of the jth nonzero AC coefficient is recomputed using the descriptors produced at the output. If this index is equal to that of the input sequence, the zero run is error-free and the coding continues up to the last

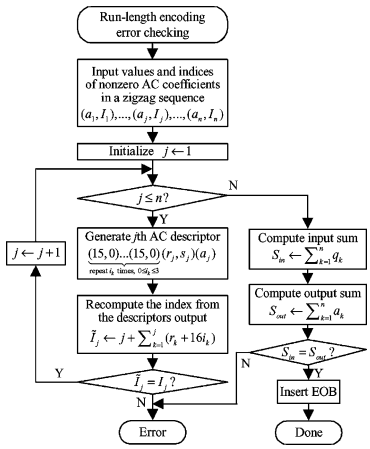


Fig. 16. Error checking algorithm for run-length coding.

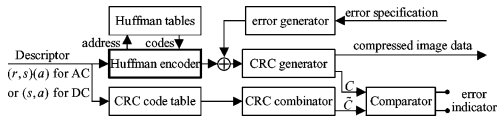


Fig. 17. Huffman encoder with CRC check codes generated.

(index, value) pair. Otherwise, if they are not equal, an error signal is generated and sent to the error handler. Finally, a single error can be detected via check sums

$$S_{\text{in}} = \sum_{k=1}^{n} q_k, \quad S_{\text{out}} = \sum_{k=1}^{n} a_k$$
$$S_{\text{in}} = S_{\text{out}} \Longrightarrow \text{AC values error} - \text{free}. \tag{34}$$

Since the processed data are all integers, comparison between input and output sums is exact because there is no roundoff errors. Overflow is not present by the small amplitude of the quantized DCT coefficients. The descriptor sequence is terminated by an EOB symbol.

Due to the variable length of compressed data stream output, an effective way to detect errors in the Huffman encoder uses the CRC codes. CRC codes are used primarily for encoding serial data messages. In this, the serial data is treated as a polynomial's coefficients, where the first bit in k-bit long serial data stream is treated as the coefficient of x^{k-1}, while the last bit is attributed to the coefficient of x^0. This input data polynomial is used as the dividend in a modulo-2 division process. The remainder when the input data polynomial is divided by the CRC polynomial is the CRC code of the data stream. According to the work of Nebus [24], if all error patterns are equally likely in a data block of k bits, then the probability that a CRC code with polynomial of degree r will detect an error is $1 - (2^{k-r} - 1)/(2^k - 1)$ As k goes to infinity, this probability approaches $1 - 2^{-r}$. For a 16-bit CRC polynomial, the probability of detecting an error is 0.999 985.

The proposed error detection strategy for the Huffman encoder is shown in Fig. 17. To detect bit errors in the compressed data output, a CRC codeword, C, of the compressed image data is computed. The CRC code table contains all the possible CRC codes of the descriptors. For the descriptors obtained from the encoding of an 8×8 DCT block, a set of CRC codes obtained

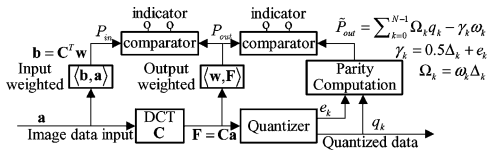


Fig. 18. Implementation of error detection across the boundary between 2-D DCT and quantization stages.

from the *CRC code table* are combined yielding a CRC codeword, $\tilde{C}$, via the CRC combinator unit. The characteristic of the $\tilde{C}$ code is, if no error occurs in either the compressed data or the CRC code table, then the C and $\tilde{C}$ codewords are identical. Otherwise, they will be different. More information about the computation of CRC code can be found in [25].

H. Errors Across 2-D DCT and Quantization Boundary

Errors detection for data across stage boundaries is important. A fault in an interfacing device may lead to the mismatch between the output data of one stage and the input data of the next stage. In this section, a fault tolerance model for the data between the 2-D DCT and quantization boundary will be developed. For the sake of simplicity, consider a positive DCT coefficient y. The quantized coefficient q can be expressed with the introduction of quantity, e

$$q = \frac{y}{\Delta} + 0.5 + e; \quad e = \left\lfloor \frac{y}{\Delta} + 0.5 \right\rfloor - \left(\frac{y}{\Delta} + 0.5 \right)$$
$$\text{with } \gamma = 0.5\Delta + e \cdot \Longrightarrow y = q\Delta - \gamma. \tag{35}$$

The output weighted sum P_{out}, as shown in (19), can be computed from the output of the quantization stage. It also means the error at the boundary can be detected by recomputing the output weighted sum using the quantized coefficients, q_k, and the quantization errors, γ_k. Let $\tilde{P}_{\text{out}}$ denotes the parity computed at the quantization output, which can be calculated

$$\tilde{P}_{\text{out}} = \sum_{k=0}^{N-1} w_k y_k = \sum_{k=0}^{N-1} (w_k \Delta_k) q_k - \sum_{k=0}^{N-1} w_k \gamma_k$$
$$= \sum_{k=0}^{N-1} \Omega_k q_k - \sum_{k=0}^{N-1} w_k \gamma_k; \quad \Omega_k \triangleq w_k \Delta_k. \tag{36}$$

The quantity Ω_k can be pre-determined because the weight factors w_k and the quantization values Δ_k are all known from the error detection for the DCT stage. The errors in 2-D DCT and quantization stages are covered by their own error detection structures. Thus, if there is no error at the boundary between these two stages, $|\tilde{P}_{\text{out}} - P_{\text{out}}|$ must be below a chosen threshold determined by roundoff tolerances. Otherwise, if error occurs across this boundary, $|\tilde{P}_{\text{out}} - P_{\text{out}}|$ exceeds the roundoff tolerances

$$\sigma = P_{\text{out}} - \tilde{P}_{\text{out}}$$
$$|\sigma| < \tau \Rightarrow \text{No error}; \quad \tau = \text{roundoff threshold}. \tag{37}$$

Error detection across this boundary is depicted in Fig. 18.

I. Error Detection for JPEG Compression Format Stage

The JPEG standard itself allows the use of restart markers to help the decoder with resynchronization. However, errors in the entropy code contents cannot be detected using restart markers,

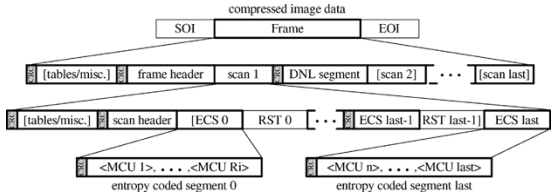


Fig. 19. CRC codes for a compressed image output.

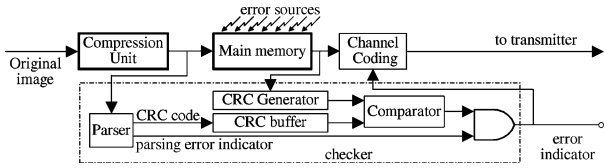


Fig. 20. Proposed fault tolerant system to ensure that output code stream format is correct before transmission.

except for the rare case when an entropy codeword is changed into a restart marker. CRC codes are very an excellent tool to detect the existence of such errors at the final system output. Proposed check code locations in a nonhierarchical syntax data structure are shown in Fig. 19. Each entropy coded segment (ECS) or marker segment has a 16-bit CRC code parity in its beginning. To maintain the JPEG standard stream format, these CRC codes will be removed from the data structure after the error checking is completed. A CRC code generation via table lookup is described in [25], where the JPEG data streams are expressed in bytes in order to speed up the polynomial division process. The structure for the error detection of JPEG compressed data is shown in Fig. 20, where data streams output from the compression unit are transferred to the main memory and the parser units. The parser is responsible for the format verification and CRC codes generation. These CRC codes are transferred to a CRC buffer and readily checked. When the main memory, whose contents can be effected by the internal chip errors or the external charge particles, receives a read request signal from the channel encoder, it transfer data to both the CRC Generator and the channel encoder. The codes generated by the CRC generator are compared with those in the CRC buffer. In the error-free condition, both CRC codes are identical. Otherwise, if these CRC codes are not identical, errors must exist in either the main memory or the checker. When the CRC code method is employed, the encoder and decoder share the same generator polynomial $g(x)$. The computation of CRC code via table look-up is possible if the frame length is an integer number of bytes. One hardware implementation of channel CRC codes can be found in [26].

IV. Error Detection Performance Results

A. For the 2-D DCT

Fig. 21 shows the performance, using the SNR computed for image data domain, of the proposed error detection mechanism for various detection thresholds. Random errors with magnitudes having error to data ratios extending from 10^{-5} to 1 were injected into one of the 16 1-D FDCTs at randomly selected locations. Without the protection, the signal to noise ratio performance drops along with the error strength, as indicated by the

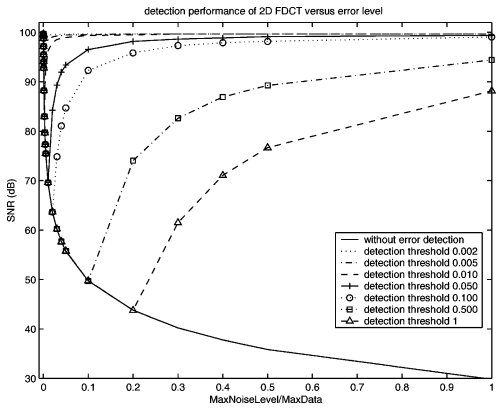


Fig. 21. Error detection performance of the 2-D FDCT system.

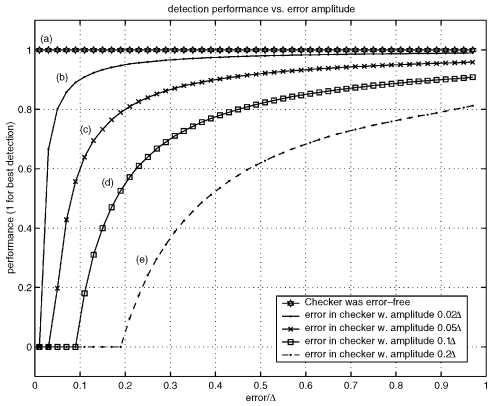


Fig. 22. Error detection performance versus the error amplitudes in the quantizer: (a) when only errors occurred in the quantizer; (b)–(e) when the checker was getting errors with amplitudes 0.02Δ, 0.05Δ, 0.1Δ, and 0.2Δ, respectively.

first solid line. However, when the error detection mechanism is active, the DCT transform performance was greatly improved, especially in the large error amplitude region. The transform performance is less improved for low error amplitude regions because detection values δs do not exceed detection thresholds. Nevertheless, the error effects are below roundoff noise levels which is generally acceptable.

B. For the Quantization

The detection method for the quantization stage is verified using a software implementation of JPEG image compression system. Twenty million 8×8 blocks of random data input were transformed yielding DCT coefficients that were input into the quantizer. Randomly located numerical errors with various amplitudes were generated and added to the DCT coefficients immediately before they are quantized. The checker was also allowed to experience computational system errors, by adding random numerical values results produced by the multiplier before the comparator. The detection performance is shown in Fig. 22 for the injected errors as described above. The experiment was conducted for a default luminance quantization table defined by the JPEG standard. The error detection performance curves are plotted for five error conditions of the checker. The experiment shows that all errors were detected without false alarms when the checker was error-free. When the checker is influenced by computer-induced errors, the performance is varied with the magnitude of the injected errors as demonstrated in the curves (b)–(e). The experiment indicates that with this design strategy, no miss detection is found in all cases. The miss detection here can be thought as the existence of incorrect quantized coefficient outputs without notification generated by the checker.

C. For the Entropy Encoder

In simulations for the entropy encoder, the checker was assumed error-free. Any errors in symbol r_s during run-length coding and any single errors appearing in the a_j symbols were detected. The CRC codes with 16-bit redundancy were used in error checking of the Huffman codewords output. The detection performance of these CRC codes was investigated in [27]. One particular issue that has been considered in the implementation was fitting the bits in the memory buffer. Each Huffman codeword of length n representing a descriptor was stored temporarily in a 32-bit memory buffer. Since $n \leq 26$ for an AC descriptor and $n \leq 20$ for a DC descriptor according to the default Huffman code tables, $32 - n$ most significant redundant bits of the memory buffer were filled with zeros. When the data were retrieved out from the buffer, these zero bits were discarded. Any errors occurring in these bits did not affect the compressed data output. However, the CRC generator encoded all bits in the memory buffer. Therefore, if an error only occurred in the unused bits, the detection algorithm would still indicated that the compressed data had an error. This is a particular false alarm circumstance. False alarm will slowdown the process due to the unnecessary repeat the encoding process. To avoid the uses of these redundant bits, the Huffman codewords lengths are saved along with their contents. In the checking step, all the bits beyond the corresponding word lengths are cleared to zero.

D. For the JPEG Compressed Data Format

The protection performance for JPEG compression format is dependent on the error rate and the length of compressed data stream. Given the length of a CRC codeword is v, each CRC codeword can contain up to k data bits and $r = v - k$ parity bits. The basic length v is generally chosen as $2^{r-1} - 1$ allowing a maximum $k = (2^{r-1} - 1) - r$ data bits to be checked. Given the CRC-16 polynomial $(g(x) = x^{16} + x^{12} + x^5 + 1)$, the smallest integer m for which $g(x)$ divides $(1 + x^m)$ is $2^{16-1} - 1 = 32\,767$. Hence, the maximum codeword length v is $v = 32\,767$, which corresponding to the data length $k = 32\,767 - 16 \approx 4$ KB. The CRC-16 polynomial can detect all single, double, triple and odd number of errors. Furthermore, it can detect all burst errors of length 16 or less [28]. In this particular design, the compressed data stream is partitioned into segments with length less than 2 KB. Each segment was associated with a CRC code parity. Bit errors are randomly injected in the code stream and the error detection performance versus the number of bit errors is shown in Fig. 23. The error coverage is quite high for different error rates.

Detection Perfomance From 100,000 Error Injections

Number of Errors	Number of Miss Detections	Error Detection Probability
1	0	1.00000
2	2	0.99998
3	0	1.00000
4	2	0.99998
5	1	0.99999
10	1	0.99999

Fig. 23. Error detection performance in the compressed stream output.

E. Protection Overhead Discussions

The overhead for introducing error-detecting protection throughout JPEG compression operations is difficult to assess because it depends on the metric used for expenses. However, it is possible to estimate computational overheads for each subsystems based on the number of operations in processing an 8×8 data block. A subtraction is comparable to an addition and is counted the same below. Comparisons are also like subtractions. The error-detecting overhead may be assessed in subsystems as follows.

1) For Each 8×8 2-D DCT: There are 256 multiplications, 240 additions and 16 comparisons overhead. The original computational cost for an 8×8 DCT are 416 additions, 256 multiplications. For an 8-bit number system, the cost of a multiplication is approximately fifty times greater than that of an addition while a division is one hundred times. Then, the extra calculations for the fault tolerance is about 93%.

2) For Quantization: There are 64 multiplications, 128 additions and 64 comparisons for fault tolerance computing. The original quantization requires 64 additions, 64 divisions, and 64 truncations. A truncation is equivalent to a shift operator, whose overhead is about the same as that of an addition. The percentage overhead for the fault tolerance of the quantization stage is around 52% when divisions are considered twice as costly as multiplications.

3) For Run-Length Coding: The number of overhead operations is variable, depending on the number of nonzero coefficients, N_Z, where $N_Z \leq 64$, and the locations of these nonzero coefficients in the zig-zag sequence. The number of additions inside the loops (Fig. 16) does not exceed $2N_Z$ (in the worse case, each loop needs 2 additions, one for the computation of $\tilde{I}_j$ and the other for the increment of j). The total number of comparisons in these loops is N_Z. The total number of additions in the computations of S_{in} and S_{out} is $2(N_Z - 1)$. Thus, the number of additions for fault detection does not exceed $(4N_Z - 2)$, while the total number of comparisons is exactly $(2N_Z + 1)$. Thus, the overall cost for the run-length coding stage does not exceed $(6N_Z - 1)$ additions. For the original run-length coding, one subtraction is required for computing the differential DC coefficient. The number of additions used to count the running zeros in a zig-zag sequence is variable but does not exceed 32. The total number of comparisons in detecting the last nonzero coefficient in a zig-zag sequence is 63. Thus, the overall cost for the original run-length coding is equivalent to 96 additions. The fault tolerance computation overhead of the run-length coding stage is $100 * (6N_Z - 1)/96\%$. For a typical $N_Z = 10$, this overhead is about 61%.

4) For Across the Boundary Between 2-D DCT and Quantization Stages: This boundary error detection only requires the computation of $\tilde{P}_{\text{out}}$, which costs 2×63 multiplications and 64 additions. One comparison is used to check between P_{out} and $\tilde{P}_{\text{out}}$. Since there were no original computations at the cross boundary between the 2-D DCT and the quantization stages, the fault tolerance computation overhead of the cross boundary is totally extra.

5) For the Output Code Stream Format: The computation overhead depends on the CRC computation method. In this paper, the CRC code for a data sequence is computed via table look-up as described in [25]. The software implementation for this computation method mainly uses XOR, AND and SHIFT logical operators, beside the use of a memory space to hold a code table. All of these type of logical operations are a fraction of the computational logic required for additions since they are integral parts of addition logic. The computation of CRC code for a data sequence of L bytes required $2L$ XORs, L SHIFTs and $2L$ ANDs. The cost for producing of the L bytes of compressed JPEG image stream depends on the contents of the image data input and the compression mode. However, the number of computations required organizing the compressed data output is far more than the computational overhead of the checking using CRC codes, both of which are relatively cheap with regard to additions. The overhead for the extra checking should be below 25%, by conservative measures.

V. Conclusion

The methods and analysis of the fault tolerance for the JPEG image compression systems were presented. The work concentrated on the algorithm level of the processing so that hardware and software implementations are both properly protected. Computer fault tolerance for discrete cosine transform (DCT), entropy coding, quantization and compressed format construction are all considered. A fault tolerance system was developed by dividing the JPEG compression systems into functional modules and modeling faults in each module by adding external values to items in the data paths. The error detection structures and algorithms were successfully developed and they not only detected subsystem errors but also protected errors introduced in the interface between subsystems.

References

[1] C. Constantinescu, "Trends and challenges in VLSI circuit reliability," *IEEE Micro*, vol. 23, no. 7, pp. 14–19, Jul. 2003.

[2] P. Shivakumar, M. Kister, S. W. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *Proc. Int. Conf. Dependable Systems and Networks*, Washington, DC, 2002, pp. 389–398.

[3] H. Cha, E. Rudnick, J. Patel, R. Iyer, and G. Choi, "A gate-level simulation environment for alpha-particle induced transient faults," *IEEE Trans. Comput.*, vol. 45, no. 11, pp. 1248–1256, Nov. 1996.

[4] P. E. Dodd *et al.*, "Single-event upset and snapback in silicon-in-isulator devices and integrated circuits," *IEEE Trans. Nucl. Sci.*, vol. 47, no. 2, pp. 2165–2174, Apr. 2000.

[5] T. Karnik, P. Hazucha, and J. Patel, "Characterization of soft errors caused by single event upsets in cmos processes," *IEEE Trans. Depend. Secure Comput.*, vol. 1, no. 2, pp. 128–143, Apr./Jun. 2004.

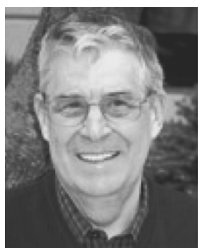
[6] M. N. Lovellette, K. S. Wood, D. L. Wood, J. H. Beall, P. P. Shrivani, N. Oh, and E. J. McCluskey, "Strategies for fault-tolerant, space-based computing: lessons learned from the argos testbed," in *Proc. IEEE Aerospace Conf.*, vol. 5, Mar. 2002, pp. 2109–2119.

[7] M. N. Lovellette, A. Campbell, K. Clark, K. S. Wood, D. L. Wood, A. Ross, J. H. Beall, and G. Clifford, "Implications of the different classes of exceptions experienced during the cots processor test flight on the argos satellite," in *Proc. IEEE Aerospace Conf.*, vol. 5, Mar. 2003, pp. 2481–2491.
[8] A. Messer, P. Bernadat, G. Fu, D. Chen, Z. Dimitrijevic, D. Lie, D. D. Mannaru, and D. Milojicic, "Susceptibility of commodity systems and software to memory soft errors," *IEEE Trans. Comput.*, vol. 53, no. 12, pp. 1557–1568, Dec. 2004.
[9] W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard*. New York: Thomson, 1993.
[10] M. Petsalis, M. Soleymani, and M. Swamy, "Effects of errors and error recovery in images compressed by the JPEG still image compression standard algorithm," in *Proc. Canad. Conf. Electrical Computer Engineering*, vol. 1, Sep. 1994, pp. 396–400.
[11] K. Rao, *Discrete Cosine Transform*. New York: Academic, 1990.
[12] J. Jou and A. Abraham, "Fault-tolerant FFT networks," *IEEE Trans. Comput.*, vol. 37, no. 5, pp. 548–561, May 1988.
[13] M. Tsunoyama and S. Naito, "A fault-tolerant fft processor," in *Proc. Dig. 21st Intl Symp. Fault-Tolerant Computing*, Jun. 1991, pp. 128–135.
[14] F. Lombardi and J. Muzio, "Concurrent error detection and fault location in an FFT architecture," *IEEE J. Solid-State Circuits*, vol. 27, no. 5, pp. 728–736, May 1992.
[15] C. Oh and H. Youn, "On concurrent error detection, location and correction of FFT networks," in *Proc. Dig. 23rd Int. Symp. Fault-Tolerant Computing*, vol. 23, 1993, pp. 596–605.
[16] G. Oh, H. Youn, and V. Raj, "An efficient algorithm-based concurrent error detection for FFT networks," *IEEE Trans. Comput.*, vol. 44, no. 9, pp. 1157–1162, Sep. 1995.
[17] G. Redinbo, "Concurrent error detection in fast unitary transform algorithms," in *Proc. Int. Conf. Dependable Systems and Networks*, Gottenburg, Sweden, Jun. 2001, pp. 37–46.
[18] S. Yu and E. E. J. Swartzlander, "DCT implementation with distributed arithmetic," *IEEE Trans. Comput.*, vol. 50, no. 9, pp. 985–991, Sep. 2001.
[19] S. Ghosh, S. Venigalla, and M. Bayoumi, "Design and implementation of a 2D-DCT architecture using coefficient distributed arithmetic," presented at the IEEE Computer Society Annu. Symp. VLSI, May 2005, pp. 162–166.
[20] H. C. Chen, T. S. Chang, and C. W. Jen, "A low power and memory efficient distributed arithmetic design and its DCT application," in *Proc. IEEE Asia-Pacific Conf. Circuits and Systems*, Dec. 2004, pp. 805–808.
[21] K. Gaedke, J. Franzen, and P. Pirsch, "A fault-tolerant DCT-architecture based on distributed arithmetic," in *Proc. IEEE Int. Symp. Circuits and Systems*, May 1993, pp. 1583–1586.
[22] P. Wintz, "Transform picture coding," *Proc. IEEE*, vol. 60, no. 7, pp. 809–820, Jul. 1972.
[23] J. Y. Jou and J. A. Abraham, "Fault-tolerant algorithms and architectures for real time signal processing," in *Proc. Int. Conf. Parallel Processing*, Aug. 1988, pp. 359–362.
[24] J. Nebus, "Parallel data compression for fault tolerance," *Comput. Design*, vol. 22, pp. 127–134, Apr. 1983.
[25] D. V. Sarwate, "Computation of cyclic redundancy checks via table-lookup," *Commun. ACM*, vol. 31, pp. 1008–1013, 1988.
[26] R. Nair, G. Ryan, and F. Farzaneh, "A symbol based algorithm for hardware implementation of cyclic redundancy check (CRC)," in *Proc. VHDL Int. Users' Forum*, Oct. 1997, pp. 19–22.
[27] T. Baicheva, S. Dodunekov, and P. Kazakov, "Undetected error probability performance of cyclic redundancy-check codes of 16-bit redundancy," *Proc. Inst. Elect. Eng. Commun.*, vol. 147, pp. 128–143, Oct. 2000.
[28] T. Ramabadran and S. Gaitonde, "Computation of cyclic redundancy checks via table look-up," *Micro. IEEE*, vol. 8, pp. 62–75, Aug. 1988.

Cung Nguyen (M'02) received the B.S. degree from the Department of Electrical Engineering, University of Missouri, Rolla (UM Rolla), in 1999, and the M.S. and Ph.D. degrees from the University of California, Davis (UC Davis), in 2002 and 2004, respectively.

In 1999, he was a Research Intern at the UM Rolla Engineering Research Laboratory, where he successfully developed a smart robot fire fighter model. From January 1997 to June 1998, he held a Computer Laboratory Assistant position at the Electrical Engineering Department, UM Rolla, from November 1995 to August 1999. He was a member of the mathematics tutoring staff at Penn Valley Community College, Kansas City, MO. He is a Graduate Research Fellow of the Graduate Assistant in the Area of National Needs (GAANN) fellowship at UC Davis.

Dr. Nguyen is a member Phi Kappa Phi honor society. He was the recipient of the 1995 United States Achievement Academy Award. His research interests include reliable distributed systems, computer networks, operating systems, and VLSI and fault-tolerance design strategies for signal- and image-coding systems.



G. Robert Redinbo (F'97) received the B.S., M.S., and Ph.D. degrees in electrical engineering from Purdue University, West Lafayette, IN.

He is a Professor of electrical and computer engineering at the University of California, Davis, where he is also a member of the Center for Image Processing and Integrated Computing. He has taught at Purdue University; the University of Wisconsin, Madison, where he served as a staff member of the Space Sciences and Engineering Center; and Rensselaer Polytechnic Institute, Troy, NY, where he was a founding member of the Center for Integrated Electronics. His industrial experience includes Martin Marietta, the MITRE Corporation, and IBM as a Visiting Scientist, Data Systems Division. His U.S. Government employment includes NASA, Defense Communications Agency, and the Army Security Agency. He has been a Consultant to Sandia National Laboratories. He took a one-year leave at the Reliability Laboratory, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, and recently completed a sabbatical at the Platform Components Division, Intel. His research interests encompass computer engineering, fault-tolerant computing, error control in digital designs, particularly communication and signal processing configurations, and real number codes.

Dr. Redinbo has been active in several IEEE organizations, including serving as a member of the Ad Com; Acoustics, Speech, and Signal Processing Society; and as an Associate Editor of the IEEE TRANSACTIONS ON COMPUTERS. He has chaired and served on a variety of program committees for IEEE-sponsored conferences and is a member of the Executive Committee, University of California MICRO Research Program. He is a Registered Professional Engineer.