SENRA Academic Publishers, British Columbia
Vol. 7, No. 3, pp. 2639-2647, October 2013
Online ISSN: 1920-3853; Print ISSN: 1715-9997

Canadian Journal of pure&applied sciences

Short Communication

AN-VE: AN IMPROVED HAMMING CODING TECHNIQUE

*Egwali Annie O and Akwukwuma V V N
Department of Computer Science, Faculty of Physical Sciences
University of Benin, PMB. 1154, Benin City, Nigeria

ABSTRACT

Many communication channels are subject to noise, and thus errors may be introduced during transmission. Error codes can be units of deliberate error injection or faults, which alters and cause hazards during communication, this makes error detection and correction important in the computing environment. Coding techniques restricted to detecting errors only are either limited to analyzing only the length of the encoded message bits or repeats every transmitted stream of bit(s) several times in order to check for correctness, which is not effective if similar error occurs within the same position on all clusters of bits in the encoded message. Error detecting and correcting techniques are more thorough by introducing extra redundant codes to detect the actual position of errors and correcting them, however if more than one error occurs, it becomes difficult to detect all errors and decode correctly. We therefore propose a hybrid error detecting and correcting technique, AN-VE, that simultaneously detect the existence of faulted codes right from the transmitter domain, analyzes all error positions in the encoded message via the use of extra parity bits, decode all errors correctly and verify error messages with the original message. We evaluate our approach using simulated and real data.

Keywords: Error, code, parity, hamming, parity.

INTRODUCTION

As present society relies on the fault-free operation of computing systems, system fault-tolerance has become a serious issue that needs addressing. Common agreement exists that large cluster of system codes always contain faults and thus precautions must be taken to avoid system failure. Failure of generated and transmitted codes often can be caused by external or internal factors that can or cannot be avoided, predicted, or corrected. Therefore, techniques are needed that guarantee correct data representation and transmission in the presence of errors (Kahn, 1996). To enable reliable delivery of digital data over reliable or unreliable communication channels, digital code redundancy techniques have been classified into two basic types: error detecting code technique and error detecting and correcting code technique. Error detecting technique is most commonly realized using a suitable hash function that adds a fixed-length tag to a message, and facilitates receivers to verify the delivered message by re-computing the tag and comparing it with the one provided. The enormous variety of dissimilar hash function designs is because of their simplicity or their suitability for detecting errors of different kinds. Error detecting and correcting code technique have the ability to detect, locate and correct errors. Consequently any error correcting code can be used for error detection.

*Corresponding author email: egwali.annie@yahoo.com

Coding techniques restricted to detecting errors only are either limited to analyzing just the length of the encoded message bits without specifying the actual bits with errors, or every transmitted stream of bit(s) is repeated several times in order to check for correctness, which does not prove effective if similar error occurs within the same position on all clusters of bits in the encoded message. In order for a system to deliver its expected service in the presence of errors caused by faults or units of deliberate error injection, some extra redundant codes are needed. Redundancy involves the inclusion of some extra codes in order to check the correctness or the consistencies of the results produce, and if the need arises, concurrent computations are chosen. Also the effects of faults can be masked with no specific indication of their occurrence, thus error effects are hidden from the rest of the system. In addition, faulty codes can be removed or replace in response to system failure, a process usually triggered either by internal error detection mechanisms in the faulty coded unit(s) of the software or by detection of errors in the output(s) of these units. These redundant codes are evident in some error detecting and correcting techniques, which makes them more thorough, because it helps to detect the actual position of the errors and correct the errors. Nevertheless, if more than one error occurs in an encoded stream message bit, error detecting and correcting techniques have difficulties in detecting all errors and decoding them correctly. Codes fault tolerance is very necessary, but can itself be dangerously error-prone because of the additional effort that must be

involved in the programming process. The additional redundancy may increase size and complexity and thus adversely affect information, software and by extension hardware reliability.

Fletcher (1982) developed a Checksum algorithm that involves detecting errors commonly introduced by humans in writing down or remembering identification numbers (Stallings, 2003). The checksum of a message is a modular arithmetic sum of a stream of message bits (SMB) of a fixed length, which could be negated by means of a one's-complement prior to transmission to detect errors resulting in all-zero messages (Fletcher, 1982). In repetition code technique, involves error detection, every transmitted stream of bit(s) is repeated several times in order to check for correctness (Filiol, 2003; Courtois, 2002). Unfortunately, repetition codes prove not to be effective if similar error occurs within the same position on all clusters of bits in the stream. Berger (1961) developed the Berger code which can detect all unidirectional errors, that is errors that only flip ones into zeroes or only zeroes into ones, such as in asymmetric channels. The check bits of Berger codes are computed by summing all the zeroes in the stream of message bits, and expressing that sum in natural binary. Berger codes can detect any number of one-to-zero bit-flip errors, as long as no zero-to-one errors occurred in the same stream of message bits. Berger codes can detect any number of zero-to-one bit-flip errors, as long as no one-to-zero bit-flip errors occur in the same SMB but cannot correct any error (Wiki, 2009).

Hamming (1969) posited the hamming error and correcting coding technique which are the earliest linear error correcting code technique. It involves the use of an extra parity bit to ensure the identification of a single error. However, if more than one error occurs, the Hamming Decoder block decodes incorrectly. Peterson (1960) proposed parity coding technique, which can only detect single errors and any odd number of errors. In this technique extra bits are added to the source bits so that the derived bits with value 1 in the set of bits are either even or odd (Peterson and Brown, 1961). When using even parity, the parity bit is set to 1 if the number of ones in a given set of bits (not including the parity bit) is odd, making the entire set of bits (including the parity bit) even. When using odd parity, the parity bit is set to 1 if the number of ones in a given set of bits (not including the parity bit) is even, keeping the entire set of bits (including the parity bit) odd. In other words, an even parity bit will be set to "1" if the number of 1's + 1 is even, and an odd parity bit will be set to "1" if the number of 1's +1 is odd (Wiki, 2010). This coding technique is applicable in data storage and retrieval from or into the computer memory. A shortfall with this technique is that for an odd flipped bit codes, an erroneous code with an odd flipped bit will be assumed to be correct. Also parity coding technique can only detect single errors and any odd number of errors. According to Wiki (2010a), parity does not indicate which bit contained the error, even when it can detect it. The data must be discarded entirely and re-transmitted from scratch. On a noisy transmission medium, a successful transmission could take a long time or may never occur.

Borden codes denoted as $B_{i/j}$ *are* a set of codes of length j for which exactly i bits are ones. The union of codes with i being the set of values congruent to $\lfloor j/2 \rfloor mod(k+1)$ is the Borden (j, k) code. For example, to derive the Borden (7, 3) code, by substituting values for j and k, we will have: $\lfloor 7/2 \rfloor mod(3+1) = 3$. Hence i belong to the set {0, 3, 6}. This means that source codes of length 7 (e.g. 0000000, 0011100, 0101010, 0111111, 1111011), which have no bits, three bits or six bits of digit 1 belongs to the Borden code set. A shortfall with this technique is that although the Borden (j, k) can detect k unidirectional errors (e.g. an erroneous conversion of 0 to 1 or 1 to 0), it cannot detect both erroneous conversion at the same time.

MATERIALS AND METHODS

A hybrid model called AN-VE that incorporates the unique features inherent in the cyclic redundancy checking (CRC) technique and the Hamming error detecting and correcting (HEDC) technique was proposed. AN-VE offers a more robust error detecting and correcting mechanism right from the transmitter domain, unlike repetition coding technique, which is not effective at detecting errors if similar error occurs within the same position on all clusters of bits in the encoded message bit stream. AN-VE is also more efficient than the CRC technique at detecting errors, which only analyzes the length of the encoded message bits against the initial message bits because it both verify the length of the encoded message and addresses all error positions via the use of extra parity bits. AN-VE also performs better that the HEDC technique, which is not able to detect all errors accurately if more than one error occurs in a decoded stream of message bits. AN-VE decodes and corrects all errors correctly and verifies decoded error messages with the original message. We evaluate our approach using simulated and real data.

AN-VE Error Detecting and Correcting Method

AN-VE (f, g) code like the hamming code consists of "g" data bits and the encoded data bits of length f. g is defined by the equation: $g = 2^a - a - 1$, *and* f is defined by the equation, $f = 2^a - 1$, were "a" denotes the parity bits such that $a \geq 3$. AN-VE is a technique that offers a more robust error detecting and correcting mechanism. Correcting and detecting error codes involves the following three phases.

Phase One: Creation of Message Bits

This phase executes the four steps process of the Hamming coding technique to create a stream of message bits. For example, for the following stream of message bits: 11010011101100

- *Step 1* and *Step 2* yields: _ _1_101-_0011101_100
- *Step 3* and *Step 4*: Using * to denote the parity bit position, the following results are derived for each position:
- Position 1 yields: * ***_1_1***0***1_0***011***0***1***_1***0***0***
  This is an even parity hence position 1 is set to 0: ***0_1_1***0***1_0***0***1***1***0***1***_1***0***0***
- Position 2 yields: 0****1***_1***01***_0***01***11***01***_1***00***
  This is an even parity hence position 2 is set to 0: 0***01***_1***01***_0***01***11***01***_1***00***
- Position 4 yields: 001****101***_001***1101***_100
  This is an odd parity hence position 4 is set to 1: 001***1101***_001***1101***_100
- Position 8 yields: 0011101****0011101***_100
  This is an even parity hence position 8 is set to 0: 0011101***00011101***_100
- Position 16 yields: 0011101000111101****100***
  This is an odd parity hence position 16 is set to 1: 00111010001110***1100***

Consequently, the expected created SMB is 0011101-000111011100.

Phase Two: Error Verification

After the initial phase, if the created SMB is suppose to be 0011101000111011100, but due to noise the message bits received is 00111***1***10001***0***10111***1***0, the system is able to detect this error because the second phase of AN-VE checks each inputted bit in the received SMB for accidental changes. During this stage, the system first check the length of the input code bits received for changes (see Tables 1 and 2) by lining input bits in a row, and a (n+1)-bit pattern that acts as a Cyclic redundancy

Table 1. AN-VE Error Detecting Method (created and received SMB are equivalent).

Created SMB:	0	0	1	1	1	0	1	0	0	0	1	1	1	0	1	1	1	0	0
Divisor	1	0	1	1															
Received SMB:	0	0	1	1	1	0	1	0	0	0	1	1	1	0	1	1	1	0	0
Divisor		1	0	1	1														
Received SMB:	0	0	1	1	1	0	1	0	0	0	1	1	1	0	1	1	1	0	0
Divisor			1	0	1	1													
Received SMB:	0	0	0	1	0	1	1	0	0	0	1	1	1	0	1	1	1	0	0
Divisor				1	0	1	1												
Received SMB:	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	0	0
Divisor					1	0	1	1											
Received SMB:	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	0	0
Divisor						1	0	1	1										
Received SMB:	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	0	0
Divisor							1	0	1	1									
Received SMB:	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	0	0
Divisor								1	0	1	1								
Received SMB:	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	0	0
Divisor									1	0	1	1							
Received SMB:	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	0	0
Divisor											1	0	1	1					
Received SMB:	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	0	0
Divisor												1	0	1	1				
Received SMB:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
Divisor													1	0	1	1			
Received SMB:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
Divisor														1	0	1	1		
Received SMB:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
Divisor															1	0	1	1	
Received SMB:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
Divisor																1	0	1	1
Result	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Table 2. AN-VE Error Detecting Method (Created and Received SMB are not equivalent).

Created SMB:	0	0	1	1	1	0	1	0	0	0	1	1	1	0	1	1	1	0	0
Received SMB:	0	0	1	1	1	*1*	1	0	0	0	1	*0*	1	0	1	1	1	*1*	0
Divisor	1	0	1	1															
Received SMB:	0	0	1	1	1	*1*	1	0	0	0	1	*0*	1	0	1	1	1	*1*	0
Divisor		1	0	1	1														
Received SMB:	0	0	1	1	1	1	1	0	0	0	1	0	1	0	1	1	1	1	0
Divisor			1	0	1	1													
Received SMB:	0	0	0	1	0	0	1	0	0	0	1	0	1	0	1	1	1	1	0
Divisor				1	0	1	1												
Received SMB:	0	0	0	0	0	1	0	0	0	0	1	0	1	0	1	1	1	1	0
Divisor					1	0	1	1											
Received SMB:	0	0	0	0	0	1	0	0	0	0	1	0	1	0	1	1	1	1	0
Divisor						1	0	1	1										
Received SMB:	0	0	0	0	0	0	0	1	1	0	1	0	1	0	1	1	1	1	0
Divisor							1	0	1	1									
Received SMB:	0	0	0	0	0	0	0	1	1	0	1	0	1	0	1	1	1	1	0
Divisor								1	0	1	1								
Received SMB:	0	0	0	0	0	0	0	0	1	1	0	0	1	0	1	1	1	1	0
Divisor									1	0	1	1							
Received SMB:	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	1	1	0
Divisor										1	0	1	1						
Received SMB:	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	1	1	0
Divisor											1	0	1	1					
Received SMB:	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0
Divisor												1	0	1	1				
Received SMB:	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0
Divisor													1	0	1	1			
Received SMB:	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0
Divisor														1	0	1	1		
Received SMB:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
Divisor															1	0	1	1	
Received SMB:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
Divisor																1	0	1	1
Result	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

check divisor is positioned underneath the left-hand end of the row. If the input bit above the leftmost divisor bit is 0, the bit is left and the divisor is moved to the right by one bit. If the input bit above the leftmost divisor bit is 1, the divisor is XORed into the input. The divisor is then shifted one bit to the right, and the process is repeated until the divisor reaches the right-hand end of the input row. Since the leftmost divisor bit zeroed every input bit it touched, when this process ends the only bits in the input row that can be nonzero are the n bits at the right-hand end of the row, which will always be less than the divisor.

Phase Three: Error Correction

After checking for changes, if the created SMB does not match the received SMB (i.e. the system reads "0000000000000001000" gotten from the received SMB "00111***1***10001***0***1011***1***0" instead of "0000000000000000111" from the created SMB "0011101000111011100"), AN-VE established that the received code block contains data error and take corrective measures to detect the actual bit locations containing the errors. The affected check bits positions are established, which are positions 2 check bit and 4 check bit for error bit at position 6 of received SMB;

Table 3. AN-VE Corrective Mechanism.

Created SMB:	0	0	1	1	1	0	1	0	0	0	1	1	1	0	1	1	1	0	0
Received SMB:	0	0	1	1	1	*1*	1	0	0	0	1	*0*	1	0	1	1	1	*1*	0
AN-VE Parsing:	_	_	_	_	_	_	_	_	_	_	_	_	_	_	_	_	_	*	0
HEDC Correction:	_	_	_	_	_	_	_	_	_	_	_	_	_	_	_	_	_	0	0
Created SMB:	0	0	1	1	1	0	1	0	0	0	1	1	1	0	1	1	1	0	0
Received SMB:	0	0	1	1	1	*1*	1	0	0	0	1	*0*	1	0	1	1	1	*0*	0
AN-VE Parsing:	_	_	_	_	_	_	_	_	_	_	_	*	1	0	1	1	1	0	0
HEDC Correction:	_	_	_	_	_	_	_	_	_	_	_	1	1	0	1	1	1	0	0
Created SMB:	0	0	1	1	1	0	1	0	0	0	1	1	1	0	1	1	1	0	0
Received SMB:	0	0	1	1	1	*1*	1	0	0	0	1	*1*	1	0	1	1	1	*0*	0
AN-VE Parsing:	_	_	_	_	_	*	1	0	0	0	1	1	1	0	1	1	1	0	0
HEDC Correction:	_	_	_	_	_	0	1	0	0	0	1	1	1	0	1	1	1	0	0
Created SMB:	0	0	1	1	1	0	1	0	0	0	1	1	1	0	1	1	1	0	0
Received SMB:	0	0	1	1	1	*0*	1	0	0	0	1	*1*	1	0	1	1	1	*0*	0

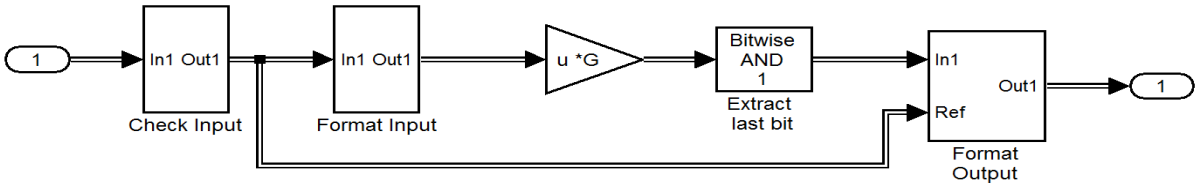

Fig. 1. Hamming Encoder.

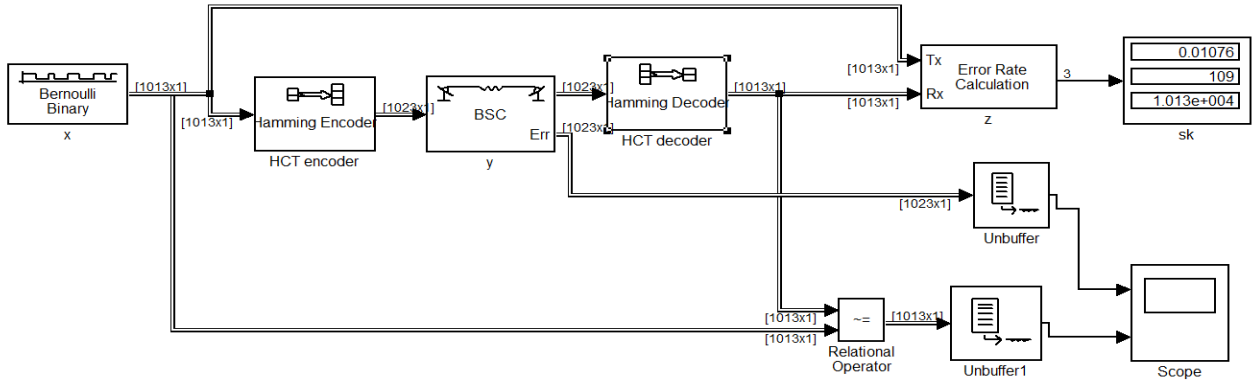

Fig. 2. Simulink of the HEDC Communication Components.

positions 4 check bit and 8 check bit for error bit at position 12 of received SMB; positions 2 check bit and 16 check bit for error bit at position 16 of received SMB. To effectively correct all errors, unlike the conventional hamming technique that can only handle one error correction in a SMB, AN-VE devices a parsing procedure that parses each n-bit binary position on the received SMB which are lined in a row and compares it with the n-bit binary position of the created SMB which are lined in a row starting from the extreme left. The parity check bits of the first bit at variance between the two sets of n-bit binary position in a row are verified to detect the error, which is then, corrected using the hamming error correcting and at each parsing stage, the system only acknowledges the positions of the other bits in the row and not their values. The system repeats this process till all n-bit binary positions is parsed and both created and received codes are equivalent (see Table 3).

EXPERIMENT

To demonstrate the efficiency of AN-VE over HEDC in detecting and correcting errors during communication, we analytically simulate the performance of the techniques in the presence of error in form of noise in a channel of communication using simulink that runs in Matlab. Generally in every communication system the basic components of communication are the original data bits,

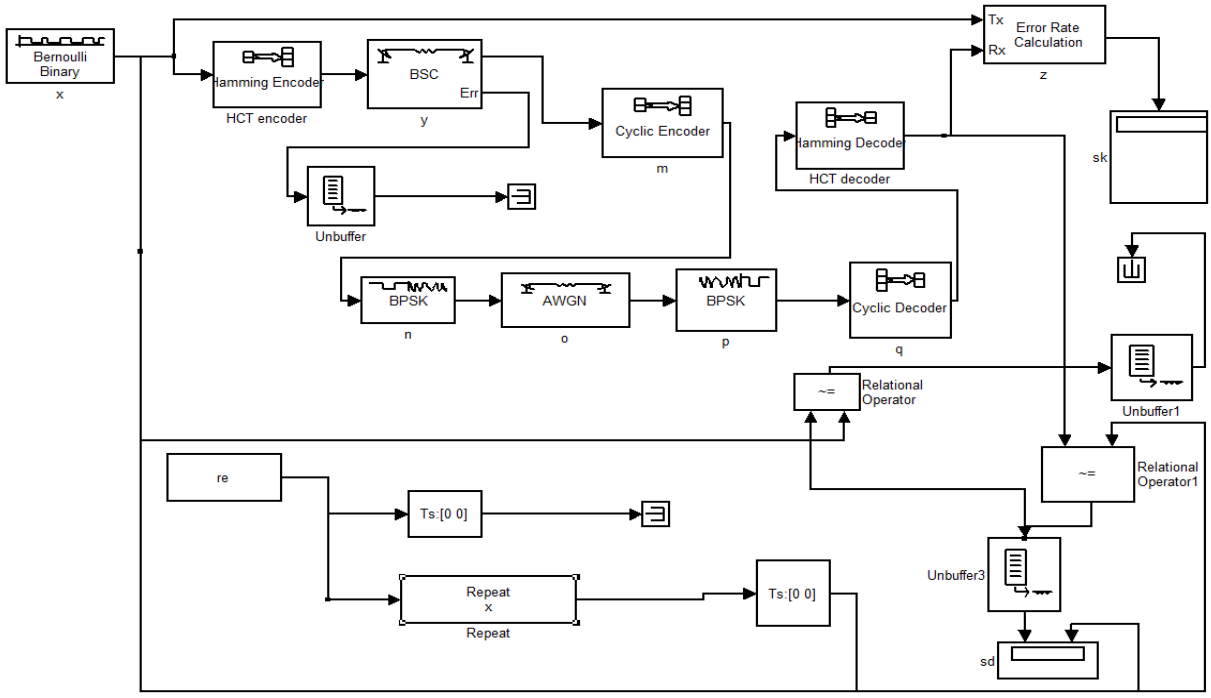

Fig. 3. Simulink of the AN-VE Communication Components.

Fig. 4. Readings of Error Data in HEDC after Computation.

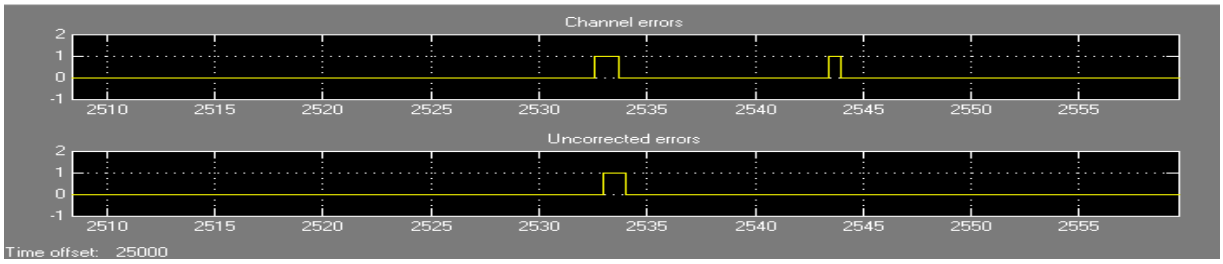

Fig. 5. Uncorrected and Corrected Errors of HEDC.

which consists of vectors of size *g*, the communication source symbolized as *x*, the channel of communication represented as *y*, the output sink denoted as *sk* and the decoded data bits indicated as *b*. To execute an operation, the original data bits are inputted via the communication source through the channel, and finally the decoded data bits is displayed at the output sink.

HEDC:

The Bernoulli Binary Generator which generates a random binary sequence is the source *x* for the signal in this model. Next the Hamming encoder encodes the original data bits *g* before it is sent through the channel (see Fig. 1). The Binary Symmetric Channel (BSC) simulates a channel with noise. The channel generates a random binary signal, and then switches the symbols 0

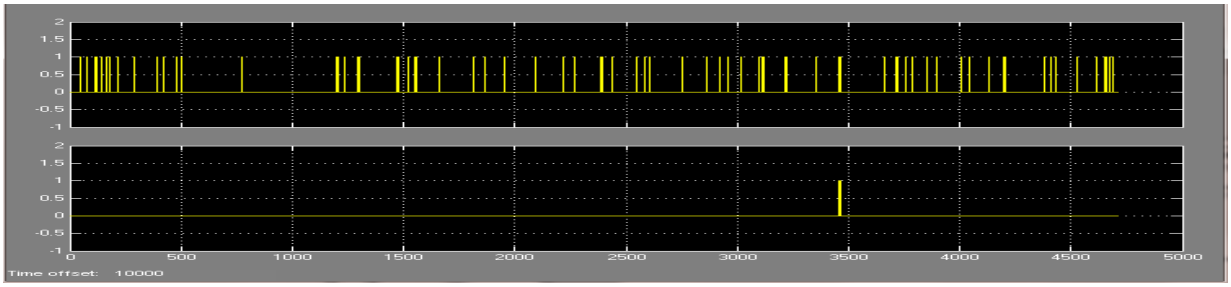

Fig. 6. Readings of Error Data (AN-VE) After Computation.

Table 4. Computational Readings.

Transmitted bits	Encoded data bits length (f)	Data bits (g)	Parity bits (s)	Bits error rate (err)		Number of errors	
				HEDC	AN-VE	HEDC	AN-VE
9.474e+004	7	4	3	0.001066	0.0057	101	54
5.007e+004	15	11	4	0.001997	0.0002397	100	12
2.415e+004	31	26	5	0.00414	0.000207	100	5
1.664e+004	63	57	6	0.006068	0.000201	101	4
1.02e+004	127	120	7	0.0101	0.00103	103	2
8151	255	247	8	0.01227	0.00114	100	9
9036	511	502	9	0.01162	0.001771	105	16
1.013e+004	1023	1013	10	0.01076	0.001382	109	14
1.018e+004	2047	2036	11	0.01041	0.001081	106	11

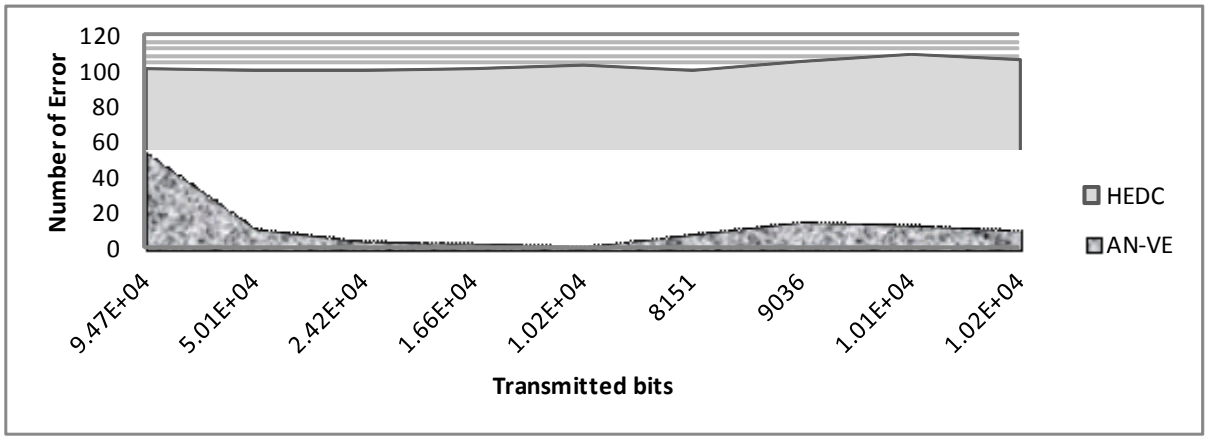

Fig. 7. Available Errors in Transmitted Bits after Applying HEDC and AN-VE Techniques.

and 1 or the reverse in the signal, according to a specified error probability, to simulate a channel with noise.

The Hamming decoder decodes the data after it is sent through the channel. It verifies if an error is created in the original data bits by the noise in the channel, identifies the error and decodes the data received to the original data bits correctly. The bits error rate represented as *err* is computed at z, which in this case detects and computes the error rate of the channel using the values of its two input ports, the transmitted signal *Tx* and the received signal *Rx*. The computation is based on the following equation (1), which denotes the probability of two or more errors occurring in encoded data bits of length *f*.

$$err = 1 - (0.99)^{f} - f(0.99)^{f-1}(0.01) \quad (1)$$

The bits error rate result is displayed in the first box of *sk* after the termination of execution. The second box of *sk* displays the number of errors occurrences. The third box of *sk* displays the total number of bits (b_n) transmitted. The logical inference and relationship between the components are represented in figure 2, which contains a sample of the readings of HEDC [(1023, 1013)].

AN-VE:

For AN-VE, the source *x* is the Bernoulli Binary Generator. Next the Hamming encoder encodes the original stream of message bits *g* before it is sent through the Binary Symmetric Channel (BSC) that adds binary errors to the input signal (see Fig. 3). The cyclic encoder creates the systematic cyclic bits with message length K and encoded SMB length N in the binary cyclic encoder. The code rate is:

$$\text{Code rate} = g_e/f_e = \text{message bits length/ encoded SMB length}$$

where g_e is the message length and f_e is the length of the derived encoded data bits and "a" denotes the check bits such that $g_e = 2^a - a - 1$, $f_e = 2^a - 1$, $a \geq 3$. Then the input SMB is modulated using the binary phase shift keying (BPSK) method, which is a technique for modulating a binary signal onto a complex waveform by shifting the phase of the complex signal. In digital baseband BPSK, the symbols 0 and 1 are modulated to the complex numbers exp(t) and -exp(t), respectively, where t is a fixed angle. Thus for t = 0, these numbers are just 1 and -1. The AWGN channel add white Gaussian noise to the input codes and it is more robust than the binary symmetric channel in some specific applications because it accepts both real or complex codes and it supports multichannel input and output codes inputs as well as frame-based processing. The Hamming decoder parses through the n-bit binary position on the received SMB bits and decodes the data after it is sent through the channel. It verifies if an error is created in the original data bits by the noise in the channel, identifies the error and decodes the data received to the original data bits correctly. The system repeats this process till all n-bit binary positions is parsed and both transmitted and received codes are equivalent. After each *AN-VE* Parsing, the bits error rate (*err*) of the created SMB and received SMB is computed at z*,* which detects and computes the error rate of the channel using the values of the transmitted signal *Tx* and the received signal *Rx* ports. The block compares the two signals and checks for errors. The output depicted as *sk* is a vector with three entries: bit error rate, number of errors and total number of bits transmitted.

RESULTS AND DISCUSSION

The readings of the upper section of figure 4 reveal the channel of uncorrected errors in an encoded SMB generated by the channel coding of HEDC, which are pulses represented as 1s. The readings of the lower section of figure 4 shows the same encoded SMB but with fewer errors at the end of each 5000 time steps.

A proximity assessment of upper section of figure 4 shows that in the single encoded SMB, two 1s represented as the wider rectangular pulse, are errors not corrected, whereas the narrower rectangular pulse to the right of the upper section represents a single error, which is corrected (see Fig. 5). The two 1s, which are uncorrected, accounts for the uncorrected errors in the lower section.

Figure 6 shows the performance of AN-VE with the upper and lower sections of the figure revealing the channel of uncorrected and corrected errors respectively. It is evident that with the same number of bits transmitted (see Table 4), AN-VE performed better, for the two 1s which are uncorrected, are far less than is the case with HEDC, this is also shown in figure 7, which reveals the number errors left over in the transmitted bits after the application of both techniques.

CONCLUSION

Fault tolerance solutions can be implemented using various methodologies. Each method has its own tradeoffs in terms of strength in detecting errors, potency in correcting errors, portability, and ease of use. We proposed a detecting and correcting technique that is more robust than HEDC because although HEDC is quite useful in cases where only a single error is of significant probability, the technique can only be used to detect up to two simultaneous bit errors and correct single errors, both cannot be done simultaneously. Also because the technique can only correct one error in each transmitted stream of message bits, if more than one error occurs, the Hamming decoder does carry the hazard of miscorrecting double errors. Conversely, AN-VE enables the detection of any number of simultaneous bit errors and corrects all errors and both can be done simultaneously. Furthermore unlike the HEDC which deals with error detection after data transmission, AN-VE addresses error detection right from the transmitter domain. AN-VE can detect the erroneous bit in a transmitted code because every transmitted code is repeated several times in order to verify its accuracy.

REFERENCES

Berger, JM. 1961. A note on an error detection code for asymmetric channels. Information and Control. 4: 68-73.

Courtois, NT. 2002. Cryptanalysis of block ciphers with overdefined systems of equations. Advances in Cryptology – AsiaCrypt 2002, Lecture Notes in Computer Science 2501, Springer-Verlag. 267-287, doi:10.1007/3-540-36178-2_17.

Filiol, E. 2003. Plaintext-dependent Repetition Codes Cryptanalysis of Block Ciphers - the AES Case, Published on eprint on 8th of January 2003. Available at: http://eprint.iacr.org/2003/003

Fletcher, JG. 1982. An Arithmetic Checksum for Serial Transmissions. IEEE Trans. on Comm. 30(1): 247-252.

Hamming, RW. 1980. The Unreasonable Effectiveness of Mathematics. The American Mathematical Monthly. 87:81-90.

Hamming, RW. 1969. One Man's View of Computer Science. Journal of the ACM. 16 (1):3-12.

Kahn, D. 1996. The Codebreakers: The Story of Secret Writing, New York: Macmillan Publishing Co., 1967. Available at: http://www.cse.iitk.ac.in/users/anuag/crypto.pdf.

Nicolas, T. 2003. Did Filiol Break AES?, Published on e-print on 4th of February 2003. Available at: http://eprint.iacr.org/2003/022.

Peterson, WW. 1960. Encoding and Error-Correction Procedures for the Bose-Chaudhuri Codes. IRE Transactions on Information Theory. IT-6:459-470.

Peterson, WW. and Brown, DT. 1961. Cyclic Codes for Error Detection. Proceedings of the IRE. 49:228. doi:10.1109/JRPROC.1961.287814.

Stallings, W. 2003. The TCP/IP Checksum. Available at: http://tchandouts.com/07315/Checksum.pdf

Wiki 2010. Parity Bit. Available at: http://en.wikipedia.org/wiki/Parity_bit.

Wiki 2010[a]. Hamming Code. Available at: http://en.wikipedia.org/wiki/Hamming_code.

Received: March 20, 2013; Accepted: Aug 12, 2013