

Low Power and Area Efficient Integer Correcting Codes

Dr.P.Pattunnarajam

Associate Professor of ECE, Prince Shri Venkateswara Padmavathy Engineering College

Abstract : To rectify burst faults of two consecutive bit locations inside a byte, look-up table-based error detection and correction integer codes are provided in this study. The convenience of encoding and decoding methods, as well as the fact that a look-up table-based error control technique takes very minimal memory, are two of its key advantages.

Keywords— Joint Picture Expert Group (JPEG), Matrix Laboratory (MATLAB), and Spatial Co-ordinate Design are all acronyms for Discrete Cosine Transformation (DCT) (SCD).

1. INTRODUCTION

The main concept underlying error detection and repair is to add redundancy (i.e., additional data) to a message so that receivers may verify for consistency and recover damaged data. Using a proper hash function is the most popular way for identifying problems (or checksum algorithm). A hash function adds a fixed-length tag to a message, enabling receivers to check for validity by recalculating the tag and comparing it to the one given.

Unlike binary-oriented codes, which have been researched intensively for over 60 years, codes over the ring Z_m of integers modulo m have received less attention. One of the key reasons for this is because integer codes have weak error control capabilities [1], which means that although their capacity to correct single mistakes of certain kinds may be beneficial in some situations, it is not suitable for error control coding. As a consequence, research on the application of integer codes in fields like frame synchronization [1], coded modulation [1], peak shift correction [2], and so on abound in the literature, but studies on forwarding error correction [1] are few.

Given this condition, the purpose of this work is to show how the idea of integer codes may be enhanced, i.e., how codes can be built to correct any burst of length l within a b -bit byte ($1 \leq b$). Furthermore, the given codes, known as integer (BTEC) b codes, have very simple encoding/decoding processes, and a look-up table (LUT)-based error control mechanism uses very little memory. They might be useful in several situations, especially when the network nodes are computers or other processor-based devices.

2. LITERATURE REVIEW

The main concept underlying error detection and repair is to add redundancy (i.e., additional data) to a message so that receivers may verify for consistency

and recover damaged data. Using a proper hash function is the most popular way for identifying problems (or checksum algorithm). A hash function adds a fixed-length tag to a message, enabling receivers to check for validity by recalculating the tag and comparing it to the one given.

Unlike binary oriented codes, which have been researched intensively for over 60 years, codes over the ring Z_m of integers modulo m have received less attention. One of the key reasons for this is because integer codes have weak error control capabilities [1], which means that although their capacity to correct single mistakes of certain kinds may be beneficial in some situations, it is not suitable for error control coding. As a consequence, research on the application of integer codes in fields like frame synchronization [1], coded modulation [1], peak shift correction [2], and so on abound in the literature, but studies on forward error correction [1] are few.

Given this condition, the purpose of this work is to show how the idea of integer codes may be enhanced, i.e., how codes can be built to correct any burst of length l within a b -bit byte ($1 \leq b$). Furthermore, the given codes, known as integer (BIEC) b codes, have very simple encoding/decoding processes, and a look-up table (LUT)-based error control mechanism uses very little memory. They might be useful in a number of situations, especially when the network nodes are computers or other processor-based devices.

$$M = \{1, a, \dots, ar, b, \dots, bs\}$$

and

$$M = \{\pm 1, \pm a, \dots, \pm ar, \pm b, \dots, \pm bs\}$$

This yields new conditions on the existence of perfect 3- and 4-shift codes. Further, it can be shown that splittings of Z_p by $(\pm 1, \pm 2, \pm 3)$ exist exactly if Z_p is also split by $\{1, 2, 3\}$.

III. ENCODING AND DECODING

Let b be the number of bits in a single byte, l be the number of correct burst errors, and c_1, c_2 be the

constants. The check byte must be calculated according to the formula [1], $CB = [C1.B1 + C2.B2 + \dots + Ck.Bk] \pmod{2b-1}$

The check byte, along with the information byte, is sent to the receiver. Various types of adders and multipliers are used to calculate the check byte.

The check byte must be calculated using the formula [1] with the received codeword.

$$= [C1.1 + C2.2] \pmod{255}$$

Then the syndrome is calculated using

$$S = [-B] \pmod{2b-1}.$$

If the calculated syndrome is '0,' there is no error and the code word received is correct. If the syndrome is not '0,' an error has occurred in one of the bytes. Using the look up table, we can then determine which byte contains the error.

After completing this task successfully, the decoder will have information about the error location and weight, allowing it to perform the error correction procedure using [1].

For errors within a data byte, $B_i = [i+] \pmod{2b-1}$.

For errors within a check-byte, $CB = [+]$ (mod 2b-1).

A. Error Correction Procedure Based on Lookup Tables [1]

TABLE I

t	Set of error weights
1	$e_{b,1} \in \{\pm 2^r.(1): r=0,1,\dots,b-1\}$
2	$e_{b,2} \in \{\pm 2^r.(1,3): r=0,1,\dots,b-2\}$
3	$e_{b,3} \in \{\pm 2^r.(1,3,5,7): r=0,1,\dots,b-3\}$
.	.
.	.
.	.
1	$e_{b,l} \in \{\pm 2^r.(1,3,5,\dots,2^{l-1}): r=0,1,\dots,b-l\}$

After calculating S, the decoder will execute an error control procedure to determine if the input codeword is correct ($S=0$) or not ($S \neq 0$). In this circumstance, a precomputed LUT is required, as shown in tables II and III, where the entries are sets of triples. Based on this, the decoder can quickly determine whether a nonzero value of S indicates an uncorrectable error or one that can be corrected using the equations $B_i = [i+] \pmod{2b-1}$ for errors inside a data byte and $CB = [+]$ (mod 2b-1) for errors within a check-byte.

Although this error-control approach looks to be simple, the number of search-and-compare operations determines its efficiency (in terms of processing time). Since a consequence, sorting the components in ascending order is extremely desired, as it enables the

employment of quick searching methods like binary search.

TABLE II

	1	2	3	4	6	8	12	16	24	32	48	63	64	96	127	Third Byte in Error
	128	159	191	192	207	223	231	239	243	247	249	251	252	253	254	
	254	253	252	251	32	249	247	64	63	207	243	24	96	239	128	First byte in error
	127	16	159	231	12	48	192	191	8	6	223	4	3	2	1	
	254	253	252	223	251	24	249	192	191	247	48	159	243	16	128	The second byte in error
	127	239	12	96	207	8	64	63	6	231	4	32	3	2	1	

TABLE III

	1	2	3	4	6	8	12	16	24	32	48	63	64	96	127	Third Byte in Error
	128	159	191	192	207	223	231	239	243	247	249	251	252	253	254	
	7	14	21	28	31	42	56	62	69	81	84	87	93	112	124	First byte in error
	13	14	16	16	17	17	18	19	19	21	22	22	23	24	24	
	1	3	2	8	1	4	6	3	9	3	4	7	4	1	8	
	9	18	27	33	36	39	54	57	66	72	78	99	10	11	13	The second byte in error
	13	14	14	15	17	18	18	19	20	21	21	22	22	23	24	
	2	4	7	6	7	3	9	8	1	6	9	2	8	7	6	

3. EXAMPLE OF INTEGER CODE CORRECTION CONCEPT

A. Reversible Burst An error occurred within a data byte.

Let $b = 8, l = 2, C1 = 7, C2 = 9$, and suppose we want to send 16 bits of data, $D = 10101100\ 11000110$. $CB = [C1.B1 + C2.B2] \pmod{255} = 181 = 101101012$ after calculating the value of check-byte CB $CB = [C1.B1 + C2.B2] \pmod{255} = 181 = 101101012$

The codeword $CW = 10101100\ 11000110\ 10110101$ will be 24 bits in length. In light of this, consider the following scenarios.

Case 1 (correctable burst error within a data byte): Assume an error on the 13th and 14th bit occurred during data transmission ($CW = 10101100$

11001010 10110101). In that case, the decoder will perform the calculation. $= [C1. 1 + C2. 2] \pmod{255} = 217$

$$S = [-] \pmod{255} = 36,$$

to determine whether the value $S = 36$ is part of the set. After completing this task successfully, the decoder will obtain information about the error location and weight, allowing it to perform the error correction procedure, i.e., $B2 = [2+] \pmod{255} = [202+251] \pmod{255} = 110001102$

B. Burst Error within a Check Byte that can be Corrected

After calculating S , the decoder will do an error check to see whether the input codeword is correct ($S=0$) or not ($S \neq 0$). In this case, as shown in tables II and III, a precomputed LUT with sets of triples as entries is required. Based on this, the decoder can readily determine whether a nonzero value of S indicates an uncorrectable error or one that can be corrected using the equations $B_i = [+] \pmod{2b-1}$ for errors inside a data byte and $CB = [+] \pmod{2b-1}$ for errors within a check-byte.

$$[+] \pmod{255} = [185+251] \pmod{255} = 101101012.$$

4. EXPERIMENTAL RESULT

A. Outcome and Discussion

1. Coding: (for 16-bit data)

Figure 1 depicts the encoding simulation using ModelSim for the data given below.

$C1=7$ $C2=9$ $B1=10101100$ $B2=11000110$ $C1=7$ $C2=9$

The check byte value received is 10110101=(181)10.

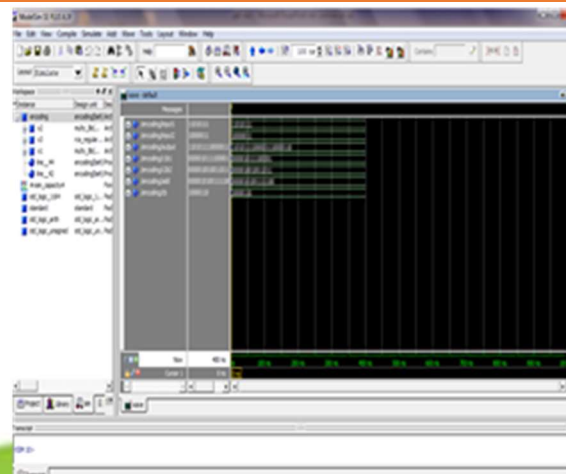


Fig 1 Encoding output waveform (16 bit)

2. Decoding (for 16-bit data)

Figure 2 depicts the decoding simulation using ModelSim for the data given below.

Received $C_b = 181$ Syndrome = 0 (without error)

By introducing error in 13th and 14th bit

$B1 = 10101100$ $B2 = 11001010$ $C1=7$ $C2=9$

Received $C_b = 217$ Syndrome = 36 (with error)

LUT is used to correct the output.

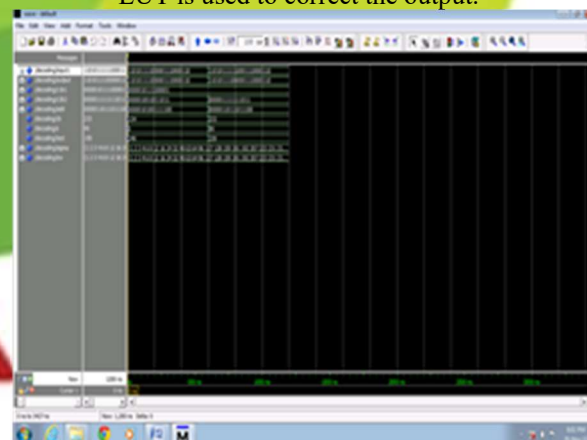


Fig 2 Decoding output waveform

3. Encoding: (for 32-bit data)

Encoding is performed for each 16 bit of data, and a check byte is calculated for each 16 bit.

Figure 3 depicts the encoding simulation using ModelSim for the data given below.

B1= 11010111 B2= 10000011 C1=7 C2=9
 B3= 11010111 B4= 10000011
 The received check byte value is
 10000111=(134)₁₀
 The received check byte value is
 10000111=(134)₁₀

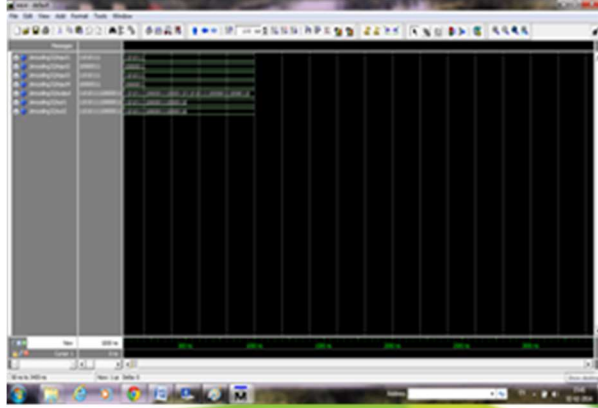


Fig 3 Encoding output waveform (32 bit)

4. Decoding (for 32-bit data)

Figure 4 depicts the decoding simulation using ModelSim for the data given below.

Cb1 received = 134, Syndrome = 0 (without a mistake)
 Cb2 = 134 Syndrome = 0 (without error)
 By introducing error in 1st and 2nd bit
 B1= 00010111 B2= 10000011 C1=7 C2=9
 B3= 11010111 B4= 10000011
 Received Cb1 = 65 Syndrome=186 (with error)

LUT will be used to correct this error.

Cb2=134 Syndrome = 0 (without error)

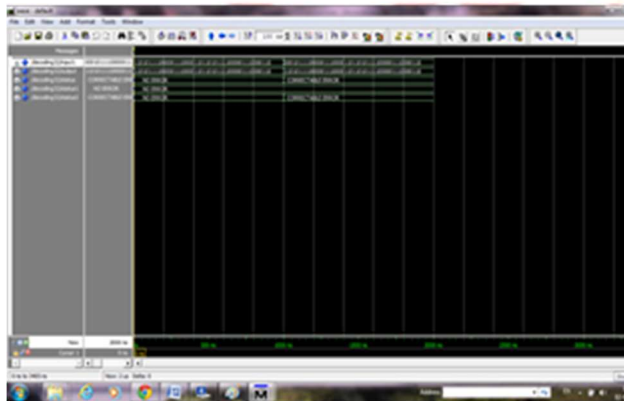


Fig 4 Decoding output waveform (32 bit)

TABLE IV
 COMPARISON WITH OTHER ADDER AND
 MULTIPLIER SETUPS

Parameter	CS adder& Array Multiplier	CS adder& Wallace tree multiplier	RCA &array multiplier	RCA & Wallace tree multiplier
POWER	30 mW	29mW	29mW	29mW
AREA	9358 gates	9570 gates	9186 gates	9222 gates
DELAY	31.235ns	31.235ns	31.260ns	31.260ns

To correct a burst error of two consecutive bit locations inside a byte, this study presents look-up table-based error detection and repair integer codes. These integer codes would be ideal software candidates. These codes offer several benefits: they are systematic, employ basic arithmetic to encode/decode data bits, require moderate-size LUTs for error control operations, and are ideally suited for implementation on general-purpose processors. When network nodes include computers or other processor-based devices, these codes are valuable.

5. CONCLUSION

This study provides look-up table-based error detection and correction integer codes to repair a burst error of two consecutive bit locations inside a byte. These integer codes are ideal candidates for use in software. These codes offer many benefits, including that they are systematic, employ basic arithmetic to encode/decode data bits, require moderate-size LUTs to execute error control processes, and are ideally suited for general-purpose usage processors. When network nodes include computers or other processor-based devices, these codes are valuable.

REFERENCES

- [1] V.I. Levenshtein and A.J.H. Vinck, "Perfect (d, k)-Codes Capable of Correcting Single Peak Shifts," IEEE Transactions on Information Theory, vol. 39, no. 2, Mar. 1993, pp. 656-662.
- [2] U. Tamm, "Splittings of Cyclic Groups and Perfect Shift Codes," IEEE Transactions on Information Theory, vol. 44, no. 5, Sept. 1998, pp. 2003-2009.
- [3] G. Umanesan and E. Fujiwara, "Parallel Decoding Cyclic Burst Error-Correcting Codes," IEEE Transactions on Computers, vol. 54, no. 1, Jan. 2004, pp. 87-92.