A 3-Tier, Novel, Efficient and Secure Method for Transmission of Decimal Numbers

Dr. Altaf Mukati*

*Professor & Dean, Faculty of Engg. Sciences, Bahria University, 13 National Stadium Road Karachi, Pakistan
Email: altafmukati@gmail.com, altaf.mukati@bimcs.edu.pk

Abstract: The decimal numbers being used in banking transactions, PIN codes and in several other applications have to be secured and efficient while being transferred over spatial or temporal channels. The present paper is limited to decimal data, i.e. BCD (Binary Coded Decimal) numbers only, which can be expanded to include all types of data as well. In the proposed scheme, the BCD numbers are first encoded (1st tier). The 4-bit outputs of an Encoder are taken as four minterm, which is then minimized through a Logic Minimizer (2nd tier). The encoder is designed to produce the terms, which when minimized, contain a single PI (Prime Implicant). Subsequently Huffman Coding or Shannon-Fano Coding is applied for Compression purposes (3rd tier). On the receiving or retrieving side, parsing of data is carried out to recover PIs which are then expanded to get 4-bit data block. Finally, these 4-bit blocks are decoded to get BCD numbers.
[Altaf Mukati. **A 3-Tier, Novel, Efficient and Secure Method for Transmission of Decimal Numbers.** *J Am Sci* 2013;9(9):98-101]. (ISSN: 1545-1003). http://www.jofamericanscience.org. 14

Keywords: BCD; Logic minimizer; Huffman coding; Data compression

1. Introduction:

Data compression has revolutionized information technology and communication applications. One of the main outcomes of this revolution is the ever growing internet and the fast development of mobile and multimedia communications [1].

Data compression has an impact on multimedia applications. It will not be practical to put images, audio or video on websites without the use of data compression algorithms [2,3].

Data compression is popular for two reasons: (a) People like to accumulate data and hate to throw anything away. No matter how big a storage device one has, sooner or later it is going to overflow. Data compression seems useful because it delays this inevitability (b) People hate to wait a long time for data transfers. When sitting at the computer, waiting for a web page to come in or for a file to download, we naturally feel that any thing longer than a few seconds is a long time to wait [4]. The spread of computing has led to an explosion in the volume of data to be stored on hard disks and sent over the Internet. This growth has led to a need for "data compression", that is, the ability to reduce the amount of storage or Internet bandwidth required to handle the data [5].

"Compression ratio" is the key factor, which is the measure of the size of a compressed file to the original uncompressed file. For example, suppose a data file takes up 50 kilobytes (KB). Using data compression software, that file could be reduced in size to, say, 25 KB, making it easier to store on disk and faster to transmit over an Internet connection. In this specific case, the "compression ratio" is of 2:1 [3,6]

Data compression can be either "lossless" or "lossy". Lossless data compression is used when the data has to be uncompressed exactly as it was before compression. Text files are stored using lossless techniques, since losing a single character can be in the worst case make the text dangerously misleading. Archival storage of master sources for images, video data, and audio data generally needs to be lossless as well. However, there are strict limits to the amount of compression that can be obtained with lossless compression. Lossless compression ratios are generally in the range of 2:1 to 8:1[3,5,7].

Lossy compression, on other hand, works on the understanding that the data doesn't have to be stored perfectly. Much information can be simply thrown away from images, video and audio data when uncompressed; the data will still be of acceptable quality. Compression ratios can be an order of magnitude greater than those available from lossless methods.

For example, video is generally compressed using lossy compression, as viewing a reconstruction of a video sequence, the original is generally not important as long as the difference do not results in annoying artefacts [2].

The Huffman or Shannon-Fano coding, suggested in this paper, are lossless data compression techniques. The present scheme has been worked out for the transmission or storage of decimal numbers only, represented in the form of BCD numbers. Sometimes the data files only contain such numbers, which are of sensitive nature and need secure and efficient transfer through spatial or temporal channel.

2. The proposed method:

The whole scheme on the sending or storage side can be depicted as in Figure1.

2.1 Encoder:

Why do we require an "Encoder" (here more appropriately "code converter")? If we input BCD numbers directly to the "Logic Minimizer", then some terms or functions will comprise of two PIs. We want here all minimized functions to contain a single PI, as we want to use each PI as a symbol to be used at data compression stage. More PIs may result in more symbols which in turn will not be helpful to get high data compression ratios. Table 1 shows the BCD numbers and corresponding encoded words.

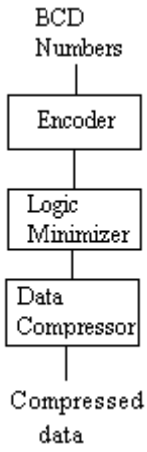

Figure 1. The proposed method of data compression

Table 1. Inputs and outputs of proposed encoder

No.	ABCD	WXYZ
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	1010
7	0111	1100
8	1000	1000
9	1001	1111

Note that the encoding is actually done for the BCD numbers 6,7 & 9 due to the reason that 0110, 0111 & 1001 produce two PIs when minimized. All the output words of the encoder will now produce single PI for each input. The present scheme requires even number of digits to input. If there are odd numbers, one may place four zeros in the beginning. This will be clear later in this paper when the scheme will be worked out through an example.

One the receiving or retrieving side, a decoder circuit will be used to do the opposite function. Both encoder and decoder circuits are shown in Figure 2 & Figure 3 respectively.

2.2 Logic minimizer

Not too much work has been done on logic minimization with respect to data compression earlier. Augustine and some others have done some good work [8,9]. This can be implemented in hardware or software depending on the requirement. We can group consecutive PIs into 2, 4, 8 or higher, to obtain better compression ratios. For the purpose of demonstration and understanding, we have grouped two consecutive PIs in this paper. Hence, the logic minimizer here will be basically a 3-input K-map solver, which takes eight bits input as eight minterms, i.e. A'B'C', A'B'C, A'BC', A'BC, AB'C', AB'C, ABC' & ABC. Each resultant term will be containing two PIs. "0000" and "1111" at the input of logic minimizer, are treated differently due to their property that four 0s or four 1s and 8 0s or 8 1s represent the same minimized functions. The adjacent four bits before/after are taken as four bit group representing four minterms A'B', A'B, AB', AB.

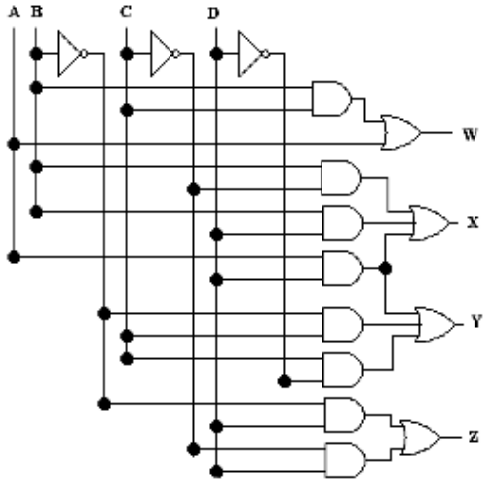

Figure 2. Design of proposed encoder

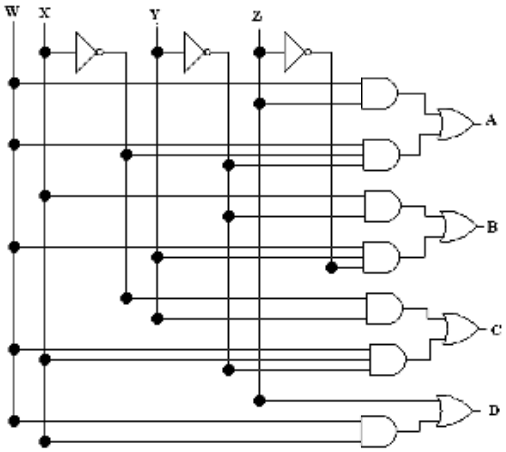

Figure 3. Design of proposed decoder

2.3 Data compressor

Any lossless data compressing scheme can be used. Huffman or Shannon-Fano schemes will be more appropriate here. The data compressor in the proposed method will get 8 bits in sequence, producing two PIs. In case of "0000" or "1111" input, the logic minimizer works differently. It will be clear in the example given in this paper.

A table is built simultaneously in the temporary storage as the minimization process is in progress. All PIs along with the frequency of occurrences is being updated in the table. When the entire data, which may range from few digits to several kilobytes, is completed then the table so prepared is used for data compression. As per Huffman or Shannon-Fano coding algorithms, a symbol which has high frequency of occurrences are assigned with shorter codes and the symbols with less occurrences, get larger codes.

3. The proposed scheme

3.1 Compression procedure

1. Input 4-bit BCD number into an specified encoder.
2. Make the group of bits into 2^n bits where n = 2,3,4...
3. Minimize the group using K-map solver and store PIs along with their frequency of occurrences in a temporary storage.
4. If "0000" or "1111" appears at the input of logic minimizer, then these are treated as single 4-bit group instead of group of 2^n bits.
5. Apply any lossless compression technique, like Huffman or Shannon-Fano coding.
6. Replace each PI with the corresponding code.
7. Send or store the compressed data along with frequency table.

3.2 Decompression procedure

1. Receive or retrieve the compressed data along with the frequency table.
2. Perform "parsing" to retrieve PIs.
3. Expand two consecutive PIs to get 8 bits. In case of "0" or "1" retrieval anytime in the compressed data, those have to be recovered as "0000" or "1111".
4. Start inputting 4-bits at a time to the decoder to get back the original BCD.
5. Repeat above to find the entire BCD data.

4. Demonstration of scheme

Let the small string of digits "37780050378957" is to be sent into spatial channel (transmission) or temporal channel (storage). The larger data file spread over several kilo bytes can be worked out in the similar manner through software implementation of data compression scheme.

The above digits in BCD form will be as follows:

0011011101111000000000001010000001101111000 1001010100111..... (1)

Data after encoding would be as follows:

0011110011001000000000001010000001111001000 111101011100

After logic minimizer (taking 8 bits), the PIs would be as follows:

A'B+AB' AB+BC 0+0 B'+0 A'B+AB' AB+1 A'B+AC'

The frequency table prepared in the temporary storage for data compression is shown in Table 2.

Table 2. PIs versus frequency of occurrences

Function	Frequency	Probability	Symbol
0	3	0.2145	E
A'B	3	0.2145	F
AB'	2	0.143	G
AB	2	0.143	H
BC	1	0.0715	I
B'	1	0.0715	J
1	1	0.0715	K
AC'	1	0.0715	L

The Huffman coding is applied by arranging symbols in the first column with respect to their frequency of occurrences as shown in Table 3.

Table 3. Huffman coding

E	0.2145	E	0.2145	E	0.2145	IJKL	0.286
F	0.2145	F	0.2145	F	0.2145	E	0.2145
G	0.143	G	0.143	G	0.143	F	0.2145
H	0.143	H	0.143	H	0.143	G	0.143
I	0.0715	KL	0.143	KL	0.143	H	0.143
J	0.0715	I	0.0715	IJ	0.143		
K	0.0715	J	0.0715				
L	0.0715						

IJKL	0.286	EF	0.429	GHIJKL	0.572
GH	0.286	IJKL	0.286	EF	0.429
E	0.2145	GH	0.286		
F	0.2145				

The Huffman tree can be constructed from table 3, is shown in Figure 4.

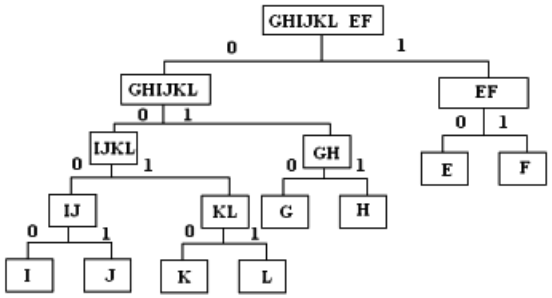

Figure 4. Huffman tree

The compression codes generated can be shown in the table 3. The compressed codes can be given as follows:

11010011000010100001101101001100101100011

Table 3. Huffman codes

Function	Symbol	Huffman Codes
0	E	10
A'B	F	11
AB'	G	010
AB	H	011
BC	I	0000
B'	J	0001
1	K	0010
AC'	L	0011

Total number of bits in "uncompressed" data = 56
Total number of bits in "compressed" data = 40
Compression Ratio = 40/56 = 1:0.71 or 29%

The decompression process starts by parsing the compressed data string. This can be shown in the table 4.

Table 4. Expanded functions

Parsed data	Symbols recovered	Expanded function
11010	A'B+AB'	A'BC+A'BC'+AB'C+AB'C'
0110000	AB+BC	ABC+ABC'+ABC+A'BC
1010	0,0	0000, 0000
000110	B',0	AB'+A'B', 0000
11010	A'B+AB'	A'BC+A'BC'+AB'C+AB'C'
0110010	AB,1	AB, 1111
110011	A'B+AC'	AB'C+AB'C'+ABC'+AB'C'

The data recovered from the expanded functions can be shown in table 5.

Table 5. Expanded functions v/s data recovered

Expanded function	Data recovered
A'BC+A'BC'+AB'C+AB'C'	00111100
ABC+ABC'+ABC+A'BC	11001000
0000, 0000	00000000
AB+A'B, 0000	01010000
A'BC+A'BC'+AB'C+AB'C'	00111100
AB, 1111	10001111
AB'C+AB'C'+ABC'+AB'C'	01011100

The data recovered is input to the decoder shown in figure 3. The final data that will be recovered as shown in table 6.

Table 6. Truth table of decoder & recovered data

WXYZ	ABCD	Data recovered	BCD data
0000	0000	0011	0011
0001	0001	1100	0111
0010	0010	1100	0111
0011	0011	1000	1000
0100	0100	0000	0000
0101	0101	0000	0000
1010	0110	0101	0101
1100	0111	0000	0000
1000	1000	0011	0011
1111	1001	1100	0111
-	--	1000	1000
--	--	1111	1001
--	--	0101	0101
--	--	1100	0111

Hence the original BCD data obtained from the decoder can be given as follows:

00110111011110000000000010100000011011110001001010111.....(2)

Bit stream at (2) is exactly the same as at (1). Hence the digits finally obtained are: 37780050378957.

5. Conclusion

The 3-tier proposed method successfully worked for compression and decompression of BCD data. The encoding method used and then Boolean minimization carried out provide 1st level of protection to the data. Although cryptography is not used, but it can be used if required. The method has a good scope for future work. In this paper, the two consecutive PIs are used to make it a 3-input K-map problem, however, the larger groups can be tried to get higher compression ratios. The other methods can also be explored to treat 0s and 1s. For large volume of data, the whole system can be developed around software.

References

1. El Qawasmeh, Eyas. and Alfitiani Arif. "Development and Integration of a New Compression Technique using Boolean minimization" Jordan University of Science and Technology and Arab Academy for Banking and Financial Sciences, 2011.
2. Adler and Mitzenmacher, M. (2001). "Towards Compression Web Graphs." Proc. of the IEEE Data Compression Conference, Utah USA, pp 203-212
3. Sayood K.. "Introduction to data compression" third edition 2006. Morgan Kaufmann.
4. Salomon D. "Data Compression: The Complete Reference", Department of Computer Science, California State University, Northridge CA, USA, 3rd ed., ISBN 0-387-40697-2, Springer 2004.
5. Al-laham, M. et.al. "Comparative Study between various Algorithms of Data Compression Techniques", IJCSNS International Journal of Computer Science and Network Security, VOL.7 No.4, pp 281-291 April 2007.
6. Compressed Image Formats: JPEG, PNG, GIF, XBM, BMP, John Miano, August 1999
7. Managing Gigabytes: Compressing and Indexing Documents and Images, Ian H H. Witten, Alistair Moffat, Timothy C. Bell , May 1999
8. Augustine, J., Feng, W., and Jacob, J. (1995). "Logic Minimization Based Approach for Compressing Image Data", Proc. IEEelhi, India, 1995, pp 225-228.
9. R P Damodare, J Augustine and J Jacob. "Lossless and lossy image compression using Boolean functions", IIS, India 1996.