Maximum Likelihood Decoder for Variable Length Codes

*Syed Misbahuddin and **Mohammed Talal Simsim

***Abstract*- Variable Length Codes (VLC) are used to transfer same amount of digital information in relatively short period of time. In variable length coding, the characters with higher probability of occurrence are assigned shorter bits sequence and the characters with less probability of occurrence are assigned relatively longer bits sequence. However, due to variable length nature of codes, the decoding circuitry at the receiving end loses the synchronization due to single or multiple bit inversions. This typically happens when data is transmitted through a Binary Symmetric Channel (BSC). This paper investigates synchronizing scheme to control the error propagation due to single or multiple bit inversions through BSC. The hardware implementation of the proposed algorithm has been presented using a hardware description language. The functional level simulation of the implementation is discussed to test the proposed algorithm.**

Index Terms: Huffman coding, Variable Length Coding, VLC Decoder

I. INTRODUCTION

Variable Length Codes (VLC) are considered to be optimal source codes. A typical example of VLC is a Huffman Code [1]. In this coding scheme, the source symbols having higher probabilities of occurrences are represented by shorter bit sequences while the symbols having lower probabilities of occurrence are assigned longer bit sequences. The variable length coding minimizes average codeword length and therefore improves data transfer rates. With this coding scheme, more information may be transferred in a given time frame. For this reason, it is desirable to apply VLC to digital video transmission. However, there are some issues associated with VLC [9]. The main limitation is the loss of the synchronization by the receiver. In the following, we discuss the general problem of loss of synchronization and briefly outline the objectives of this research which addresses some of those problems.

During the data transmission using variable length coding through a binary symmetric channel, single or multiple bits of data may be inverted depending upon the channel characteristics. A single or multiple errors within a certain span of transmitted digits will cause incorrect decoding, which will, moreover, spill into the adjacent symbols and, therefore, will lead into loss of synchronization with the transmitting end. Therefore, bits inversions causes veral subsequent symbols to be wrongly decoded in addition to its parent symbols. The loss of synchronization will continue until the last bit of a decoded sequence coincides with the last bit of a codeword in the uncorrupted sequence. This limitation of VLC is illustrated in Example I.

Example 1

We illustrate the loss of synchronization with two cases of variable length codes mentioned below:

a) Case 1

Table I: Probability distribution of five code symbols for case 1

Symbol	Probability	Codeword
A	0.4	00
B	0.2	01
C	0.2	10
D	0.1	110
E	0.1	111

Consider the sequence ABAEBCD corresponding to the bit stream 000100 111 01 10110. Let us assume that the 3rd bit is inverted. It can be verified that the decoded sequence is given by ADBDDD. The decoder is out of synchronization after the first character and incorrectly decodes the next 5 characters before regaining synchronization.

b) Case 2

Table 2: Probability distribution of five code symbols for case 2

Symbol	Probability	Codeword
A	0.4	0
B	0.2	100
C	0.2	110
D	0.1	101
E	0.1	111

For case 2, assume symbol sequence ABAEBCD is transmitted from the transmitter. Corresponding bit stream for this symbol sequence will be according to Table 2:

0 100 0 111 100 110 101

During the transmission through BSC if bit number 4 is inverted then the bit sequence at the receiving end will be:

0 100 1 111 100 110 101

It is obvious, that due to the received corrupted bit sequence, the decoded symbol sequence will be ABECACD. This case shows that due to single bit inversion, we lost 3 source symbols.

*Department of Computer Engineering, Sir Syed University of Engineering and Technology, Karachi, drmisbah@ssuet.edu.pk
** Department of Electrical Engineering, Umm Al-Qura University, Makkah, Saudi Arabia, msimsim@uqu.edu.sa

Example1 shows that error propagation is triggered due to a simple bit inversion in any place. The error span is finite but still undesirable as instead of losing the single symbol where the bit inversion occurs, several symbols are incorrectly decoded.

The issue of error propagation has drawn attention of many researchers. T.J Ferguson and J.H Robino witz have investigated the construction of Huffman codes having inherent self synchronizing properties [3]. Some other procedures for construction of synchronizing codes have been studied by P.G. Neuman [4][5].These codes are included with non zero probabilities in the code dictionary. Assumption is that once the decoder at the receiving side is out of synchronization, then error propagation will continue until a synchronizing codeword comes. During the interim period, the decoder may continue producing erroneous data i.e. it will lose many valid characters. A solution to this problem proposed by Ferguson and Robinowitz[3] is to include special synchronizing code words having high source probabilities. But the general dynamic behavior of a decoder operating on data transmitted through a channel corrupted by Gaussian noise has not been widely investigated. In such cases, every transmitted bit might be inverted. In such channels, the synchronization problem becomes more complex. The bit inversion through a noisy channel may itself corrupt synchronizing the code words. Therefore, the purpose of synchronizing code is not achieved.

In this paper, synchronization recovery scheme is proposed to limit the error propagation due to bit inversions. In the proposed synchronizing scheme, the transmitter (encoder) inserts special sync pulses in the output bit stream. The main idea behind the scheme is as follows: From a particular code dictionary, all possible combinations of n code words are saved in a memory inside the receiving unit of the communication system along with information about their lengths in a separate memory buffer. The transmitter generates sync pulses after every group of *n* variable length code words. The synchronizing algorithm at the receiving side compares received group with all possible combinations of *n* variable length code words whose total lengths is equal to the received data length. As a result of execution of the decoding algorithm, variable length code words which are nearest to the received sequence will be decoded.

The paper organization is as follows: section II sheds light on the details of the proposed algorithm. Section III describes the Application Specific Integrated Circuits (ASIC) design concepts applied to the design of maximum likelihood decoder using a hardware description language. The simulation example is discussed in section III also. Finally, conclusion is discussed in section IV.

II. PROPOSED SYNCHRONIZING SCHEME

We have seen that due to variable length nature of codes, merely a single bit inversion in a codeword it triggers an error propagation due to which, several codes are wrongly decoded. To control this error propagation, we present a synchronizing scheme in this paper. The proposed scheme suggests that at the transmitting side of communication system, sync pulses are generated along with the information required for transmission. These sync pulses are inserted into the bit stream in a predetermined manner. The loss of synchronization could be restricted considerably with the help of these sync pulses. The sync pulses are inserted in the following way: After encoding the source sequence, *n* code words are used to make one group of codes, where *n* is determined by system requirement. A sync signal is inserted at the end of each group. Due to the variable nature of codes, the sync signals may be placed unevenly. At the receiving end, the decoder first determines the position of the sync signal. Since there must be precisely "*n*" code words, the decoder attempts to decode the received sequence into exactly "*n*" code words. In case of single or more bit errors in the received data, the received bits may not be decoded into exact "*n*" code words. Indeed, even when they are decoded into "*n*" code words, some or all of the decoded characters may be erroneous. However, the loss of synchronization would be automatically terminated by the beginning of every sync signal and there would be no error propagation. This synchronizing scheme will bring some overhead to the system. These are reduced throughput and increased average transmitter power. The reduction of throughput is self evident because the sync signals occupy time slots that would otherwise be utilized by information bits.

We consider the following M-ary ASK signaling scheme, although other schemes using frequency and phase modulation are also possible. Let the three signals S_0, S_1, S_{sync} corresponding to 0, 1and sync respectively are:

$$S_0 = ACOS(t)\ 0 \le t \le T$$
$$S_1 = -ACOS(t)\ 0 \le t \le T$$
$$S_{sync} = \pm kACOS(t)\ 0 \le t \le T$$

All signals are of equal duration and frequency. The amplitude for the binary signals is "*A"* while the amplitudes for sync signals are made *+kA* and *-kA* in an alternating fashion. At the receiving side, the sync pulses can be considered as the marks of the start and the end of a group of *n* VLC code words. The data elements between two sync pulses are transferred into a memory called INBUF by a special hardware we call *preprocessor*. Obviously, if each data element at each

memory element is processed individually then error in a group of code words would not affect the processing on any other data element. The preprocessor prepares to count the lengths of the received data in another memory array called LNBUF. After the completion of this initial process, the actual decoding sequence is initiated. The sync pulses are inserted after every *n* VLC code words. Each set of *n* code words can be stored at one memory location in the input buffer, INBUF. In a typical 26 alphabet variable length code dictionary the maximum length of a code word would be ten [2]; therefore, a set of 3 variable length codes could be adjusted in a 32 bit register. In the design of the decoder we illustrate the application of the proposed scheme, by setting a maximum value of *n* equal to 2. The memory elements used are 8 bits in length.

A. Decoding process

The proposed algorithms assume that the combinations of *n* variable length codes obtained from a particular set of variable length codes dictionary are stored in a memory buffer called CODEDICTIONARY. Due to variable nature of codes, the length of *n* codes will be different; therefore, corresponding length information of all combinations is stored in another memory buffer called DICLNTH.

The decoding process enters into the execution phase starting from the first data element in INBUF. The received *n* code words are saved in INBUF are compared with all valid combinations of *n* code words stored in CODEDICTIONARY. The code word from the CODEDICTIONARY is selected in favor of the received data which is closest to the received data. This means that the suggested scheme also performs some error correction. If there are single or multiple bit inversions in the received data then the error would be cramped within that data element and would not disturb the decoding process of the subsequent data sets.

EXAMPLE II

The functionality of the proposed synchronizing algorithm is illustrated in here using the VLC dictionary given in Example I (case 1).

Transmitted sequence without sync pulses:

110111 01 10 10 00
Corresponding symbols:
D E B C C A

Received sequence with an error in the first bit and corresponding decoded code symbols:

0101110110 10 00
B B D D C A

It is obvious that due to a single bit inversion, we lost 4 source symbols.

Transmitted sequence with sync pulses:
S 110 111 S 01 10 S 10 00 S

Received sequence with an error in the first bit.
S 010111 S 0110 S 1000 S

Decoded sequence
110 111 0110 1000=DEBCCA

It is noted Total six bits received between two sync pulses. In the VLC dictionary there are two codes with three bits representations namely D and E with codes 110 and 111, respectively. The proposed synchronizing scheme decodes the received sequence in favor of closest match which is 110 111 (D and E).

With the insertion of synchronization pulses at the transmitting end and appropriate processing of these pulses at the receiver, we are now in a position to limit error propagation due to loss of synchronization caused by channel errors.

III. HARDWARE IMPLEMENTATION OF PROPOSED SYNCHRONIZING SCHEME

In this section, we will discuss the Application Specific Integrated Circuit (ASIC) design of Maximum Likelihood Decoder (MLD) to implement the synchronizing algorithm introduced in section II. The proposed decoding ASIC consists of three levels; (i) A preprocessor (ii) The decoder and (iii) A post processor. Figure 1 shows the simplified diagram of VLC decoding scheme implementing the proposed synchronizing scheme.

A. Preprocessor

The preprocessor is at the input of the proposed MLD system. It accepts the variable length code words along with the sync pulses from some external component of a communication system. Sensing the sync pulses, the preprocessor scans the set of *n* code words and saves them in a buffer INBUF along with the corresponding lengths in another buffer called LNBUF. The preprocessor uses some special hardware to count the length of the data between two sync pulses. It indicates the availability of the data in INBUF and LNBUF to MLD by sending a signal on a line AVAIL. The decoding process begins when the data availability signal is received.

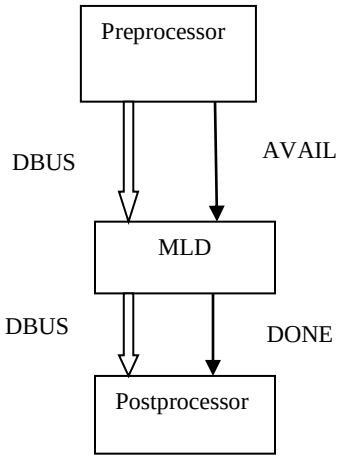

Figure 1: VLC Decoding System

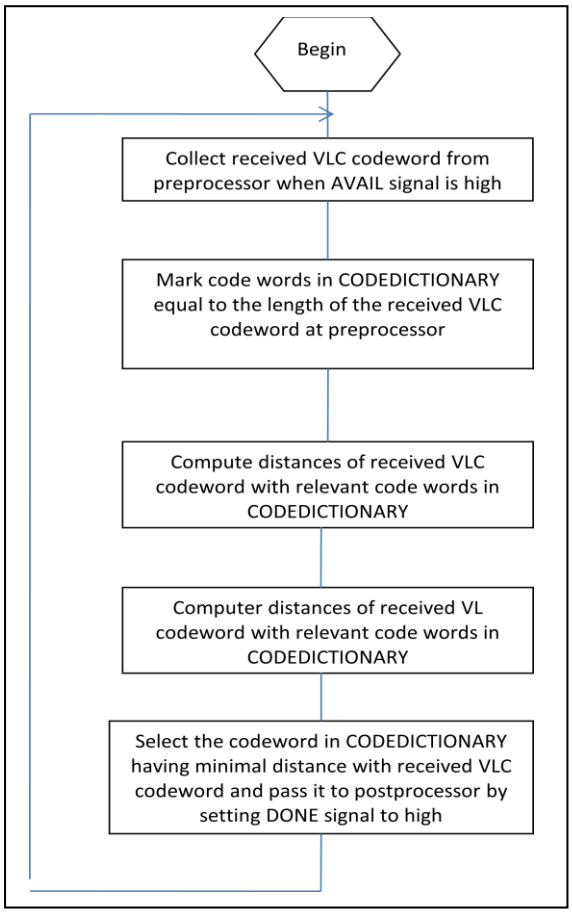

Figure 2: Flowchart of MLD

B. Maximum Likelihood Decoder (MLD)

The maximum likelihood decoder for VLC is the central processing unit of the proposed decoding ASIC system. It includes a Central Control Unit (CCU), memory units to save different data related to VLCs and an Arithmetic Logic Unit (ALU) to perform arithmetic and logical operations. The control unit inside the decoder generates various control signals which regulates the working of all units of the decoder. These control sequences are synchronized with the master clock of the whole system.

C. Postprocessor

When data is processed by the MLD, a DONE signal is sent to postprocessor. High level on DONE indicates to the postprocessor that the data has been processed and is ready to be transferred to the destination of the communication system which could be a computer output console or another host computer. Postprocessor clears DONE line to "0", once it accepts the data from MLD. There is no guarantee that the corrected data is the actual data which was transmitted by the transmitter. But this decoding strategy will correct some errors while at the same time will ensure that there will be no error propagating regardless of the decoding outcome.

D. Design of Control Unit of MLD

The synchronizing algorithm implemented by MLD has been tested by simulating the proposed system. For designing MLD we have used AHPL hardware description language [6]. AHPL is a Register Transfer Language (RTL) used for designing digital systems. Even thought, AHPL is a simple language, it can be used to model complex digital systems.

Similar to other digital systems, the proposed decoding system has two major components namely Information Processing Unit (IPU) or data unit and the Control Unit (CU). The IPU defines the basic information transfers and operations which can be performed on the information as it flows through the system. The sequence in which the operations are performed in a multistep computation is determined by the control unit. The control section will cause register transfers to take place in the data unit by sending signals on a set of control lines.

The AHPL sequence for MLD waits for a signal from the preprocessor and it sends a DONE signal to the postprocessor when it completes the decoding process. The control unit inside MLD sends signals to collect the arrived group of VLC and its length information from the preprocessor. MLD maintains a dictionary of all *n* combinations in a lookup table called CODEDICTIONARY. MLD control unit executes the following AHPL steps summarized in flowchart shown in Figure 2:

a. Length of the received group of VLC is compared with the lengths of all combinations stored in CODEDICTIONARY.
b. The pointers of the groups in CODEDICTIONARY whose lengths have been found equal to the received group of VLC are recorded.

c. The distances between the received code and possible groups from CODEDICTIONARY are computed and stored in a buffer called DISTANCES.
d. Minimum distance is determined and the corresponding VLC group from CODEDICTIONARY is selected in favor of received code on Maximum likely hood basis.
e. DONE signal is sent to postprocessor and AHPL sequence is returned to observe signal from preprocessor.

E. *Simulation of MLD*

The Maximum Likelihood Decoder has been simulated using AHPL functional level simulator [8]. The simulator inputs the compiled out of AHPL compiler and user supplied directives or parameters. The simulator parameters include initialization of all register arrays and input values of the system. The simulator executed the control unit of MLD for 470 clock pulses and generated the decoded VLC group. We provided a group of variable length code group with corruption in a bit to the simulator found that simulator generated the closest VLC group. For example, we input 0011, which is not a valid codeword group. The MLD produced 00 01, which is the closest possible code.

IV. CONCLUSION

Variable length codes assign shorted bit sequence to the symbols having higher probability of occurrence and longer bit sequence to the symbols having low probability of occurrences. This approach allows transferring the same information in relatively shorter period of time. However, a mere single bit inversion causes several characters to be wrongly decoded. In this paper a synchronizing scheme is presented to limit error propagation of VLC through a noisy BSC having nonzero crossover transition probabilities. The ASIC implementation of the proposed algorithm has been performed using hardware description language. The simulation of the design verified its functional accuracy.

ACKNOWLEDGMENT

The authors would like to express their appreciation to King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia for providing the environment to complete this research work.

REFERENCES

[1]D. Huffman, "A method for construction of minimum redundancy codes", *proceedings Institute Radio Engineers*, Vol. 40, pp 1098-1101, Sept. 1952.

[2] J.C. Maxted and J.H. Robinowitz, "Error Recovery of Variable Length Codes", *IEEE Transactions of Information Theory*, Vol. IT-31, No. 6 November 1985, pp 794-801.

[3] T.J. Ferguson and J.H. Robinowitz "Self Synchronizing Huffman Codes", IEEE Transactions of Information Theory, Vol. IT-30, 1984, pp. 687-693.

[4] P.G. Neuman "On a Class of Efficient Error Limiting Variable Length Codes", IRE Transactions of Information Theory Vol IT-8, 1962.

[5] P.G. Neuman "Error Limiting Coding Using Information lossless sequential machines", IEEE Transactions of Information Theory Vol IT-10, 1964.

[6] F.J. Hill and G.R. Peterson, *Digital Systems: Hardware Organization and Design*, John Wiley and Sons, New York, NY, 1987.

[7] Syed Misbahuddin, *An Investigation of Error Recovery of Variable Length Codes*, MS Thesis, Department of Electrical Engineering, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, January 1988.

[8] M.M. Al-Sharif, *Functional Level Simulator for Universal AHPL*, MS Thesis, University of Arizona, 1983.

[9] Thomas P. Bizon, Mary Jo Shalkhauser, and Wayne A. Whyte. Jr. "*Real-Time Transmission of Digital Video Using Variable-Length Coding*", Data Compression Conference (DCC '93) sponsored by the Institute of Electrical and Electronics Engineers, Snowbird, Utah, March 30-April 1, 1993.