IMPLEMENTATION OF LOSSLESS DATA COMPRESSION AND DECOMPRESSION TECHNIQUE USING VERILOG

B. Anil kumar[1], G. Niharika[2], R. Yamini Krishna[3], M. Sravan[4]

[1] *Assistant Professor, Dept of ECE, Malla Reddy Institute Of Engineering And Technology, Hyd., TS, India.*

[2] *B.Tech Student, Dept of ECE, Malla Reddy Institute Of Engineering And Technology, Hyd., TS, India*

[3] *B.Tech Student, Dept of ECE, Malla Reddy Institute Of Engineering And Technology, Hyd., TS, India*

[4] *B.Tech Student, Dept of ECE, Malla Reddy Institute Of Engineering And Technology, Hyd., TS, India*

Abstract-Data compression is the reduction or elimination of redundancy in data representation in order to achieve savings in storage and communication costs. Data compression techniques can be broadly classified into two categories: Lossless, Lossy schemes. Here in our project we use Xilinx tool. Xilinx ISE (Integrated Synthesis Environment) is a software tool used for synthesis and analysis of HDL designs, enabling the developer to synthesize their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli. In this project our proposal is that implementing the Lossless data compression technique using a new kind of compression coding i.e., ARITHMETIC coding. Arithmetic coding yields better compression because it encodes a message as a whole new symbol instead of separable symbols. By combining an adaptive binary arithmetic coding technique with context modeling, a high degree of adaptation and redundancy reduction is achieved. The most important advantage of this Arithmetic coding are : 1)Key lengths are flexible 2)Compressing the data by certain code word 3)Interval used by encoder is [0,1]. After data compression the output is a code word, which is a lowest limit value in entire range.

Keywords—*Data Encoding ; Data Decoding ;Xilinx; Verilog.*

I. INTRODUCTION

Data compression is a method of encoding rules that allows reduction in the total number of bits to store or transmit a file.

Currently, two basic of classes of data compression are applied in different areas one of these is lossy data compression, which is used to compress image data files for communication. The other one is lossless data compression that is used to transmit or archive text or binary files to keep their information intact at any time. Data compression techniques are mainly used to decrease memory size requirement for the image. In lossless, the original data is exactly reconstructed after the decompression whereas, lossy may lose some information from the image data.

II. LITERATURE SURVEY

Comparison between Lossy & Lossless Data compression

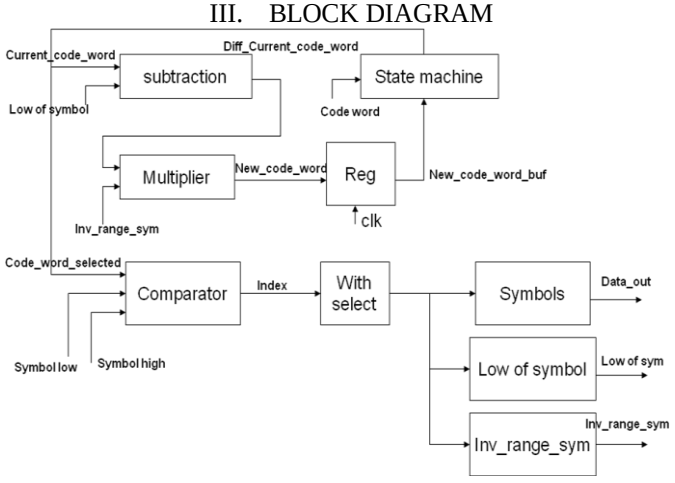
2.1.Lossy Compression

Lossy compression method provides superior compression ratio than lossless compression. The compression ratio is high in this method. The decompressed data is not accurately same to the original data, but close to it. Different types of lossy compression techniques are used, it is characterized by the quality of the reconstructed data and its adequacy for application.

2.2.Lossless Compression

In Lossless compression scheme the reconstructed data, after compression, is numerically identical to the original image. Most lossless compression programs do two things in sequence: the first step generates a statistical model for the input, and the second step is to map input data to bit sequences.

III. BLOCK DIAGRAM



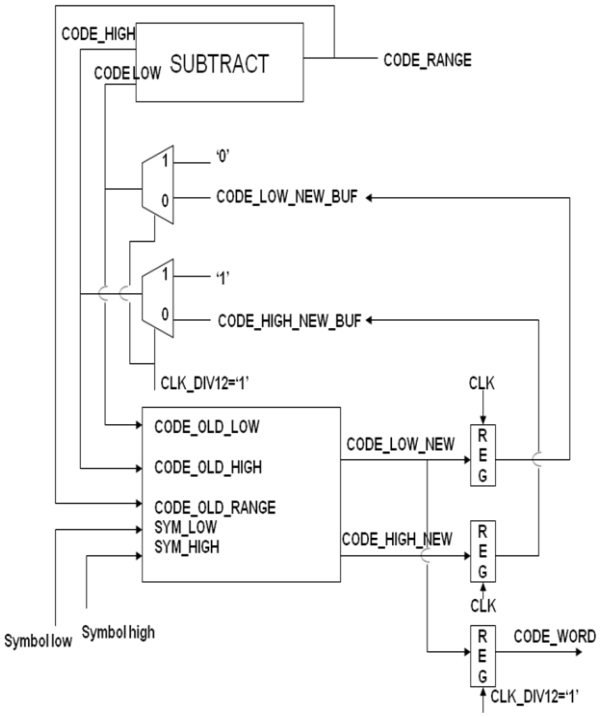


Fig.3.1: Encoding Architecture

A. ENCODING ARCHITECTURE:

Encoding architecture is shown in figure 3.1, where initially symbol low , symbol high, code_old_low, code_old_high and code_old _range are given as inputs to form code_old_new and code_high_new, these are connected to the registers where the data is temperarly stored untill the clk=1, these process is repeated for 10 times to complete the whole data. After receiving the first data the outputs of registers are given as inputs to multiplexer i.e. code_low_new_buffer and code_high_new_buffer and given as inputs to the substracter, where the code_low is substracted from code_high and gives a code_range. This process is repeated for 10 symbols of data.

B. DECODING ARCHITECTURE-Decoding architecture is shown in figure 3.2. In this multiplication, substraction, inv_range_sym, state machine (S0 to S10) are done.

IV. DESCRIPTION

Text message of 10 symbols [including space character (-)]

(i) SWISS MISS = 10 symbols

Symbol	No.Of Times Frequently Used Symbol	Probability	Code
S	5	5/10 = 0.5	[0.5 , 1.0]
W	1	1/10 = 0.1	[0.4 , 0.5]
I	2	2/10 = 0.2	[0.2 , 0.4]
M	1	1/10 = 0.1	[0.1 , 0.2]
-	1	1/10 = 0.1	[0.0 , 0.1]

Table.1:

The overall interval for the above example is – [0.0, 1.0].

Ranges for particular symbol formed showed in below

P(-) =0.1	P(M)=0.1	P(I)=0.2	P(W)=0.1	P(S)=0.5

0 0.1 0.2 0.4 0.5 1.0

Table.2:

Arithmetic coding encodes the entire message into a single number that is a fraction n where ($0.0 \leq n \leq 1.0$). The system utilizes an arithmetic coder in which the overall length within the range [0,1] allocated to each symbol.

A. 4.1Encoding process:

Based on the equations encoding is done. For starting symbol low value will be initial value i.e. '0' , and range will be [1.0 - 0 = 1.0]. Therefore range is '1'.

EQUATIONS:

NEW HIGH =
OLD LOW + RANGE * HIGH RANGE(X) ------ [1.1]
NEW LOW =
OLD LOW + RANGE * LOW RANGE(X) ----- [1.2]

B. 4.2 (B) Decoding Process:

Decoder gets only low value of the encoder 10th position value i.e. last symbol of 'S'. Therefore low new value will be = " 0.71753375 ".

Equation:

LOW RANGE (X) =

(NEWLOW-OLDLOW)*INVERSE RANGE …..[1.3]

When the '0' is obtained at the final symbol then the decoding process is completed. And the given text message is correct when the '0' is not obtained at the last symbol of decoding process then the given text message will be a wrong message.

4.3 Compression Technique:

Arithmetic coding:

Arithmetic coding completely bypasses the idea of replacing an input symbol with a specific code. Instead, it takes a stream of input symbols and replaces it with a single floating point output number. The longer (and more complex) the message, the more bits are needed in the output number It was not until recently that practical methods were found to implement this on computers with fixed sized registers. The output from an arithmetic coding process is a single number less than 1 and greater than or equal to 0. This single number can be uniquely decoded to create the exact stream of symbols that went into its construction. In order to construct the output number, the symbols being encoded have to have a set probabilities assigned to them. For example, if I was going to encode the random message "SWISS MISS", It would have a probability distribution that looks like this:

Character	Probability
SPACE	1/10
I	2/10
M	1/10
S	5/10
W	1/10

Once the character probabilities are known, the individual symbols need to be assigned to a range along the "probability line", which is nominally 0 to 1.

It doesn't matter which characters are assigned which segment of the range, as long as it is done in the same manner by both the encoder and the decoder. The nine character symbol set use here would look like this:

Character	Probability	Range
S	0.5	0.5 - 1.0
W	0.1	0.4 - 0.5
I	0.2	0.2 - 0.4
M	0.1	0.1 - 0.2
SPACE	0.1	0.0 - 0.1

Each character is assigned the portion of the 0-1 range that corresponds to its probability of appearance. Note also that the character "owns" everything up to, but not including the higher number. The most significant portion of an arithmetic coded message belongs to the first symbol to be encoded. When encoding the message "SWISS MISS", the first symbol is "S". In order for the first character to be decoded properly, the final coded message has to be a number greater than or equal to 0.5 and less than 1.0. What we do to encode this number is keep track of the range that this number could fall in. So after the first character is encoded, the low end for this range is 0.5 and the high end of the range is 1.0. After the first character is encoded, we know that our range for our output number is now bounded by the low number and the high number. What happens during the rest of the encoding process is that each new symbol to be encoded will further restrict the possible range of the output number. The next character to be encoded, 'W', owns the range 0.40 through 0.50. If it was the first number in our message, we would set our low and high range values directly to those values. But 'W' is the second character. So what we do instead is say that 'W' owns the range that corresponds to 0.40-0.50 in the new sub range of 0.5 – 1.0. This means that the new encoded number will have to fall some where in the 50^{th} to 60^{th} percentile of the currently established range. Applying this logic will further restrict our number to the range 0.7 to 0.75.

The algorithm to accomplish this for a message of any length is shown below :

```
Set low to 0.0
Set high to 1.0
While there are still input symbols do
   get an input symbol
   code_range = high - low.
   high = low + range*high_range(symbol)
   low = low + range*low_range(symbol)
End of While
Output low
```

Following this process through to its natural conclusion with our chosen message looks like this: TABLE.3:

New Character	Low value	High Value
	0.0	1.0
S	0.5	1.0
W	0.7	0.75
I	0.71	0.72
S	0.715	0.72
S	0.7175	0.72
SPACE	0.7175	0.71775
M	0.717525	0.71755
I	0.717530	0.717535
S	0.7175325	0.717535
S	0.71753375	0.717535

So the final low value, 0.71753375 will uniquely encode the message "SWISS MISS" using our present encoding scheme. Given this encoding scheme, it is relatively easy to see how the decoding process will operate. We find the first symbol in the message by seeing which symbol owns the code space that our encode message falls in. Since the number 0.71753375 falls between 0.5 and 1.0, we know that the first character must be "S". We then need to remove the "S" from the encoded number. Since we know the low and high ranges of S, we can remove their effects by

reversing the process that put them in. First, we subtract the low value of S from the number, giving 0.4350675. Then we divide by the range of S, which is 0.5. This gives a value of 0.4350675. We can then calculate where that lands, which is in the range of the next letter, "I".

The algorithm for decoding the incoming number looks like this:

get encoded number Do
find symbol whose range straddles the encoded number
output the symbol
range = symbol low value - symbol high value
subtract symbol low value from encoded number
multiplication encoded number by range
until no more symbols

Note that I have conveniently ignored the problem of how to decide when there are no more symbols left to decode. This can be handled by either encoding a special EOF symbol, or carrying the stream length along with the encoded message.

The decoding algorithm for the "SWISS MISS" message will proceed something like this:TABLE.4:

Encoded Number	Output Symbol	Low	High	Range
0.4350675	S	0.5	1.0	0.5
0.350675	W	0.4	0.5	0.1
0.753375	I	0.2	0.4	0.2
0.50675	S	0.5	1.0	0.5
0.0135	S	0.5	1.0	0.5
0.135	SPACE	0.0	0.1	0.1
0.35	M	0.1	0.2	0.1
0.75	I	0.2	0.4	0.2
0.5	S	0.5	1.0	0.5
0.0	S	0.5	1.0	0.5
0.0				

V. RESULT

So new interval [0.71753375 , 0.717535]

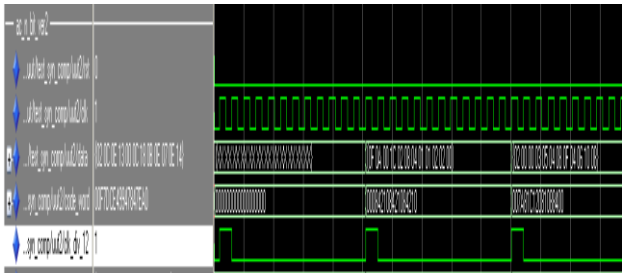
Conversion of encoding codeword in to bits(80 bits): The codeword received is converted in to bits as follows:

CODE WORD = 0.71753375 = 80 BITS

0=INTEGER, 71753375 = FRACTIONAL PART

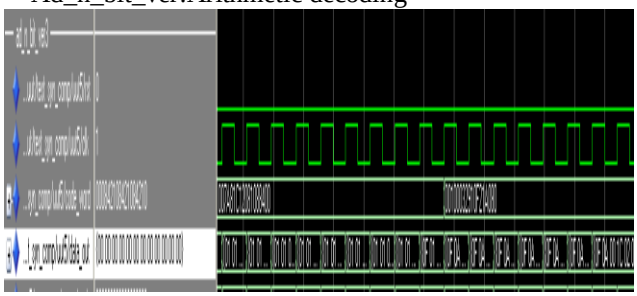
CODE WORD=0.71753375 MULTIPLYING WITH 2(Binary value)

Considering the integer values as 80 bits of data = 10110111 10110000 01001010 10110110 00000110 10110111 10101010 00100101 11011000 10110110.



Ac_n_bit_ver: Arithmetic Coding

Ad_n_bit_ver:Arithmetic decoding



VI. CONCLUSION AND FUTURE SCOPE

In this paper compression of data is done for 80 bits without losing the input data accurate during the compression, in future the compression of data can be increased and new techniques of implementation can be possible so that more accurate and fast compression can be done.

REFERENCES

[1]. Ben-Gal (2008). "On the Use of Data Compression Measures to Analyze Robust Designs" (PDF). 54 (3). IEEE Transactions on Reliability: 381–388.

[2]. IEEE Transactions on Reliability: 381–388.

[3]. McIntyre, D. R., and Pechura, M. A. 1985. Data Compression Using Static Huffman Code-Decode Tables. Commun. ACM 28, 6 (June), 612-616.

[4]. Reghbati, H. K. 1981. An Overview of Data Compression Techniques. Computer 14, 4 (Apr.), 71-75.

[5]. Storer, J. A., and Szymanski, T. G. 1982. Data Compression via Textual Substitution. J. ACM 29, 4 (Oct.), 928-951.

[6]. Tanaka, H. 1987. Data Structure of Huffman Codes and Its Application to Efficient Encoding and Decoding. IEEE Trans. Inform. Theory 33, 1 (Jan.), 154-156.

[7]. Welch, T. A. 1984. A Technique for High-Performance Data Compression. Computer 17, 6 (June), 8-19.

[8]. Witten, I. H., Neal, R. M., and Cleary, J. G. 1987. Arithmetic Coding for Data Compression. Commun. ACM 30, 6 (June), 520-540.

[9]. Cappellini, V., Ed. 1985. Data Compression and Error Control Techniques with Applications. Academic Press, London.
[10]. Faller, N. 1973. An Adaptive System for Data Compression. Record of the 7th Asilomar Conf. on Circuits,Systems and Computers (Pacific Grove, Ca., Nov.), 593-597.
[11]. Cormack, G. V. 1985. Data Compression on a Database System. Commun. ACM 28, 12 (Dec.), 1336-1342.