Some Remarks on Image Compression based on Reed-Muller Transform

Kamran Iravani

VLSI Technology, Inc.
1109 McKay Drive, M/S 21A
San Jose, CA 95131

Marek A. Perkowski

Department of Electrical Engineering
Portland State University, P. O. Box
751, Portland, OR 97207

Abstract

Reddy and Pai [1] have proposed to use the Positive polarity Reed-Muller transform for image data compression. In this paper an attempt has been made to improve this technique by using the more general Fixed Polarity Reed-Muller (FPRM) form. It will be shown that the Fixed Polarity Reed-Muller form, although better than positive polarity, does not improve the compression factor enough to warrant its use as an image compression method. It will be also shown that some crucial errors in the paper by Reddy and Pai make it impossible to evaluate the quality and compression factors of their approach.

1. Introduction

We believe that the techniques developed in the area of logic synthesis will find applications not only for circuit design, but also in image processing, compression, pattern recognition and other areas. This also applies specifically to EXOR logic.

In recent decades there has been a great interest in image data compression. This is due to the fact that digital representation of images usually requires a large number of bits, while in many applications it is desired to represent, store, transmit or process the image with a fewer number of bits.

Based on the applications there are two general categories of image compression techniques, lossy image compression and lossless image compression. In lossy image compression the reconstructed image is not exactly the same as the original but the compression factor is high, while in lossless image compression the reconstructed image is exactly the same as the original one but the compression ratio is not as high. Therefore there exists always a trade-off between the compression factor and the quality of the image.

Another important factor in image compression techniques is speed of the process. In some applications it is important for the process to be fast. The speed of the process is directly related to the number of operations needed for the compression, and also to hardware realization of the compressor.

In general, the compression techniques based on Fourier transform have a high compression factor with good quality compared to the other techniques. The only disadvantage of this method is that the process is slow because it requires a large number of multiplication operations.

The Walsh, Arithmetic and Haar transforms used in logic synthesis have been also applied in image compression and processing, and they are faster.

Another approach to obtain a fast method for image compression has been proposed by Reddy & Pai [1] who used the positive polarity Reed-Muller transform for this purpose. In their method the pixel matrix of an image is divided into eight matrices from the Most Significant Bit (MSB) plane to the Least Significant Bit (LSB) plane. On each plane the Reed-Muller transform is performed, and finally the run-length coding [5],[6] is used to compress the data. This method is fast because the operations needed are AND and Exclusive-OR which are done on bits. Such a method would also allow for very fast and inexpensive realization in hardware. But despite the fact that Reddy & Pai have claimed this method to have a good compression factor, we will show that this method does not give a good compression factor. They have claimed that with compression factor of 3.2, i.e. 2.5 bits per pixel, the quality of the reconstructed image is very good. We will also demonstrate that the paper published by Reddy & Pai contains several mistakes which, unfortunately, make their results and conclusions unacceptable.

It has been the purpose of this paper to improve the Reed-Muller image compression using the Fixed-Polarity Reed-Muller transform [2], [3], [4], which is a more general case. It will be shown that the results are still poor and a good compression factor cannot be obtained.

In this paper, first the Reed-Muller expansion of a switching function is briefly reviewed, then the Fixed-Polarity Reed-Muller form will be introduced. Next, image compression using Fixed-Polarity Reed-Muller form will be discussed with the simulation results. Then the paper published by Reddy & Pai will be investigated and criticized. Finally the paper is concluded by some remarks on this method of image compression.

2. Reed-Muller Transform

Any switching function of n variables can be defined by 2^n coefficients in a sum of product form as in Eq. (2.1):

$$f(x_0, x_1, \ldots, x_{n-1}) = d_0\bar{x}_{n-1}\bar{x}_{n-2}\ldots\bar{x}_0 + d_1\bar{x}_{n-1}\ldots\bar{x}_1 x_0 + \ldots + d_{2^n-1}x_{n-1}x_{n-2}\ldots x_0 \quad (2.1)$$

where $\left(d_0, d_1, \ldots, d_{2^n-1}\right)$ are the coefficients of the products which represent the values in the output column of the truth table of the function. These coefficients can be represented in vector form D, called truth vector.

The function can also be represented by the Reed-Muller canonical form over Galois field (2) as in Eq. (2.2):

$$f(x_0, x_1, \ldots, x_{n-1}) = a_0 \oplus a_1x_0 \oplus a_2x_1 \oplus a_3x_0x_1 \oplus \ldots \oplus a_{2^n-1}x_0x_1\ldots x_{n-1} \quad (2.2)$$

where $\oplus$ denotes modulo-2 addition and $\left(a_0, a_1, \ldots, a_{2^n-1}\right)$ are the coefficients of the expansion. These coefficients can be represented in vector form A, called the function vector.

These two representations are related to each other by a transform matrix as in Eq. (2.3)

$$A = TD \quad (2.3)$$

where T can be written in a recursive form as in Eq. (2.4):

$$T = T_n = \begin{bmatrix} T_{n-1} & 0 \\ T_{n-1} & T_{n-1} \end{bmatrix} \qquad \text{for } n \geq 1 \quad (2.4)$$

$$T_0 = 1$$

3. Fixed polarity Reed-Muller form

In the previous section the Reed-Muller expansion form was shown as in Eq. (2.2). In that form all the variables are in the positive form, while any variable x_i can be substituted with its negation $\bar{x}_i$, and still retain the canonical form. In the case that the variables are allowed to take negative polarities, we have the *Generalized Reed-Muller (GRM)* canonical form. And if each variable is restricted to retain the same polarity in all terms, i.e. either positive or negative but not both, the canonical form is called the *Fixed Polarity Reed-Muller* form.

For a function with n variables the number of possible arrangements of polarities is 2^n, so there are 2^n possible fixed polarity Reed-Muller forms. For a specific function the number of terms varies based on the polarity of the variables. For instance, if the Reed-Muller form of a function is $f(x_0, x_1, x_2) = x_2 \oplus x_0 x_2 \oplus x_1 x_2 \oplus x_0 x_1 x_2$, and the polarity of the variables is selected as $\bar{x}_0 \bar{x}_1 x_2$, the result will be $f(x_0, x_1, x_2) = \bar{x}_0 \bar{x}_1 x_2$.

Therefore by finding appropriate polarities, one can reduce the number of terms of a function in fixed polarity Reed-Muller form. The best polarity gives the least number of terms in a specific function.

4. Fixed Polarity Reed-Muller Image compression method

In the last section it was explained that if the best polarities for the variables are selected, the number of terms of each function can be reduced. Using this fact we are going to investigate if it can be used to compress the data of an image.

Every image consists of a set of pixels. In a grey-level picture, each pixel is usually represented by one byte. Therefore an image can be modeled by a matrix of pixel values, where each pixel value can vary in the range of 0 to 255 (256 levels of grey). In other words each pixel value consists of 8 bits, and each bit can be either 1 or 0. The picture matrix can be converted to 8 matrices from the most significant bit plane to the least significant bit plane, and each element of these matrices is either 1 or 0.

The algorithm to compress the image is as follows:

```
for( i=Most significant bit plane to least significant bit plane )
    {
        for  ( each plane )
            {
                (a) fetch a block of size (N × N)
                (b) find the best polarity of the variables
                (c) Compute Reed-Muller transform of the block
                (c) employ run-length coding on the resultant transform domain bit plane
            };
    };
```

In this method each block of size $(N \times N)$ is considered as a Karnaugh map, so the number of 1s in each block corresponds to the number of minterms. After finding its fixed polarity Reed-Muller form with the minimum number of terms [2], [3], [4], the new map usually contains fewer number of 1s.

For run-length coding three methods are being used: One dimensional Run-Length Coding (RLC) [6], Relative Address Coding (RAC) [5] and Relative Element Address Designate (READ) [6]. RAC and READ give almost the same result, and both of them are better than RLC. Since READ is more complicated than RAC but has almost the same effect on compression, RAC has been considered to be the best method of coding for this purpose. To determine the relative address distance codes in RAC, Huffman coding was used which gives the best compression factor.

As it will be shown later, our simulation results did not show a good compression factor (compression factor is the ratio of the size of the original image to the size of the image after being compressed). In fact assuming that the Reddy and Pai's simulation results were correct, we expected to obtain a much better compression factor than what we really obtained. After doing more research it has been understood that there exist some crucial errors in Reddy&Pai's work which make their simulation results totally unacceptable (these errors will be explained in detail later).

We cannot get a good result because of the following important point:

When Run-Length Coding is used, the compression ratio mostly depends on the number of transition elements (a transition element is the element which is different from its previous element in the same line) in a bit plane matrix, and not on the number of nonzero elements. For example in a 16 bit row of a block, it is possible to have 8 zeros and 8 ones with just one transition element as in Fig. 1 (a), alternatively it is possible to have 13 zeros and 3 ones with 6 transition elements as in Fig. 1 (b). Obviously the latter one will be less compressed while the number of nonzero elements in it is much smaller. Therefore the pattern of the elements is also important, and just reducing the number of nonzero elements might not be sufficient to obtain a higher compression ratio.

0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1

(a)

0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0

(b)

Fig. 1. The number of 1s in (b) is smaller, but the number of transition elements is larger.

5. Experimental Results

In this section the simulation results of the compression method based on Fixed-Polarity Reed-Muller form is discussed. For this method the size of the blocks has been taken to be 16×16, in order to be comparable with Reddy & Pai's simulation results.

In Table 1 compression factors of the four MSB planes are shown. These factors have been calculated based on 10 different pictures of natural scenes. Also the average numbers of transition elements for these images are shown. As it can be seen, the number of transition elements is high

Bit Plane	MSB	2nd MSB	3rd MSB	4th MSB
Compression Factor	1.9	1.2	1	1
Average number of Transition elements for Images of size 256x256	11000	19000	25500	29000

Table 1. Compression factor and the number of transition elements for typical images

which causes the compression factor to be low.

The data in this table are based on the fact that there is no information loss after compressing the bit planes.

As we know, the Least Significant Bit planes do not have as much effect on the values of pixels as the MSB planes. Therefore loss of information can be allowed in these planes. The simplest way to compress the data based on losing information is to consider some specific area, and calculate the number of 1s. If they are more than zeros, we change all the elements to one, otherwise all the elements are changed to zero.

The simulation results showed that to obtain a good quality for reconstructed images, we should not lose information in the four MSBs plane, and in this the case compression ratio is usually more than 4 bits per pixel. In other words the compression factor is usually less than 2, which is low compared to other commercially used compression techniques such as JPEG.

Fig. 3 shows the original image of a lady, and Fig. 4 shows the reconstructed image with compression factor of 2, i.e. 4 bits per pixel. It can be observed that with this compression factor the quality of the reconstructed image is good. Fig. 5 and Fig. 6 also shows the original and reconstructed image of a house, and we can see that the reconstructed image has a good quality. But compression factor of 2 is too low to make this method as a good candidate for image compression techniques.

6. A critique of the Reddy & Pai's paper on Reed-Muller Image Compression

Reddy & Pai's algorithm to compress the image is as follows:

for(i=Most significant bit plane to least significant bit plane)

begin

for (each plane) do

begin

(a) fetch a block of size ($N \times N$)

(b) Compute the Reed-Muller transform of this block

(c) employ runlength coding on the resultant transform domain bit plane

end;

end;

The idea here is to decrease the number of 1s by taking Reed-Muller transform, so that run-length coding results in a lower number of bits per pixel. The type of Run-Length coding which has been used, is Relative Address Coding (RAC). For their experimental results they have chosen blocks of size (16×16), and they have used RAC method with relative address distance code as in Table 2.

They have also suggested that, since in Reed-Muller transform all coefficients except the last one, are locally sensitive, a permutation of the input sequence will alter the number of non-zero coefficients (1s) in the Reed-Muller domain. Therefore an optimum permutation of the given input sequence results in a minimum number of non-zero coefficients in the Reed-Muller domain.

Relative distance	**Positive distance**	**Negative distance**
1	00	10
2	01	11
3	0000	1010
4	0001	1011
5	0100	1110
6	0101	1111
7	000000	101010

Table 2. Codes used by Reddy & Pai for relative distances.

In the following the above method will be investigated:

As it can be seen, the body of the inner loop consists of three steps. There is nothing wrong with step (a), but the problems with steps (b) and (c) cannot be ignored.

The problems with step (b)

In paper [1] the authors have not clearly explained what they have done, but from what they have written, it is obvious that it must be one of the following cases and nothing else:

1) The authors have just simply computed the positive polarity Reed-Muller transform of the block.

2) The authors have performed a permutation on the input sequence, and then computed the positive polarity Reed-Muller transform.

In the first case we cannot obtain a good compression factor by just computing the Reed-Muller transform of the block. In this case even for the MSB plane the compression factor on the average is less than 1.6, which is not good at all. We can get a better compression factor even if we don't use a transformation, and just directly encode the plane (we have developed a new EXOR-based method by Exoring the adjacent lines and adjacent planes [9] with a much better compression factor and much simpler hardware). In addition to that, we used fixed polarity RM transform, and although it is more general than RM transform and should further decrease the number of terms, still the result is not that good, and the compression factor for the MSB plane is less than 2.

In the second case although the authors have mentioned something about permutation, they have not found any method to find the best permutation, and still this problem remains unsolved. The authors have clearly suggested this problem for those who are interested in this issue, and want to work on it in the future. Therefore if they have applied a permutation, it has been done exhaustively which is not acceptable. It is obvious that finding the best permuted input sequence exhaustively cannot be realized by any hardware, because it requires too many cases to be checked, which is impossible. In addition to that, it is also needed to send a large amount of information with each compressed block to make it possible to reconstruct the permuted matrix, which would further decrease the compression factor.

The Problems with step (c)

In this step there are some obvious mistakes, and in the following these mistakes are discussed:

The authors have used Relative Address Coding (RAC) to code the data after performing the Reed-Muller transform. The code used by Reddy & Pai is in Table 2. These codes are short, and it seems that the reason for obtaining a good compression factor is to use such short codes. But these codes are all wrong and if these codes are used to code the data, the data cannot be reconstructed. In the following, four problems with the data in this table have been explained in the order of their importance.

1) In Relative Address Coding [5] the relative address is computed either with respect to the transition elements in the current line or those in the previous line. And then for all runlengths a method of coding must be used. For example Huffman Coding can be used which is the best method for coding. In the paper by Yamazaki [5] about RAC, other codes have been used which are not as good as Huffman from the point of view of compression, but they take less memory space. In any case, all of these coding methods must satisfy the following condition:

“ Each code must not be equal to a first part of another code.”

Huffman coding satisfy this condition and also the method used by Yamazaki satisfies them. Otherwise the codes cannot be detected.

Obviously the codes in Table 1 do not satisfy this condition, therefore they cannot be used to code the data. To make the problem clear, assume a part of a block after Reed-Muller coding is as in Fig. 2 According to Table 2 the code for element A must be 01 because the relative distance is +2, and the same code will be chosen for element B because the relative distance for B is +2 as well. Since A and B are adjacent transition elements, the code for them will be 0101. Now if the receiver receives the code 0101, it cannot identify if it is the code for the relative distance +6 or it is the code for two relative distance +2. This problem can be found in all of the codes in this table.

```
              C                 E
              ↓                 ↓
    . . . 0   1   1   1   1   1   1   0   0   0   0   0   0 . . .
. . .   0   0   0   0   1   1   1   1   1   1   0   0   0   0 . . .
                        ↑                 ↑
                        A                 B
```

Fig. 2. An example showing contradiction in Reddy & Pai coding

2) The authors have mentioned that depending on the probability of the occurrence, the RAC distances (0, +1, -1) are given special code words. According to Table 2 the authors have defined the codewords for +1 and -1 but not for 0. In any case the obvious thing is that all the codes must be

predefined. It would be impractical to first find the probability of the occurrence of the runlengths for every single matrix of an image, and then according to that, find the codewords for them.

Another important problem is that usually the most common relative distance is 0, so it should have the shortest codeword, which in most of the cases consists of just one bit, i.e. either 0 or 1, or two bits. But according to Table 2 it is impossible to have a short codeword for relative distance 0, because in this case the condition explained previously cannot be satisfied.

3) Since the size of the blocks has been chosen to be 16×16, when the RAC method is used, there is a need to define the codes for 16 positive relative distances, and not just for 8. From Table 2 it seems that the authors have not understood the RAC method sufficiently well. Probably they have thought that since the distance of each transition element from one of the row ends is less or equal than 8, it is enough to just define 8 positive relative distances. But it is impossible to decode the data this way, and 16 positive relative distances have to be defined, although for negative relative distances 8 codes are needed.

4) In RAC sometimes we need to compute the relative distance with respect to the transition elements in the current line. But according to Table 1 it seems that the authors of this paper have not defined them. In any case there are two possibilities:

a) They have defined relative distances and used them but they just did not mention it in the paper. In this case, either their codes are all wrong like those in Table 2, or the authors use very long codewords, because the condition mentioned for the coding must not be contradicted at least for the rest of the codewords. But with such long codewords it is impossible to obtain a good compression factor.

b) They have not defined the relative distances, and therefore have not used any codewords for the relative distances in the current line. In this case the coding will not be optimum, and the method is not the RAC method anymore. Some modifications have to be done to make the method work at the price of a decreased compression factor.

From the above discussion it is obvious that the reconstruction of the image is impossible with this type of coding.

6. Conclusion

An image compression method based on Fixed Polarity Reed-Muller transform was created and analyzed in order to improve the method introduced by Reddy & Pai [1]. Despite our several attempts and various variants tested, we were not able to find a good compression factor. Therefore we believe that this paper demonstrates that FPRM transform cannot be a good candidate for image compression, although the method is fast.

We have also clearly demonstrated that the paper published by Reddy & Pai on Reed-Muller image compression contains several errors which make their results totally unacceptable.

Although the experimental results presented here exclude the positive polarity Reed-Muller transform and the Fixed-Polarity Reed-Muller transform from being used directly in image encoding, we still believe that the theories and algorithms developed in EXOR logic synthesis may be useful in some areas of image compression and image processing. Just one of these possibilities is presented in [8].

References

1. B. R. K. Reddy, A. L. Pai, “Reed-Muller Transform Image Coding”, *Computer Vision, Graphics, and Image Processing,* Vol. 42, 1988, pp. 48-61.

2. D. H. Green, *Modern Logic Design*, Electronic Systems Engineering Series, 1986.

3. Y. Z. Zhang, P. J. W. Rayner, “Minimization of Reed-Muller Polynomials with Fixed Polarity”, *IEE proceedings,* Vol. 131, Pt. E, No. 5, September 1984, pp. 177-186.

4. A. Sarabi, M. A. Perkowski, “Cube Based Method for Optimal and Quasi-Optimal Minimization of Consistent Generalized Reed-Muller Expansions”, Department of Electrical Engineering, Portland State University, 1992.

5. Y. Yamazaki, Y. Wakahara, H. Teramura, “Digital Facsimile Equipments “Quick-FAX” Using a New Redundancy Reduction Technique”, *National Telecommunications Conference.* 1976, pp. 6.2.1-6.2.5.

6. R. Hunter, A. H. Robinson, “International Digital Facsimile Coding Standards”, *Proceedings of the IEEE,* Vol. 68, No. 7, July 1980, pp. 854-867.

7. D. A. Huffman, “A Method for the Construction of Minimum Redundancy Codes”, *Proceedings IRE,* Vol. 40, 1962, pp. 1098-1101.

8. M. A. Perkowski, T. Ross, D. Gadd, J.A. Goldman and N. Song, “Application of ESOP Minimization in Machine Learning and Knowledge Discovery”, *Submitted to International Workshop on Applications of the Reed-Muller Expansion in Circuit Design,* 1995.

9. K. Iravani, “An Analysis of Approaches to Efficient Hardware Realization of Image Compression Algorithms”, *Master Thesis*, Portland State University, 1994.