IMAGE COMPRESSION BASED ON REED-MULLER TRANSFORMS

Kamran Iravani and Marek Perkowski †,

VLSI Technology, Inc.,
1109 McKay Drive, M/S 21A, San Jose, CA 95131,
† Portland State University,
Department of Electrical Engineering,
Portland, Oregon 97207,
Tel: 503-725-5411, Fax: 503-725-4882, *mperkows@ee.pdx.edu*

Abstract— Wavelet Methods are revolutionizing the area of image processing and particularly compression. Haar transform is the simplest wavelet. Binary and multiple-valued nonsingular (NS) transforms [1] have similar local properties but use only logical operations and do not require additions and subtractions of Haar. It is then interesting to investigate the applications of NS transforms in these areas. The simplest of NS transforms is the Positive Polarity Reed-Muller Transform, which was first used by Reddy and Pai [2] for image data compression. In this paper an attempt will be made to improve their technique by using the more general (multi-polarity) Fixed Polarity Reed-Muller (FPRM) forms. It will be shown that the Fixed Polarity Reed-Muller form, although

better than the positive polarity, does not improve the compression factor enough to warrant its use as a lossless 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. We believe, however, that because of their specific properties, simplicity and speed of calculations, nonsingular transforms will find their applications niche in the areas of image compression, reconstruction, and recognition.

I. INTRODUCTION

The computer, telecommunications and video-media applications have developed rapidly the field of multimedia which requires high performance and speed digital video and audio capabilities. Digital storage media, image displays, and communication networks that use digital transmission channels require significant amounts of storage and high data rates to meet the needs of interactive visualization, home entertainment and multi-media systems. Multimedia systems are then the main reason of interest in image data compression. This is due to the fact that digital representation of images usually requires large numbers of bits, while in most applications it is desired to represent, store, transmit or process the image with a fewer number of bits. Good examples are still digitized images: a single DIN A4 color picture scanned at 300 dpi with 8 bits/pixel/ color produces 30 MBytes of data.

Based on the applications, there are two general categories of image compression techniques, **lossy** image compression and **lossless** image compres-

sion. 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 processing. 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 the hardware realization of the compressor.

In general, the compression techniques based on Discrete Cosine and Fourier transforms have a high compression factor with good quality compared to other techniques. The only disadvantage of these methods is that the process is slow because it requires a large number of multiplication operations. The Indeo-C algorithm of Intel used a transform coding called Fast Slant Transform (FST), with only additions and shifts. The (non-sinusoidal) Haar transform is the fastest of all known complete unitary transforms. It is local, thus used for data compression of non-stationary ("spiky") signals, as well as processing operations such as edge extraction. Discrete Wavelet Transform (DWT) uses Haar functions to code images and is one of the most promising techniques today. The Arithmetic, Walsh (Hadamard), and Haar transforms are used in image compression, processing and related problems, and especially there is recently a very strong interest in Haar transform for video compression.

Recently, the techniques developed in the area of logic circuit synthesis find applications not only for circuit design, but also in Machine Learning, image processing, compression, pattern recognition and other areas [3, 4, 5, 6]. This also applies specifically to spectral approaches to logic (Walsh, Reed-Muller, Haar, arithmetic, adding, nonsingular, Christenson) [7, 8, 9, 10, 11, 12, 13, 14, 1, 15, 16, 17, 18].

The Arithmetic, Walsh, and Reed-Muller transforms (to lesser degree Haar) are fundaments of the most important spectral approaches to logic synthesis. These transforms are faster, since multiplications are not required and only addition/subtraction/shift/logic operations are used. Haar is a special kind of wavelet transforms, and therefore a question arises, what are the mutual relationships of spectral logic transforms and wavelet approaches.

Among hundreds of papers on Reed-Muller and nonsingular transforms, we were able to find just one related to signal processing and compression; an approach to obtain a fast method for image compression has been proposed by Reddy and Pai [2] 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 [19, 20] is used to compress the data.

The approach of Reddy and Pai would be potentially a very fast and inexpensive method because only Exclusive-OR operations would be executed

on bits, and adders/subtractors of Walsh or Haar transforms are not used. Because of existance of Butterfly-like structures similar to other "fast" transforms for Reed-Muller, such a method would also allow for very fast and inexpensive realization in hardware.

But despite the fact that Reddy and Pai have claimed this method to have a good compression factor, we will show that this method does not have it for lossless images. 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 and Pai contains several mistakes which, unfortunately, make their particular results and conclusions unacceptable, without, however, negating the general innovativeness and potentials of the originated by them approach.

It has been the purpose of this paper to improve the Reed-Muller image compression using the Fixed-Polarity Reed-Muller transform [8, 18, 16], 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 (section II), then the Fixed-Polarity Reed-Muller form will be introduced (section III). Section IV explains butterly method for software/hardware realization of transform algorithms. In section V, image compression using Fixed-Polarity Reed-Muller form will be discussed. Experimental results are given in section VI. Then the paper by Reddy and Pai will be investigated and criticized in section VII. Section VIII concludes the paper

and outlines future work.

II. REED-MULLER TRANSFORM

Any switching function of n variables can be defined by 2^n coefficients in a sum of product (SOP) form as in Eq. 1:

$$\begin{aligned} f(x_0, x_1, ...x_{n-1}) \;&=\; d_0\overline{x_{n-1}}\;\overline{x_{n-2}}...\overline{x_0} \;\oplus \\ d_1\overline{x_{n-1}}\;\overline{x_{n-2}}...\overline{x_1}x_0 \;&\oplus\; ... \;\oplus\; d_{2^n-1}x_{n-1}x_{n-2}...x_0 \end{aligned} \tag{1}$$

where $(d_0, d_1, ..., d_{2^n-1})$ 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 Positive Polarity Reed-Muller (PPRM) canonical form over Galois field (2) as in Eq. 2:

$$\begin{aligned} f(x_0, x_1, ...x_{n-1}) \;&=\; a_0 \;\oplus\; a_1x_0 \;\oplus \\ a_2x_0x_1 \;&\oplus\; ... \;\oplus\; a_{2^n-1}x_0x_1x_{n-1} \end{aligned} \tag{2}$$

where $\oplus$ denotes modulo-2 addition and $(a_0, a_1, ..., a_{2^n-1})$ 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. 3:

$$A = TD \tag{3}$$

where T can be written in a recursive form as in Eqs.4,5.

$$T_0 = 1, \tag{4}$$

$$T = T_n = \begin{bmatrix} T_{n-1} & 0 \\ T_{n-1} & T_{n-1} \end{bmatrix} \qquad for \quad n \geq 1 \tag{5}$$

III. FIXED POLARITY REED-MULLER FORM

In the previous section the Reed-Muller expansion form was shown as in Eq. 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 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 polarities is 2^n, so there are 2^n possible fixed polarity Reed-Muller forms. For a specific function the number of terms (i.e. non-zero coefficients of the transform) varies based on the polarity of the variables. For instance, if the Reed-Muller form of a function is $f(x_0, x_1, x_2) = x_2 \oplus x_0x_2 \oplus x_1x_2 \oplus x_0x_1x_2$, and the polarity of the variables is selected as $\bar{x_0}\bar{x_1}x_2$, the result will be $\bar{x_0}\bar{x_1}x_2$.

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.

IV. BUTTERFLIES FOR FAST FPRM TRANSFORMS

The advantage of all "fast" transforms, Fourier, Cosine, Walsh, Hadamard, Slant, and Haar is that Butterfly structures can be developed for them, which next leads to efficient combinational, pipelined, systolic, memory-based (Ping-Pong,etc), and parallel or distributed hardware or software realizations. Much information about the properties of an uderlying transform can be gained by analysis of such butterflies. For instance, it can be observed that Fourier and Walsh are global transforms, thus any point of the image space has the same importance and affects all coefficients. In contrast, Haar is a local transform, which means that local data differences in image space are used, and some coefficients depend only on some (in a sense geometrically local) image data. Binary Reed-Muller transforms are similar to Haar in their locality, but their realizations are simpler because none of them involves adding/subtracting. Instead, basically only EXOR operations are used, which are very fast both in hardware and in software.

We will illustrate a Butterfly for the PPRM transform, but very similar butterflies for FPRM and higher order nonsingular (polynomial) transforms can be created [21, 1, 22]. Assume that the image of letter **A** from Fig. 1a is represented by a Karnaugh Map of function $f(a, b, c, d)$ from Fig. 1b. Using of a Butterfly to calculate the PPRM transform of the function from Fig. 1b

is shown in Figure 2. The rows in the first column correspond to all minterms of four variables in natural binary ordering. Each minterm corresponds to one of inputs to the butterfly. Minterms with value 1 (ones, true minterms) are denoted by black bullets. Minterms with value 0 (zeros, false minterms) are denoted by lack of bullets. Every next column corresponds to the *"folding operation"* with respect to one of input variables, d,c,b, and a. The Butterfly should be interpreted (evaluated) from left to right by synchronized moving of the bullets from a column to the next column. Each joining of a horizontal and diagonal arrows represents an EXOR operation. Using rules $0 \oplus 0 = 0$, $1 \oplus 1 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, the bullets are moved from left to right and they become PPRM coefficients when they reach the last column in the right. For instance, if the two input to EXOR element are bullets, the output is no bullet. If only one input is a bullet, the output is also a bullet. Propagation of values through the butterfly reveals that the PPRM for function $f(a,b,c,d) = \sum(0,1,2,3,4,6,8,10,11,12,13,14,15)$ from Fig.1 is $f = 1 \oplus bd \oplus ad$, as shown in the last column.

This Butterfly structure can be interpreted as a combinational network of EXOR gates, and thus all the coefficients can be calculated in one pulse of the clock. Conversely, a D-type flip-flop can be associated with every intersection of a row and a column of the butterfly, thus leading to a pipelined structure giving first set of transform coefficients after $n = 4$ pulses, and in every next pulse (assuming one pulse to load all flip-flops in the first column). Similarly, the memory-based architectures (Ping-Pong and other), based on Address

Generators, can be developed from this butterfly. Design methods well-known from DSP architectures based on FFT, DCT, WT or HT transforms can be used to develop them.

It can be seen in Figure 2 that minterm $0000 = \bar{a}\ \bar{b}\ \bar{c}\ \bar{d}$ (denoted by $a'b'c'd'$ in the figure), influences (globally) the values of all coefficients, because there are arrows going from it to all points in the last column that includes PPRM coefficients. Similarly, minterm $abcd$ influences only the value of coefficient $abcd$ and minterm $abc\bar{d}$ influences only the value of coefficients abc and $abcd$. Thus every minterm has some local scope of coefficients influenced by it. Observe, that for positive polarity of variables, 1111, the coefficients are all sets of non-negated variables (constant 1 corresponds to an empty set). We call 1111 the **polarity minterm** of PPRM. Its **opposite polarity minterm**, 0000, is the most sensitive to value changes in pixels, because it propagetes its value to all coefficients. Similarly cell 1111 is the least sensitive. Thus, in Negative Polarity RM, 0000, the polarity minterm is 0000 and the 1111 is the opposite polarity minterm. Similarly, all 2^n fixed polarity FPRMs can be created, each with its polarity minterm and opposite polarity minterm. In our case, there are $2^n = 2^4 = 16$ polarities, thus 16 possible most sensitive to changes minterms in the Kmap, thus, every pixel of the initial image can be made most important for the transform. Therefore, transforms for certain image regions can be created. A Grey-coded Kmap from Fig. 1b is only an example, and binary-coded (Marquand) or other encodings of k-dimensional images are possible.

Comparison of all Haar functions to all Reed-Muller functions reveals their similarity. Both transforms represent local behavior and both can be generalized to 2^n different polarities [23].

An interesting modification of this method can be used for image reconstruction [3]. Assume noise in a transmitted bit changes this bit to unknown value, which we will represent by a don't care in the map. Thus FPRM transform on an incomplete function leads to simple FPRM forms that assign values to don't cares in such a way that the maximum number of coefficients becomes zero [12, 11]. Using Occam's Razor principle, the particular FPRM transform polarity and one of its possible assignments of values to don't cares are selected that minimize the number of non-zero coefficients. Similarly, the lowest order coefficients can be discarded. Then a butterfly-based inverse transform is done from coefficients back to the image space, with noise removed and the image smoothed. This way lossles transmission of high performance in presence of noise can be achieved for limited black and white images.

Binary Reed-Muller logic is the same as Galois Field with 2 elements and Galois multiplication and addition. This logic can be generalized to a Galois Field(k), where $k = p^r$, p=prime, $r > 1$, and most of the important properties from GF(2) are inherited in GF(k), including the existence of Butterflies. Galois Field(k) Butterflies require GF(k)-addition and GF(k)-multiplication operations, which however can be easily built with look-up tables, multivalued- or binary-logic elements, and thus realized efficiently in hardware, or in software. For instance, it can be proved that only k two-input

EXOR gates and no other gates are sufficient to realize Galois addition in Galois Field(2^k) [24]. Thus, while binary images are naturally good for PPRM, FPRM and all binary nonsingular transforms, two approaches can be used for grey-scale and color images. One is a separate coding of image planes, as in Reddy and Pai, and in this paper. The other is to use higher-order Galois Fields and nonsingular transforms in them, which we are presently investigating [24].

V. FIXED POLARITY REED-MULLER IMAGE COMPRESSION METHOD

In last sections 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 a still 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 signif. bit plane)
```

```
{
 for  ( each plane )
   {
     (a) fetch a block of size (N x N)
     (b) find the best polarity of the variables
     (c) Compute Reed-Muller transform of the block
     (d) employ run-length coding on the resultant
           transform domain bit plane
   };
};
```

In this method each block of size **N x N** is considered as a Karnaugh map, so the number of 1s in each block corresponds to the number of true minterms. After finding its fixed polarity Reed-Muller form with the minimum number of terms [8, 18, 16], the new map (in transform space) usually contains fewer number of 1s than the image space map.

For run-length coding three methods are being used: One dimensional Run-Length Coding (RLC) [20], Relative Address Coding (RAC) [19] and Relative Element Address Designate (READ) [20]. 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 and 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. 3a, alternatively it is possible to have 13 zeros and 3 ones with 6 transition elements as in Fig. 3b. 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.

VI. EXPERIMENTAL RESULTS

In this section the simulation results of the compression method based on Fixed-Polarity Reed-Muller form are discussed. For this method the size of the blocks has been taken to be 16 x 16, in order to be comparable with Reddy and 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 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 there are more ones 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 of reconstructed images, we should not lose information in the four MSB planes, 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. 5 shows the original image of a lady, and Fig. 6 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. 7 and Fig. 8 also show 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 a good candidate for image compression techniques.

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

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

```
for(i=Most significant bit plane to least signif. bit plane)
 begin
  for  ( each plane ) do
   begin
    (a) fetch a block of size ( )
    (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 runlength 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 x 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.

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 [2] 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:

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 EXOR-ing the adjacent lines and adjacent planes [15] 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 the (positive polarity) 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 for checking to be a practical method. 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 and Pai is in Table 2. These codes are short, and it seems that the reason for obtaining a good compression factor was 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 [19] 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 [19] 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 [25] coding satisfy this condition and also the method used by Yamazaki satisfies them [19]. 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 as in Fig. 4. 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.

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 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 x 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.

VIII. CONCLUSIONS AND FUTURE WORK

An image compression method based on Fixed Polarity Reed-Muller transform was created and analyzed in order to improve the method introduced by Reddy and Pai [2]. Despite our several attempts and various variants tested, we were not able to find a good compression factor. Therefore we believe that our paper demonstrates that FPRM transform cannot be a good candidate for lossless compression of still images, although the method is very fast. We have also clearly demonstrated that the paper published by Reddy and Pai on Reed-Muller image compression contains several errors which unfortunately 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 lossless image compression, PPRMs and FPRMs can be still useful for lossy image compression, video coding, image recognition (feature creation) [3, 4, 5], and image reconstruction [3]. We believe that the theories and algorithms developed in nonsingular logic may be useful in some areas of image compression and image processing, especially when local features are important, similarly to the applications of Haar transform and wavelets. These research and application areas should be further identified and investigated.

One of possibilities is to use FPRM-based lossy image compression for feature creation in Machine Learning method with supervision. In [4] Exclusive Sum of Products (ESOP) expressions were minimized with good results for

this task. Observe, that an FPRM minimizer that could deal with incompletely specified functions can be used for this task, instead of the ESOP minimizer, in features domain. Moreover, FPRM minimizer could be first used also in image domain, to create features from the Karnaugh Map image representation discussed above. Then some high-order coefficients or their functions become feature values to be used in supervised learning.

References

[1] M. Perkowski, A.Sarabi, and F. Beyl, "Fundamental Theorems and Families of Forms for Binary and Multiple-Valued Linearly Independent Logic," *Proc. Reed-Muller'95*, pp. 288-299.

[2] B.R.K. Reddy, and A.L. Pai, "Reed-Muller Transform Image Coding," *Computer Vision, Graphics, and Image Processing,* Vol. 42, 1988, pp. 48-61.

[3] T. D. Ross, M.J. Noviskey, T.N. Taylor, and D.A. Gadd, "Pattern Theory: An Engineering Paradigm for Algorithm Design," *Final Technical Report WL-TR-91-1060,* Wright Laboratories, USAF, WL/AART/WPAFB, OH 45433-6543, August 1991.

[4] M.A. Perkowski, T. Ross, D. Gadd, J.A. Goldman and N. Song, "Application of ESOP Minimization in Machine Learning and Knowledge Discovery," *Proc. Intern. Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, 1995.

[5] M. Perkowski, M. Marek-Sadowska, L. Jozwiak, T. Luba, S. Grygiel, M. Nowicka, R. Malvi, Z. Wang, and J. Zhang, "Decomposition of Multiple-Valued Relations," *Proc. ISMVL'97*, pp. 13-18.

[6] J. Han, "Data Mining Techniques", *Proc. 1996 ACM-SIGMOD Int'l Conf. on Management of Data (SIGMOD'96)*, Montreal, Canada, June 1996 (Tutorial).

[7] M. Davio, J.P. Deschamps, and A. Thayse, "Discrete and Switching Functions," *McGraw Hill*, 1978.

[8] D.H. Green, *Modern Logic Design,* Electronic Systems Engineering Series, 1986.

[9] T. Sasao (editor), "Logic Synthesis and Optimisation," *Kluwer Academic Publishers,* 1993.

[10] T. Sasao (editor), "Representation of Boolean Functions," *Kluwer Academic Publishers,* 1996.

[11] A. Zakrevskij, "Minimum polynomial implementation of systems of incompletely specified Boolean functions," *Proc. Reed-Muller'95,* pp.250-256.

[12] M. Perkowski, P. Lech, Y. Khateeb, R. Yazdi, and K. Regupathy, "Software-Hardware Codesign Approach to Generalized Zakrevskij Staircase Method for Exact Solutions of Arbitrary Canonical and Non-canonical Expressions in Galois Logic," *6th Intern. Workshop on Post-Binary ULSI Systems*, May 27, 1997, pp. 41-44.

[13] M. Perkowski, "A Fundamental Theorem for Exor Circuits," *Proc. Reed-Muller'93*, pp. 52-60.

[14] M. Perkowski, A.Sarabi, and F. Beyl, "XOR Canonical Forms of Switching Functions," *Proc. of Reed-Muller'93*, pp. 27-32.

[15] K. Iravani, and M.A. Perkowski, "An Analysis of Approaches to Efficient Hardware Realization of Image Compression Algorithms," *Dept. Electrical Engineering Report,* Portland State University, June 1997.

[16] A. Sarabi, and M.A. Perkowski, "Cube Based Method for Optimal and Quasi-Optimal Minimization of Consistent Generalized Reed-Muller Expansions," *Department of Electrical Engineering Report*, Portland State University, May 1992.

[17] A. Sarabi, and M.A. Perkowski, "Fast Exact and Quasi-Minimal Minimisation of Highly Testable Fixed-Polarity AND/XOR Canonical Networks," *Proc. DAC,'92*, pp. 30-35.

[18] Y.Z. Zhang, and P.J.W. Rayner, "Minimization of Reed-Muller Polynomials with Fixed Polarity," *IEE Proceedings,* Vol. 131, Pt. E, No. 5, Sept. 1984, pp. 177-186.

[19] Y. Yamazaki, Y. Wakahara, and H. Teramura, "Digital Facsimile Equipments "Quick-FAX" Using a New Redundancy Reduction Technique," *National Telecommunications Conference,* 1976, pp. 6.2.1-6.2.5.

[20] R. Hunter, and A. H. Robinson, "International Digital Facsimile Coding Standards," *Proc. of the IEEE*, Vol. 68, No. 7, July 1980, pp. 854-867.

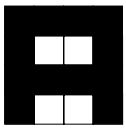
[21] B.J. Falkowski, and S. Rahardja, "Fast Transforms for Orthogonal Logic," *Proc. IEEE 28th ISCAS,* Seattle, WA, May 1995.

[22] M. Perkowski, L. Jozwiak, R. Drechsler, and B. Falkowski, "Ordered and Shared, Linearly-Independent, Variable-Pair Decision Diagrams," *Proc. 1st Intern. Conf. on Information, Communication and Signal Processing (ICICS'97)*, Singapure, Sept. 9-12, 1997.

[23] S. Rahardja, and B. Falkowski, "Application of Sign Haar Transform in a Ternary Communication," *Int. J. Electronics*, Vol. 79, No.5, pp. 551-559, 1995.

[24] K.M. Dill, K. Ganguly, R.J. Safranek, and M.A. Perkowski, "A New Linearly Independent, Zhegalkin Galois Field Reed-Muller Logic", *Proc. RM'97*, Oxford, Sept. 1997.

[25] D.A. Huffman, "A Method for the Construction of Minimum Redundancy Codes," *Proc. of IRE*, Vol. 40, 1962, pp. 1098-1101.

(a)  (b)

ab \ cd	00	01	11	10
00	1	1	1	1
01	1	0	0	1
11	1	1	1	1
10	1	0	0	1

Figure 1: *Encoding of a binary image with a Karnaugh map.*

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 256 x 256	11000	19000	25500	29000

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

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 and Pai for relative distances.*

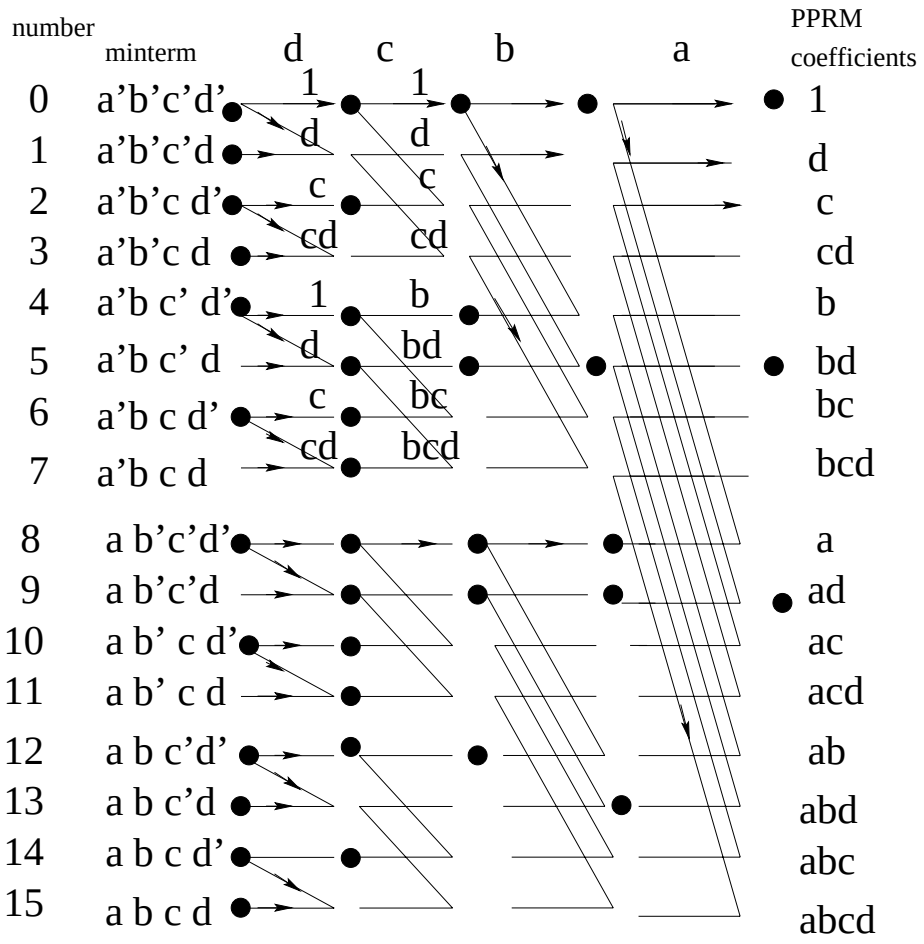


Figure 2: *Using a Butterfly structure with only EXOR gates to calculate the PPRM transform for function from Fig. 1b.*

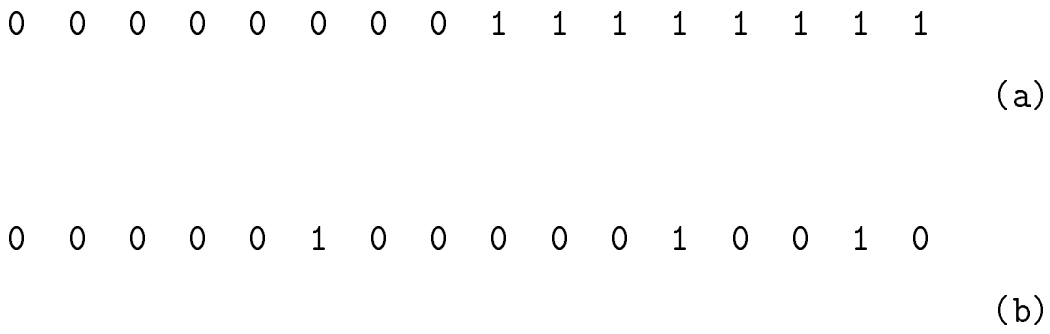


Figure 3: *Example to Run-Length Coding. The number of 1s in (b) is smaller, but the number of transition elements is larger.*

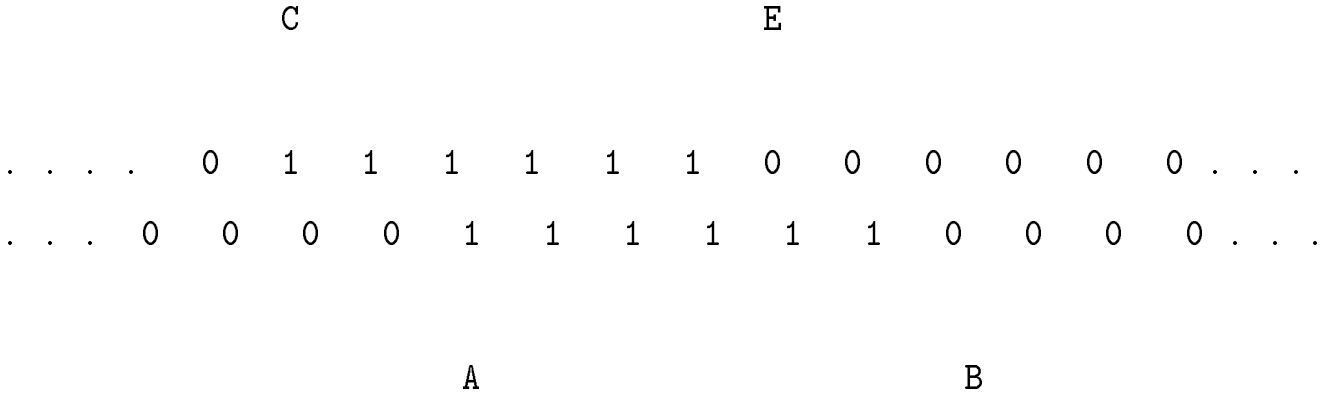


Figure 4: *An example showing contradiction in Reddy and Pai coding.*

Fig. 6.1 : Original image (8 bits per pixel)

Figure 5: *Original image of Lena. 8 bits/pixel.*

Fig. 9.2 Compressed image (2.5 bits per pixel)

Figure 6: *Compressed image Lena. 2.5 bits/pixel.*

Fig. 6.3 : Original image (8 bits per pixel)

Figure 7: *Original image of House. 8 bits/pixel.*

Fig. 9.4 Compressed image (2.5 bits per pixel)

Figure 8: *Compressed image of House. 2.5 bits/pixel.*