Mojette Transform Software – Hardware Implementations and its Applications

Jan Turan[1], Peter Szoboszlai[2], and Jozsef Vasarhelyi[3]

[1]Department of Electronics and Multimedia Telecommunications, Technical University of Kosice, Park Komenského 13, 041 20 Košice, Slovak Republic, e-mail: jan.turan@tuke.sk
[2]Magyar Telekom, Hungary 1117 Budapest, Magyar tudosok korutja 9., e-mail: szoboszlai.peter@telekom.hu
[3] Department of Automation, University of Miskolc, Miskolc-Egyetemvaros, Hungary, e-mail: vajo@mazsola.iit.uni-miskolc.hu

***Abstract* — In the last ten years there were intensive researches to find new methods for handling internet image processing and distributed databases. One of the many methods is the Mojette transform (MT). The MT is used mainly in image processing applications, but can be used also for distributed databases also. The MT implemented under different operating systems and for different processors presented in helps the performance analyses of the MT and also let to conclude the need for reconfigurable hardware implementation. The Mojette Transformation Tool (MTTool) is an implementation of MT and inverse MT (IMT) in .Net environment. The software development provides us with the endless possibility of different variations of the MT implementation in a shorter time frame and on lower costs. Also the testing with such a tool is much easier and it's also better for demonstration and training purposes. There is also analyzed the hardware implementation of MT and IMT based on Field Programmable Gate Arrays (FPGA) using reconfigurable platform. The paper tries to conclude the necessity for hardware implementation for real time processing. The paper outline the development work in order to create an embedded reconfigur-able hardware based on FPGA board.**

***Index Terms* — Mojette Transform, MTTool, image processing, distributed databases, FPGA, embedded systems, MoTIMoT co-processor**

I. INTRODUCTION

The Mojette Transform (MT) is originated from France where J-P. Guédon named it according an old French class of white beans, which were used to teach children computing basics of arithmetic with simple addition and subtraction [1-5]. He named MT after the analogy of beans and bins. The bins contain the sum of pixel values of the respective projection line [1]. There are several different variations of MT applications nowadays which are used in different areas such as tomography [2], internet distributed data bases [3], encoding, multimedia error correction [4,5,7], or The Mojette Transform Tool (MTTool) which was created for testing purposes, by the way it can also be used for demonstrations and training purposes as well.

However the MTTool development isn't finished yet we already gained many experiences with it, and we see how it could become more helpful for further projects both in software and in hardware development in the area of MT applications. So the main purpose to build such an environment was that with the help of it we could try to compare the software implementation of the MT later on with the hardware implementations.

In the first section we introduce the mathematical background of the MT and the IMT briefly and this is followed by the description of the transforms implementation in several versions, concentrated not on the stodgy programming stuff rather on a basic description.

Next we share some of our test results which were focusing on the time consumption of the MT and IMT and after that we also introduce the differences between the first and the other versions from the Graphical User Interface point of view illustrated with some figures as well.

Until today the aim of the researches was software implementation of the MT and IMT [1-5,7]. This kind of implementation can be used only on still images, because they cannot process the data in real-time. Motion pictures, video streams and distributed data bases require run-time processing of the frames. Only specialized hardware or embedded systems with real-time operating systems can ensure this.

This paper presents one implementation method to implement the MT and IMT in Field Programmable Gate Array (FPGA) using reconfigurable platform.

II. MOJETTE AND INVERSE MOJETTE TRANSFORM

A. Mojette Transform

The main idea behind the Mojette transform- MT (similarly to the Radon transform) is to calculate a group of projections on an image block [5]. The MT [1,11,16,17] projects the original digital 2D image:

$$F = \left\{ F\left(i, j\right); i = 1,...,N;\ j = 1,...,M \right\} \tag{1}$$

to a set of K discrete 1D projections with:

$$M = \left\{ M_k\left(1\right); k = 1,...,K; 1 = 1,...,1_K \right\} \tag{2}$$

MT is an exact discrete Radon transform defined for a set $S = \{(p_k, q_k),\ k = 1,...,K\}$ specific projections angles:

$$M_K\left(l\right) = proj\left(p_k, q_k, b_l\right) = \sum_{(i,j)\in L} F\left(i, j\right)\delta\left(b_l - iq_k - jp_k\right) \tag{3}$$

where *proj* (p_k,q_k,b_l) defines the projection lines p_k, q_k, and $\delta(x)$ is the Dirac delta in the form:

$$\delta(x)=\begin{cases}1, if\ \_x=0\\ 0, if\ \_x=1\end{cases} \tag{4}$$

and

$$L=\{(i,j);b_l-iq_k-jp_k=0\} \tag{5}$$

is a digital bin in the direction θ_k and on the set b_l.

So the projection operator sums up all pixels values which centers are intersected by the discrete projection line *l*. The restriction of angle θ_k leads both to a different sampling and to a different number of bins in each projection (p_k, q_k). For a projection defined by θ_i, the number of bins n_i can be calculated by:

$$n_i=(N-1)|p_i|+(M-1)|q_i|+1 \tag{6}$$

The direct MT is depicted in Fig. 1 for a 4x4 pixel image. The set of three directions *S={(-1,2),(1,1),(0,-1)}* is resulting 20 bins.

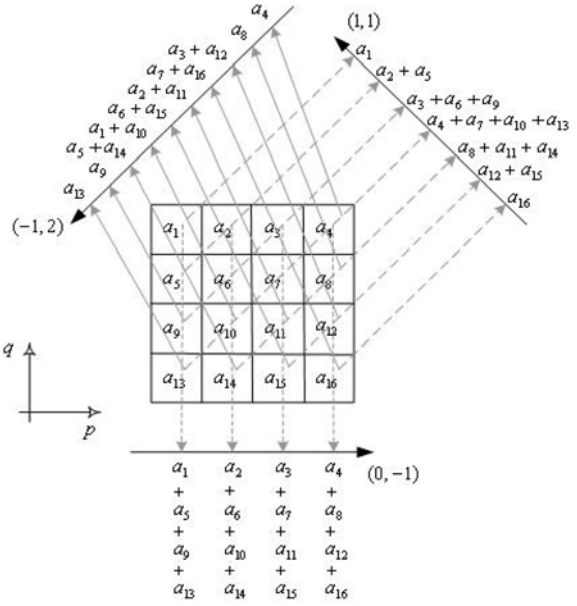

Fig.1. The set of three projections computed from a 4x4 image.

The MT can be performed by directly addition of the image pixel values in grey scale images and for bitmaps we can add the different bitmap color table values.

B. *Inverse Mojette Transform*

Computing Inverse Mojette Transform (IMT), bins are back-projected. A single pixel-*bin* correspondence must be found at each iteration cycle in order to reconstruct a pixel value. When it has been done, this pixel value is substituted in the adequate coordinate of a blank image and subtracted from the corresponding bins in each projection of the original image. To find this single pixel-*bin* correspondence examination of the projections of unary image is necessary. The *bin* values in those projections show the number of pixels which are stored in the given *bin*. Therefore the modification of the unary projections has to be done parallel (Fig.2). The IMT is iterating this process until the image is completely reconstructed. For the reconstruction both projection sets needed (one is the MT of the image and the other is the MT of a unary image where each pixel value is 1). Fig.3 shows the first step of the reconstruction process.

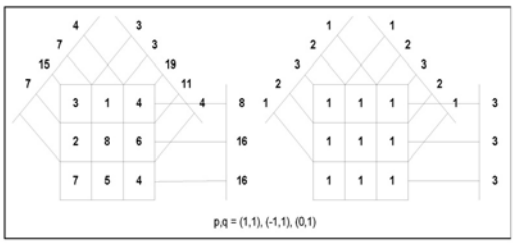

Fig.2. Mojette Transform on integers.

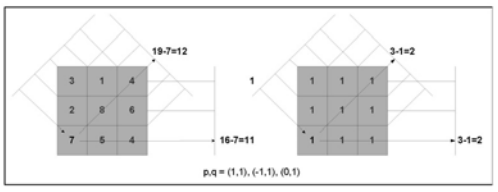

Fig.3. Inverse Mojette Transform – first step.

C. *Condition of Reconstruction*

The key of the transform with regard to the controlled redundancy is that the number of computed projections can be larger than we need for the reconstruction (inverse transform). Thus, we can control a first step of redundancy with the number of projections. The question is then to know how many projections and which projections give an adequate set to reconstruct the image (i.e. compute the Inverse Mojette Transform). The result for rectangular image was given by Katz in [6].

According to the Katz's Lemma [6] image of dimension KxL can be reconstructed with the *Mf* set of directions (p_i, q_i), i=1, 2, …, I if

$$K \le P_i=\sum_{i=1}^{I}|p_i| \quad or \quad L \le Q_i=\sum_{i=1}^{I}|q_i|, \tag{7}$$

where K and L are the dimensions of the image p_i the horizontal projection coordinate (x axis) and q_i the vertical projection coordinate (y axis).

The condition of reconstruction ensures that the number of *bins* (N_b) is greater than the number of pixels (N_p) therefore the number of equations will be larger than the number of unknown variables during the reconstruction process. At the same time the condition of reconstruction answers the question whether is any projection in the projection set which is unnecessary to reconstruct the original image.

MT for binary image of simply binary data is defined with the operations „AND" and „XOR" respectively instead of the multiplication „*" and addition „+" [1,5,11].

III. MOJETTE TRANSFORM IN MTTOOL

In MTTool the implementation of the MT was applied in three different ways [15-17]. This is due to the fact that this application is still in development and the three different ways were constructed not in the same time but in the last years.

TABLE I
MT IMPLEMENTATION AND IT'S MAIN DIFFERENCES

Nr.	Image Format	Projections	MT and IMT
1	PGM	p={1,-1,3,-3}; q={quarter of the image size}	Addition and subtraction
2	BMP	p={2,-2}; q={1} and p={3,-3,2}; q={1}	Addition and subtraction
3	BMP	p={2,-2}; q={1} and p={3,-3,2}; q={1}	Matrix

A. The First Version

In the first release the declaration of some rules, which had to be somehow flexible and at the same time they shouldn't be very complex was one of the hardest to decide. We had to declare the image sizes with which we had to work later and to look for useful relation between the picture size and the vectors we use in the MT and IMT. After calculating with several file sizes, it was clear we had to choose it so, that it should be easy to remember and have something common with all the other images as well. We decided to take the picture size $2^n x 2^n$ where n is equal to 8 and 9, but can be changed later on easily. So the picture size is 256 x 256 and 512 x 512. In the Picture Preview we can open and display any kind of PGM or BMP file it doesn't matter which size the picture got, but some of the images are increased or decreased to fit on the screen.

TABLE II
IMAGE DISPLAY

Original size	Displayed size	Ratio
1600 x 1200	400 x 300	0,25
1599 x 1199	799 x 599	0,5
1024 x 768	512 x 384	0,5
Height < 1024	Height +180	Other

The first step in the MT after checking the restrictions is to make a vector from the pixels of the image. To define the size of this vector is easy, when we are following this simple rule $(1, 2^n \times 2^n)$. If n=8 this results the vector (1, 65536), in which every line contains a pixel value from the picture. Because the PGM picture is a 256 grayscale image, a PGM file contains just pixel values from 0 to 255. In case of a BMP image we could process it three times because of the different bitmap color table values. [13,14]

In the second step we are executing the MT. The vector p is predefined for the four projection directions and the q vector has the same value in each four case (quarter size of the $2^n \times 2^n$ image). We generate four files for the four different projections, these are the following:

1) originalfilename.pgm.moj1 (1, q)
2) originalfilename.pgm.moj1 (-1, q)
3) originalfilename.pgm.moj1 (3, q)
4) originalfilename.pgm.moj1 (-3, q)

From the existing MT files (moj1,…, moj4) we get the original PGM picture with the IMT. All of the four MT files are needed in this case to rebuild the original image completely without any error. If any of the MT files is defect or incomplete the IMT doesn't give back the original image. Each of the four files is containing a vector described above. The next step of the IMT is, to read the first and last vectors of the third and forth MT files and put it on their place. So we have in all four corner of the picture filled up with the valid pixel values. Step 1, 2, 3 and 4 on the following figure:

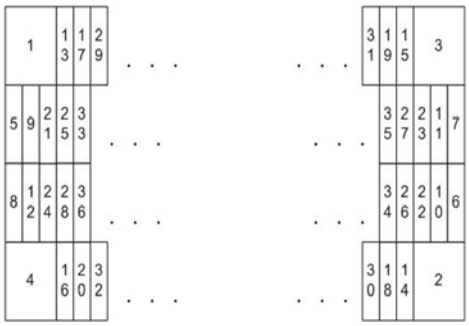

Fig.4. First 30 steps of the IMT.

After recreating the pixel values we only need to add the new header for the file and the restoration of the original image is already performed.

B. The Second and Third Version

These solutions are differing from the previous one in such a way that these are applied on BMP images and in these cases we are performing the MT and IMT on the three different bitmap color table. We use the same algorithm for the three different color maps and collecting the bins into 3 separate files which are differing only in their extensions and of course in their content. On the bitmap images we are using the directions $S_1=\{(2,1),(-2,1)\}$ and $S_2=\{(3,1),(-3,1),(2,1)\}$ for the block sizes 4 and 8. Although the MT is also prepared for the block size 16 and 32 but the implementation of the IMT isn't done yet. In the second version we use simple addition and subtraction different from the one mentioned in the first version, since here we have block sizes 4 and 8 and there we are performing the MT and IMT on the whole image at once and not step by step. In the third version instead of addition and subtraction we use matrices for the MT and IMT on the above mentioned block sizes. The MT with matrices is implemented in the following way, where bi is the bin resulted from the following equation:

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \\ b_9 \\ b_{10} \\ b_{15} \\ b_{16} \\ b_{17} \\ b_{18} \\ b_{19} \\ b_{20} \end{bmatrix} = \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0 \end{bmatrix} * \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \\ a_9 \\ a_{10} \\ a_{11} \\ a_{12} \\ a_{13} \\ a_{14} \\ a_{15} \\ a_{16} \end{bmatrix} = \begin{bmatrix} 10 \\ 123 \\ 37 \\ 137 \\ 254 \\ 319 \\ 433 \\ 68 \\ 6 \\ 234 \\ 125 \\ 267 \\ 312 \\ 8 \\ 45 \\ 178 \end{bmatrix} \quad (8)$$

The inverse matrix for the previous example (for the 4x4 matrix size) is implemented as it's showed on the next equation, where a_i are the original values of the matrix:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \\ a_9 \\ a_{10} \\ a_{11} \\ a_{12} \\ a_{13} \\ a_{14} \\ a_{15} \\ a_{16} \end{bmatrix} = \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&1&0&0&0&0&-1&0&0&0&-1 \\ 0&0&0&1&0&0&0&1&0&0&-1&0&0&0&-1&0 \\ 0&0&0&0&0&0&0&-1&0&0&0&1&0&0&0&1 \\ 0&0&0&0&0&0&0&0&-1&0&0&1&0&0&1&0 \\ 0&0&0&0&1&0&0&0&1&0&0&0&0&-1&0&0 \\ 0&0&0&0&0&1&0&0&0&1&0&0&-1&0&0&0 \\ 0&0&0&0&0&0&0&0&0&-1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&-1&0&0&1&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&-1 \\ 0&0&0&0&0&0&0&1&0&0&0&0&0&0&-1&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0 \end{bmatrix} * \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \\ b_9 \\ b_{10} \\ b_{15} \\ b_{16} \\ b_{17} \\ b_{18} \\ b_{19} \\ b_{20} \end{bmatrix} = \begin{bmatrix} 10 \\ 123 \\ 25 \\ 35 \\ 12 \\ 102 \\ 252 \\ 241 \\ 2 \\ 78 \\ 255 \\ 23 \\ 178 \\ 45 \\ 6 \\ 234 \end{bmatrix} \quad (9)$$

IV. EXPERIMENTS WITH MTTOOL

With the built in ZIP and Huffman coding opportunities (Fig. 5) we can decrease the size of any vectors which are created from the projections of MT [13-17]. The Huffman lossless encoding and decoding algorithm was chosen due to it's binary block encoding attribute and not because of it's compression capability. Good data compression can be achieved with Zip and Unzip which is also implemented. The possibility of time measuring with simple tools, such as labels or easily generated text files which are including the test results can give us a good insight into the MT and IMT. From these results we can estimate and predict the consumed time on hardware implementation and also the cost of it.

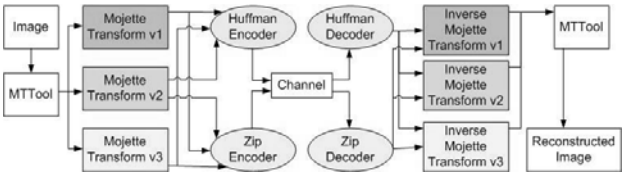

Fig. 5. Logical system architecture of the MTTool.

The time measurement was applied on three different images with three different image sizes and with three different periods. The images were a black and white PGM files with pixel values of 0 and 255 and the LENA.PGM. First test ran only once, after that the second test was running for 6 times in a row, and the last test was running 24 times. Each test was performed on the sizes of 16x16, 32x32 and on 512x512. the results of the two smallest image sizes are nearly identical and the result were nearly always under 20 millisecond for MT and IMT, but on the 512x512 image size we could see the following difference:

TABLE III
TEST RESULT OF THE MT AND IMT WITH THE FIRST VERSION

IMAGE	Black (512x512)		White (512x512)	
	Minute: Second: Millisecond	MT and IMT in Millisecond	Minute: Second: Millisecond	MT and IMT in Millisecond
MT start	57:14:277		0,1621528	
MT end IMT start	57:15:439	1162	0,1623032	1893
IMT end	57:15:910	471	0,1687963	561
MT start	57:22:259		0,1761806	
MT end IMT start	57:23:411	1152	0,1744792	1733
IMT end	57:23:891	480	0,1699653	550

From this we can see that the difference between the black and the white images is more than 50 percent, when it comes to the MT, and only 20 percent when we apply the IMT on the Mojette files.

V. USER INTERFACE OF THE MTTOOL

Basically the MTTool program surface is bade on MDI structure of Microsoft. The parent window contains the main menu and nearly all child windows are opened here. The only exception is the Help/Contents (help.html) which appears outside from the parent window in the default internet browser application.

At the beginning the main goal of the implementation was to represent both the original and the restored image together with the Mojette files which contained the bins of the Mojette transform described in [11,13-17]. We created two image boxes to show the "before and after" images for the user, and we duplicated the information on the two tabbed bar to help the users in comparing the original values, the projections and the inverse values to each other. On the Fig. 6. we can see, the amount of information was too much to check it visually even if the different projections were separated not only by their colors but also were listed after each other. Due to the image size of 512 x 512 we had too many bins to display. That's also one of the many things we decided to make different versions as well.

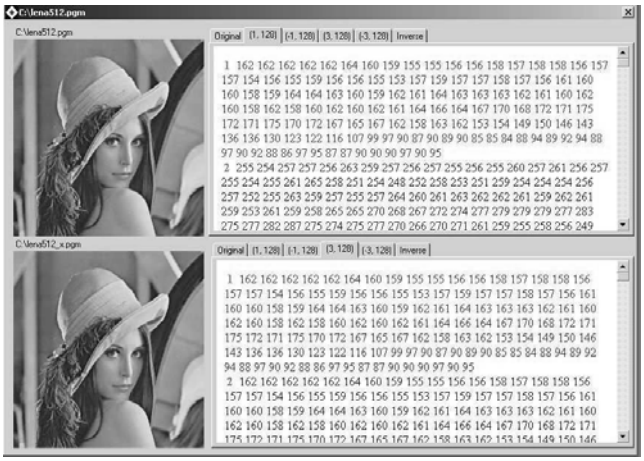

Fig. 6. MT GUI of the first version.

Multiple images in multiple instances can be opened so we are able to compare them not only by the image itself but also by their properties as well we can see the following information from the pictures such as: Original size, Colors used, Pixel size, Size of the DIB-element, Actual size and Last write time.

Later on we developed the Mojette transform in a different way and this gave us the opportunity to create a new interface as well for them (Fig. 7). On an image we can perform the Mojette Transform (Block) with the help of the following child window, where all the steps are visible for a better overview. First we have to choose between two differently based Mojette Transform type such as Algorithm and Matrix. After that we can choose from the predefined Block sizes, which we want to use to perform the MT. Here we will automatically see the Number of blocks, Number of projections, the Projections and the number of bins used for the chosen Block size. If both parameters are set we can click the Next button. As soon as we pushed the Next button we will see the result of the MT with the created Red, Green and Blue bin values grouped together and ordered by the blocks. Here we can check the values of each Block where the MT was performed, and see what are the bin values stored in these Blocks.

Mojette block window (Fig. 7) represents the blocked version of the MT. As it can be seen we can choose between Matrix and Algorithm versions with different blck sizes from 4 up to 32 from this and from the hardcoded Projections the other values are calculated.

Although this information is also stored in a file we are reading it out from the memory. With the help of Show button and the X and Y values we can choose and display the different Blocks and bin values. By hitting the Next button we can create the bin file which contains all the data presented in the step before. Here we have the opportunity to add error to the Blocks, which will result vertical lines in the reconstructed image from the faulty bin file. However we can only see this error on the image if we reconstruct the bin into a BMP file with the IMT which can transform the bin file automatically, so we don't need to set the IMT like we had to do it with the MT.

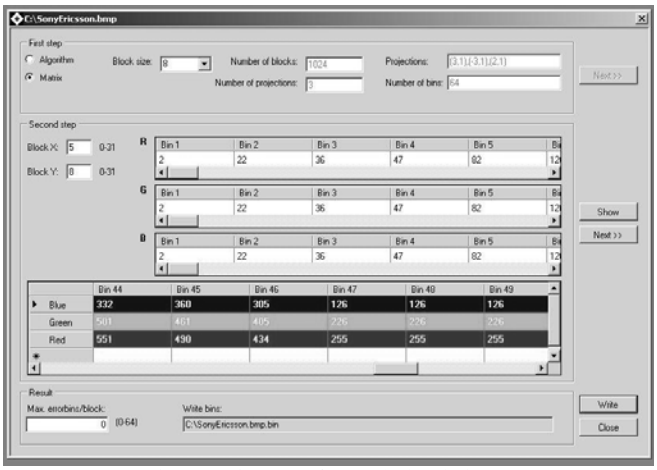

Fig. 7. Mojette transform (Block) window.

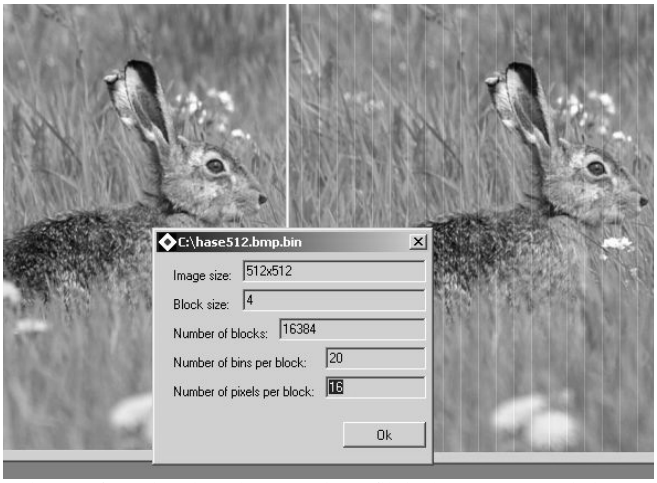

Fig. 8. Information about the restored BIN file in the Inverse MT window.

The Inverse Mojette (Block) window (Fig. 8) shows us some information about the restored image, we can see that the restored image on the right contains error. This was caused in the Mojette Transform phase where we overwrite the value 0 in the 'Max. errorbins/block' field to add some noise to the Mojette Transform. We can also gain information about the MT which was performed on the image.

VI. LOW POWER – LOW COMPLEXITY HW DESIGN METHODOLOGY FOR SELECTED ALGORITHM- MT AND IMT

Power consumption is a very important question of our days. Many energy saving technologies and techniques were developed in different fields. The spread of portable computers and communication devices inspires the developers to design low power consumption devices (CPUs, storage devices, etc.) [8]. In the computer technology there are several options when low power consumption devices are needed (DSPs, μPs). One of these options is using field programmable gate arrays (FPGAs) [9]..

The "MOTIMOT" co-processor denotes the hardware implementation of the direct and inverse Mojette transform as co-processing elements of an embedded processor [12,16,17].

To construct a new hardware for a special task is difficult and expensive both in time and costs [8,10]. Estimating the hardware and software needs to implement a MOTIMOT processing system there is necessary to map the tasks what such a system should implement. These tasks are not limited to the direct and inverse Mojette transformation, but also posses embedded computer functions with or without (depends on the application) real time operating system kernel. Fig.9 shows the functional block scheme of the MoTIMoT co-processor. Both of the Virtex II PRO and Virtex IV FPGAs (FPGAs used for implementation) contain an embedded power PC 405 RISC processor (PPC) [9]. This general purpose processor can manage the MoTIMoT hardware, its driver and the incoming and outgoing data. The MT and IMT blocks are connected to the PPC via the processor local bus (PLB).

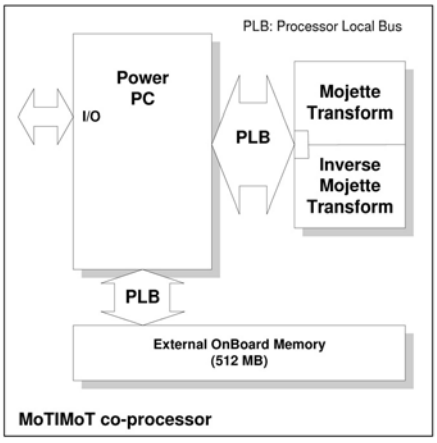

Fig.9. Block schema of MoTIMoT co-processor.

Probably calculation of MT or IMT of an image is not necessary in the same time, therefore only one of the configuration files is loaded into the device. At this point we can use the advantage of partial run-time reconfiguration.

The images are received in real-time thru an Ethernet connection or from a digital camera connected to the platform via an USB port, after processing the image or the frames these are returned to the sender or sent to a client (PC, PDA, etc.). On the motherboard there is an external onboard memory with the size of 512 MB. In our days this resource is available on almost any mobile devices. There are several types of cellular phones which contain place for a micro secure digital card.

A. *Low complexity system design for stepping window method*

Fig. 10 shows the symbolic logic circuitry of MT on a 4x4 image. The input registers ($IR_{x,y}$, x = 1, .., 4; y = 1, .., 4) contain the values of pixels of the image while the output registers ($OR_{i,j}$, i = 1, .., 3; j = 1, .., max_*bin*_number$_i$) contain the values of bins. Three projection lines (1,1; -1,1; 1,2) gives an adequate set of bins and the set of projections meets the reconstruction criteria. The Mojette operator is the XOR logic operator and the different colour (greenscale) of the line means separate the three projection lines and their bins. Fig. 10 represents the MT of a 4x4 pixel size image. This size do not meet with real image sizes of coarse, even so the 4x4 pixel size window is usable in the stepping window method. The register size depends on the width of input data (byte, word, double word) but notice that the larger the data width the more hardware resources are required.

The symbolized logic circuit of the reconstruction (IMT) is depicted in Fig.11. The pixels can calculated in two ways. At first every pixel value calculable from *bin* values only and secondly a pixel value is calculable from *bin* values and the already calculated pixel values. The second version gives more simple (low-complexity) hardware. In Fig. 11 the input registers (IR) contain the *bin*-values, the output registers (OR) contain the original pixel-values, while the IMT operator is the XOR as in the MT was. The number of registers is the same in both cases (MT and IMT).

As a matter of fact the number of input registers can be smaller than the number of bins because of the redundancy.

The single pixel-*bin* correspondences (most of projection lines contain some) give the original pixel value without any calculation this is called zero level. Other bins contain more pixel values. The number of XOR operations need to be performed on them to get the original pixel value is the number of its level. In this case there are five levels from zero to four. In software solution it means a cycle ("for") from zero to five and every cycle contains another cycle ("for") from one to n_{bl} where n_{bl} is the number of bins on the same level. If the window is chosen larger the number of level will increase with it. Using the already calculated pixel values the complexity of the whole system will increase also of course. However increasing of complexity will be slower then when only the *bin* values are used in the calculations.

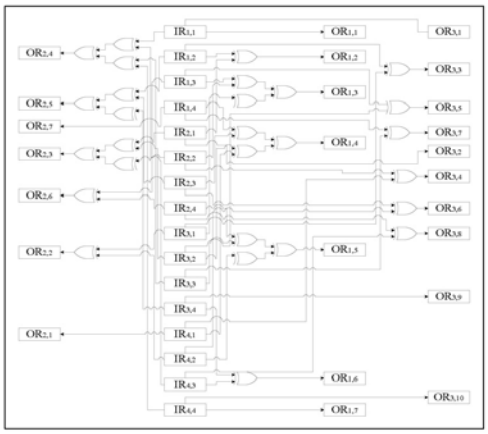

Fig.10. Logical model of the MT on a 4x4 image

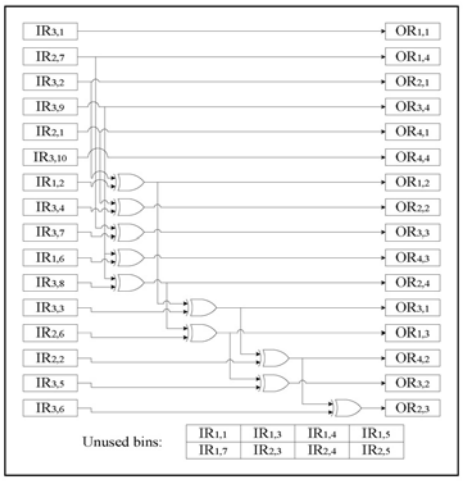

Fig.11. Symbolized logic circuitry of a 4x4 size image reconstruction

Definitely there is a window size limit (WSL) for every FPGA chip. The WSL is the block size what the given FPGA can process in parallel. The WSL depends on the number of logic cells in the given chip and other resources. To enlarge the window size above to the WSL more FPGA chips must be applied. Next figure (Fig.12) shows a MT/IMT computing system. The data source means image or other types of data pre-processing before the MT is not necessary. Post-processing after the MT can be any kind of lossless compression method to decrease the redundancy. Data sink can be a storage device (e.g. PC). In the decoding process the data source is the file of projection lines. Pre-processing is necessary (uncompress the files). Post processing is not necessary here. The data sink in this case is the user's application.

As it is shown in Fig. 12 when larger window size must chosen then one need to multiply the WSL hardware. The sub-picture limited by the broken line shows a multi-chip MT/IMT system. The distribution and merge of data (by the multiplexer and demultiplexer units) also managed by an FPGA chip or if the hardware resources of the FPGA chip allows it, can managed by the on-chip general purpose processor (eg.: the PPC in the Xilinx Virtex II Pro FPGA). A four FPGA solution of the stepping window method for MT is depicted in Fig. 13. The sub-window size is fit to the possibilities of the given FPGA and can not be larger than the WSL.

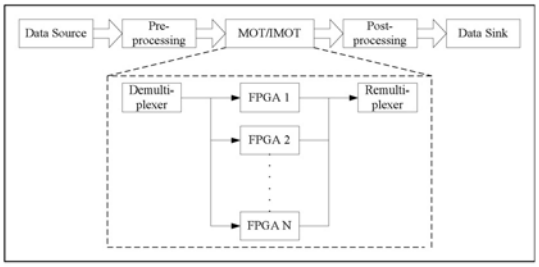

Fig. 12 MT/IMT system block-schema

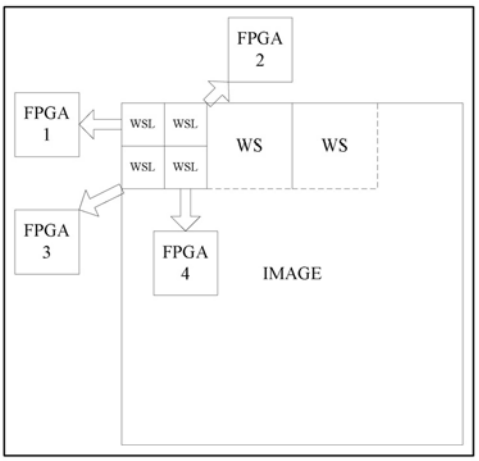

Fig.13. Stepping window method with four FPGAs

With this method the window size (area) equal to the window size limit (area) multiplied by four (if n FPGAs are in use of course the WSL will be multiplied by n). It results shorter computing time but larger power consumption. To find the balance between the power consumption and speed, it is necessary to known where the algorithm of the Mojette transform will be used.

B. Low complexity system design for sliding window method

The sliding window method differs to the stepping window method in its basic. At the stepping window method as it is suggested by its name, there are no common pixels of the windows in two neighbor steps. Contrarily the sliding window method moves the window only with one row or column (depends on the direction of processing) forward. It means most of the pixels are common in two windows, which are neighbors of each other.

Another difference compared to the stepping window method that while at the stepping window method the MT/IMT computing is one single step, the MT/IMT computing has two parts at the sliding window method. First part is to calculate the final value of the given *bins*/pixels and calculate the next temporal value of the other *bins*/pixels in the window and second part is to move the data: write out the final values, move the temporal values into the corresponding registers and read in new data.

Fig. 14 shows the symbolic logic circuitry of the sliding window method where $p_i = \{1, -1, 3, -3\}$ and $q_i = 1$. The virtual image size is defined by P and Q where Q = 4 and P = file size/Q. This means that the size of sliding window is chosen independently the file size. The size of sliding window is given by the hardware resources (number of logic circuits, memory, etc.) of the given FPGA.

Note if q is chosen larger (q=2) the logic circuitry and the size of sliding window (number of registers) will be multiplied by the new q value. When larger window size is required then the WSL, more FPGAs can be used. Four FPGAs using system is depicted in Fig. 15.

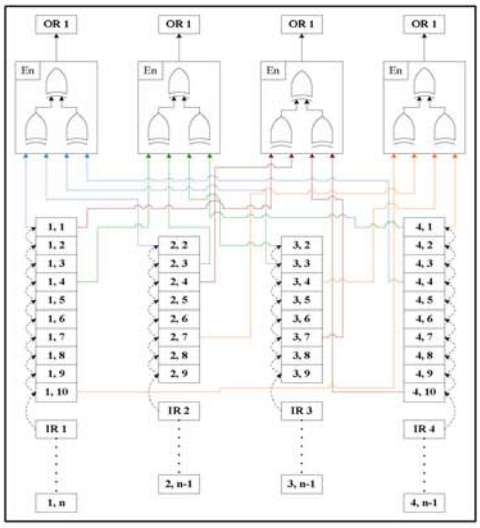

Fig. 14 Symbolic logic circuitry for sliding window method.

Each of the FPGAs contain the same logic circuitry and the size of WSL is the same but the virtual WSL of the whole system is four times larger then the WSL of an individual FPGA. Every FPGA processes a four word width data stream transparently, so the whole system processes a sixteen word width data stream. The projection coordinate q is equal to one in every projection of every FPGA, therefore the corresponding projection lines are merged into one line. Four FPGAs and four projection lines give sixteen projection lines. The merge of the mentioned projections will result four *bin* vectors where the resultant q coordinate is equal to four ($q_r = 4$). This way the permanently processed data width is multiplied by four. The IMT computing system will reconstruct the original data whether it contains only one FPGA (a larger one) or four FPGAs.

Fig. 16 shows the merge process after the MT computing. Every FPGA has four outputs (four *bin* vectors). As it is depicted in the above mentioned figure the corresponding *bin* vectors of the FPGAs are merged into one *bin* vector.

C. IMT

The logic circuitry of the IMT compared with the MT shows that the reconstruction process is more complicated. There are single-pixel *bin* correspondences which result pixel values very simply but the bins of every other projection line need correction. It is necessary, because the *bin* value corrections generate new single-pixel *bin* correspondences and ensures the continuity of the reconstruction process.

The symbolic logic circuitry of the IMT computing SLW co-processor is depicted in Fig. 17. The picture does not show the total system only a part of it. In the image Pi (i=1..8) means the reconstructed pixel values, PLi (i=1..4) are the projection lines, while the rectangles above them represents the bins of the projection lines. In the image "tr" means temporary register which are necessary for the second step, the *bin* value correction. The outputs of Unit 1 are the reconstructed pixel values and it has an enable input. Unit 2 gives the *bin* values after the correction and works with the same enable signal as the Unit 1. In the image (Fig. 17) the bit correction unit is depicted only for one projection line but the other three units are very similar. After the *bin* correction the new values are stored in temporary registers.

In the second phase the correct *bin* values from the temporary registers are written back to the original registers. The third phase is to slide the window. Move the values of the register forward by two places, and read in new data.

The MT and IMT functions are implemented as separate hardware co-processors of the main processor. This is possible in two ways. The main processor can be an off-chip or an on-chip solution. The implemented algorithms are realized as separate co-processors and they work either in parallel or using run-time reconfiguration (This method was not tested yet) using relatively low working frequencies (100-300 MHz). This way can be obtained very high processing speeds.

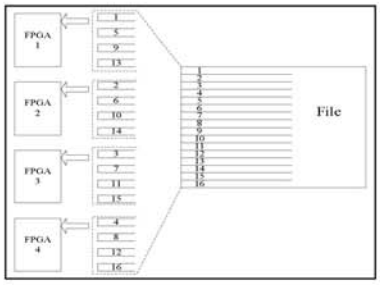

Fig.15. Sliding Window Method with four FPGAs.

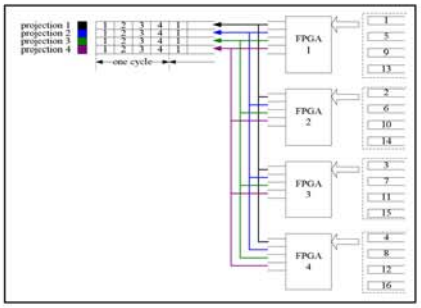

Fig.16. Merge process.

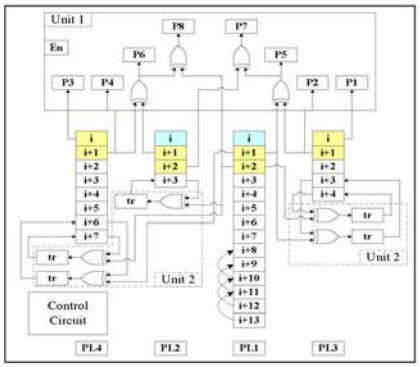

Fig.17. Symbolic logic circuitry for IMT (SLW)

VII. CONCLUSIONS

The paper outlined the different ways, how the MT and IMT is currently implemented in the MTTool. In the MTTool software version more tests should be performed to get more

accurate results, and possible directions should be introduced in near future how the MT should be implemented in a hardware platform. There was presented the hardware implementation of MT and IMT in FPGAs using parallel implementation of the Mojette algorithm. For complex 256x256 size images the MT/IMT can be implemented using multiple FPGA chips as MoTIMoT co-processor.

ACKNOWLEDGMENT

This work was partially supported from the grants VEGA 01/0045/10 and project COST IC0802.

REFERENCES

[1] Guédon J-P., Normand N.: The Mojette Transform: The First Ten Years, Proceedings of DGCI 2005, LNCS 3429, 2005, 79-91.

[2] Guédon J-P., Normand N.: Spline Mojette Transform Application in tomography and communication, In EUSIPCO, sep. 2002.

[3] Guedon, J. P., Parrein, B., Normand, N.: Internet Distributed Image Databases. Int. Comp. Aided Eng., Vol. 8, 2001, 205 – 214.

[4] Parrein B., Normand N., Guédon J-P.: Multimedia Forward Error Correcting Codes For Wireless Lan, Annals of Telecommunications (3-4) 448-463 March-April, 2003.

[5] Normand, N., Guedon, J. P.: La transformee Mojette: une representation recordante pour l'image, Comptes Rendus Academie des Sciences de Paris, Theoretical Comp. SCI. Section, 1998, 124 – 127.

[6] Katz, M.: Questions of uniqueness and resolution in reconstruction from projections. Springer Verlag, Berlin, 1977.

[7] Autrusseau, F., Guedon, J. P.: Image Watermarking for Copyright Protection and Data Hiding via the Mojette Transform, Proceedings of SPIE, VOL. 4675, 2002, 378 – 386.

[8] Turán, J., Ovsenik, L., Benca, M., Turán, J. Jr: Implementation of CT and IHT Processors for invariant Object Recognition System, Radioengineering, Vol. 13, No. 4 2004, 65-71.

[9] Xilinx, *ML403 User Guide*, http:\\xilinx.com/ml403

[10] Wu,J.-Olivier,C.-Chatellier,C.-Poussard,A-M.: Redundant Transformation on the Binary Data and Its Applications to Image Coding and Transmisssion, IEEE APCCAS 2000, Dec 2000, 763-766.

[11] Szoboszlai,P. – Turán,J.: Implementation of the Mojette Transform and HAAr Wavelet in .NET, Carpathian Journal of Electrical Engineering, Vol. 1, No. 1, 2008, 63-68.

[12] Serfőző, P., Vásárhelyi, J., Szoboszlai, P.: Test Result of Mojette Transform Implementation On Embedded PowerPC, Proceedings of 9th International Carpathian Control Conference ICCC 2008, May 25-28, 2008, 24-29.

[13] Serfőző, P., Vásárhelyi, J., Szoboszlai, P., Turán, J.: Performance Requirements of the Mojette Transform for Internet Distributed Databases and Image Processing, OPTIM 2008, Brasov, Romania, May 22-24,2008, 78-82.

[14] Szoboszlai,P.-Turán,J.–Vásárhelyi,J.-Serfozo,P.: Mojette Transform Based Real-Time Video Surveillance System In: Carpathian Journal of Electronic and Computer Engineering, Vol., 3,No.2, 2009, ISSN 1844-9689, 1-6.

[15] Szoboszlai, P., Turán, J., Vásárhelyi, J., Serfőző, P.: The Mojette Transform Tool and Its Feasibility, MACRo 2009, Târgu Mureş, Romania, Acta Universitatis Sapientiae, Electrical and Mechanical Engineering, ISSN 2065-5916, No.1, 2009, 163-174.

[16] Szoboszlai,P.-Vasarhelyi,J.-Turan,J.-Serfőző,P.: Mojette Transform implementation in software and hardware, ESDA'09

[17] Szoboszlai, P., Vásárhelyi, J., Turán, J., Serfőző, P.: MTTool Software Tool and Low Complexity Hardware for Mojette Transformation, Studies in Computational Intelligence, ISBN 978-3-642-15219-1, Springer-Verlag, Berlin, Heidelberg, Vol. 313, 2010, DOI: 10.1007/978-3-642-15220-7, 15-26.

AUTHORS BIOGRAPHY

Ján TURÁN (Prof., Ing., RNDr., DrSc.), was born in Šahy, in 1951. He received an Ing. degree in physical engineering from the Czech Technical University, Prague, in 1974 and a RNDr. degree in experimental physics from the Charles University, Prague, in 1980. He received a CSc. (PhD.) and DrSc. degree in radio electronics from the Technical University, Košice, in 1983 and 1992 respectively. Since March 1979, he has been at the Technical University of Košice, as a professor of electronics and telecommunications technology. His research interests include multimedia signal processing and fiber optics communications and sensing.

Péter SZOBOSZLAI (Ing.) was born in Miskolc,in 1980. He received his Ing.. degree in computer sciences from University of Miskolc, in 2006. From 2006 he worked at Ericsson Ltd., Budapest, in 2008 he joined Magyar Telekom PLC as IT manger. He started his PhD studies at the Technical University of Košice, in 2006 in the field of electronics and multimedia communications. His research interest include image processing, digital signal processing and applications of FPGA.

József VÁSÁRHELYI (ass. prof, Ing., PhD.) was born in Cluj, in 1958. He received his Ing.. degree in electronics and telecommunications from Technical University of Cluj, in 1983. He received a PhD. degree in the field of electronics at Technical University of Cluj in 2004. He joined the Technical University of Miskolc since 1992. His research interest includes applications of FPGA, dynamic reconfiguration systems and their applications, system on chip and embedded systems.