A Novel Architecture for Motion Estimation

Subarna Chatterjee[1]

[1] **Department of IT, Bengal Engineering and Science University,**
Shibpur , Howrah – 711103, WB, India

Abstract

The work of motion estimation is highly used in the surveillance applications especially for military activities. One of the key elements of many video compression schemes is motion estimation, for the removal of video temporal redundancy. This paper presents an architecture for motion estimation using modified Full Search BM algorithm eliminating the SAD distortion criterion. The quality of the algorithm used was compared with Full Search through software implementations. The quality of BMA results was considered satisfactory, The designed hardware considered a search range of [-25, +24], with blocks of 16x16 pixels. The architecture was described in VHDL and mapped to a Xilinx Virtex-II Pro VP70 FPGA.

Defining efficient techniques for video processing is of special interest due to the existence of a wide variety of applications in the fields of entertainment, computer vision, surveillance, security etc. The different techniques are mainly compared in the terms of algorithmic efficiency, hardware requirement, processing speed and error performances. In this paper we have designed a new parallel processing architecture to perform the video analysis for motion estimation. We have also done the simulation and FPGA based synthesis of the proposed architecture for the most commonly used target hardware to analyze the hardware cost.

Keywords: *Motion Estimation, Macro Blocks, Zero Matrix, Edges, Frame.*

1. Introduction

During warfare or in animal world, the fighter or the hunter always tries to protect himself by adopting slow or fast movements and hides against opponents by adopting camouflaging and decides and acts for sudden attack at an ultimate moment; and here comes the motivation to take remote pictures of the environment and analyze the pictures, identify the presence of motion, if any, and to conclude that there must be an enemy, if there is motion. Analysing the motion of the pictures, an appropriate action is taken, depending on the generated signal. To implement the process, we need architectures which estimate the motion of the pictures.

Already several papers have been published proposing different architectures based on the Full-Search Block-Matching (FSBM) [1,2,13-17] motion estimation (ME) algorithm which is the most popular algorithm for ME and also demands most computation. H.Yeo et all [1] proposed pipelined systolic array architecture where they have achieved high throughput rate but the hardware complexity is also high. L.C.Liu et all [17] proposed a Frame-level pipelined FSBM architecture, where the search range increases and this increases the required number of computation and storage space; this will result in an increase in processing time and memory bandwidth. Some authors proposed block-level [2,14] pipelined architectures which have many disadvantages over frame-level pipelined architectures in their complex control, increased number of memory accesses , reduced data reuse, reduced processor utilization ratio. All the proposed architectures based on FSBM algorithm are scalable with the search range and depend on the value of the search range. The problem with FSBM ME algorithm is its requirement of large number of search data accesses and most search areas are overlapped. If the search range is set half of the block size, significant reduction in input-output bandwidth can be achieved without sacrificing performance.

The task of motion estimation is used in many areas and it requires heavy computation which has created a need for developing new ways of implementations, such that, it decreases the time required for the application. The aim is to propose a ME algorithm which will reduce the search data access and hence the required number of computation; and propose a suitable architecture model for ME, considering highest throughput, low pin-count, low memory bandwidth and reduce computational complexity to handle huge amount of data by modifying the existing FSBM algorithm. With this view the work focused on the simulation of the ME algorithm in a parallel logic and also tries to define an efficient architecture model for ME which will parallelize the task of ME so that multiple blocks can be processed simultaneously keeping in mind the number of processors required and also the processing time needed and simulate the hardware design.

In this paper, the video or image is captured using an image sensor and stored as successive frames in a video file format. The set of frames is then processed to detect the motion of the moving objects. A proposal for hardware implementation of the algorithm is made and architectural synthesis of the same for different board is tested. Performance and implementation cost of the parallel

memory architecture are estimated and compared to some previous memory architectures. The parallel memory can keep the data path fully utilized in video processing function implementations. This ensures high-speed operation and full utilization of the processing resources.
Main Results:
The organization of the paper is as follows. In Section II the Review of Previous work of ME has been done. In Section III we have defined edge detection algorithm. Methodology of motion analysis with its hardware estimation is being described in Section III and Section IV. Performance analysis and comparison with the other related works is described in Section V and concluding remarks in Section VI.

2. Review of Previous work of Motion Estimation

The work of ME is highly used in the surveillance systems of the defense department. In systems such as MPEG[4], ME[12][18], eliminates temporal redundancies in image sequences and thus accounts for most of the compression. To perform ME, the Full Search Block Matching (FSBM) [1][13][14][15] algorithm is preferred because of it's optimal precision, simplicity and low control overhead.
Block-based ME technique is being widely used in video compression applications, for the removal of video temporal redundancy. The six-level nested Do-loop Full-Search Block-Matching ME algorithm proposed by H.Yeo and Y.H.Hu[1] is as shown below.

```
do v = 1 to $N_v$
do h = 1 to $N_h$
    MV (h, v) = (0, 0);
    $D_{min}$ (h, v) = $\propto$;
    do m = -p to p
    do n = -p to p
        MAD (m, n) = 0;
        do i = 1 to N
        do j = 1 to N
            MAD (m, n) = MAD (m, n) + | x ((h-1)N + i, (v-1)N + j) – y ((h-1)N + i + m, (v-1)N + j + n)|;
        enddo j, i ;
        if $D_{min}$ (h, v) > MAD (m, n)
            $D_{min}$ (h, v) = MAD (m, n);
            MV (h, v) = (m, n);
        end if ;
enddo n, m, h, v ;
```

A typical video frame consists of $N_h \times N_v$ blocks of pixels where N_h is the number of $N \times N$ blocks in each row and N_v is the number of block-level rows in each frame. The value of N, which is the macroblock size, is arbitrary. The

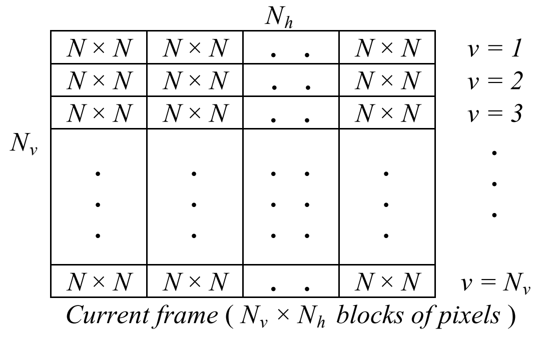


Fig. 1 Current frame block distribution.

value of N is 8 or 16 generally. Figure 1 shows the distribution of blocks in the current frame.
For ME each image block in the present frame is to be matched to an $N_h \times N_v$ region in a reference frame. Each of these blocks has to be considered individually and block matching computation has to be performed in a predefined search area in the reference frame. The search is restricted to a specific area in the reference frame which is called search area. The search range is [- p, p] where $p \le N$. The block matching algorithm has to be applied for each block in the current frame. For each block first the motion vector (MV) of that block is initialized to 0 and $D_{\min}$ value for that block to $\propto$. The search range for this block is as shown in Figure 2.

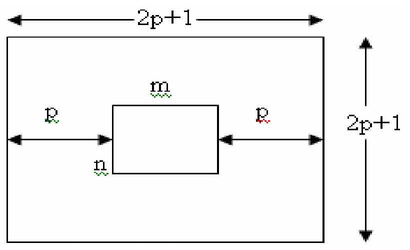


Fig. 2 Search range for each block.

Next the value of two variables m and n has to be varied from $-p$ to p. For each value of m and n we initialize the value of the variable MAD for that m and n combination to 0. As seen in Figure 2 there will be $(2p+1)^2$ candidate blocks. So, for each combination value of m and n we will consider one among the $(2p+1)^2$ candidate blocks. For the block in the current frame we will compute the MAD (Mean Absolute Distortion) value with each of these candidate blocks. So, we will get total $(2p+1)^2$ number of MAD values. The function MAD in Eq (1) is as below:

$$MAD(m,n) = \frac{1}{N}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}| x(i,j) - y(i+m, j+n) |$$

Here, $x(i,j)$ denotes the luminance pixels of the current frame and $y(i+m, j+n)$ denotes the luminance pixels of the reference frame.

After the complete iteration of the do loops the variable $D_{\min}$ for that block will contain the lowest MAD value among the $(2p+1)^2$ MAD values. The variable MV for that block will contain the values of m and n for which give the best matching block, i.e., the candidate block with lowest MAD value. The values of m and n will denote displacement of the block in the current frame with respect to the reference frame, which is known as motion vector. Total $N_h \times N_v$ number of motion vectors is generated for a frame. With help of these motion vectors we can easily construct or predict the current frame.

For each position value of MB the algorithm consider $(2p+1)^2$ candidate blocks. For the block in the current frame it will compute the MAD (Mean Absolute Distortion) value with each of these candidate blocks that is, total $(2p+1)^2$ number of MAD values computation. In general case considering the search range [-16,16] the total number of MAD values computation needed is $(2p+1)^2 = (2 \text{ x } 16 + 1)^2 = 33^2 = 1089$.

Several papers published before different architectures are proposed which are based on the FSBM[1][13][14][15] ME algorithms. These solutions are able to find the optimal results in terms of blocks matching. However, this type of architecture uses a very high amount of hardware resources due to this large number of computation. To reduce the hardware resources cost the complexity of the Full Search has to be reduced with little losses in the results quality.

The objective is to propose a ME algorithm which will reduce the number of computation and not consider the $(2p+1)^2$ candidate blocks for each MB; which result in improvements in performance, storage space and computational complexity.

3. Edge Detection

Edges are often considered as primary image artifacts for extraction by low-level processing techniques, and the start point for many computer vision techniques. Edge detecting an image significantly reduces the amount of data and filters out useless information, while preserving the important structural properties in an image.The Canny edge detection algorithm [3][5][6] is a commonly tused technique for optimal edge detection. It utilizes some improved techniques which results in: (i) low error rate,(ii well localized edge points,(iii)only one response to a single edge etc.

3.1 Steps for Edge Detection

Step1: The initial image is smoothened by convolving with a Gaussian filter with specified standard deviation σ, to reduce noise. Mathematically we can write the 2-D convolution [6] as:

$$O(i,j) = I(i,j) \otimes K(k,l)$$
$$= \sum_{k=1}^{m}\sum_{l=1}^{n} I(i+k-1, j+l-1)K(k,l) \qquad (2)$$

If the image has M rows and N columns, and the kernel has m rows and n columns, then the size of the output image will have M - m + 1 rows, and N - n + 1 columns. where:
K(k,l) = convolution kernel; I(x,y) = original image; O(x,y) = filtered image;
m by n= size of convolution kernel

Step2: The gradient of the image is obtained by feeding the smoothened image through a convolution operation with the derivative of the Gaussian in both the vertical and horizontal directions. $G=(G^2_x+G^2_y)^{1/2}$ and edge direction $\theta=\tan^{-1}(G_y / G_x)$ computed at each point.

Step3: Edge point is defined to be a point whose strength is locally maximum in the direction of the gradient.

Step4: Non-maximum suppression is used to trace along the edge in the edge direction and suppress any pixel value (sets it equal to 0) that is not considered to be an edge.

3.2 Proposed Steps for Motion Detection

Canny edge detection algorithm is used to detect the edges of the image frames. Edges of the frames are detected and saved, which is then compared with the initial frame to find the rate of change of information followed by the predicted velocity. The steps of the proposed algorithm can be briefed as follows:

1. Video is captured, and stored as a video file.
2. Conversion of each JPEG (RGB) frame into a gray scale image.
3. Detection of the edge pixels in the successive frames.
4. Divide each image into a set of MBs.
5. Storing the index of the zero MBs i.e. the MBs which doesn’t have any edge pixels, for each frame.

6. Finding the number of similarities in the index values between the frames. From this we can calculate the difference in the motion information for each movie.

3.3 Result Analysis for Motion Detection Algorithm

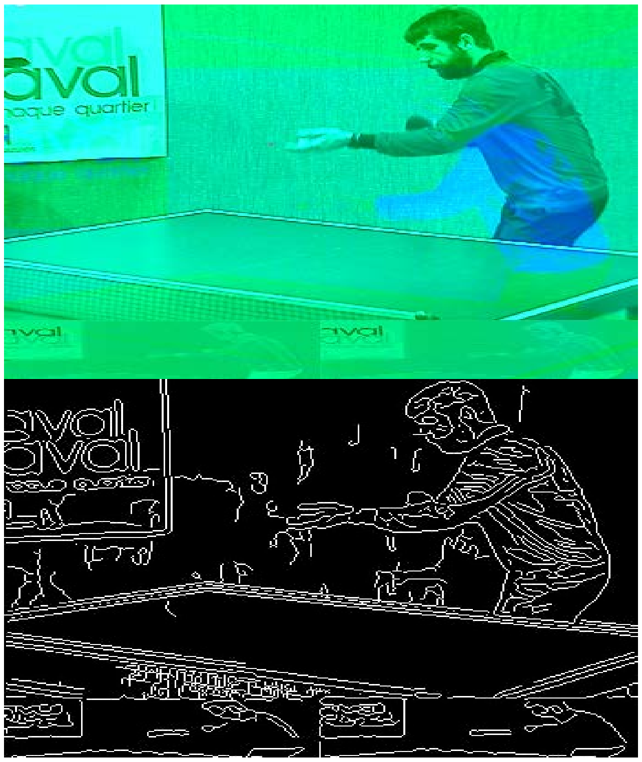


Fig. 3 Upper figure shows the 1st frame of the video file and lower figure the frame after edge detection.

Validity of the proposed Motion detection algorithm is simulated using MATLAB 7.1 from Mathworks Inc[11]. Here 112 frames of a video file have been taken. Each frame is a still image of pixel size 288 × 350. The first frame has more similarity with the second frame than with the one hundred and twelfth frame in terms of same index repetition. The number of zero MB(16 ×10) for the first frame is 60,and for second frame is 67 out of 630 MB, same index repeat on 58 number of MB; so difference is [(60 - 58)+(67 - 58)] = 11 number of MB for first frame and second frame. The number of zero MB(16x10) for 112th frame is 80 same index repeat on 29 number of macro blocks so difference is [(60 - 29) + (80 -29)] = 82 number of MB. This difference in information concludes that there exists motion of the object in the movie.

4. Hardware Estimation

4.1 Hardware Estimation Of Canny Edge Detection Algorithm

From Eq (2) putting i=j=1 we get,

$$O(1,1) = \sum_{k=1}^{m} \sum_{l=1}^{n} I(k,l)K(k,l)$$

Now putting m=n=5 that is the size of the convolution kernel we get,

$$O(1,1) = I_{11}K_{11} + I_{12}K_{12} + I_{13}K_{13} + I_{14}K_{14} + I_{15}K_{15} + I_{21}K_{21} + I_{22}K_{22} + I_{23}K_{23} + I_{24}K_{24} + I_{25}K_{25} + I_{31}K_{31} + I_{32}K_{32} + I_{33}K_{33} + I_{34}K_{34} + I_{35}K_{35} + I_{41}K_{41} + I_{42}K_{42} + I_{43}K_{43} + I_{44}K_{44} + I_{45}K_{45} + I_{51}K_{51} + I_{52}K_{52} + I_{53}K_{53} + I_{54}K_{54} + I_{55}K_{55}$$

$$= Y_{11}+Y_{21}+Y_{31}+Y_{41}+Y_{51}$$

Where,

$$Y_{11} = I_{11}K_{11} + I_{12}K_{12} + I_{13}K_{13} + I_{14}K_{14} + I_{15}K_{15}$$
$$Y_{21} = I_{21}K_{21} + I_{22}K_{22} + I_{23}K_{23} + I_{24}K_{24} + I_{25}K_{25}$$
$$Y_{31} = I_{31}K_{31} + I_{32}K_{32} + I_{33}K_{33} + I_{34}K_{34} + I_{35}K_{35}$$
$$Y_{41} = I_{41}K_{41} + I_{42}K_{42} + I_{43}K_{43} + I_{44}K_{44} + I_{45}K_{45}$$
$$Y_{51} = I_{51}K_{51} + I_{52}K_{52} + I_{53}K_{53} + I_{54}K_{54} + I_{55}K_{55}$$

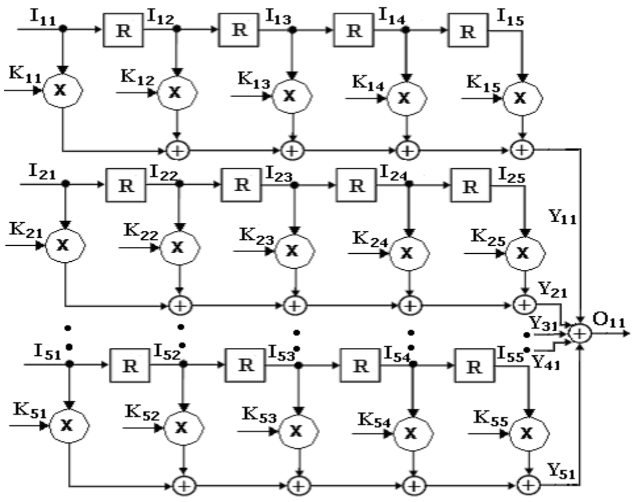


Fig. 4 Parallel architecture of Gaussian filter.

Figure 4 shows the hardware estimation of the Gaussian Filter, which is the core of the Canny Edge Detection Algorithm. The filter comprised of four delay operators(4 D FF), five multiply operators, and four addition operators. Figure shows a fully-parallel hardware architecture for this algorithm. The fully-parallel architecture simply maps each arithmetic operator in the algorithm into a separate hardware element. From figure we see execution take $(T_m+4T_a+T_a) = T_m+5T_a$ clock cycle, where T_a = addition time and T_m = multiplication time, let T_m =1ns and T_a = 1ns;then T_m+5T_a = 1+5 = 6 ns when the size of kernel is 5 by 5 matrix. So in general case when it is of order n then this architecture need n+1 clock cycle.

Similarly using the ITU-R601 format (720×576 at 25fps) each frame is of size 576×720 , it get the 1st sample in ($T_m+4T_a+T_a$) clock cycle;

$$O(i,j) = O(1,1) + O(1,2) + O(1,3) + \ldots\ldots + O(1,575) + O(1,576) +$$
$$O(2,1) + O(2,2) + \ldots\ldots + O(2,575) + O(2,576) +$$
$$O(3,1) + O(3,2) + \ldots\ldots + O(3,575) + O(3,576) +$$
$$O(4,1) + O(4,2)) + \ldots\ldots + O(4,575) + O(4,576) +$$
$$\ldots\ldots + O(720,1) + O(720,2) + \ldots\ldots + \; + O(720,576) +$$
$$O(720,1) + O(720,2) + \ldots\ldots + \; O(720,576)$$

$$O(1,575) = I_{1575}K_{11} + I_{1576}K_{12} + I_{2575}K_{21} + I_{2576}K_{22} + I_{3575}K_{31} + I_{3576}K_{32} + I_{4575}K_{41} + I_{4576}K_{42} + I_{5575}K_{51} + I_{5576}K_{52}$$

$$O(1,576) = I_{1576}K_{11} + I_{2576}K_{21} + I_{3576}K_{31} + I_{4576}K_{41} + I_{5576}K_{51}$$

$$O(719,1) = I_{7191}K_{11} + I_{7192}K_{12} + I_{7193}K_{13} + I_{7194}K_{14} + I_{7195}K_{15} + I_{7201}K_{21} + I_{7202}K_{22} + I_{7203}K_{23} + I_{7204}K_{24} + I_{7205}K_{25}$$

$$O(720,1) = I_{7201}K_{11} + I_{7202}K_{12} + I_{7203}K_{13} + I_{7204}K_{14} + I_{7205}K_{15}$$

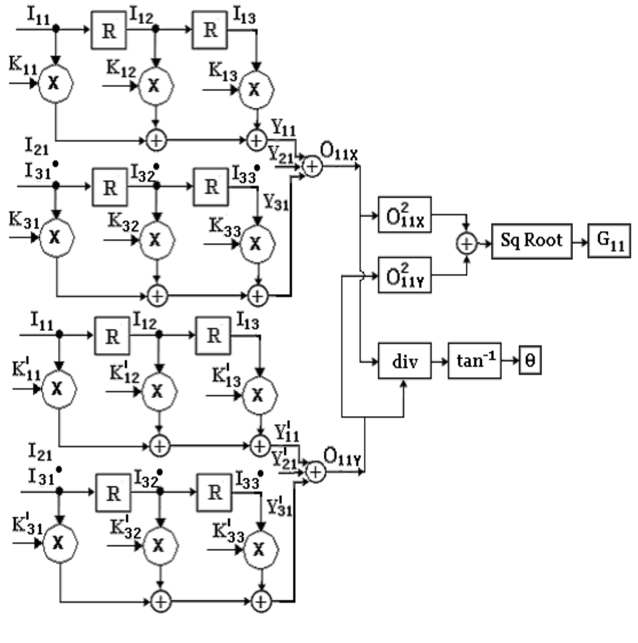


Fig. 5 Gradient and Edge direction calculation.

Considering $T_m = T_a = 1$ clock cycles output of the 1st frame that is 1st sample is available at 6 + (576-1) = 581 clock cycles; hence Sampling time ≥ 581 clock cycles or ≥ $T_m + 5T_a + (576-1)$ sec.

Figure 5 shows the proposed architecture of the gradient and edge direction calculation. The filter comprised of two delay operators (2 D FF), three multiply operators, and three addition operators for gradient calculation in the horizontal direction and same elements are used for gradient calculation in the vertical direction. The spatial gradient amplitude $G = (G^2_x + G^2_y)^{1/2}$ and edge direction $\theta = \tan^{-1}(G_y / G_x)$ computed at each point. The fully-parallel architecture simply maps each arithmetic operator in the algorithm into a separate hardware element. From figure we see execution take $(T_m+2T_a+T_a+T_m+2T_a) = 2T_m+5T_a$ clock cycle, where T_a = addition time and T_m = multiplication time, let T_m =1ns and T_a = 1ns;then $2T_m+5T_a$ = 2+5 = 7 ns when the size of kernel is 3 by 3 matrix. So in general case when it is of order n then this architecture need 2n+1 clock cycle.

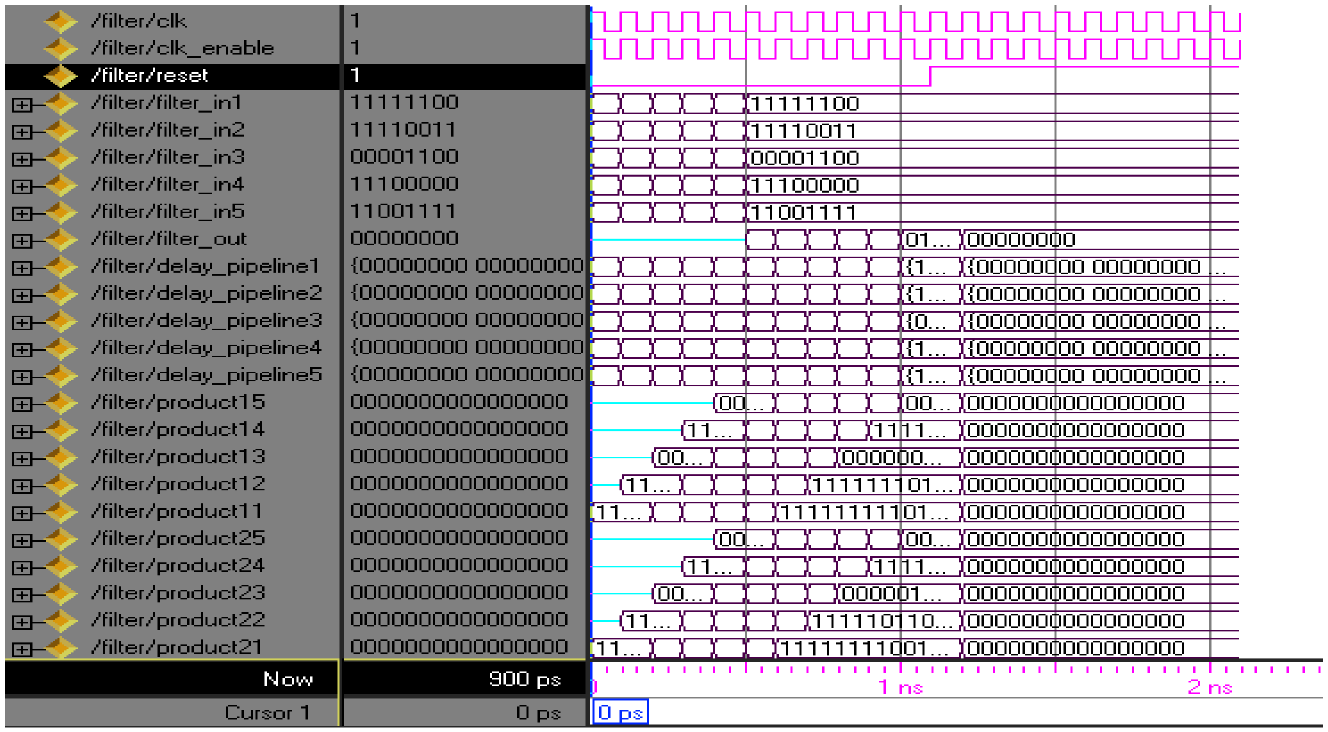
4.2 Result Analysis of Gaussian filter

Fig 6 shows the simulation output of the Gaussian filter using Xilinx-ISE6 and Modelsim 6.1 simulator. At first clock product $I_{11}K_{11}$, $I_{21}K_{21}$, $I_{31}K_{31}$, $I_{41}K_{41}$, $I_{51}K_{51}$ is calculated; second clock product $I_{12}K_{12}$, $I_{22}K_{22}$, $I_{32}K_{32}$, $I_{42}K_{42}$, $I_{52}K_{52}$ is calculated; similarly at third ,fourth and fifth clock product $I_{13}K_{13}$, $I_{23}K_{23}$, $I_{33}K_{33}$, $I_{43}K_{43}$, $I_{53}K_{53}$,$I_{14}K_{14}$, $I_{24}K_{24}$, $I_{34}K_{34}$, $I_{44}K_{44}$, $I_{54}K_{54}$,$I_{15}K_{15}$, $I_{25}K_{25}$, $I_{35}K_{35}$, $I_{45}K_{45}$, $I_{55}K_{55}$ is calculated. At sixth clock the first filter output is generated from the filter_out port. After that at every clock the output is generated.

Table 1: Device utilization summary

	Spartan-3 S50	**Virtex – v50**	**Virtex-II V2000**	**Virtex-II Pro VP70**	**Virtex-IV VFX60**
# Slices	372 (48%)	787(102%)	372 (3%)	372 (1%)	350 (1%)
# Slice FFs	240 (15%)	240(15%)	240 (1%)	240 (0%)	200 (0%)
# 4 input LUTs	498 (32%)	1283(83%)	498 (2%)	498 (0%)	498 (0%)
# bonded IOBs	50 (40%)	50 (27%)	50 (8%)	50 (5%)	50 (12%)
# GCLKs:	25 (625%)	1 (25%)	25 (44%)	25 (7%)	1 (3%)
Memory Usage	76804 KB	78196 KB	113584KB	217284KB	238472KB

The percentage value in the parenthesis shows utilization ratio of the amount of available resources. The comparison of the synthesis results of the proposed parallel architecture of the Gaussian filter taking different device family are being summarized in Table 1. Table 1 shows the usage of different blocks in FPGA ie LUT, RAMs. The comparison of the synthesis results taking different Device-Family shows that the designed architecture utilizes maximum resource for Spartan-3 device.



```
/filter/clk                  1
/filter/clk_enable           1
/filter/reset                1
/filter/filter_in1           11111100
/filter/filter_in2           11110011
/filter/filter_in3           00001100
/filter/filter_in4           11100000
/filter/filter_in5           11001111
/filter/filter_out           00000000
/filter/delay_pipeline1      {00000000 00000000
/filter/delay_pipeline2      {00000000 00000000
/filter/delay_pipeline3      {00000000 00000000
/filter/delay_pipeline4      {00000000 00000000
/filter/delay_pipeline5      {00000000 00000000
/filter/product15            0000000000000000
/filter/product14            0000000000000000
/filter/product13            0000000000000000
/filter/product12            0000000000000000
/filter/product11            0000000000000000
/filter/product25            0000000000000000
/filter/product24            0000000000000000
/filter/product23            0000000000000000
/filter/product22            0000000000000000
/filter/product21            0000000000000000
Now                          900 ps
Cursor 1                     0 ps
```

Fig. 6 Simulation output of Gaussian filter.

4.3 Architectural connectivity of Motion Detection Algorithm

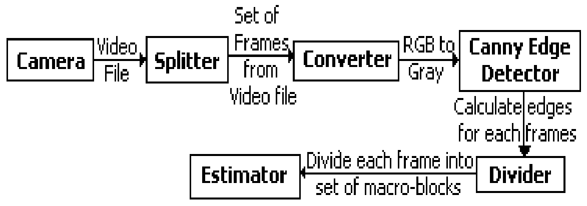

Camera
Video File
Splitter
Set of Frames from Video file
Converter
RGB to Gray
Canny Edge Detector
Calculate edges for each frames
Divider
Divide each frame into set of macro-blocks
Estimator

Fig. 7 Block Diagram of Motion Detection.

Figure 7 shows the basic architectural blocks of motion detector hardware. Capture the video file by camera. Split the .avi file into set of frames, now convert the RGB image to gray scale image. Calculate the edges by Canny edge detector .Divide each frame into set of macro-blocks. When control signal (c) of the estimator is zero then test for equality for each macro-block with zero matrix and store the index of zero macro-blocks(16×16) for each frame. Then count the number of zero matrix(16×16) for each frame. When control (c) is 1 then Check if same index repeat for the successive frames and calculate the difference.

Using the ITU-R601 format (720×576 at 25fps) each frame is of size 576×720.We divide it into 256 MB each of size 36×45. Now we have to test for equality with zero matrix(36×45) and count the number of zero MB. Figure 6 is the algorithm for equality test of one MB of size $N_v \times N_h$ of a frame with zero MB of same size.

```
do i = 0  to $N_v$
    do j = 0  to $N_h$
        if  zero( i,j) == m(i,j)
            p = p+1;
        endif;
    enddo j;
enddo i;
if p == $N_v \times N_h$
   q=1;
else q=0;
end;
```

Fig. 8 Algorithm for equality test of one MB of size $N_v \times N_h$

Considering N_v =36 and N_h=45 the processing of j loop can be implemented in an estimator with 2 ALU objects and will execute in two clock cycles. For processing for a 36 × 45 macro-block is implemented as a single serial processing stage (minimum resource approach), it will execute in 90 clock cycles. For next checking it takes 2 clock cycles.

Figure 9 demonstrates the proposed internal architecture of the Estimator block, which utilizes multiple processing elements. Here we have taken 256 processing elements. Each Processing element (P_1,P_2, P_3,.....P_{256}) has one memory element M(Each memory consists of 36register (R_1,R_2,R_3…R_{36})), 36Comparator to test the equality and store the result in the registers (I_1,I_2,I_3…I_{36}),1Adder for addition and store the result in register $R\_{}_1$,1Comparator to test the equality and store the result in the $I\_{}_1$ register. At the end of each cycle the sum of all the $I\_{}_1$ registers are being calculated and are store in the counter register (count).

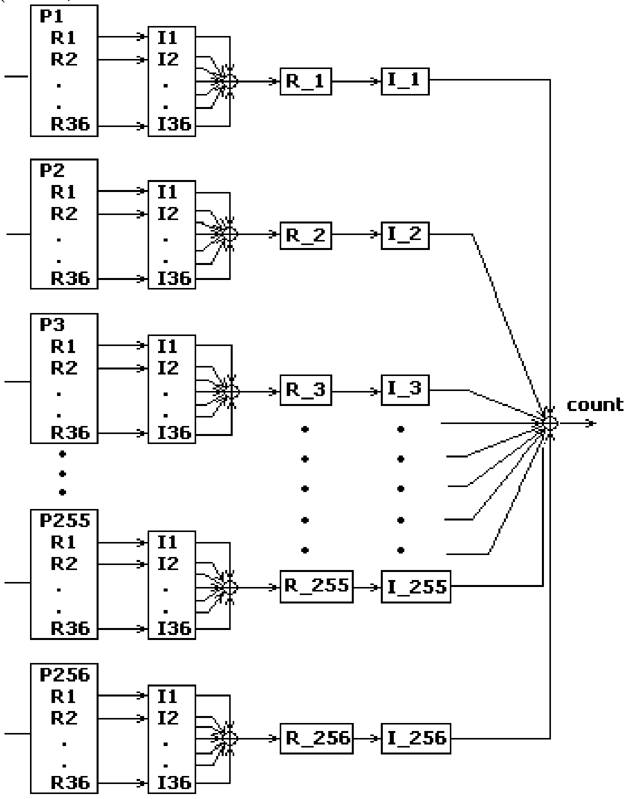


Fig. 9 Block Diagram of Estimator.

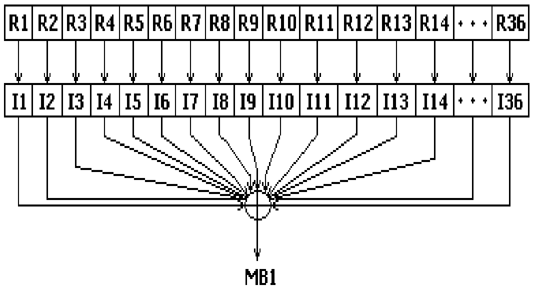

R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14 ··· R36
I1 I2 I3 I4 I5 I6 I7 I8 I9 I10 I11 I12 I13 I14 ··· I36
MB1

Fig. 10 Parallel pipeline processing approach.

Figure 10 is the block diagram Of the steps required for comparison of each macro-blocks(36×45) of a frame with zero matrix using parallel approach in the first execution phase on a processing element(P). 1st row pixels of the macro blocks(36×45) of the frame enter the registers (R_1,R_2…R_{36}). They are compared with a constant zero value, if equal then registers (I_1,I_2, I_3…I_{36}) will be set to one otherwise they will be set to zero. Count the number of zero macro blocks for each frame. We have to check whether the value of register $R\_{}_1$,$R\_{}_2$,$R\_{}_3$,…$R\_{}_{256}$ (MB1 in figure 5) is 1620 then set $I\_{}_1$,$I\_{}_2$,$I\_{}_3$,….$I\_{}_{256}$ to one otherwise set $I\_{}_1$,$I\_{}_2$,$I\_{}_3$,….$I\_{}_{256}$ to zero. Lastly count the sum of $I\_{}_1$,$I\_{}_2$,…$I\_{}_{256}$. For one frame it takes 3+ (45-1)+2 = 49 clock cycles.

The proposed architecture in Figure 9 employs 256 processing elements The number of clock cycles required is 49 for calculating the index and number of zero macro-blocks for each frame. Here we have taken 25 frames for motion detection so total clock cycles required is 25×49 =1225 for calculating the index and number of zero macro-blocks for the movie. Checking of same index repetition for the 1st frame and 2nd frame take 2 clock cycles. As the execution phase occur in parallel in pipeline manner so the total checking takes 1225 clock cycles to calculate the difference. So total clock cycles required depends on the number of frames and the size of frames in the video-file.

4.2 Improvements from Design Point of View

To detect only moving objects, frame subtraction is performed. Since there is not enough space to store past frames entirely, we store only the zero MBs of the 1st Frame after ED.

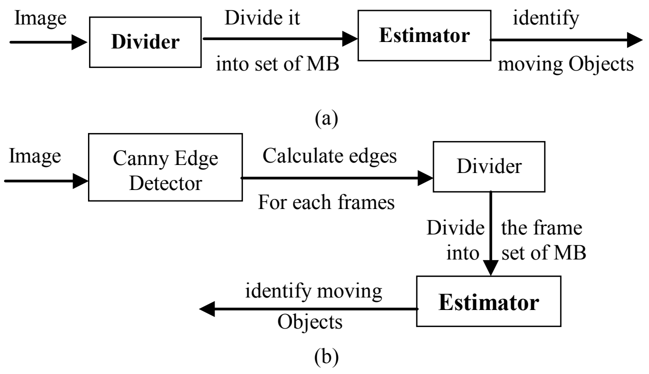


Fig. 11(a) Block Diagram of the FSBM algorithm for ME (b)Block diagram of the proposed ME algorithm.

ME[12][18] with this modified algorithm the distortion criterion is not calculated for all samples. In this method, for each pixels calculated, only the zero MB pixels are considered and the rest of the pixels are discarded. Results shows that with this algorithm less than a quarter of the block samples are calculated, increasing the performance and decreasing the complexity of the ME operation.

```
do k = 0 to $N_v \times N_h - 1$
   do i = 0 to N-1
      do j = 0 to N-1
         MB (k, i, j) = I (i + m , j + n)
```

```
            enddo j;
            n=n+N;
        enddo i;
        m=m+N;
    enddo k;
```

Time Complexity for division of a frame into a set of MBs $= T(n\times n)$
$= N_h \times N_v \times N \times N$
Considering image size of 8×8 matrix and size of each MB of 2×2 and number of MB 16,
$T(8\times8) = 4 \times 4 \times 2 \times 2$
$= 16 \times 4$
$= 64$

Time Complexity for execution of the **FSBM algorithm for ME** used by the estimator $=T(n\times n)$
$= N_h \times N_v \times (2p+1)^2 \times N \times N$
Considering image size of 8×8 matrix and size of each MB of 2×2 and number of MB 16 ,
$T(8\times8) = 4 \times 4 \times (2p+1)^2 \times 2 \times 2$
$= 16 \times (2 \times 2 + 1)^2 \times 4$
$= 64 \times (4 + 1)^2$
$= 64 \times 5^2$
$= 64 \times 25 = 1600$
Total time complexity in this case is = 64+1600 clock cycles;

Time Complexity of the **proposed ME algorithm** $= T(n\times n)$
$= N_h \times N_v \times N \times N$
Considering image size of 8×8 matrix and size of each MB of 2×2 and number of MB is 16,
$T(8\times8) = 4 \times 4 \times 2 \times 2$
$= 16 \times 4$
$= 64$
Total time complexity in this case is
= 13 + 64 + 64 clock cycles;
= 64 + 77 clock cycles;

Ratio of total time is *1600:77* = 20.78:1 ≈ 21:1. So the number of computation and the processing time required for ME can be reduced by applying this ME algorithm.

4.4 Result Analysis of Estimator

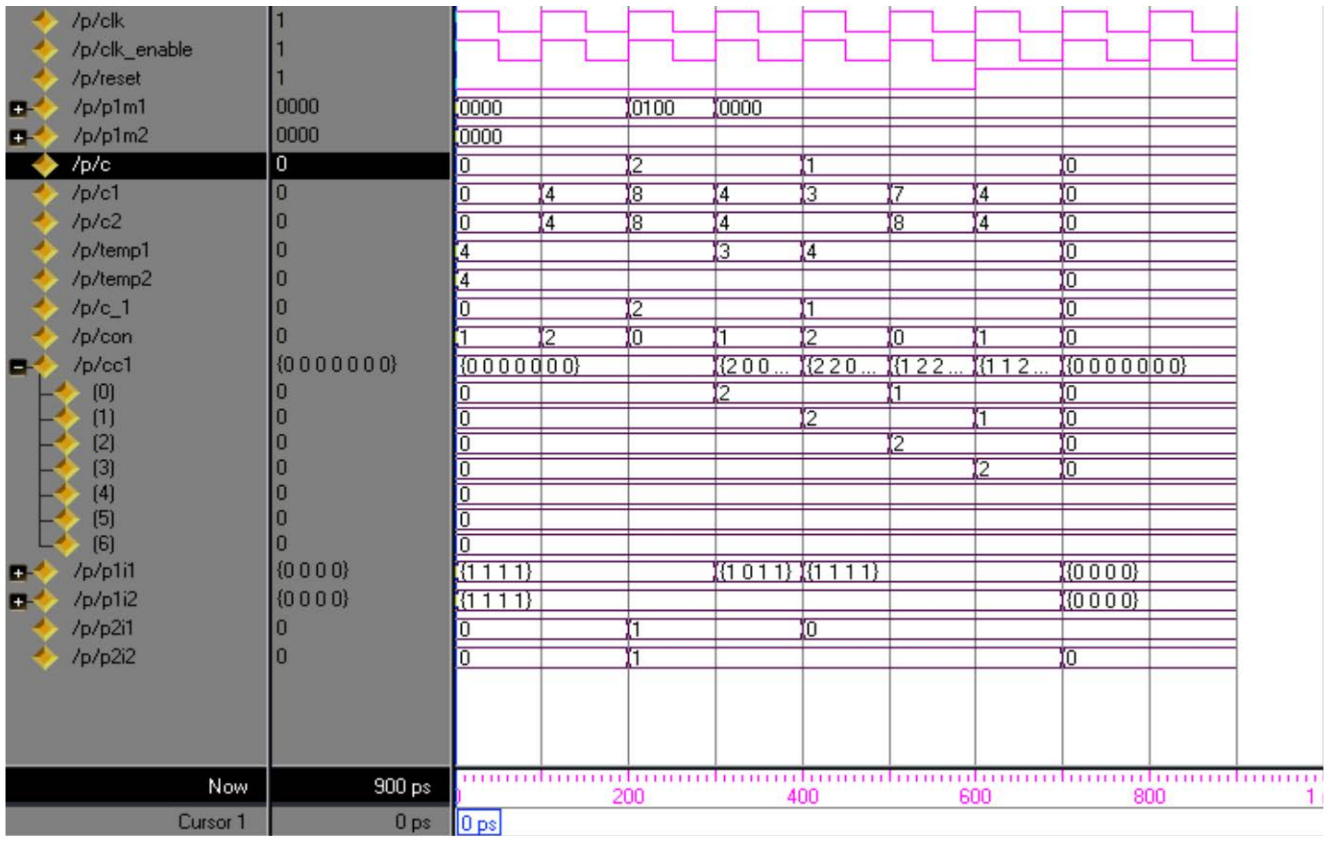


Fig. 12 Simulation output of the estimator.

Figure 12 shows the simulation output of the estimator using Modelsim 6.1 simulator. At first clock memory elements (p_1m_1,p_1m_2) get the data of the first row of the 4×2 matrix. It compare the four data value of p_1m_1 and p_1m_2 with a constant zero if tested result is equal then register p_1i_1 and p_1i_2 is set to 1 otherwise it is set to zero, both p_1i_1 and p_1i_2 are 4×1 matrix. Now we add the four value of p_1i_1(0,1,2,3) and p_1i_2(0,1,2,3) and store the value in temp1 and temp2.At the 2nd clock we get the number of zero MB for the 1st row of the 4×2 MB here it is 4 and we give the input for the next row of the 4×2 matrix p1m1 and p1m2. At the 3rd clock we get the number of zero MB for each 4×2 MB here it is 8.Now as here we take the frame size as 4×4 and divide it in two 4×2 MB. At 3rd clock cycle we get the result of the comparison of the value of c_1 & c_2 with 8 if it is 8 then $p_2i_1$and p_2i_2 is set to one otherwise they are set to zero, c is the counter register which count the total number of zero MB for each frame here it is two for the first frame and 1 in the 2nd frame. So the same index repeats for one MB i.e. the number of similar zero matrix of MB for the 1st and the 2nd frame is one.
Counter cc1 count the reputation of zero macro-blocks in the successive frames. The counters cc_1 is an array of registers so that we can get the difference in the successive frames. During the execution of the second frame the value of p_1i_1 and p_1i_2 is compared with 8 and another checking is there if $p_2\_i_1$ and $p_2\_i_2$ are one(for previous frame it is a zero matrix MBs);if both the condition satisfies $p_2\_i_1$,$p_2\_i_2$ is updated with the new test results if equal then it is set to one otherwise set to zero.

5. Performance Analysis and comparison

With the proposed architecture, ME is performed block by block in a raster scan order. Using the ITU-R601 format (720×576 at 25fps), the functionality of the proposed scalable architecture is thoroughly tested. In total, around 1225clock cycles are required to perform real-time ITUR601 resolution (with block size 45×36) ME.

Table 2: Performance Comparison

Type	#PE	Pipeline Level	Data Operation	Search Range(p)	Input Pins	#CC /Block
Ours	256	Frame	TDO	NA	256	2.46
Liu,Li et al	1089	Frame	TDO	-16/+16	32	256
Yeo & Hu	1024	Frame	BDO	-16/+15	72	256
He,Bi,Mao	256	Frame	TDO	-16/+16	16	1089
Lai &Chen	1024	Block	BDO	-16/+15	40	256
Lee & Lu	256	Block	BDO	-16/+15	24	1024
Tuan et al	256	Block	BDO	-16/+15	16	1024

BDO: Broadcasting Data operation, TDO : Transmittent Data operation, DO: Data operation. # CC: Clock Cycle,

The comparison of the proposed architecture with the other existing architectures[1][9][13][14][15] presented in the Table 2. In Table 2 number of PE, Pipeline Level, type of Operation, Search Range (p), Input pin count, number of clock cycles required to estimate the motion per block have been compared. All the proposed architecture is scalable with the search range p so depends on the value of p. The Full Search Block Matching ME algorithm consider each of these blocks individually and compute the block matching in a predefined search area in the reference frame. The search range is [*-p, p*] where $p \le N$ ($N \times N$ = size of the frame). For each position value of MB the algorithm consider $(2p+1)^2$ candidate blocks. For the block in the current frame it will compute the MAD (Mean Absolute Distortion) value with each of these candidate blocks that is, total $(2p+1)^2$ number of MAD values computation. As the proposed architectures considers the search range [-16, 16] total number of MAD values computation needed is $(2p+1)^2 = (2 \times 16 + 1)^2 = 33^2 = 1089$. ME[9][12][18][19] with this modified algorithm is based on finding the same index repeatation of zero MB with the next frame and discarding the rest of the MB pixels; we are not considering the $(2p+1)^2$ candidate blocks for each MB. So we get a huge number of reduction in computation which implies a decreases in storage place ; results in an increases in the performance of the algorithm as well as the proposed architecture for ME operation.
In summary the clock cycles per frame in the proposed structure is minimum while smaller PE components are used.

6. Conclusions

The architecture level techniques, such as parallel architectures, are more effective than sequential. By using these techniques the power consumption can be reduced further more without degrading the system performance. Parallel architectures reduce the power consumption but increase the cost of the silicon area than sequential architecture.
The proposed architecture can reduce the processing cost, which increases the speedup as well as the throughput .The parallel memory can keep the data path fully utilized in video processing function implementations. This ensures high-speed operation and full utilization of the processing resources. The future work will be vlsi fabrication of the estimated hardware.

Acknowledgments

I like to acknowledge the reviewing work of my M.E.(CSE) thesis by Prof. Amitabha Sinha, Director, School of Information Technology, West Bengal University of Technology, Salt Lake, Sector-I, Kolkata-700 064, India for and his moral support, guidance, for the completion of my work. I also acknowledge the moral support of concerned staff of West Bengal University of Technology. Also I must acknowledge the inspiration I received from Prof. A. K. Ray, then Head of the Department, School of Medical Science & Technology, IIT Kharagpur, and Professor of Electronics & Electrical Communication Engineering, IIT Kharagpur, Kharagpur-731 302, WB, India.

References

[1] H. Yeo,Y.H. Hu, "A Novel Modular Systolic Array Architecture for Full Search Block Matching Motion Estimation" IEEE Transactions on Circuits Systems for Video Technology, Vol. 5, No. 5 , Oct. 1995, pp.407-416.
[2] L. Vos , M. Stegherr, "Parameterizable VLSI architectures for the full-search block-matching algorithm," IEEE Transactions on Circuits and Systems, vol. 36, no. 10, Oct. 1989 , pp. 1309–1316 .
[3] N. Roma, L. Sousa, "Parameterizable hardware architectures for automatic synthesis of motion estimation processors," IEEE Workshop on Signal Processing Systems- Design and Implementation(SiPS'01), Sept. 2001, pp. 428–439.
[4] S. Agha V. M. Dwyer "Algorithms and VLSI Architectures for MPEG4 Motion Estimation" Electronic Systems and Control Division Research 2003.
[5] D. Zhāng G. Lu "An Edge and Color Oriented Optical Flow Estimation Using Block Matching" Gippsland School of Comp & Info Tech Monash University Churchill, Victoria 3842.
[6] M. Venkatesan , D.V. Rao "Hardware Acceleration of Edge Detection Algorithm on FPGAs" Department of Electrical and Computer Engg University of Nevada Las Vegas, NV 89154.
[7] K. N. Ngan, A. A. Kassim, H. S. Singh, "Parallel image-processing system based on the TMS32010 digital signal processor" IEE Proceedings E, Vol. 134, No.2, Mar 1987, pp.119-124.
[8] F. Durand, J. Dorsey "Fast Bilateral Filtering for the Display of High-Dynamic Range Images" ACM Transactions on Graphics, 2002, pp.249-256.
[9] Subarna. Chatterjee et al, "Parallel Hardware Design for Motion Estimation", International Journal of Recent Trends in Engineering(IJRTE),Vol.1,No.1, Aug 2009, pp.653-657.
[10] H. S. Neoh, A. Hazanchu**k** **"**Adaptive Edge Detection for Real-Time Video Processing using FPGA" Altera Corporation**,** 101 Innovation Dr. San Jose CA 95134.
[11] MATLAB Reference Manual from Matworks,Inc.
[12] Subarna Chatterjee et al., "Parallel Hardware Design for Motion Estimation", International Journal of Recent Trends in Engineering (IJRTE-2009), Vol-1, No-1, pp. no. 653-657, ISSN-1797-9617, May - 2009.
[13] S. Kittitornkun ,Y.H.Hu "Frame-Level Pipelined Motion Estimation Array Processor" IEEE Trans. on Circuits and Systems for Video Technology, Vol. 11, No. 2, Feb 2001.
[14] J. C. Tuan, T. S. Chang, and C. W. Jen, "On the data reuse and memory bandwidth analysis for full-search block-matching VLSI architecture," IEEE Transactions on Circuits and Systems for Video Technology, Vol. 12, No. 1, Jan 2002, pp. 61-72.
[15] HE Wei-feng, BI Y un-long, MAO Zhi-gang "Efficient Frame-Level Pipelined Array Architecture for Full-Search Block-Matching Motion Estimation" IEEE International Symposium on Circuits and Systems, Japan ,Vol. 3, May 2005, pp. 2887- 2890.
[16] Douglas L. Perry, "VHDL Programming by Example", Tata McGraw-Hill Edition 2002, Fourth Edition.
[17] L.C. Liu, J.C. Chien, H. Y. H.Chuang, and C. C. Li "A Frame-Level FSBM Motion Estimation Architecture with Large Search Rang" IEEE Conference on Advanced Video and Signal Based Surveillance(AVSS'03).
[18] Subarna Chatterjee et al., "Design of Parallel Architecture for Motion Estimation" International Workshop on Mobile Systems(WoMS'08)July 2008, pp-113-116.
[19] Subarna Chatterjee et al., "Architecture Design of Efficient Video Processing Technique for Motion Analysis" IEEE WIE National Symposium on Emerging Technologies (WieNSET'07) Jun 2007, pp. 141-144.

Subarna Chatterjee received her Bachelor's degree in Information Technology, from MCKV Institute of Engineering, Liluah, Howrah, then under Vidyasagar University, in 2004 followed by Post Graduate Diploma in Embedded Computer System in 2006(Dec), and M.E. (CSE) in 2009(July) from West Bengal University of Technology , Kolkata. She is presently pursuing her PhD at Bengal Engineering and Science University, Shibpur, Howrah. She was awarded Gold Medal from the university for stood 1st in M.E.(CSE)-2009. She has authored 10 research papers in International and National journals and conferences. Her current research interest includes Medical Imaging, Image Processing, Pattern Recognition and Machine Intelligence.