FPGA-Based Real-Time Video-Object Segmentation with Optimization Schemes

M.M. Abutaleb, A. Hamdy, and E.M. Saad

Abstract— This paper presents the implementation of a segmentation process to extract the moving objects from image sequence taken from a static camera used for real time vision tasks. Various aspects of the underlying motion detection algorithm are explored and modifications are made with potential improvements of extraction results and hardware efficiency. The whole system is implemented on a single low cost FPGA chip, capable of real-time segmentation at a very high frame rate that reaches to 1130 fps. In addition, to achieve real-time performance with high resolution video streams, dedicated hardware architecture with streamlined data flow and memory access reduction schemes are developed. Data flow reduction of 38.6% is achieved by processing only one distribution at time through the hardware. Also, substantial memory bandwidth reduction of 60% is achieved by utilizing distribution similarities in succeeding neighboring pixels as well as word length reduction.

Keywords— Hardware optimization, Motion detection, Real-time implementation.

I. INTRODUCTION

THE motion detection is an essential processing component for many video applications such as video surveillance, military reconnaissance, robot navigation, collision avoidance, and path planning. Most of these applications demand a low power, compact, light weight, and high speed computation platform for processing image data in real time. There are three conventional approaches to motion detection: temporal differencing [1]; optical flow analysis [2], [3]; and background subtraction [4-6]. Motion detection by background subtraction can be divided into adaptive and non-adaptive background methods. Non-adaptive methods need off-line initialization; errors in the background accumulate over time. A common method of adaptive backgrounding is to average the frames over time [7]. This creates an approximate background. This is effective where objects move continuously and the background is visible for a significant portion of time, it is not robust for scenes with many moving objects. It cannot handle a multimodal backgrounds caused by the repetitive motion of the background.

Stauffer algorithm [8], [9] is representative of an adaptive method which uses a mixture of normal distributions to model a multimodal background image sequence. This method deals robustly with slowly-moving objects as well as with repetitive background motions of some scene-elements. Repetitive variations are learned and in that way, a model for the background distribution is preserved. A practical implementation of Stauffer algorithm in [10] provides values for all model parameters. With a large number of calculations due to the pixel-wise processing of each frame, Stauffer algorithm [8] could only achieve a low frame rate, far from real-time requirements. Hence, in this work we use a fast motion detection algorithm based on a multi-modal distribution to extract the moving objects by modeling each pixel as a mixture of three (or larger) distributions with a small number of calculations to achieve a high frame rate for real-time requirements.

A few high-performance implementations of motion detection algorithms exist, and the fastest of these involve the use of special hardware features to achieve their high performance. Implementations based on general-purpose microprocessors have had to be of low computational complexity in order to achieve processing speeds above a few frames per second, and exclude algorithms of moderate or high computational complexity. A solution to implement complex algorithms at frame rates is to build custom hardware. The downside of this approach is that design of custom hardware has typically been a lengthy and expensive process. It may take months to develop and verify a design, and Application-Specific Integrated Circuits (ASICs) incurs costs ranging from hundreds to hundreds of thousands of dollars. There is an option available that bridges the gap between the ease of design associated with software and the performance associated with hardware. The advent of reconFigurable logic hardware in the form of Field-Programmable Gate Arrays (FPGAs) allows designs to be quickly developed and prototyped at relatively low cost [11].

In recent years, different schemes have been proposed to implement motion detection algorithms and achieve real-time computation. Using a pipeline image processor, Correia and Campilho [12] propose a design which can process the Yosemite sequence of 252×316 size in 47.8ms. FPGAs have been used to process larger images at faster speed. An algorithm proposed by Horn and Schunck [13] was implemented [14]. It is an iterative algorithm where the accuracy depends largely on the number of iterations. The classical Lucas and Kanade approach was also implemented [15] for its good tradeoff between accuracy and processing efficiency. Two step search block matching algorithm [16] was first implemented and then ported onto an Altera NiosII processor [17] where some hardware-based support was used

Authors are with department of electronics, communications, and computer, faculty of engineering, Helwan university, Cairo, Egypt.

to meet application timing requirements. A fast and accurate motion estimation algorithm [18] was modified for FPGA hardware implementation [19]. This design is able to process images of size 640×480 at 64 frames per second (fps).

This paper describes a low cost and less area implementation of a fast motion detection algorithm to extract the moving objects from image sequence of size 768 x 576 pixels at a very high frame rate that reaches to 1130 frames per second in a single FPGA chip which is adequate for most real-time vision applications. 13.4% area optimization and 38.6% data flow reduction are achieved to enhance the computational complexity. Furthermore, a word length and memory access reduction schemes are implemented, resulting in more than 60% memory bandwidth reduction. This paper is organized as follows. In section 2, the algorithm is formulated and our suggestions are introduced. In section 3, the hardware implementation of the design is discussed. In section 4, Different optimization schemes are proposed and discussed. Conclusion is given in section 5.

II. ALGORITHMS

The adaptive-background method for motion detection [8] is applied to the frame pixels. To achieve this, each pixel of the reflectance component is modeled as a mixture of K (three to five) Gaussian distributions (each pixel has K of mean-values, variance-values and weights-values maintained at time t). At any given time t, the history of a particular pixel $\{x_0, y_0\}$ is:

$$\{X_1,...,X_t\} = \{I(x_0,y_0,i): 1 \le i \le t\} \tag{1}$$

where I is the image sequence. The probability of observing the given value of a current pixel is:

$$P(X_t) = \sum_{k=1}^{K} \omega_{k,t} * \eta(X_t, \mu_{k,t}, \Sigma_{k,t}) \tag{2}$$

where $\omega_{k,t}$ is an estimated weight of the kth Gaussian in the mixture at time t, $\mu_{k,t}$ is its mean value, $\Sigma_{k,t}$ is its covariance matrix (or the variance in the case of monochromatic image, which is our case), and η is a Gaussian probability density function:

$$\eta(X_t, \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(X_t - \mu_t)^T \Sigma^{-1}(X_t - \mu_t)} \tag{3}$$

For each new pixel, the intensity value is thus checked against the existing Gaussian distributions, until the match is found. The matching criterion is defined as a pixel's value lying within 2.5 standard deviations interval from the mean of a current distribution. The mean and variance parameters of the distribution matching are updated as:

$$\mu_t = (1-\rho)\mu_{t-1} + \rho(X_t) \tag{4}$$

$$\sigma_t^2 = (1-\rho)\sigma_{t-1}^2 + \rho(X_t - \mu_t)^T(X_t - \mu_t) \tag{5}$$

$$\rho = \alpha\eta(X_t|\mu_k,\sigma_k) \tag{6}$$

where ρ is the learning factor and α is the learning rate. The weights of all distributions are updated as:

$$\omega_{k,t} = (1-\alpha)\omega_{k,t-1} + \alpha(M_{k,t}) \tag{7}$$

where Mk,t=1 for the matching model and Mk,t=0 for the remaining models. If none of the K distributions matches the current pixel value, the least probable distribution is replaced with a distribution having the current value as its mean, an initialized high variance, and a low weight. The next step is to sort all the components in the mixture in the decreasing order of ratios (ω/σ). It is possible to set a threshold T, which will separate distributions representing the background. The pixels that do not fit the background distributions are considered to belong to the moving objects.

There are several problems for implementing the Stauffer algorithm [8] such as a heavy use of square roots and squares in the equations, floating point calculations, and the need for evaluating exponential for the density function calculation. So, the VHDL code is hard to be synthesizable for this algorithm and the requirements for real time (video rate) are not suitable to be achieved by this algorithm. Hence, we propose a fast and efficient algorithm as in [20] to extract the moving objects in each video frame for software and hardware implementation. In the proposed method, each pixel is modeled as mixture of three distributions (k=3) as in Stauffer's method but each distribution is represented only by mean-value and weight-value maintained at time t. The mixture here is ordered in decreasing order of weight (ω) values. Each pixel is checked against the distributions, until the match is found. The matching condition is achieved if the variation of the pixel within *R%* (matching ratio, 10% to 40%) from its mean value.

/* **Check_Distribution_Condition** */
$$(1 - R\%)\,\mu_{k,t-1} \le X_t \le (1 + R\%)\,\mu_{k,t-1} \tag{8}$$

For the matching distribution, we store the current pixel as the processed pixel value $X_{P,t}$ for the next time t+1.

/* **Match_Update** */
$$X_{P,t} = X_t \tag{9}$$

Also, we can apply the temporal differencing as proposed by Arseneau, S. and Cooperstock, J. [21] where the pixel is considered foreground pixel if $|X_t - X_{t-1}| > T$ where T is the threshold value. But, we replace the previous pixel value X_{t-1} by the previous processed pixel value $X_{P,t-1}$ for time t-1.

/* **Check_Foreground_Condition** */
$$| X_t - X_{P,t-1} | > T \tag{10}$$

We found that the adaptive value for T as the half of *R%* from the pixel mean value gave good results than fixed threshold value. The weight parameter of the matching distribution for foreground pixel will be updated as:

/* **Foreground_Match_Update** */
$$\omega_{k,t} = (1-\alpha)\,\omega_{k,t-1} \tag{11}$$

while the mean and weight parameters for none foreground pixel will be updated as:

/* Background_Match_Update */

$$\omega_{k,t} = (1-\alpha)\,\omega_{k,t-1} + \alpha \qquad (12)$$

$$\mu_{k,t} = (1-\alpha)\,\mu_{k,t-1} + \alpha\,X_t \qquad (13)$$

where we select α as the learning factor as proposed by P. Wayne and Johann [10]. If the current pixel is not matched with any distribution, this pixel will be classified as foreground pixel. The mean of the third distribution of that pixel will be replaced by its intensity value and its weight will be selected as lower value than other distributions. Also, the processed pixel will be equal to the value of the mean of the first distribution.

/* None_Match_Update */

$$\omega_{3,t} = \text{Low}, \quad \mu_{3,t} = X_t, \quad X_{P,t} = \mu_{1,t-1} \qquad (14)$$
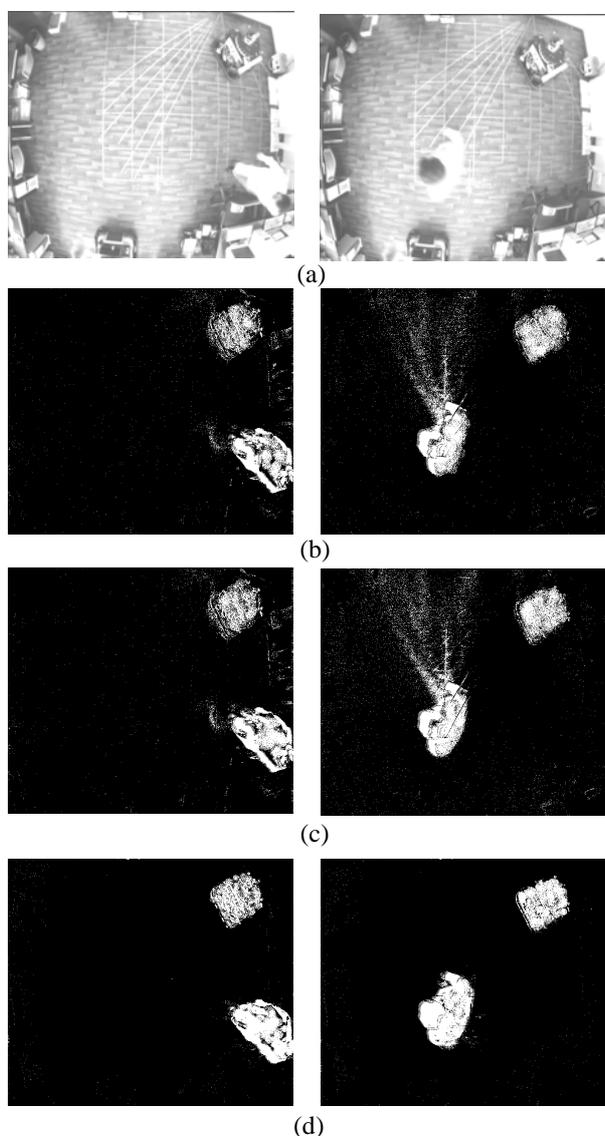

(a)

(b)

(c)

(d)

Fig.1. frames 16 (left) and 37 (right) from the lab sequence (a) and its corresponding foreground image obtained by Stauffer (b), modified Stauffer (c) and proposed (d) algorithms with R%=20%

A. Experimental Results

The software version is implemented in Matlab scripts to detect the moving objects for 768 x 576 frames using a 2GHz P4, 256MB RAM computer. Figure 1 shows the results of applying Stauffer algorithm [8], modified Stauffer algorithm [10] and our algorithm on the frames 16 and 37 respectively where a person moves quickly and a robot moves slowly in the lab. The effect of rapid lighting change is appeared in the results of Stauffer algorithm and a homomorphic filter can be used as pre-processing operator to extract the foreground pixels as in [6]. This pre-processing operator is not required for the extraction of foreground pixels using our algorithm due to applying the temporal differencing on the matching distribution. Post-processing operators can be applied to remove noisy pixels and fill the holes inside the blobs on the results of all algorithms.

The experimental results of average elapsed time per frame have been calculated by applying the Stauffer and the proposed algorithms. The results are T_1=121.5849 S/F, T_2=1.8816 S/F, and T_3=1.0996 S/F where T is the average elapsed time in seconds per frame (S/F). T_1 is the time if we apply Stauffer algorithm [8], T_2 if we apply the modified Stauffer algorithm as in [10], and T_3 if we apply our proposed algorithm. The comparison results are D_1 = 99% and D_2 = 42% where the reduction ratio D is defined as the ratio at which the average elapsed time per frame T is reduced by applying our proposed algorithm: $D_i = (T_i - T_3) / T_i$; i=1, 2. D_1 is the ratio if we apply Stauffer algorithm and D_2 if we apply modified Stauffer algorithm. It can be seen that the time is optimized by applying our algorithm compared to that generated with the other algorithms. Figure 2 shows the results of applying the proposed algorithm on the frames 54 and 187 respectively in another environment. Note that the first person moves quickly while the second person that appears in frame 187 moves slowly. Results show the enhanced performance of the proposed algorithm with less computation cost and higher frame rate.
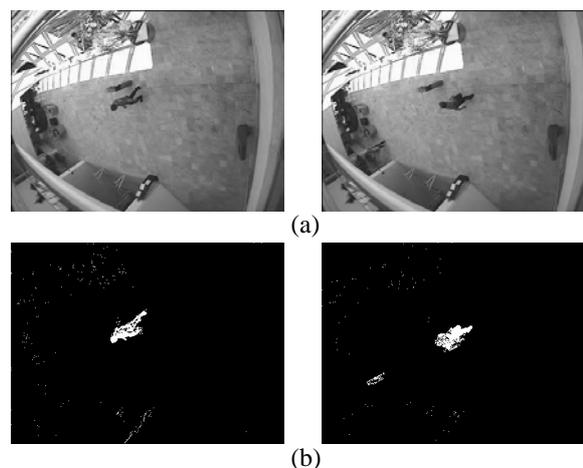

(a)

(b)

Fig.2. frames 54 (left) and 187 (right) from the second sequence (a) and its corresponding foreground image obtained by the proposed algorithm (b) with R%=30%

Two major features make the proposed algorithm a good candidate for hardware implementation. First, it is primarily composed of linear operations that can be easily implemented in hardware. Operations such as addition and multiplication can be represented efficiently in terms of number of logic blocks required, and can be computed in few, or even one, clock cycles. Second, there is no iteration or any explicit coarse-to-fine control strategy. This property makes the real time flow of data possible through the hardware. In the next sections, we will describe the implementation of the proposed algorithm and the modifications applied to the implemented architecture.

III. HARDWARE IMPLEMENTATION

Some implementations of active vision components are the same to provide easy interconnection and real time operation. The input and output memories, that can be accessed at the same time, are the most important parts of this hardware structure [22]. The other element is always an FPGA (or several), that holds the vision algorithm and the memories addressing and control tasks. Here, single-pixel multiple-distributions (SPMD) architecture is presented for the proposed motion detection algorithm to extract the foreground images in real time, taking advantage of the data and logical parallel opportunities offered by FPGA architecture. The system uses input memory to access the captured frame pixels, output memory to store the processed pixels, and input-output memory to read and store distribution parameters as shown in Fig.3.
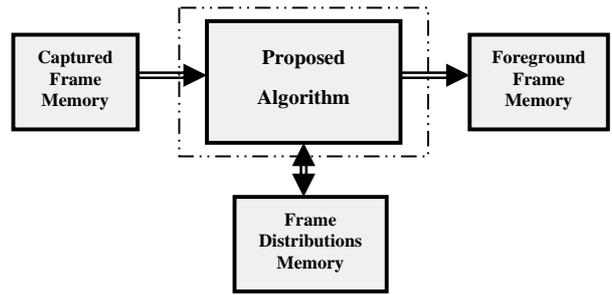

Fig.3. block diagram of the system

A. Pixel-based Architecture

Stauffer algorithm [8] and the proposed algorithm are pixel level processing for each frame to classify if this pixel as foreground pixel or not. The required system must be able to process images of size 768 x576 (442368 pixels) at real time rate (25-30 fps). Generally, it is required to process 442368 x 25 pixels per second. So, any saving or speeding up in the implementation of a single pixel architecture will be magnified 442368 x 25 times. Figure 4 shows SPMD architecture in the form of pipeline and parallel processing to achieve maximum speed up. A control signal rst initializes the distribution values and parameters of the algorithm. Ready_in_pixel is active when the data from camera (frame memory) and distributions values included the previous processed pixel value (distributions memory) are ready in the input. Pixel_in (range 0 to 255) and Foreground (0 or 255) signals represent pre- and post-processing intensity value of the current pixel.
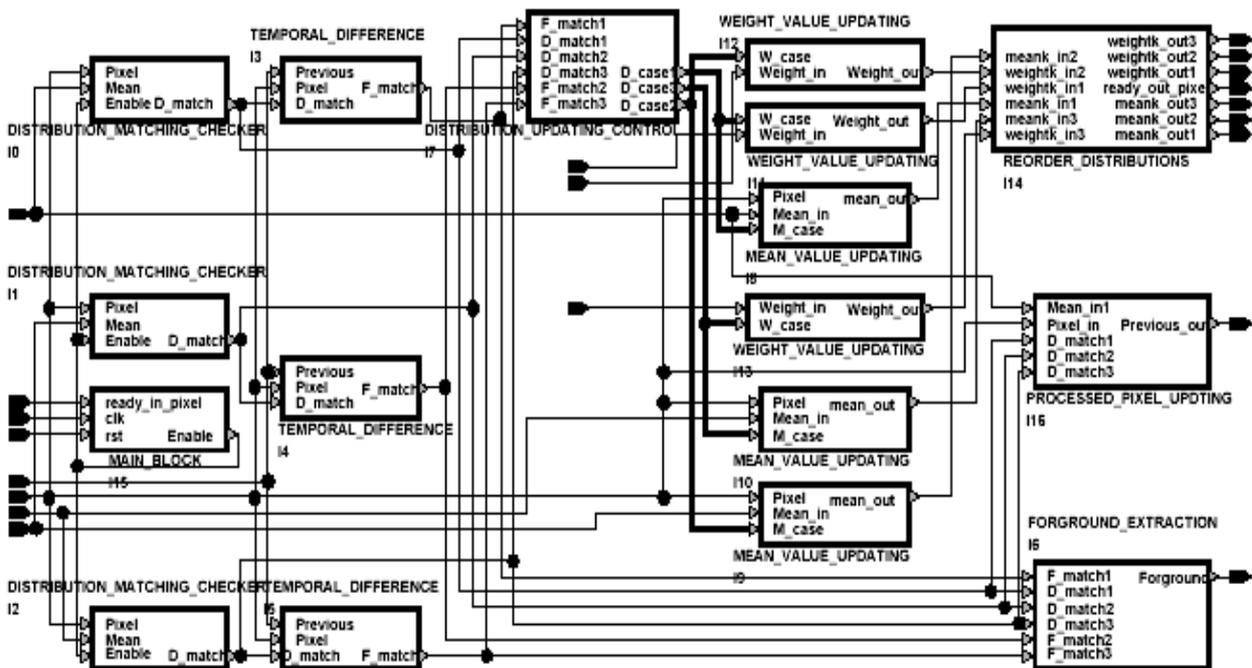

Fig.4. the structure of SPMD architecture

Mean_in1,2,3, Weight_in1,2,3, Previous_in, Mean_out1,2,3, Weight_out1,2,3 and Previous_out signals are pre- and post-processing distributions and processed pixel values. The Ready_out_pixel is active when the output data and parameters are ready in output. Each block in the shown architecture' is responsible to satisfy one or some equations in the proposed algorithm. The fixed point data representation is used here vs. floating point representation for means and weights calculations. Main block starts the operation with the first clock signal.
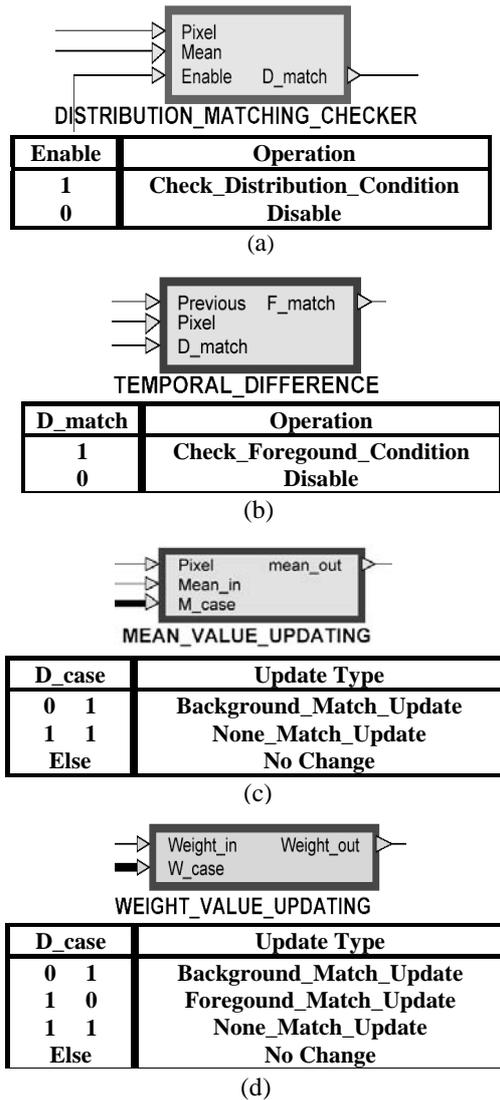
DISTRIBUTION_MATCHING_CHECKER

Enable	Operation
1	Check_Distribution_Condition
0	Disable

(a)

TEMPORAL_DIFFERENCE

D_match	Operation
1	Check_Foregound_Condition
0	Disable

(b)

MEAN_VALUE_UPDATING

D_case		Update Type
0	1	Background_Match_Update
1	1	None_Match_Update
Else		No Change

(c)

WEIGHT_VALUE_UPDATING

D_case		Update Type
0	1	Background_Match_Update
1	0	Foregound_Match_Update
1	1	None_Match_Update
Else		No Change

(d)

Fig.5. DISTRIBUTION_MATCHING_CHECKER block (a), TEMPORAL_DIFFERENCE (b), MEAN_VALUE_UPDATING block (c), and WEIGHT_VALUE_UPDATING block (d)

There are MEAN_VALUE_UPDATING, WEIGHT_VALUE_UPDATING, DISTRIBUTION_MATCHING_CHECKER and TEMPORAL_DIFFERENCE blocks for each distribution. These blocks are implemented together to perform the pipeline and parallel operation to achieve the maximum speed up. Figure 5 shows the functions of these blocks. D_match signal is assigned as one when the current pixel satisfies the corresponding distribution condition. F_match signal is assigned as one when the current pixel satisfies temporal differencing condition. Mean_out and Weight_out signals represent the new mean and weight values in the corresponding distribution for the current pixel to be applied in time t+1.

FORGROUND_EXTRACTION block outputs a foreground signal as one if the current pixel value is matched with any foreground matching condition or not matched with any distribution condition. DISTRIBUTION_UPDATING_CONTROL block, see Fig.6, outputs D_case (updating type) for three distributions. New processed pixel value is obtained by PROCESSED_PIXEL_UPDTING block, see Fig.7. REORDER_DISTRIBUTIONS block is used to order distributions in decreasing order of weights.
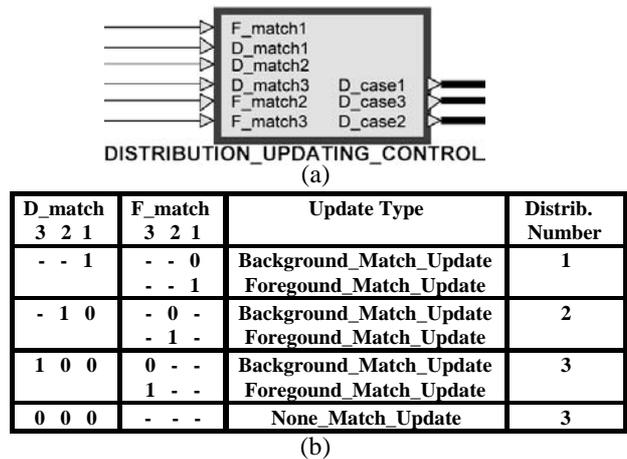
DISTRIBUTION_UPDATING_CONTROL

D_match 3 2 1	F_match 3 2 1	Update Type	Distrib. Number
- - 1	- - 0	Background_Match_Update	1
	- - 1	Foregound_Match_Update	
- 1 0	- 0 -	Background_Match_Update	2
	- 1 -	Foregound_Match_Update	
1 0 0	0 - -	Background_Match_Update	3
	1 - -	Foregound_Match_Update	
0 0 0	- - -	None_Match_Update	3

(b)

Fig.6. DISTRIBUTION_UPDATING_CONTROL block (a) and its corresponding operation

PROCESSED_PIXEL_UPDTING

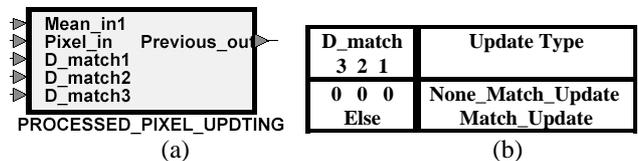| D_match 3 2 1	Update Type
0 0 0	None_Match_Update
Else	Match_Update

(a) (b)

Fig.7. PROCESSED_PIXEL_UPDTING block (a) with its corresponding operation (b)

B. Experimental Results

The implemented SPMD architecture using low cost available Spartan-II development system with Xilinx chip 2s200fg456 has 54.1 MHz maximum frequency and uses 209 CLB slices with 8.89% utilization. Hence, ten pixels can be processed in parallel and this means that 500 M Pixels / second (10 x 50 MHz) can be processed. This system is able

to process images of size 768 x 576 (442368 pixels) at 1130 frames per second. This rate can be reduced due to access data with the external memories but it is guaranteed to process images in the real time or faster. To speed up this system, the bigger area and higher performance FPGA chip or more than one FPGA chip can be used. For instance, the implementation in Xilinx chip 2v1000fg456 has 95.5 MHz maximum frequency and 4.12% CLB slices utilization. The output foreground signals from the implemented system are the same results that as obtained by the implemented software version.

IV. OPTIMIZATION SCHEMES

A. Word length Reduction

Slow background updating requires large dynamic range for each parameter in the distributions. This is due to the fact that parameter values are changed slightly between frames, but could accumulate over time. In this section, parameter word length reduction is investigated for potential memory bandwidth reduction. According to (13), the mean of each distribution is updated using a learning factor α. The difference of mean between current and previous frame is derived from the following equation;

$$\Delta\mu = \mu_{k,t} - \mu_{k,t-1} = \alpha \left(X_t - \mu_{k,t-1} \right) \tag{15}$$

Given a small value for α, e.g. 0.0001, a unit difference between the incoming pixel and the current mean value results in a value of 0.0001 for $\Delta\mu$. In practice, the bits for fractional parts should be somewhere in the range of 10-14 bits. To be able to record this slight change, 18 bits have to be used for the mean value, where 10 bits accounts for the fractional part and 8 bits are used for the integer one. Fewer bits can be achieved by ignoring small deviations of the incoming pixel from current mean, while picking up only large ones. The extreme case is when only the largest deviation is picked, e.g. where the incoming pixel is in the range of the matching ratio *R%* off the current mean. Larger than that, the incoming pixel will be not match the current distribution. With an upper bound for the matching range, a very small fractional value is derived for $\Delta\mu$.

To reduce the number of bits that is needed for each distribution, a word length reduction scheme is proposed. From equation 15, a small positive or negative number is derived depending on whether the incoming pixel is larger or smaller than the current mean. Instead of adding a small positive or negative fractional number to the current mean, a value of 1 or -1 is added. The overshooting caused by such coarse adjustment could be compensated by the update in the next frame, e.g. without illumination variation, the mean value will fluctuate with a magnitude of 1.The proposed parameter updating scheme keeps track of the relatively fast value changes in the dynamic scene while fluctuates around a constant value in the latter static scene. However, with the primary goal to reduce word length, the proposed scheme results in limited improvements to the segmentation results. Nearly no visual difference can be observed in the segmented

results from the proposed and normal schemes. With a proposed parameter updating, only integers are needed for mean specification, which effectively reduce the word length from 18 down to 8 bits in each distribution.

For the SPMD architecture, over 38% word length reduction and less hardware complexity are achieved by using the proposed updating scheme compared with the normal updating scheme. Thus, the proposed scheme enhance the algorithmic performance while at the same time reduce both memory bandwidth and computational complexity.

B. Area Optimization

The updated distributions have to be ordered by REORDER_DISTRIBUTIONS block for use in the next frame. In order to reduce hardware complexity found in parallel ordering network, while still maintaining the speed, a specific feature in the algorithm is explored. By observing that only one distribution is updated at a time and all distributions are initially ordered, the ordering of three (or k) distributions can be changed by rearranging an updated distribution among two (or k-1) ordered distributions. The architecture for the ordering network (k=3) is shown in Fig.8.
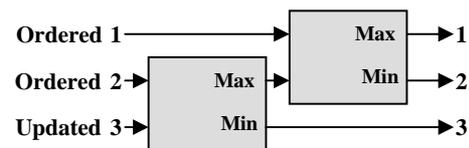

Fig.8. the ordering network

From the Figure all unmatched ordered distributions are compared with the updated one (3 in the Figure) based on weights values. After this modification, the implemented SPMD architecture uses only 181 CLB slices with the same speed. This results in 13.4% ([209-181] ÷209) more reduction in the area of the previous proposed SPMD architecture [20].

C. Data Flow Reduction

Dedicated hardware architecture, with a streamlined data flow reduction scheme, is implemented to address the computation capacity and chip input-output pins bottlenecks. This is a large improvement to the previous proposed SPMD architecture [20] which is shown in Fig.4. In this part, a thorough description of the whole system architecture with data flow reduction scheme is given.

Each pixel has a series of corresponding distributions, where are stored on off chip memories due to its size. The processing blocks of all distributions for a single pixel are implemented together in SPMD architecture, as shown in Fig.4, to achieve a very high frame rate but this requires a heavy computation capacity and input-output pins for the parameters (mean μ and weight ω values) of the distributions. The data flow can be reduced if only one distribution is processed at a time and all distributions are processed in pipeline through the same architecture. Here, off chip frame distributions memory is divided into four banks as shown in

Fig.8. Bank0 stores the processed pixel value (X_P) and the order code (C_o) for the current processing pixel. The code Co represents the decreasing order of distributions according to weights (ω) and through it the distribution parameters are streamed into FPGA in pipeline. Bank1, Bank2, and Bank3 store the three distribution parameters. Only one of the three banks is enabled by R_1, R_2, R_3 signals depending on the code Co as shown in Fig.9 and the selected parameters (P_S) will be flowed with FPGA.

Fig.8. organization of distributions memory

Code C_1 C_2 C_3	Distribution Sel. Order		
0 0 0	1	2	3
0 0 1	1	3	2
0 1 0	2	1	3
0 1 1	2	3	1
1 0 0	3	1	2
1 0 1	3	2	1

Fig.9. Distribution selection and pipeline order

A simplified conceptual block diagram of that single-pixel single-distribution (SPSD) architecture is given in Fig.10 to illustrate the data flow within the system. It consists of Control_Unit, Matching_Unit, Decoder_Unit, and Output_Unit blocks. The signals with the same names in SPMD architecture have the same functions.
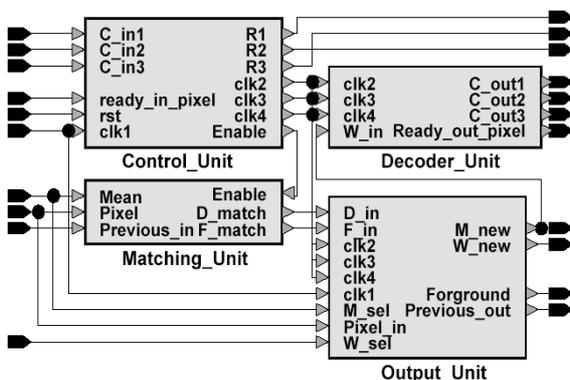

Fig.10. the structure of SPSD architecture

A Control_Unit as shown in Fig.11 consists of two blocks. Mul_Clk_Unit block generates three different clocks/enables (clk2,3,4) from the input clock clk1 as shown in Fig.12. Switch_RAM block generates enable signals (R1,R2,R3) for memory banks depending on the 3-bit code signal (C_in1,2,3) to order the entrance of selected distributions. The pipeline operation of the SPSD architecture requires four different clocks/enables. clk1, clk2, and clk3 signals are used to enter the selected mean (M_sel) and weight (W_sel) data from the first, second, and third enabled distribution memory bank respectively.
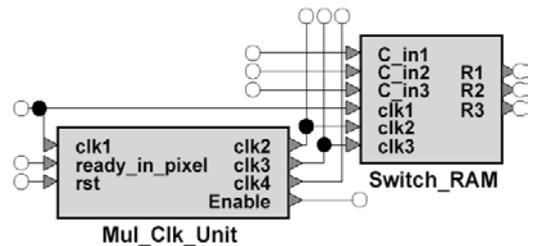

Fig.11. Control_Unit block
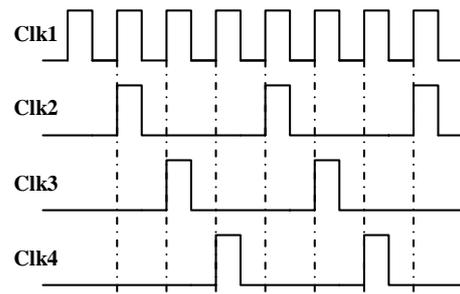

Fig.12. clocks/enables signals

Output_Unit block as shown in Fig.13 consists of six blocks. In Updating_Control block, clk2, clk3, and clk4 are used to generate a new mean (M_new) and weight (W_new) data for the first, second, and third enabled distribution memory bank respectively. Mean1_reg stores the mean value of the first selected distribution. Two Match_reg blocks store D_match and F_match signals for the first and second selected distributions with clk2 and clk3 signals respectively. While with clk4 signal: Processed_Clk block generates new processed pixel (Previous_out) data and Fourground_Clk block generates the foreground decision value.

A Matching_Unit block as shown in Fig.14 consists of Distribution_Matching_ Checker and Temporal_Difference blocks with the same functions that are previously discussed for SPMD architecture. A Decoder_Unit block as shown in Fig.15 consists of three blocks. Two W_reg blocks store weight values for the first and second selected distributions

with clk2 and clk3 signals respectively. Reorder_Clk generates the code C_o (C_out1,2,3) signals that represents the decreasing order of the updated weights (W_up1,2,3) as shown Fig.9 and Ready_out_pixel signal to end the current pixel processing.
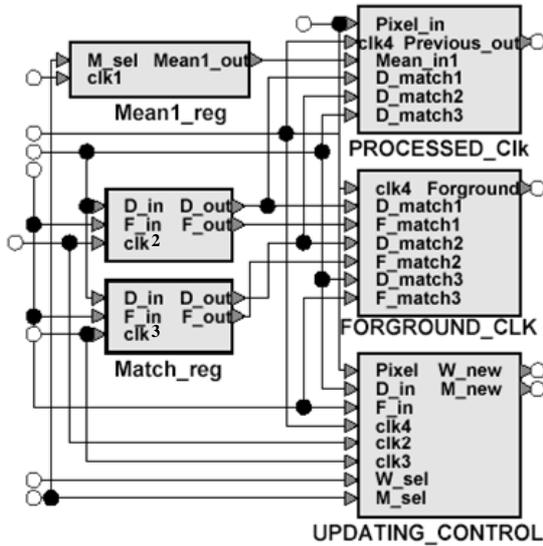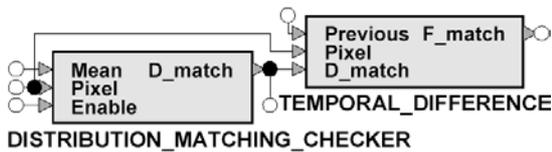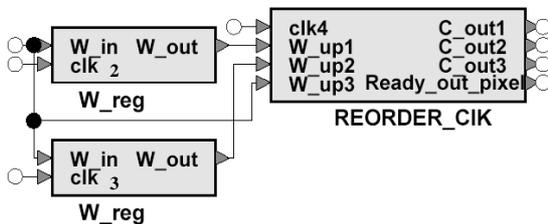
Fig.13. Output_Unit block

Fig.14. Matching_Unit block

Fig.15. Decoder_Unit block

The implemented SPSD architecture using available Spartan-II development system with Xilinx chip 2s200fg456 has 69.5 MHz maximum frequency and uses 75 CLB slices with 3.19% utilization. The SPSD architecture has higher speed and lower area than SPMD architecture. So, this system

is able to process images of size 768 x 576 at the same frame rate as SPMD architecture with 38.6% reduction in number of input-output pins.

D. Memory Access Reduction

To reduce heavy memory access incurred by accessing off chip memory banks that store one frame of distributions, an encoder and decoder blocks are designed by utilizing distribution similarities in succeeding neighboring pixels as shown in Fig.16. This is covered in this part.
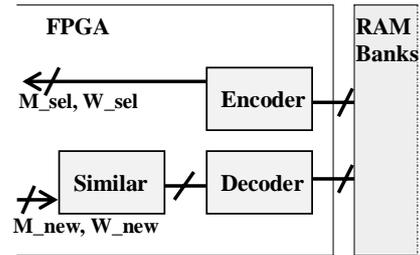
Fig.16. similarity decoder/encoder

We classify "similar" distributions in the following way: from the definition of a matching process, each distribution can be simplified as a two dimensional rectangle (mean and weight dimensions). The center of the rectangle is mean value whereas the border to the center is specified by the matching ratio *R%* off that mean. One way to measure the similarity between two distributions is to check how much of the two rectangles that overlap. If the overlap volume takes up certain percentage of both rectangles, they are regarded as "similar". The whole idea is illustrated in Figure 17. The reason for such criteria lies in the fact that a pixel that matches one distribution will most likely match the other, if they have enough overlapping volume. The percentage is an overlapping ratio parameter that can be set to different values among different situations.
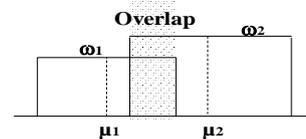
Fig.17. distribution similarity as modeled by rectangle overlapping

In the architecture, two similar distributions are treated as equivalent. By only saving non overlapping distributions together with the number of equivalent succeeding distributions, memory bandwidth is reduced. Various overlapping ratios are selected to evaluate the efficiency for memory bandwidth reduction. With a low value where less

overlapping distributions are regarded as the same, more savings could be achieved. However, more noise is generated due to increasing mismatches in the matching block. Fortunately, such noise is found non-accumulating and therefore can be reduced by the morphological filtering. The background pixels exhibit high similarity within neighboring pixels. With foreground objects entering the scene, part of distributions are replaced, which results in the decrease of number of similar distributions. The trends will continue until it reaches a certain point where most pixel locations contain a foreground distribution. The decrease will flattens out in the end. Figure 18 shows the results for different overlapping ratios and achieved memory bandwidth reductions in lab sequence. In this implementation, an overlapping ratio of 0.8 is selected, combined with word length reduction scheme, a memory bandwidth reduction of over 60% is accomplished.
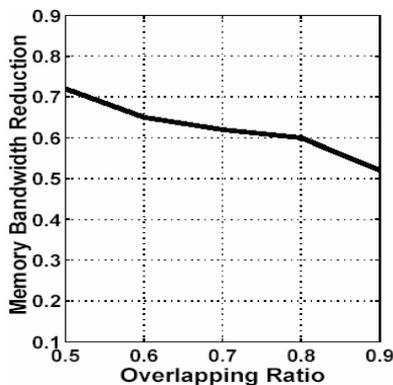

Fig.18. memory bandwidth reduction over overlapping ratio

V. Conclusion

High computation demand makes it difficult to use motion detection algorithm for real-time applications using general purpose processors. In this paper, a low cost and area efficient FPGA-based implementation of a fast motion detection algorithm has been presented to extract the moving objects from image sequence of size 768 x 576 pixels at a very high frame rate that reaches to 1130 fps which is adequate for most real-time vision applications. It has been shown that the area of the SPMD architecture [20] was reduced by 13.4% by modifying the REORDER_DISTRIBUTIONS block.

Dedicated hardware architecture, with a streamlined data flow and memory bandwidth reduction schemes, has been implemented to address the computation capacity and memory bandwidth bottlenecks. Data flow reduction of 38.6% has been achieved by using the proposed SPSD architecture in comparison to SPMD architecture [20] to enhance the computational complexity. Also, a word length and memory access reduction schemes have been proposed for the architectures, resulting in more than 60% memory bandwidth reduction.

References

[1] C. Anderson, P. Burt, and G. van der Wal, "Change detection and tracking using pyramid transformation techniques", in Proceedings of SPIE. Intelligent Robots and Computer Vision, vol. 579, pp. 72-78, 1985.

[2] J. Barron, D. Fleet, and S. Beauchemin, "Performance of optical flow techniques", International Journal of Computer Vision, vol. 12, pp. 42-77, 1994.

[3] K. Belda, J. Böhm, "Range-space predictive control for optimal robot motion", NAUN JOURNAL of CIRCUITS, SYSTEMS and SIGNAL PROCESSING, Issue 1, Vol. 1, pp. 1-7, 2007.

[4] A. Kasinski and A. Hamdy, "Efficient Separation of mobile objects on the scene from the sequence taken with an overhead camera". Proc. Int. Conf. on Computer Vision and Graphics, Zakopane, vol. 1, pp. 425-430, 2002.

[5] Y. Ivanov, A. Bobick, and J. Liu, "Fast Lighting Independent Background Subtraction". Technical Report no. 437, MIT Media Laboratory, 1997.

[6] A. Kasinski and A. Hamdy, "Robust Classification of Moving Objects Based on Rigidity Criterion using Region Growing of Optical Flow Fields," Advanced Concepts for Intelligent Vision Systems, Ghent, Belgium, pp.180-187, Sep.2003.

[7] G. Halevy and D. Weinshall, "Motion of disturbances: detection and tracking of multi-body non-rigid motion", Machine Vision and Applications, vol. 11, Issue 3, pp. 122-137, 1999.

[8] C. Stauffer and W.E.L. Grimson, "Learning Patterns of Activity Using Real-Time Tracking". IEEE Trans. vol. 22, no. 8, pp. 747-757, 2000.

[9] W.E.L. Grimson, C. Stauffer, R. Romano, and L. Lee, "Using adaptive tracking to classify and monitor activities in a site". Computer Vision and Pattern Recognition (CVPR), p. 22, June 1998.

[10] P. W. Power and J.A. Schoonees, "Understanding Background Mixture Models for Foreground Segmentation". Imaging and Vision Computing New Zealand, Auckland, NZ, pp.267-271, Nov 2002.

[11] A. Raabe, B. Bartyzel, G. Zachmann, J. K. Anlauf, "Hardware Accelerated Collision Detection -- An Architecture and Simulation Results", WSEAS TRANSACTIONS on SYSTEMS, Issue 5, Vol. 3, pp. 2025-2030, July 2004.

[12] M. Correia, A. Campilho, "Real-time implementation of an optical flow algorithm", Proc. ICIP, Vol. 4, pp. 247-250, 2002.

[13] B. Horn, B. Schunck, "Determining optical flow", Artificial Intelligence, vol. 17, pp. 185-203, 1981.

[14] J. L. Martín, A. Zuloaga, C. Cuadrado, J. Lázaro, U. Bidarte, "Hardware implementation of optical flow constraint equation using FPGAs", Computer Vision and Image Understanding, Vol.98, pp. 462-490, 2005.

[15] J. Díaz, E. Ros, F. Pelayo, E. M. Ortigosa, S. Mota, "FPGA-based real-time optical-flow system", IEEE Trans. Circuits and Systems for Video Technology, Vol. 16, no. 2, pp. 274-279, Feb 2006.

[16] Yu, N., Kim, K., Salcic, Z. "A New Motion Estimation Algorithm for Mobile Real-Time Videoand Its FPGA Implementation". IEEE TENCON-2004, Chiang Mai, Thailand, 21-24, 2004.

[17] Joseph Poh. "Smart camera – an intelligent vision sensor", Dept. of Electrical and Computer Engineering ELECTENG/COMPSYS/ SOFTENG Conference, Sept. 2005.

[18] G. Farnebäck, "Fast and accurate motion estimation using orientation tensors and parametric motion models", Proc. ICPR, vol.1, pp.135–139, 2000.

[19] Z.Y. Wei, D.J. Lee, B.E. Nelson, and M.A. Martineau, "A fast and accurate tensor-based optical flow algorithm implemented in FPGA", IEEE Workshop on Applications of Computer Vision, pp.18-23, Austin, Texas, USA, Feb. 2007.

[20] E.M. Saad, A. Hamdy, and M.M. Abutaleb, "ReconFigurable Hardware Implementation of a Fast and Efficient Motion Detection Algorithm", 10th WSEAS Int. Conf. MMACTEE'08, Sofia, Bulgaria, pp.40-45, May 2008.

[21] Arseneau, S. and Cooperstock, J.R. "Real-Time Image Segmentation for Action Recognition". Proc. IEEE PACRIM, Pacific Rim Conference on Computers, Visualization and Signal Processing, Victoria, pp.86-89, Aug. 1999.

[22] Cobos, P., Monasterio F., "FPGA Board for Real Time Vision Development Systems", 4th IEEE International Caracas Conference on Devices, Circuits and Systems (ICCDCS), pp: 1-6, 2002.