J Real-Time Image Proc
DOI 10.1007/s11554-014-0418-x

ORIGINAL RESEARCH PAPER

Embedded architecture for noise-adaptive video object detection using parameter-compressed background modeling

Kumara Ratnayake · Aishy Amer

Received: 24 August 2012 / Accepted: 17 March 2014

Abstract Video processing algorithms are computationally intensive and place stringent requirements on performance and efficiency of memory bandwidth and capacity. As such, efficient hardware accelerations are inevitable for fast video processing systems. In this paper, we propose resource- and power-optimized FPGA-based configurable architecture for video object detection by integrating noise estimation, Mixture-of-Gaussian background modeling, motion detection, and thresholding. Due to large amount of background modeling parameters, we propose a novel Gaussian parameter compression technique suitable for resource- and power-constraint embedded video systems. The proposed architecture is simulated, synthesized and verified for its functionality, accuracy and performance on a Virtex-5 FPGA-based embedded platform by directly interfacing to a digital video input. Intentional exploitation of heterogeneous resources in FPGAs, and advanced design techniques such as heavy pipelining and data parallelism yield real-time processing of HD-1080p video streams at 30 frames per second. Objective and subjective evaluations to existing hardware-based methods show that the proposed architecture obtains orders of magnitude performance improvements, while utilizing minimal hardware resources. This work is an early attempt to devise a complete video surveillance system onto a stand-alone resource-constraint FPGA-based smart camera.

K. Ratnayake (✉) · A. Amer
Department of Electrical and Computer Engineering,
Concordia University, Montréal, QC, Canada
e-mail: k_ratnay@ece.concordia.ca

A. Amer
e-mail: amer@ece.concordia.ca

Keywords Field-programmable gate arrays · FPGA · Video signal processing · Noise · Moving objects · Motion detection · Thresholding · Gaussian background update · Gaussian parameter compression

1 Introduction

Video (moving) object detection classifies pixels of video frames into multiple moving regions and a background. This error-prone task is an important initial component in many intelligent video processing applications, including video surveillance, human motion analysis, video compression, video retrieval, and semantic annotation of video scenes. However, detecting moving objects is challenging under uncontrolled conditions. Frequent and quick background adaptations are needed due to non-stationary dynamic pixels, such as changes in environmental conditions (for example, sudden illumination changes, rain, and snow), sudden scene changes (for example, moved object, cast shadows, and ghosts), wavering bushes, moving escalators, and video noise when soared to an unacceptable level.

Various approaches to detect moving objects have been presented in the literature [1–7], varying in computational complexity and accuracy. Fast and effective techniques rely on background subtraction, in which one or more background models are estimated and evolved frame by frame. The salient foreground objects are then detected by comparing current frame with a background frame. Among the myriad of background subtraction techniques, Mixture-of-Gaussian (MoG)-based methods are widely adopted because of the robustness to tolerate variations in non-stationary dynamic background pixels [8, 9]. MoG methods attempt to model each background pixel using a mixture of

M Gaussian distributions. Each distribution maintains three dynamic *Gaussian parameters*—mean, variance, and weight $(\mu_{\hat{i}}, \sigma_{\hat{i}}, \omega_{\hat{i}})$, where $\hat{i} \in \{1, \ldots, M\}$. Gradual background changes are efficiently updated with Gaussian parameters of each distribution.

While the underlined MoG-based methods improve the accuracy of object detection, these techniques require significant computational power, memory bandwidth, and storage. Despite the recent technological advances in semiconductor process technology, conventional software platforms are often unable to deliver the required performance, and thus have frequently failed or prevented the realization of real-time video surveillance systems. Hardware acceleration for MoG-based background subtraction is, thus, inevitable.

However, when targeting a hardware implementation based on fixed-point arithmetic, high dynamic range or precision of video processing parameters, here the Gaussian parameters $\mu_{\hat{i}}, \sigma_{\hat{i}}, \omega_{\hat{i}}$, is critical for accurate processing (background estimation). This effectively introduces major challenges to the implementation of MoG-based background estimation on embedded systems. First, the need for larger dynamic range of fixed-point numbers for each parameter in M Gaussian distributions requires a high memory bandwidth and a large memory capacity. Second, the resultant frequent access to external memory yields excessive power dissipation due to memory I/O port switching. On contrary, embedded systems are constrained by memory bandwidth, memory capacity as well as power consumption. In addressing the computational complexity of general MoG background subtraction methods, the hardware implementation of logical modules poses comparatively less challenges as a significant fraction of these algorithms can be parallelized. We, thus, identify the exorbitant memory bandwidth inherently required by MoG as the most challenging task in its hardware implementation. Existing compression schemes are infeasible for compressing Gaussian parameters as the resource utilization of such methods are substantially high.

Although, it may intuitively appeal for a hardware implementation based on Graphics Processing Units (GPU) [10], there are several drawbacks that prohibit the use of GPUs as the hardware accelerator. Previous research has highlighted that the power consumption of the GPUs is significantly higher than those of other counterparts [11]. GPUs have fixed computational/programming model that requires fitting the algorithms to their architectures and the only interface to the GPUs is PCI Express. On contrary, implementation of video object detection on embedded platform is challenging as embedded processing motes are often constrained by computational power, memory, power consumption, and cost. An architecture that optimizes those stringent constraints is therefore needed. Field-programmable gate arrays (FPGAs) are reconfigurable devices that contain reconfigurable fabrics, built in Intellectual Property (IP) blocks, such as embedded processors, high speed Input Outputs (IOs), multiply accumulate modules, and dedicated memory blocks. FPGAs provide flexibility into hardware as the circuit functionality can be changed by device reconfiguration after the system deployment. Operational concurrency can be achieved by exploiting the inherent parallelism available in FPGA architectures, thus FPGAs are capable of providing high performance required in embedded processing applications.

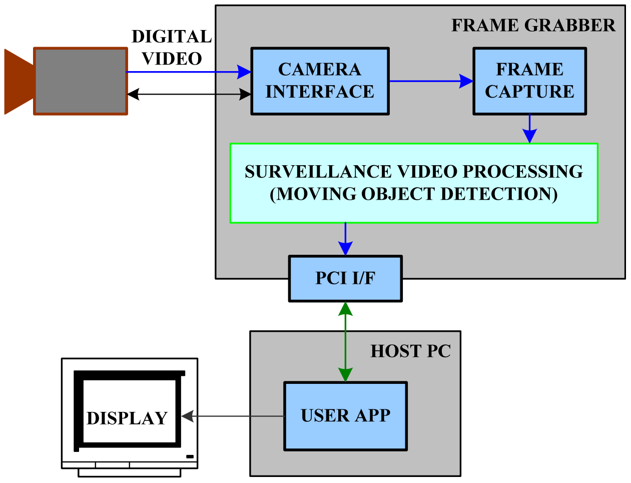


Fig. 1 Embedded development platform for the proposed FPGA-based implementation of autonomous moving object detection. The block *Moving Object Detection* is detailed in Fig. 2

In this paper, we propose an FPGA implementation of a video object detection method that combines noise estimation [12], MoG-based adaptive background modeling [13], and object-change detection (i.e., motion detection and thresholding) [14]. In brief, the motivation for integrating the methods in [12–14], is due to their inherent modularity and simplicity while producing meaningful objects under various conditions. Furthermore, we propose a novel lossless intra- and inter-Gaussian parameter compression technique suitable for implementation on resource- and power-constraint embedded video processing system. The synoptic of the proposed embedded video processing architecture is depicted in Fig. 1. The proposed work sets a basis for our attempt to implement a complete surveillance system (i.e., noise estimation, background update, motion detection, thresholding, object-feature extraction, object tracking, and event detection) onto a stand-alone FPGA-based smart camera. Moreover, the proposed research outlined in this paper significantly improves the accuracy of our early work on FPGA-based video motion detection [15], primarily due to the adaptive background modeling and noise adaptation.

The rest of the paper is organized as follows. Section 2 discusses related work to algorithms and implementations, and motivates the selected methods [12–14]. In Sect. 3, we summarize the referred moving object detection methods [12–14]. An overview of the proposed Gaussian parameter compression algorithm is outlined in Sect. 4. The proposed hardware architecture is presented in Sect. 5. Experimental results are outlined in Sect. 6, and finally, conclusion is presented in Sect. 7.

2 Related work

2.1 Algorithms

Moving object detection methods can be classified based on their automation, spatial accuracy, temporal stability, and computation load [16]. Computationally expensive methods give, in general, accurate results, however, few are tested on a large number of videos, throughout long videos, on noisy videos, and without parameter tuning. Background subtraction-based methods are popular due to their low computation and effectiveness.

Background subtraction methods require a background modeling and update stage to compensate for variable backgrounds. Background modeling techniques are fundamentally centered around: adjacent frame differencing, α-blending the previous frames (temporal averaging), median filtering, predictive filtering, and MoG [1, 3, 8, 9]. Adjacent frame differencing, the simplest form of background model, uses previous video frame as the background frame. The aforementioned method may fail to identify the interior pixels of large homogeneous moving objects. Median filtering-based modeling sets each pixel in the background frame to be the temporal median value at each pixel location of previous L video frames. Although the median filter-based techniques are computationally efficient, these methods fail in modeling non-stationary dynamic background pixels, such as those due to sudden illumination changes, temporal clutter, shadows, and ghosts. While α blending the previous frame is simple and computationally inexpensive, the method distorts the colors behind moving objects, which is commonly known as *ghosting* effect. Other background models use predictive techniques such as Kalman [17] and Wiener [18] filters. These techniques heavily depend on the predefined state transition parameters, thus they may fail in case the color distribution does not fit into a single model. In contrast, background models based on MoG are widely adopted because of the robustness to tolerate variations in non-stationary dynamic background pixels. Furthermore, MoG-based background modeling methods can address disadvantages of the other methods more effectively. These models represent spectral features of various background appearances using multiple Gaussian distributions. Gradual background changes are efficiently updated with mean, variance, and weight parameters of each Gaussian. In [19], Zivkovic presents a MoG method that automatically adjusts to the scene by adapting the MoG parameters and the number of components of the mixture for each pixel. Lee [20] proposes a more complex component classification scheme to handle bootstrapping by including a component match counter into the parameter update. However, both methods [19, 20] are computationally expensive, thus pose great challenges in their hardware realizations.

Much of the above background subtraction and modeling methods have focused on updating the background frame on the pixel level with little or no emphasis on producing connected binary blobs or regions. Such connected blobs are crucial for higher level (object-based) processing such as object tracking or event detection. To this end, we need to merge background modeling with connected-blob detection. MoG methods are specially suitable for dynamic (e.g., outdoor) scene because they implicitly handle dynamic changes, and we thus propose to use the MoG-based background modeling in [13] due to its superior accuracy while still being fast compared to related works such as [19–22]. To produce connected blobs, we propose to use the non-parametric object-change detection method in [14] due to its low computation and its noise and temporal stability. These features forgo spatial accuracy, e.g., at object boundaries in the intended application video surveillance.

Noise is present in any image or video signal and may significantly affect the performance of subsequent video processing tasks such as object segmentation. One way to adapt video processing tasks to noise, is by estimating the noise level. Noise estimation is, however, challenging in structured images. The selected noise estimation method [12] is fast and effective as it finds intensity homogeneous blocks and then estimates noise in these blocks. This makes it reliable for both high noisy and textured images.

2.2 Implementations

An FPGA implementation of object detection with multimodel background subtraction is presented by Appiah and Hunter [23], where the authors use a collection of low-pass filters and maintain background weight, match, and an updating scheme. Attempts are made to minimize the FPGA resources by reducing the floating-point computations. Oliveira et al. [24] propose another FPGA-based implementation of detecting moving objects from a static background using color images. The method is based on chromaticity and brightness distortion computational color model that facilitates distinguishing shading background from the foreground objects. However, the implementations in [23, 24] are only capable of real-time processing of low-resolution videos, yet require considerably large amount of FPGA resources and external memory. Schlessman et al. [25] present an FPGA-based heterogeneous multiprocessor architecture for detecting video objects by background subtraction. Indeed, the implementation adopts a PowerPC processor fabricated in the FPGA to realize a processing throughput of 30 frames per second (fps) for the CIF video resolution, but the whole implementation may easily be unstable due to data collision on the transferring bus. In [26, 27], a design of an embedded automated digital video surveillance system is presented. Here, the authors implement MoG-based background subtraction method morphological operations, labeling and feature extraction on a Xilinx FPGA platform. In [26, 27], authors adopt word -ength reduction scheme and utilize pixel locality to reduce the memory bandwidth by more than 70 %. However, the accuracy of MoG parameters is crucial for accurate background modeling, hence [26, 27] may fail in environments with slow moving objects. Genovese and Napoli [28] propose an optimized FPGA implementation of MoG and binary image denoising algorithms. The MoG parameters are assigned with smaller bit-widths, which can create rounding errors in fixed-point arithmetic computations causing inaccurate estimation of slow background updates as explained in Sect. 4.

3 Unified noise-adaptive video object detection through adaptive background update

The proposed framework is illustrated in Fig. 2, and detailed descriptions of each module are given next. For the remainder of this paper, $I(n, s)$ is used to denote the luminance pixel intensity of image I at spatial location $s = (i, j)$ and frame time n. s or n may be dropped for clarity if these subscripts are not relevant in the description.

3.1 Noise estimation

In this section, we briefly outline the structure-based spatial noise estimation algorithm [12], where the image is divided

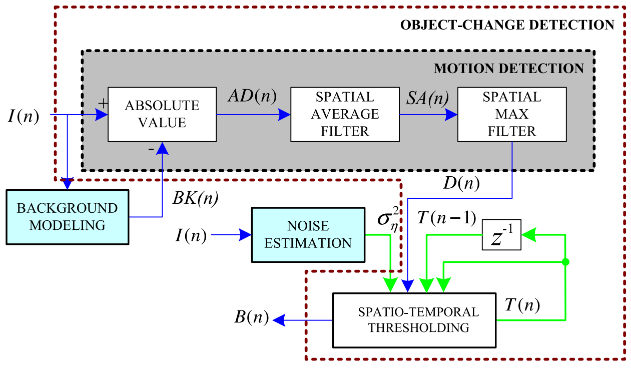


Fig. 2 Moving object detection: noise estimation [12], MoG background model [13], spatio-temporal object-change detection [14]

into blocks of size $W \times W$, and for each block, a homogeneity measure ξ_{Bh} is computed using eight high-pass directional filters. Notice that if the image dimensions are not multiple of W, then rectangular blocks are extracted at the left and bottom boundaries. The mask of the horizontal direction scanning window on the image function I for $W = 5$ is

$$I_o(i) = -I(i-2) - I(i-1) + 4 \times I(i) - I(i+1) - I(i+2) \tag{1}$$

The homogeneity measure ξ_{Bh} is obtained by adding the absolute values of all eight high-pass directional filters. The variance σ^2_{Bh} of each block is then calculated with

$$\sigma^2_{Bh} = \frac{\sum_{(i,j)\in W_{ij}} \left(I(i,j) - \mu_{Bh}\right)^2}{W \times W} \tag{2}$$

Here, W_{ij} denotes a block in which the input image is divided into, and μ_{Bh} is the sample mean defined as

$$\mu_{Bh} = \frac{\sum_{(i,j)\in W_{ij}} I(i,j)}{W \times W} \tag{3}$$

The homogeneity measures ξ_{Bh} is sorted and variances σ^2_{Bh} satisfying the condition given in Eq. 4 are averaged to obtain the global noise variance σ^2_η.

$$\left| 10\log_{10}\frac{255^2}{\sigma^2_{Bh}} - 10\log_{10}\frac{255^2}{\sigma^2_{REF}} \right| < t_\sigma \tag{4}$$

In Eq. 4, σ^2_{REF} is defined as a reference variance, which is chosen as the median of the variances of the three most homogeneous blocks. t_σ is a user-defined threshold value, which can be seen as the maximal error between the true variance and the estimated variance [12]. As in [12], we set t_σ to 3 dB.

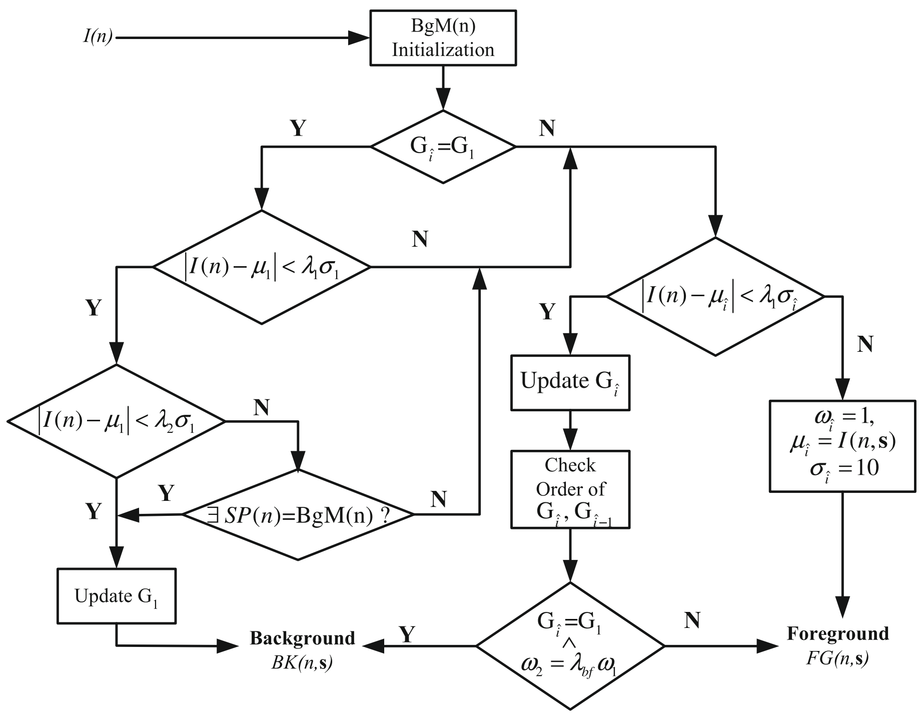
3.2 Background modeling

Figure 3 illustrates the referred background update algorithm [13], while Table 1 lists its main symbols. The background pixel is modeled by a mixture of M Gaussian components ($\mathbf{G}_1 \ldots \mathbf{G}_M$). Each component is characterized by three parameters: weight, mean, and variance $(\omega_{\hat{i}}, \mu_{\hat{i}}, \sigma_{\hat{i}})$, where $\hat{i} \in \{1, \ldots, M\}$. After receiving a new frame $I(n)$, a comparison is executed between intensities for each input pixel and the background model $BgM(n)$ to find the matched component and then the background is updated accordingly.

The algorithm in [13] uses Eq. 5 to update the mean value $(\mu_{\hat{i}})$ of a matched component:

$$\mu_{\hat{i}}(n) = \mu_{\hat{i}}(n-1) + \alpha \times \Delta\mu \tag{5}$$

where



I(n)
BgM(n) Initialization
$G_{\hat{i}}=G_1$
Y
N
$|I(n)-\mu_1|<\lambda_1\sigma_1$
$|I(n)-\mu_1|<\lambda_2\sigma_1$
$\exists SP(n)$=BgM(n) ?
Update G_1
$|I(n)-\mu_{\hat{i}}|<\lambda_1\sigma_{\hat{i}}$
Update $G_{\hat{i}}$
Check Order of $G_{\hat{i}}, G_{\hat{i}-1}$
$\omega_{\hat{i}}=1$, $\mu_{\hat{i}}=I(n,\mathbf{s})$ $\sigma_{\hat{i}}=10$
$G_{\hat{i}}=G_1 \wedge \omega_2=\lambda_{bf}\omega_1$
Background BK(n,s)
Foreground FG(n,s)

Fig. 3 Flowchart of the selected background update algorithm [13]

Table 1 Parameter definitions of the referred background update algorithm [13]

Parameter	Definition
α	Adaptation rate
$SP(n)$	Surrounding pixels
$BK(n)$	Background frame
$I(n)$	Current input frame
$BgM(n)$	Current background model
λ_1, λ_2	Hysteresis-based cluster thresholds
λ_{bf}	Secondary background component threshold

$$\Delta\mu = I(n) - \mu_{\hat{i}}(n-1) \tag{6}$$

which we here approximate to:

$$\mu_{\hat{i}}(n) = \begin{cases} \mu_{\hat{i}}(n-1)+1 & : \ \Delta\mu > \lambda_2 \times \sigma_1; \\ \mu_{\hat{i}}(n-1)-1 & : \ \Delta\mu < -\lambda_2 \times \sigma_1; \\ \mu_{\hat{i}}(n-1) & : \ \text{otherwise.} \end{cases} \tag{7}$$

Equation 7 effectively updates the mean value reducing many computations. Although such optimization minimizes FPGA resource requirement, MoG parameters still require large amount of memory bandwidth and capacity. We therefore propose to compress these parameters using a hardware-amenable lossless compression method. Details of the proposed MoG parameter compression algorithm and its resource- and power-optimized implementation are given in Sects. 4 and 5.3, respectively.

3.3 Object-change detection

The object-change detection method [14] primarily consists of two modules: motion detection, and spatio-temporal thresholding (see Fig. 2). The motion detection finds first the absolute frame difference, $AD(n)$ at time instant n, between the current frame $I(n)$ and background frame $BK(n)$. $AD(n)$ is then spatially filtered by both a 3×3 average and a 3×3 max filter. The 3×3 average filter computes the arithmetic mean of the 9 nearest neighbors of each pixel in the difference frame. Similarly, the 3×3 max filter produces the maximum pixel of the 9 nearest neighbors of each pixel in the averaged frame.

In the thresholding module, a global spatial threshold T_g is first computed as follows. The spatially filtered frame $D(n)$ is divided into K consecutive non-overlapping blocks $W_k, k \in \{1, \ldots, K\}$. The histogram of each W_k is split into L equal sections. The most frequent gray-level g_{pl} of each histogram section is found and

$$T_g = \frac{\sum_{k=1}^{K}(\lambda_l + \mu_k)}{K \times L + K} \tag{8}$$

where $\lambda_l = \sum_{l=1}^{L} g_{pl}$ and μ_k is the pixel average of W_k. Note that T_g is obtained using block local and global data. T_g is then proportionally adapted to the noise variance σ_η^2 using

$$T_\varsigma = T_g + a \times \sigma_\eta^2 \tag{9}$$

where $0 < a < 1$ (as in [14], we use $a = 0.1$) and σ_η^2 is estimated using the method described in [12]. This noise-adapted T_ς is then quantized to maintain spatio-temporal stability where quantization down to three levels yields good results [14]. The quantized threshold T_q is passed through a memory system that holds the threshold of the previous frame and determines the threshold $T(n)$ based both on new quantized threshold T_q as well as the previous threshold $T(n-1)$. Finally, $D(n)$ is globally thresholded by $T(n)$ creating a binary frame $B(n)$.

4 Proposed parameter compression algorithm

The fundamental characteristics of a compression algorithm for coding Gaussian parameters on embedded systems are lossless compression, minimal utilization of hardware resources, high throughput, and low compression ratio. Here, compression ratio is defined as the ratio of number of output bits to the number of input bits and, consequently, the smaller its value the better the compression. As Gaussian parameters are recursively updated, any slight deviation to the parameters can accumulate and degrade the robustness of the background subtraction. Thus, lossless compression of Gaussian parameters is essential. A low compression ratio is needed to reduce the massive memory bandwidth imposed by the Gaussian parameters. Moreover, hardware overhead of the implementation should be relatively small compared to the entire video processing system, and a high throughput is required to process HD-720p30 video streams in real time.

Our experiments manifest that MoG-mean and MoG-variance require 24 bits each while MoG-weight needs 16 bits to accurately estimate slow background update. For $M = 3$, each background pixel requires $3 \times (24 + 24 + 16)$ bits (24 Bytes). Moreover, if color pixels are processed for background estimation, the number of bytes required to represent the MoG parameters increase to 72 Bytes. For each new pixel, the Gaussian parameters are retrieved from memory, then updated and written back to the memory. Hence, MoG implementation demands for high memory bandwidth, capacity, and power consumption often limiting the overall performance of a real-time embedded system. Total memory bandwidth required by uncompressed M MoG components, $BW_{original}$, is calculated by

$$BW_{original} = 2 \times M \times N_G \times \gamma \times f \tag{10}$$

where N_G is the total number of bytes required to represent the parameters in a single MoG component (N_G = 8 Bytes in our study), γ is the number of pixels in a frame, and f is the frame rate. The factor 2 in the Eq. 10 is due to read and write of memory accesses. For example, $BW_{original} = 1.3$ GB/s is required for retrieval and update of MoG parameters from external memory when estimating background of a monochrome HD-720p30 (1,280 × 720, 30 fps) video

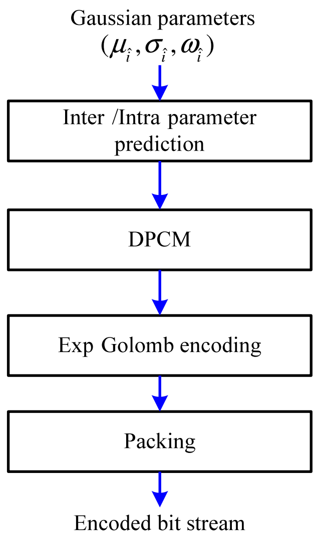

Gaussian parameters
$(\mu_{\hat{i}}, \sigma_{\hat{i}}, \omega_{\hat{i}})$
Inter /Intra parameter prediction
DPCM
Exp Golomb encoding
Packing
Encoded bit stream

Fig. 4 Flowchart of the proposed compression algorithm

source. Although 1.3 GB/s of memory bandwidth can be available on an embedded system when the clock frequency to a Double Data Rate (DDR) memory is 266 MHz with a 32 bits data bus, there are other components in the processing pipeline, such as image filtering and object-change detection modules, that typically require simultaneous access to memory. Hence, the MoG implementation on an embedded system is confronted with an acute memory bandwidth problem, and consequently requiring to compress the Gaussian parameters.

No compression algorithm catered to compress the Gaussian parameters and that satisfies all of the aforementioned conflicting requirements has been reported in the literature hitherto. We propose a novel lossless compression algorithm, summarized in Fig. 4, that exploits parallel processing architecture to achieve a high throughput with a low compression ratio while maintaining low utilization of hardware resources.

Our study shows that the Gaussian parameters of real video sequences have strong intra-correlation in both the horizontal and vertical directions. Moreover, there exists inter-correlation among the Gaussian components of means and variances $(\mu_{\hat{i}}, \sigma_{\hat{i}})$ independently. The main idea of the proposed compression algorithm is to maximally exploit these correlations using inter- and/or intra-parameter prediction for differential pulse code modulation (DPCM), which calculates a prediction error defined as the difference between the original and predicted parameters. The various prediction-error methods employed in the proposed algorithm are defined in Eqs. 11 and 12.

$$\left.\begin{aligned} \Delta_{\mu_{\hat{i}}}(i,j) &= \mu_{\hat{i}}(i,j) - \mu_{\hat{i}}(i-1,j) \\ \Delta_{\sigma_{\hat{i}}}(i,j) &= \sigma_{\hat{i}}(i,j) - \sigma_{\hat{i}}(i-1,j) \\ \Delta_{\omega_{\hat{i}}}(i,j) &= \omega_{\hat{i}}(i,j) - \omega_{\hat{i}}(i-1,j) \end{aligned}\right\} \hat{i} = 1 \tag{11}$$

$$\left.\begin{aligned} \Delta_{\mu_{\hat{i}}}(i,j) &= \mu_{\hat{i}}(i,j) - \mu_{\hat{i}-1}(i,j) \\ \Delta_{\sigma_{\hat{i}}}(i,j) &= \sigma_{\hat{i}}(i,j) - \sigma_{\hat{i}-1}(i,j) \\ \Delta_{\omega_{\hat{i}}}(i,j) &= \omega_{\hat{i}}(i,j) - \omega_{\hat{i}}(i-1,j) \end{aligned}\right\} \hat{i} \neq 1 \tag{12}$$

Here, $\Delta_{\mu_{\hat{i}}}, \Delta_{\sigma_{\hat{i}}}, \Delta_{\omega_{\hat{i}}}$ denote the prediction errors of the Gaussian parameters obtained by subtracting the predicted parameter values from the originals $\mu_{\hat{i}}, \sigma_{\hat{i}}, \omega_{\hat{i}}$, and (i,j) is the spatial coordinate. $\hat{i} \in \{1, \ldots, M\}$ represents the Gaussian component. At the beginning of each frame line, the prediction is set to zero for all parameters. Only intra-prediction is employed for weights $\omega_{\hat{i}}$, as $\omega_{\hat{i}}$ exhibit weak inter-correlation among its M components. A mixture of intra- and inter-prediction is applied for the other parameters. We limit the intra-prediction to horizontal direction only, to minimize the hardware overhead.

The prediction errors are binary coded by Exp-Golomb algorithm [29] producing *Code Words* (*CW*). Exp-Golomb codewords are regular logical structures that consist of predetermined code pattern and require no decoding tables. Each exp-Golomb codeword is constructed with M_{EG} leading zeros, "1" in the center and M_{EG} bit long information field, and can be expressed as follows:

$$[0_1, 0_2, \ldots, 0_{M_{EG}}][1][INFO] \tag{13}$$

where

$$M_{EG} = \lfloor log_2[CN + 1] \rfloor \tag{14}$$

and *INFO* is an M_{EG} bits information carrying field and can be obtained by

$$INFO = CN + 1 + 2^{M_{EG}} \tag{15}$$

In Eqs. 14 and Eq. 15, the term *CN* is called *Code Number* which is the input to the Exp-Golomb encoder. The corresponding length of each *CW*, *Code Lengths* (*CL*), is $2M_{EG} + 1$ bits. The block *Packing* packs these Exp-Golomb coded words into the memory bus width (32 bits in our case) producing the final encoded bit stream.

5 Proposed pipelined architecture and implementation

The overall system-level architecture of the FPGA design is illustrated in Fig. 5. It consists of *Multi Port Data Router (MPDR)*, background modeling, Codec for Gaussian parameter compression, noise estimation, motion detection, and spatio-temporal thresholding. Figure 6 shows the pipeline stages of the processing blocks. As can be seen from Figure 6, the proposed architecture contains four pipeline stages. *NOISE ESTIMATION* and *MOG-PARAMETER DECOMPRESSION* are mutually exclusive modules, thus they are executed in parallel. *NOISE ESTIMATION* has $N_1 + 14$ clock cycles pipeline delay due to intrinsic sorting required in the algorithm. *MOG-PARAMETER DECOMPRESSION* pipeline produces its first output in 12 clock cycles, at which point *BACKGROUND MODELING* can be started. Number of pipeline delay in *BACKGROUND MODELING* is 35 clock cycles. Both *MOTION DETECTION* and *MOG-PARAMETER COMPRESSION* are executed in parallel due to the data independence. *MOTION DETECTION* and *MOG-PARAMETER COMPRESSION* have 13 and 17 clock cycles, respectively, in pipeline delay. *SPATIO-TEMPORAL THRESHOLDING* has a pipeline delay of $N_2 = N_1 + \frac{N_1}{K}$ clock cycles, where K, as defined in Sect. 3.3, is the total number of blocks in which frame $D(n)$ is divided into. Details of the architecture for these modules are described next.

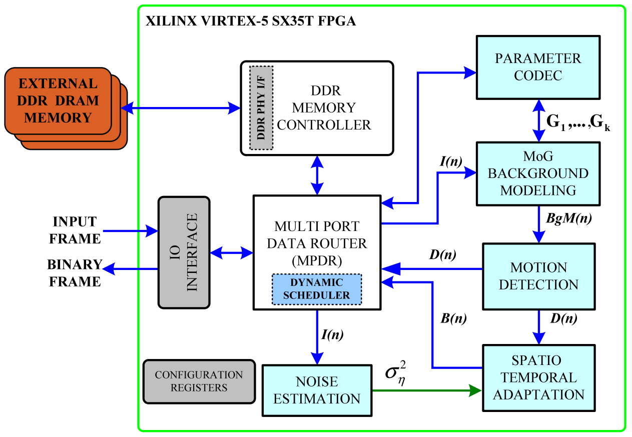


Fig. 5 System-level architecture of the proposed FPGA-based implementation of object detection

5.1 Proposed MPDR architecture

An efficient management of data transfers within a system is the key to any real-time hardware implementation. In our implementation, we designed a configurable and versatile *MPDR* architecture that can be easily configured by a simple set of registers. The proposed *MPDR* consists of 4 KB deep *First In First Out (FIFO)* memories connected to each read and write *MPDR* channels, a *DYNAMIC SCHEDULER* to manage these *FIFOs*, and a DDR memory controller. A write transfer to the memory is initialized by filling the corresponding write *FIFO* (up to a maximum of 2 KB), and sending a request to *DYNAMIC SCHEDULER*. Whenever a read FIFO is half empty, a read request is automatically initialized. An internal cache memory is used to store the addresses and transfer descriptions of each *MPDR* channel. The proposed *DYNAMIC SCHEDULER* is based on a round-robin arbitrator that arbitrates all parallel requests from each channel and serves the selected *MPDR* channel.

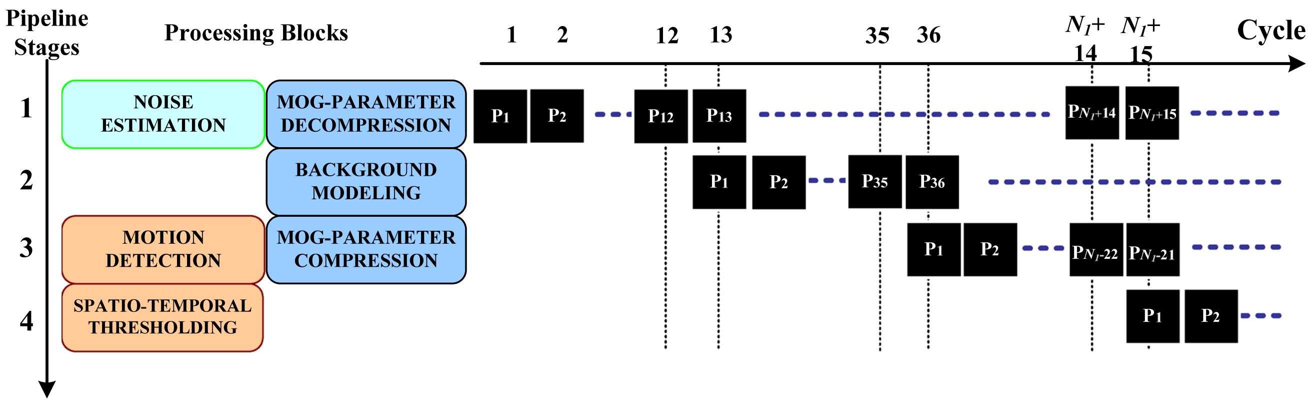
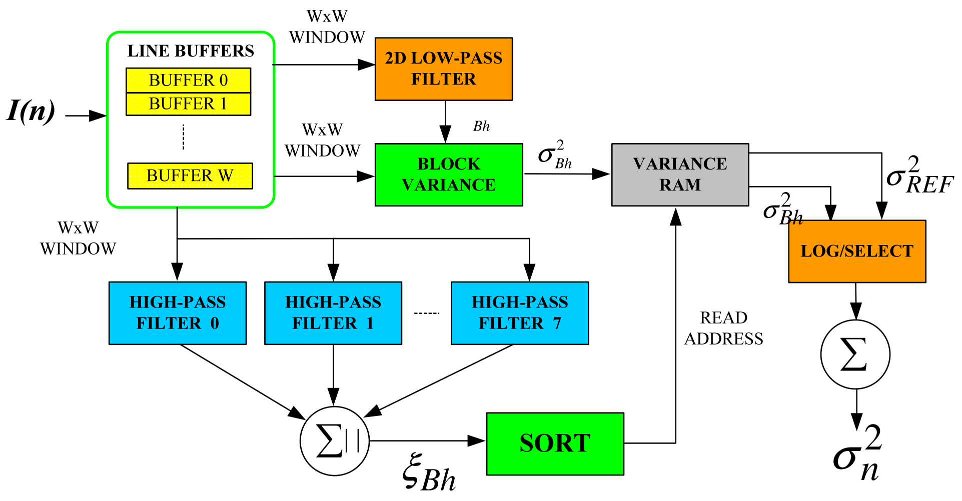


Fig. 6 Pipeline stages of main processing blocks. *Black squares* on the right side indicate stream of data (pixels or MoG parameters). N_1 is the number of clock cycles for a full frame

Fig. 7 Proposed Architecture of the noise estimation algorithm



5.2 FPGA-based implementation of noise estimation

Figure 7 illustrates the overall architecture of the noise estimation algorithm. As can seen, *LINE BUFFERS* are utilized with BRAMs, which generate $W \times W$ blocks. As in [12], we have set $W = 5$. In the 2D *LOW-PASS FILTER* block, the sample mean μ_{Bh} is produced and passed to *BLOCK VARIANCE* module which computes the variance σ^2_{Bh}. These block variances are then stored in the *VARIANCE RAM*. A set of eight *HIGH-PASS FILTERS* produces directional filters, and absolute value of the result of the directional filters are summed to produce the homogeneity measures ξ_{Bh}. The smaller values of ξ_{Bh} represent more homogeneous blocks. The *SORT* module sorts ξ_{Bh} in an ascending order. Then, the indexes of the least 10 % of ordered ξ_{Bh}, i.e., most homogeneous blocks, are sent as the read address to the *VARIANCE RAM*. *VARIANCE RAM* then outputs corresponding μ_{Bh} and μ_{REF}. Implementation of *SORT* module is based on our previous work [30], which we have presented an efficient FPGA implementation to sort large volume of data using a modified counting-sort algorithm. *LOG/SELECT* block finds the logarithmic value of μ_{Bh} and μ_{REF}, which are compared to an application-dependent threshold as seen in Eq. 4 to select only the valid variances. These variances are then summed in the accumulator to obtain the global noise variance-σ^2_n.

5.3 Proposed compression architecture

The pipeline architecture of the proposed compression engine (encoder) is shown in Fig. 8. As decompression is directly opposite of the compression, its description is omitted. The proposed architecture is composed of a linear array of *M BASIC COMPRESSION ENGINES (BCE)*, which is expanded in Fig. 8 for compressing mean parameter of the first Gaussian component. To increase the throughput, the implementation of *BCE* is pipelined in three stages—*DPCM*, *Exp-Golomb Encoder*, and *BARREL SHIFTER*. Each *BCE* can receive one MoG parameter each clock cycle and it has a pipeline latency of 9 clock cycles.

The block *DPCM* calculates the difference of the two input signals Δs and converts the result to a positive integer θ through

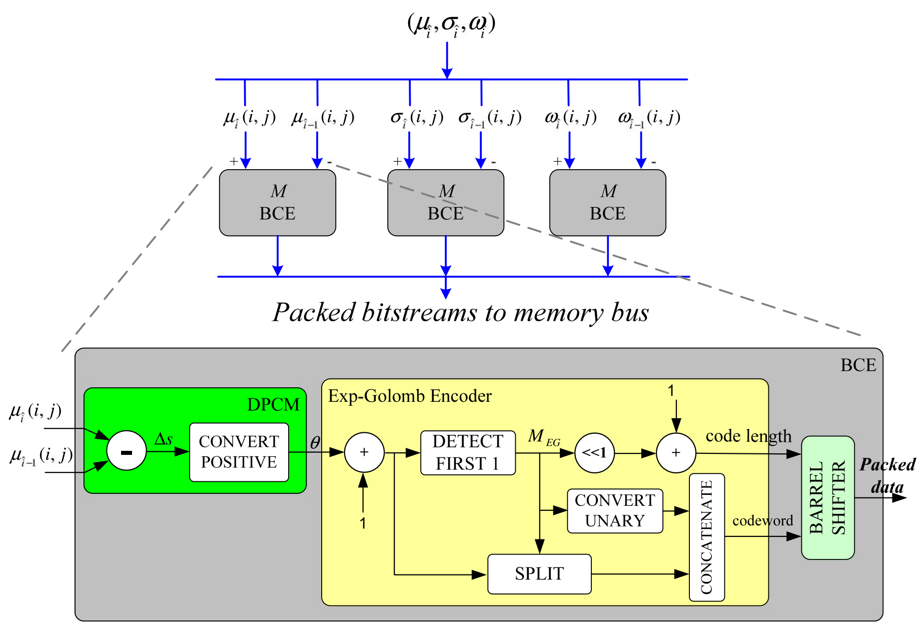


Fig. 8 Overall architecture of the proposed compression unit

CONVERT POSITIVE module, as *Exp-Golomb Encoder* takes only positive integers at its input in the proposed implementation. The signed conversion is governed by:

$$\theta = \begin{cases} 2\Delta s & : \ \Delta s \ > 0; \\ 2\Delta s - 1 & : \ \text{otherwise}. \end{cases} \tag{16}$$

Direct hardware implementation of conventional Ex-Golomb coding (Eqs. 13, 14 and 15) is inefficient due to the complex relation in *CW* and *CL*. Hence, we adopt the modified coding number technique [31] that constructs the Ex-Golomb codewords with reduced hardware complexity. The underlining principle in [31] is simply to add 1 to the original *CN*:

$$CN_{modified} = CN_{original} + 1 \tag{17}$$

Then, M_{EG} is implicitly obtained by counting the number of bits to the first 1 in the $CN_{modified}$ from its Least Significant Bit (LSB), and *INFO* is the M_{EG} LSBs in the $CN_{modified}$ [31]. Clearly, this avoids computing logarithmic operation in Eq. 14 and evaluating *INFO* field using Eq. 15.

In *Exp-Golomb Encoder* block (Fig. 8), θ is incremented by 1 to generate $CN_{modified}$. (Notice that this incrementor can be removed and merged with Eq. 16, which results in an area-optimized implementation). The block *DETECT FIRST 1* determines M_{EG}, i.e., the first occurrence of 1 in $CN_{modified}$, and returns the number of bits to its position from the LSB. *CONVERT UNARY* block converts M_{EG} into unary notation, i.e., M_{EG} zeros, followed by a single one. The result is concatenated in *CONCATENATE* block with the output of *SPLIT* module, in which *INFO* field is formed by extracting M_{EG} LSBs from $CN_{modified}$.

BARREL SHIFTER takes Golomb codewords and corresponding *code length* as inputs and packs the codewords to fit into the system memory bus, which is 32-bit wide. The packed bitstreams are stored in the external memory. The proposed architecture requires no local memory and its computational complexity is relatively low, and yields over 50 % reduction in the required memory bandwidth by Gaussian parameters.

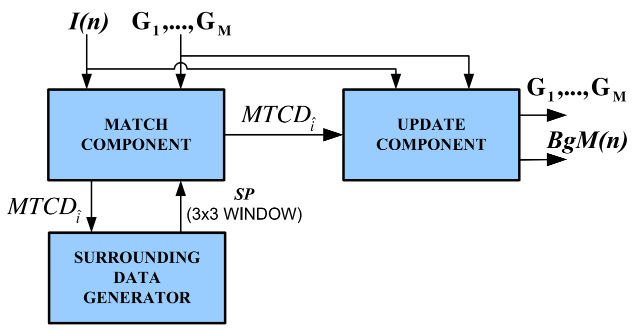


Fig. 9 Top-level data flow diagram of the MoG background modeling

5.4 FPGA architecture for MoG background modeling

The top-level block diagram of the proposed architecture for the modified MoG modeling is depicted in Fig. 9. This module takes the inputs $I(n)$ and M Gaussian components $(\mathbf{G}_1\ldots\mathbf{G}_\mathbf{M})$, and outputs the updated Gaussian components along with the current background frame $BgM(n)$. The proposed architectures of the various blocks in Fig. 9 are described next.

5.4.1 Match component module

This module determines if $I(n)$ has a matching background by comparing with the M Gaussian components. The

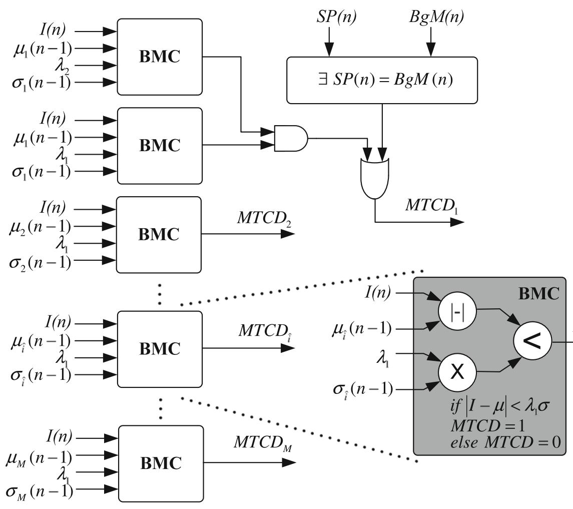


Fig. 10 The proposed architecture of *MATCH COMPONENT* block

proposed architecture is presented in Fig. 10, and its corresponding pseudo code is given in Algorithm 1. The architecture primarily consists of a *scalable* array of *Basic Matching Circuit (BMC)*. Each *BMC* produces $MTCD_{\hat{i}}$ ($\hat{i} \in \{1,\ldots,M\}$) binary output signals that determine whether a matching occurs with $\hat{i}$th Gaussian components. Matching with the $\mathbf{G}_1$ component is performed by utilizing two *BMC* components and a additional circuitry to determine if current pixel has surrounding background, i.e., $\exists SP(n) = BgM(n)$, where $SP(n)$ is produced from *SURROUNDING DATA GENERATOR* module.

```
MatchComponent();
// Determines if $I(n)$ matches with $\hat{i}^{th}$ Gaussian component -
   $MTCD_{\hat{i}}$
// $MTCD_{\hat{i}} \leftarrow 1$ match found
$MTCD_1, MTCD_2, ..., MTCD_M \leftarrow 0$;
for $i \leftarrow 1$ to $M$ do
    if $G_{\hat{i}} = G_1$ then
        if $(I(n) - \mu_1) < \lambda_1\sigma_1$ then
            if $(I(n) - \mu_1) < \lambda_2\sigma_1 \vee \exists SP(n) = BgM(n)$ then
                // $\mathbf{G}_1$ is a matched component
                // Update $(\mathbf{G}_1)$
                $MTCD_1 \leftarrow 1$ ;
            end
        end
    else
        if $(I(n) - \mu_{\hat{i}}) < \lambda_1\sigma_{\hat{i}}$ then
            // $\mathbf{G}_{\hat{i}}$ is a matched component
            // Update $(\mathbf{G}_{\hat{i}})$
            $MTCD_{\hat{i}} \leftarrow 1$;
        end
    end
end
```

Algorithm 1: Pseudo-code for *MATCH COMPONENT* circuit

5.4.2 *Surrounding data generator module*

This module generates neighborhood pixels around (i, j) of both input and background. The architecture utilizes two sets of BRAMs (RAMB16_S18_S18) as line buffers to store the previous two lines as depicted in Fig. 11, and outputs the neighborhood pixels.

5.4.3 *Update component module*

The proposed architecture of *UPDATE COMPONENT* module is rendered in Fig. 12. This module consists of two blocks for updating matched and unmatched Gaussian components, which are appropriately selected through the output *MUX* by $MTCD_{\hat{i}}$ signals. In the *UNMATCHED COMPONENT UPDATE* block, Gaussian components are reordered by weight $\omega_{\hat{i}}$, and the least weighed component, $\mathbf{G_M}$, is reset with $\mu_M = I(n)$, $\sigma_M = 10$, and $\omega_M = 1$.

The *MATCHED COMPONENT UPDATE* block utilizes an array of *BASIC UPDATE CIRCUITS (BUC)*, and the output of *BUC* is reordered by weight. The proposed architecture of BUC is shown in Fig. 13, where each parameter $(\mu_{\hat{i}}, \sigma_{\hat{i}}, \omega_{\hat{i}})$ is updated using trivial arithmetic operations. The control logic *MU CONTROL* adopts Eq. 7, which facilitates the calculation of mean parameters. *WEIGHT CONTROL* determines (per frame basis) sudden illumination change by comparing number of foreground pixels n_sdn with a predefined threshold T_sdn. If sudden illumination change occurs, then $\omega_{\hat{i}}$ is assigned to ω_1, otherwise $\omega_{\hat{i}}$ is incremented by one.

5.5 Proposed architecture for object-change detection

The selected object-change detection consists of motion detection and spatio-temporal thresholding. Our architecture for motion detection and the MPDR is scalable in that motion detection can be configured into the two modes, background $BK(n)$ and previous $I(n-1)$ frame, on the fly. In the former case, the MPDR is programmed to store the acquired frame as the background frame in the memory and it continuously reads the background frame and sends it to the motion detection module along with $I(n)$. In the later case, the MPDR transfers newly arrived $I(n)$ to the memory for future processing as well as to the motion detection module. At the same time, the MPDR reads $I(n-1)$ that was stored in the memory (during the last frame time) and sends it to the motion detection module. The output frame $D(n)$ of the motion detection is routed back to the memory and to the spatio-temporal thresholding node. The spatio-temporal thresholding block takes a full frame time to compute a threshold, hence it is necessary to buffer the frame being processed in the memory until a valid

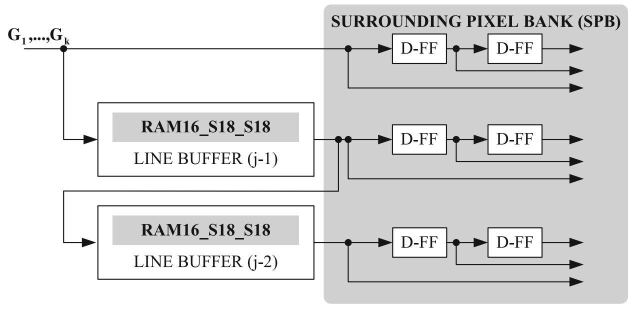


Fig. 11 Resource-optimized architecture for the *Surrounding Data* block for MoG parameters. An exact replica is employed for the input frame *I(n)*

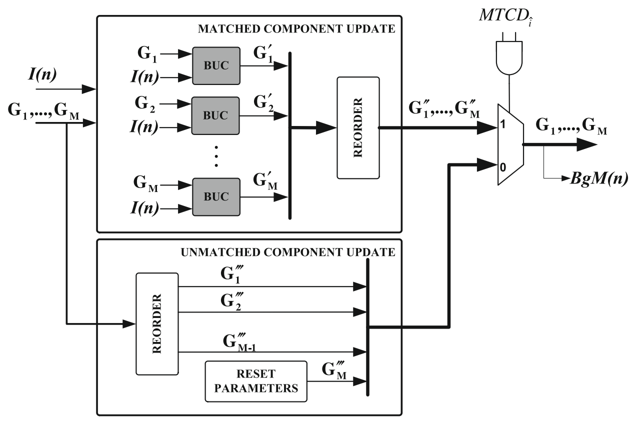


Fig. 12 Architecture of the *Update* module

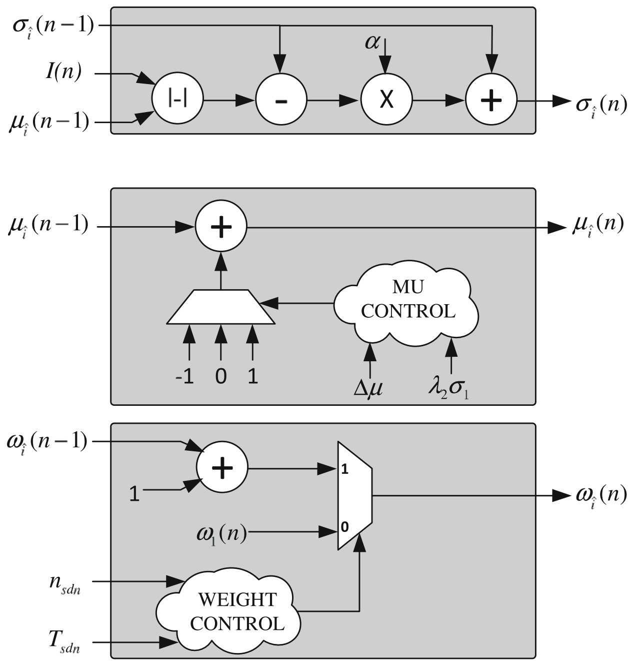


Fig. 13 The proposed architecture of the *BUC* module

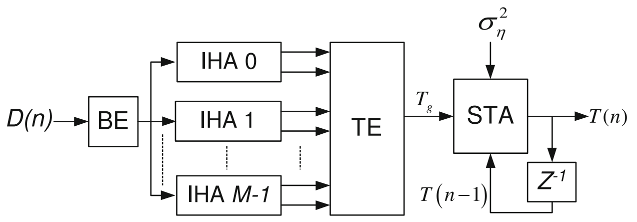


Fig. 14 High-level architecture of *Spatio-Temporal Thresholding*

threshold is available. The proposed MPDR architecture manages all these massive data parallelism in such a way that is seamless to any of the processing blocks.

5.5.1 Configurable motion detection implementation

The absolute difference frame $AD(n)$ is computed by a simple subtractor and its absolute value is routed to the spatial average and max filters. We have architected the spatial filters to be flexible and configurable in number of ways: (1) our implementation can change the size of the both filters from any configuration between 1×1 and 5×5 online, and (2) the frame resolution is programmable allowing to support different video cameras. The architecture is designed in a modular manner, so that future design expansions can be easily feasible. For instance, if the design has to support a video camera with more than 2 KB line width, it can be achieved using multiple instances of the existing modules. We also minimized the memory bandwidth that would require to write and read previous lines for two-dimensional filters using internal Block of Random Access Memories (BRAMs) as line buffers.

5.5.2 Spatio-temporal thresholding architecture

The high-level architecture designed for the spatio-temporal thresholding is shown in Fig. 14. Notice that the noise variance σ_n^2 is obtained from the architecture presented in Sect. 5.2.

The novelty of this architecture is that it does not require any external memory to extract the individual blocks. The *Block Extractor (BE)* splits the motion-detected data into M vertical blocks, which are then fed into M *Intensity Histogram Analysis (IHA)* modules. Each *IHA* generates μ_k and λ_l for the corresponding block. The *Threshold Estimator (TE)* takes those values to produce T_g for *Spatio-Temporal Adaptation (STA)* module. The *STA* consists of an adder that adds T_g to a weighted value of noise variance to get T_ς, and two priority encoders. The first encoder produces T_q by quantizing T_ς down to three quantization levels which are defined with three user programmable registers. The second priority encoder selects $T(n)$ according to T_q and $T(n-1)$.

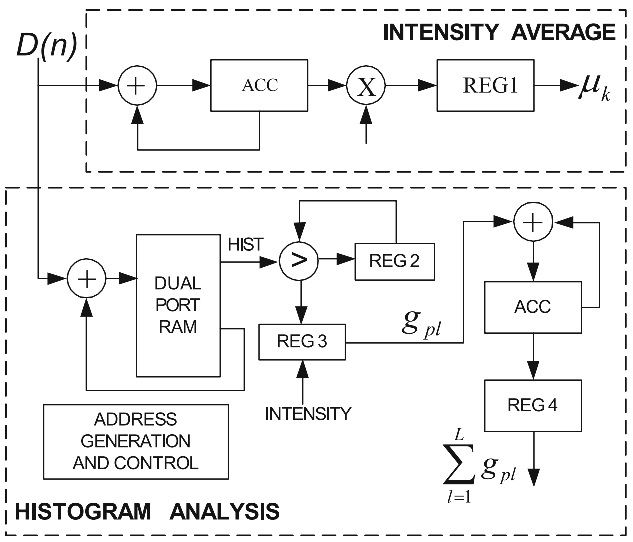


Fig. 15 Architecture of *Intensity Histogram Analysis* module

5.5.3 *Architecture of the IHA module*

The *IHA* consists of two main processing nodes—*Intensity Average* and *Histogram Analysis* which compute μ_k and λ_l, respectively, and a controller and an *Address Generation* unit, that generate the signals required to control these processing nodes. The overall architecture is shown in Fig. 15. In the *Intensity Average* module, we use a multiplier as a divider to obtain the average value. Denominator of this divider is the total number of pixels, which is known *a priori*, and we compute the inverse of denominator and program it through a configurable register.

Hence, the resource usage is minimal and the result of the average is obtained with less pipelined delay when compared to a pure divider usage. The *Histogram Analysis* block first calculates the histogram of the input frame using a BRAM and an adder. After the entire frame data for a $W \times H$ block is entered, the histogram will be available in the BRAM. As in the referred algorithm [14], we use $W = H = 3$ in our implementation. When the histogram is sequentially read, REG 2 holds the maximum value within an interval $l, l \in \{1, \ldots L\}$, and REG 3 keeps the corresponding gray value, g_{pl}. Once the complete histogram is read, g_{pl} is accumulated over the entire intervals, and the result of λ_l will be stored in the Reg 4.

5.5.4 *Architecture of the TE module*

The architecture of the threshold estimator is shown in Fig. 16. The inputs, μ_k and λ_l, to the *TE* block arrive in serially. This allows us to use two multiplexers to select the appropriate operands to the accumulator, which minimizes the resource usage. After all the data of an entire frame have arrived, T_g, will be available in the REG1.

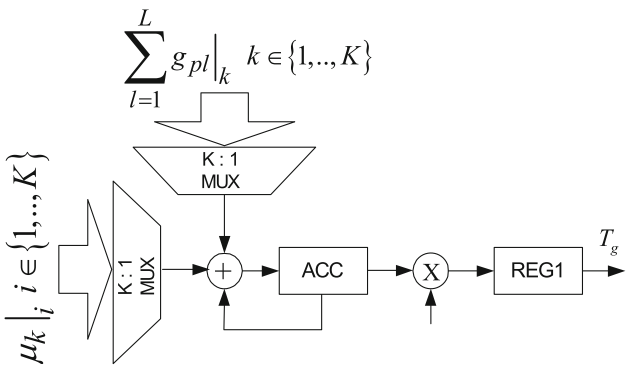

$\sum_{l=1}^{L} g_{pl}\Big|_k \quad k \in \{1,..,K\}$
K : 1 MUX
$\mu_k\Big|_i \; i \in \{1,..,K\}$
K : 1 MUX
ACC
X
REG1
T_g

Fig. 16 *Threshold Estimator* architecture

6 Experimental results

In this section, we present quantitative and qualitative experimental results of the proposed hardware architecture. We select a set of standard indoor and outdoor video sequences with varying object type, size, speed, and illumination. Furthermore, we present FPGA resource utilization and implementation results, and rationally compare some of the existing related work with the proposed architecture.

6.1 Gaussian parameter compression

Figure 17 shows the overall memory bandwidth reduction yielded by employing the proposed compression architecture for various video resolutions. Notice that memory bandwidth for uncompressed MoG parameters, $BW_{original}$, is calculated by Eq. 10, and the memory bandwidth required by compressed MoG components, $BW_{\text{compressed}}$, is calculated by

$$BW_{\text{compressed}} = 2 \times \beta \times f \qquad (18)$$

where β is total number of bytes required to represent compressed MoG parameters in a frame. In our simulation, β is empirically obtained by counting the number of bytes on the memory bus for each frame.

Table 2 lists the average compression ratio for Gaussian parameters with various video sequences. Note that a low compression ratio yields a better parameter compression. The proposed scheme employs intra-prediction for $(\omega_{\hat{i}}, \mu_1, \sigma_1)$, and inter-prediction for $(\mu_{\hat{i}}, \sigma_{\hat{i}})$, where $i \neq 1$, as such combination of prediction achieves optimal (minimal) compression ratio.

Figure 18 shows that *Bandwidth Reduction Ratio* (BRR) of the proposed Gaussian parameter compression algorithm for three video streams *Snow*, *Fog*, and *Winter*, which contain complex background scenes. BRR is defined as follows:

$$\mathrm{BRR} = 1 - \frac{BW_{\mathrm{compressed}}}{BW_{\mathrm{original}}}. \tag{19}$$

Figure 18 confirms that the proposed algorithm can reduce the overall memory bandwidth required for Gaussian parameters by over 50 %.

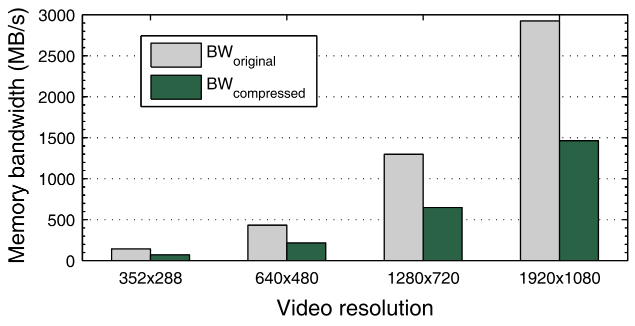


Fig. 17 Reduction of memory bandwidth by the proposed compression scheme

Table 2 Compression ratio for $(\omega_{\hat{i}}, \mu_{\hat{i}}, \sigma_{\hat{i}})$ with various video sequences

Video sequences	Prediction method					
	Intra-			Inter-		
	$\mu_{\hat{i}}$	$\sigma_{\hat{i}}$	$\omega_{\hat{i}}$	$\mu_{\hat{i}}$	$\sigma_{\hat{i}}$	$\omega_{\hat{i}}$
Snow	0.54	0.61	0.47	0.42	0.56	0.54
Fog	0.48	0.52	0.48	0.37	0.49	0.49
Winter	0.43	0.55	0.31	0.26	0.52	0.43
Campus	0.47	0.43	0.23	0.32	0.47	0.35
Survey	0.38	0.42	0.30	0.30	0.33	0.38
Hall	0.41	0.39	0.17	0.27	0.36	0.26
Average	0.45	0.49	**0.33**	**0.32**	**0.45**	0.41

Bold values indicate the optimal average compression ratio between intra- and inter-prediction methods for each Gaussian parameters

Table 3 compares FPGA resource utilization of the proposed compression implementation with three related compression architectures. Notice that the resource utilization of [32–34] is for either compression or decompression circuitry. Thus, adopting these methods for compressing and decompressing of a single Gaussian parameter would theoretically require twice the listed resources in Table 3. As MoG comprises $3 \times M$ distinct Gaussian parameters for each pixel, the methods [32–34] would require $2 \times 3 \times M$ of the listed resources in Table 3. In contrast, the resource breakdown of the proposed compression codec is for compressing and decompressing of all $3 \times M$ Gaussian parameters.

6.2 Object detection validation

In Figs. 19, 20, 21, 22, and 23 we observe the result of hardware simulations slightly deviates from the software counterpart, which is spontaneous due to the fixed-point hardware implementation.

We objectively quantify the results of the proposed FPGA implementation against the software model, which serves as the ground-truth data. We apply the following two commonly used measures of performance for comparing binary images [35]: (1) Product of Correctly Classified Proportions, PCP defined as:

$$PCP = \frac{TP + TN}{TP + FP + TN + FN} \tag{20}$$

where TP, TN, FP, and FN are the total number of true positives, true negatives, false positives, and false negatives, respectively; (2) Percentage of difference between hardware and software binarized frames, Δ_{hw}:

$$\Delta_{hw} = \frac{\sum |B_{hw}(n) - B_{sw}(n)|}{N_H \times N_V} \times 100\,\% \tag{21}$$

Fig. 18 Average bandwidth reduction ratio (defined in Eq. 19) with the proposed compression algorithm for various Gaussian parameters of complex video streams

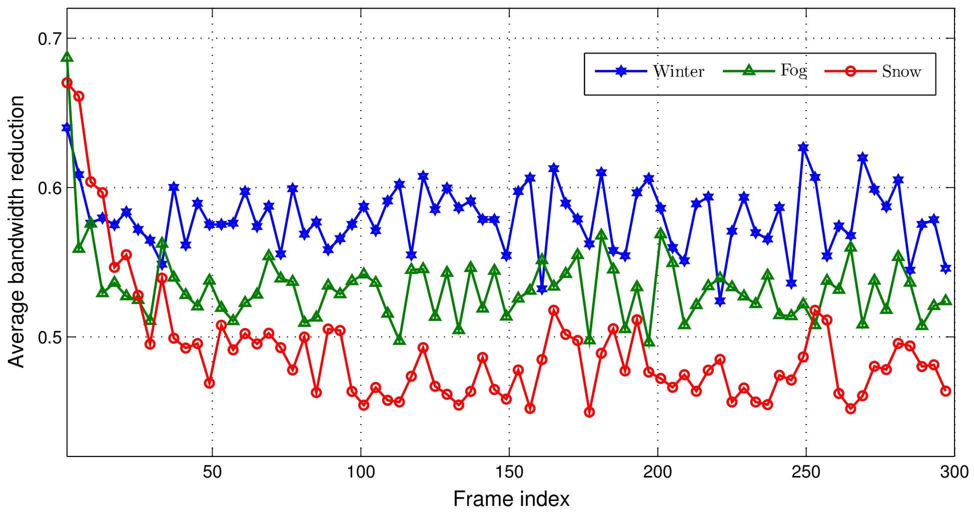


Table 3 Comparison to related compression methods

Method	Arithmetic [32]	JPEG [33]	Lossless [34]	Proposed
LUTs	823	5,880	13,784	2,154
FFs	339	1,930	6,848	1,785
BRAMs	7	2	NA	0
DSP48s	2	9	NA	0

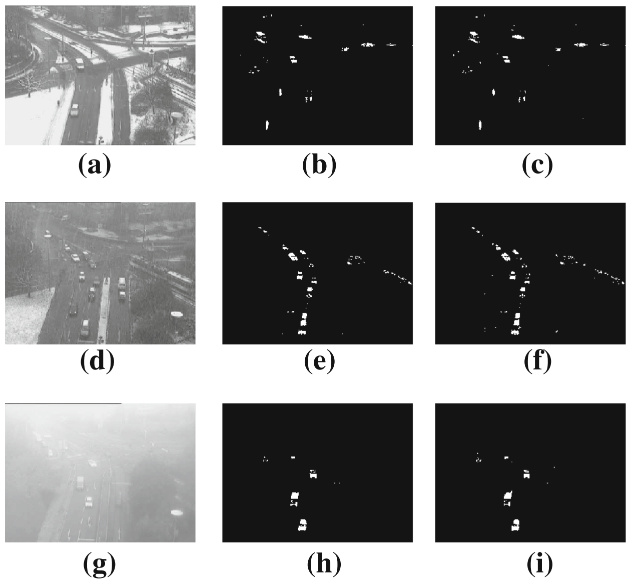


Fig. 19 Proposed FPGA implementation comparison to the software counterpart with *Winter* (*top row*), *Snow* (*middle row*), and *Fog* (*bottom row*) video sequences. *First column* is the original $I(n)$, and *second* and *third columns* are binary $B(n)$ frames with software and the proposed hardware implementation, respectively

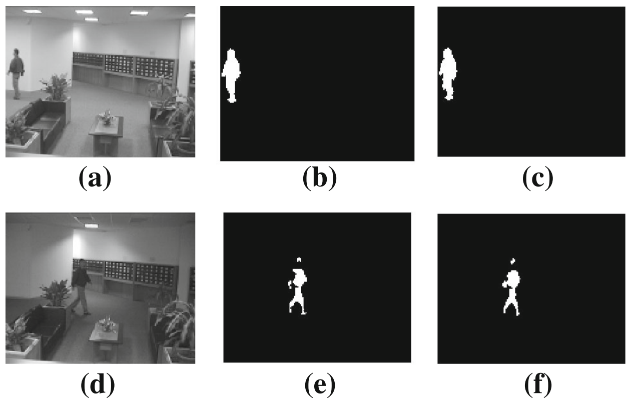


Fig. 20 Proposed FPGA implementation comparison to the software counterpart. *Top row* is frame 387 of the *Switching Light Off* video sequence and *bottom row* is frame 641 of the same video sequence after the light is switched off. *First column* is original $I(n)$, and *second* and *third columns* are binary $B(n)$ frames with software and the proposed hardware implementation, respectively

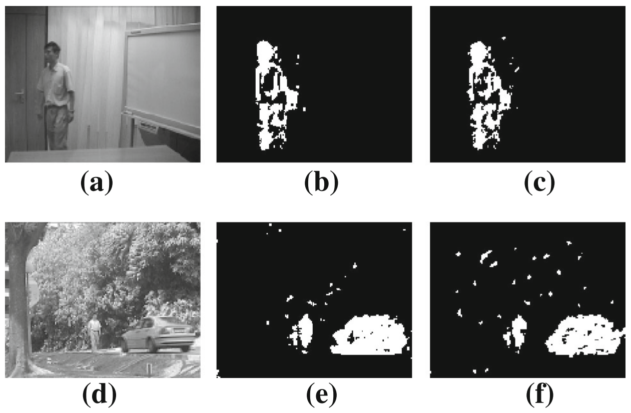


Fig. 21 Proposed FPGA implementation comparison to the software counterpart. *Top row* is frame 593 of the swaying *Curtain* video sequence and *bottom row* is frame 814 of the *Wavering Trees* video sequence. *First column* is original $I(n)$, and *second* and *third columns* are binary $B(n)$ frames with software and the proposed hardware implementation, respectively

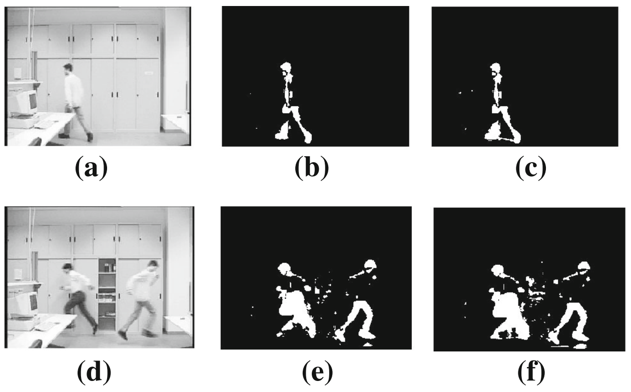


Fig. 22 Proposed FPGA implementation comparison to the software counterpart. *Top row* is frame 230 of the *Laboratory* video sequence and *bottom row* is frame 685 of the same video sequence after one of the drawers is left open. *First column* is original $I(n)$, and *second* and *third columns* are binary $B(n)$ frames with software and the proposed hardware implementation, respectively

where N_H and N_V are the number of horizontal and vertical pixels in a frame respectively, $B_{hw}(n)$ represents the binary frame of the hardware implementation, and $B_{sw}(n)$ is the software ground-truth binary frame. Notice that the higher a PCP measure is, the better match between the binary frames of hardware and software is. Conversely, a small value of Δ_{hw} indicates a good match. Figure 24 shows the mean value of PCP measure and Δ_{hw} obtained for each frame of *Campus*, *Snow*, *Fog*, and *Winter* video sequences.

Moreover, the integration of MoG background model is clearly supported by Fig. 25. The average mean of PCP measure for the tested videos has improved over the non adaptive background modeling implementation. The dips in PCP occur when hardware and software results differ by some pixels, which is due to the fixed-point implementation of the algorithm in hardware.

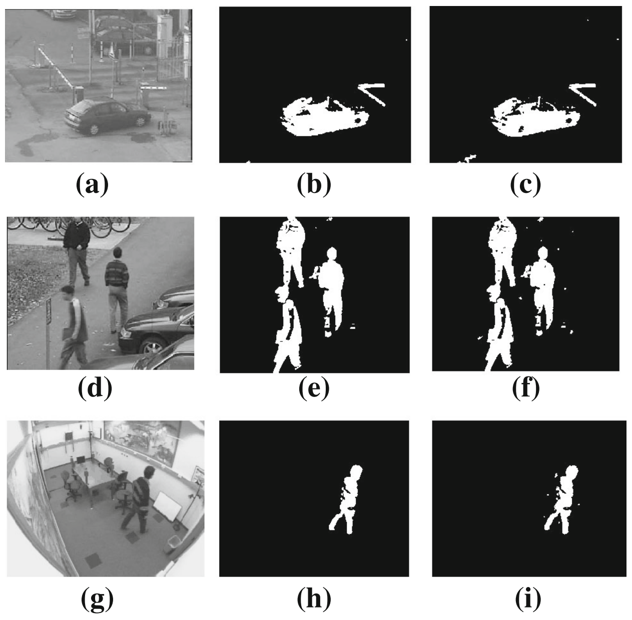


Fig. 23 FPGA simulation comparison with software counterpart. *Top row* is frame 23 of the *Campus* video sequence, *middle row* is frame 104 of the *Survey* video sequence, and *bottom row* is frame 147 of the *Intelligent Room*. *First column* is original $I(n)$, and *second* and *third columns* are binary $B(n)$ frames with software and the proposed hardware implementation, respectively

6.3 Synthesis and FPGA implementation results

We have used Aldec Active-HDL 8.2 for design capturing in VHDL (Very-high-speed integrated circuits Hardware Description Language) and simulation of the proposed architecture. The proposed architecture was implemented using Xilinx ISE Design Suite 11.4 on a Virtex-5 FPGA: XC5VSX35T with -2 speed grade and FF665 package. Power estimation was obtained using Xilinx XPower Estimator (XPE). A resource breakdown and power dissipation of various modules in the proposed architecture that processes HD-720p video streams at 30 fps are given in Table 4. All modules in Table 4 are clocked at 125 MHz, except the Memory Controller, which runs at 133 MHz. Total power consumption excludes device static power which amounts to 470 mW. It can be seen from Table 4 that the proposed compression algorithm utilize relatively low hardware resources. Moreover, the module reduces over 50 % memory bandwidth. Note that MPDR and memory controller contribute to a large part of the whole design due to parallel memory requests from various modules. For different video resolutions, resource utilization and power dissipation are listed in Table 5, which further illustrates that significant power savings are possible for high-resolution videos when integrated with the proposed compression core. These power savings are primarily due to the lower clock at the memory IO interface.

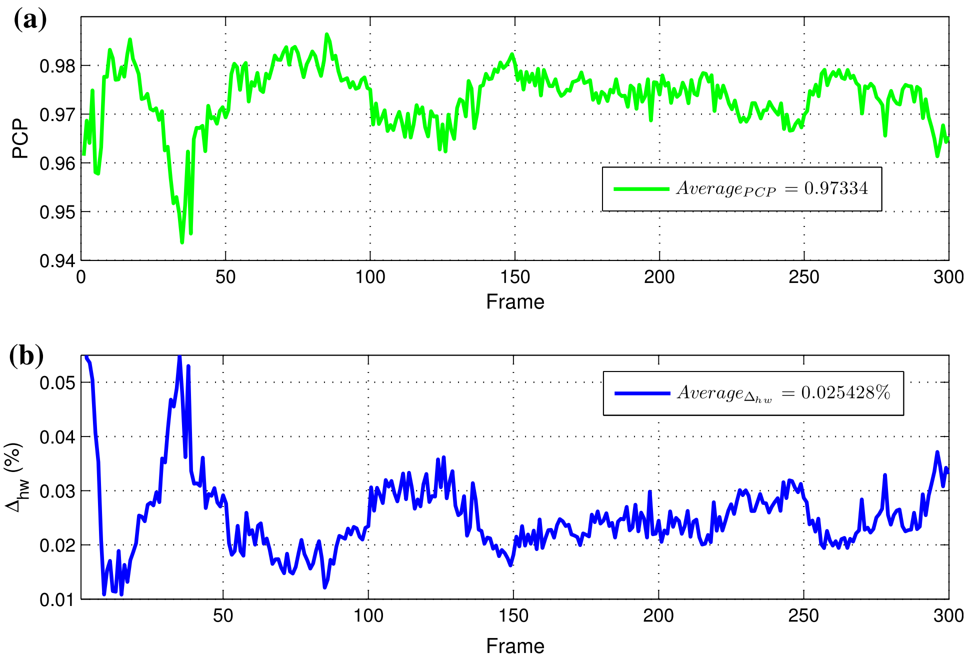


Fig. 24 Average objective measures for *Campus*,*Snow*, *Fog*, and *Winter* video sequences: **a** comparison between software and hardware implementations with averaged PCP objective measure, and **b** percentage of difference between hardware and software binarized frames Δ_{hw}

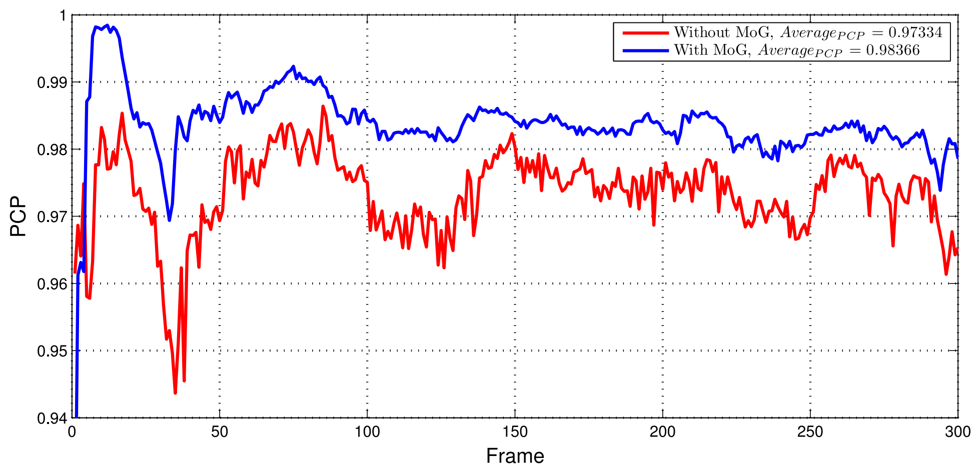


Fig. 25 Average PCP measure for object-change detection in *Campus*, *Snow*, *Fog*, and *Winter* video sequences, which is improved (closer to 1) with MoG background modeling

Table 4 Resource distribution for various modules of the proposed architecture and corresponding power dissipation for hd-720p resolution (excluding device static power) on Virtex-5 XC5VSX35T FPGA

Module	Resource				Power (mW)
	LUTs	FFs	BRAMs	DPS48s	
Compression	2,282 (27 %)	1,876 (21 %)	0 (0 %)	0 (0 %)	24 (4 %)
MoG	631 (7 %)	643 (7 %)	8 (18 %)	25 (57 %)	24 (4 %)
Change detection	1,048 (12 %)	879 (10 %)	12 (27 %)	12 (27 %)	18 (3 %)
Noise estimation	984 (12 %)	856 (9 %)	9 (20 %)	6 (14 %)	17 (3 %)
Memory controller	1,562 (19 %)	2,241 (24 %)	3 (7 %)	0 (0 %)	483 (81 %)
MPDR	1,794 (21 %)	2,438 (27 %)	10 (23 %)	1 (2 %)	24 (4 %)
Video interface	151 (2 %)	148 (2 %)	2 (5 %)	0 (0 %)	6 (1 %)
Total	8,452 (100 %)	9,081 (100 %)	44 (100 %)	44 (100 %)	596 (100 %)

Table 5 Clock frequency and power dissipation of the proposed architecture on Virtex-5 XC5VSX35T FPGA

	358 × 288			640 × 480		1,280 × 720		1,920 × 1,080
	Original	Compressed	Original	Compressed	Original	Compressed	Original	Compressed
Resource								
LUTs	5,127	7,364	5,514	7,896	5,914	8,452	7,294	9,688
FFs	6,413	8,381	6,652	8,584	7,156	9,081	8,079	9,973
BRAMs	32	32	38	38	44	44	73	73
DSP48s	47	47	47	47	47	47	47	47
DDR Clock (MHz)	125	125	125	125	266	133	599	299
Clock toggle rate (%)	25	12.5	36	18	50	50	50	50
Quotient power (mW)	471	482	476	490	485	497	501	512
Core dynamic power (mW)	109	114	123	130	287	231	614	443
IO power (mW)	172	86	247	124	730	365	1,648	821
Total power (mW)	752	682	846	744	1,502	1,093	2,763	1,776

6.4 Comparison to existing work

For CIF video resolution, the software implementation of the proposed work requires 71 ms per frame, on an average, when implemented in C++ on PC, with Intel Core i7 processor (six cores @3.2 GHz each) and 12 GB of memory. In contrast, our proposed FPGA implementation executes in less than 1 ms, accomplishing approximately a speed-up factor of 70-folds over the software implementation.

Table 6 compares the proposed implementation with the related arts [23–28] in performance and resource utilization. The architectures presented in [23–25] are based on conventional object-change detection methods using pure

Table 6 Comparison of related work on processing throughput and resource utilization

Method Resolution	[23] 640 × 480	[24] 320 × 240	[25] 320 × 240	[26] 640 × 480	[27] 640 × 480	[28] HD-1080p	Proposed HD-1080p
fps	210	42	30	25	25	24	30
LUTs	3347	14000	3056	10682	12214	1179	9688
FFs	1766	14000	1766	NA	4273	492	9973
BRAMs	57	NA	NA	NA	84	NA	73
MULTs	NA	NA	6	NA	NA	10	47
FPGA	XC2V6000	APEX20KE	VP30	VP30	VP30	LX50	SX35T

background subtraction and lack these features; therefore, these systems may fail to accurately detect video object in noisy environments [26–28] may fail to estimate slow background updates as MoG parameters are lossy and assigned with smaller bit-widths creating rounding errors in fixed-point arithmetic computations. Notice that the resource utilization listed under [28] are only for MoG and denoising modules, whereas the proposed implementation contains the complete system including the vital peripheral components, such as memory and video interfaces. The proposed implementation also features important characteristics including adaptation to inevitable video noise and a more recent MoG algorithm. When comparing the resource utilization in Table 6, it must be observed that the LUTs in the FPGAs of [28] and the proposed implementations are 6-input/1-output or 5-input/dual-output, while the LUTs in the FPGAs of the rest of the related methods are fundamentally 4-input/1-output.

7 Concluding remarks

This paper presented a resource- and power-optimized hardware architecture for moving object detection that combines noise estimation, Mixture-of-Gaussian background modeling, motion detection, and spatio-temporal thresholding. We discovered a major implementation challenge with background modeling. When targeting a hardware implementation based on fixed-point arithmetic, high dynamic range or precision of Gaussian parameters is critical to accurately model the background. The need for larger dynamic range of fixed-point numbers for each parameter Gaussian distributions requires a high memory bandwidth and a large memory capacity. The resultant frequent access to external memory yields excessive power dissipation due to memory I/O port switching. We proposed a novel lossless Gaussian parameter compression technique suitable for implementation on resource- and power-constraint embedded video processing system. The proposed architecture was targeted to resource-limited embedded platform, and thus the FPGA implementation was optimized at algorithmic, architectural, and logic levels to minimize power consumption and FPGA resource utilization. Advanced design techniques, such as pipelining and data parallelism were employed to achieve a processing throughput of 30 fps for HD-1080p video resolution. We showed that the proposed implementation significantly outperformed the existing hardware-based methods in resource utilization reduction and processing speed.

In future work, we plan to integrate anisotropic diffusion either as a pre-processing step to reduce noise or as a low-pass filter applied to the difference frame of the motion detection. Other future work includes integrating the proposed design with FPGA-based object tracking for event detection aiming at a complete surveillance video processing system onto a stand-alone smart camera.

Acknowledgments This work was supported, in part, by the Fonds de la recherche sur la nature et les technologies du Quebec (NATEQ).

References

1. Oliver, N., Rosario, B., Pentland, A.: A Bayesian computer vision system for modeling human interactions. IEEE Trans. Pattern Anal. Mach. Intell. **22**(8), 831–843 (2000)
2. Li, L., Huang, W., Gu, I.Y.H., Tian, Q.: Statistical modeling of complex backgrounds for foreground object detection. IEEE Trans Image Process. **13**(11), 1459–1472 (2004)
3. Cheung, S.C.S., Kamath, C.: Robust techniques for background subtraction in urban traffic video. Proc. SPIE **5308**, 881–892 (2004)
4. Happe, M., Lübbers, E., Platzner, M.: A self-adaptive heterogeneous multi-core architecture for embedded real-time video object tracking. J. Real Time Image Process. (2011) (Published online)
5. Chakraborty, D., Shankar, B.U., Pal, S.K.: Granulation, Rough Entropy and Spatiotemporal Moving Object Detection. Applied Soft Computing (2012)
6. Gao, H., Peng, Y., Dai, Z., Xie, F.: A new detection algorithm of moving objects based on human morphology. In: IEEE International Conference on Intelligent Information Hiding and Multimedia, Signal Processing, pp. 411–414 (2012)
7. Wang, Y.T., Chen, K.W., Chiou, M.J.: Moving object detection using monocular vision. Intell. Auton. Syst., pp. 183–192 (2013)
8. Stauffer, C., Grimson, W.E.: Learning patterns of activity using real-time tracking. IEEE Trans. Pattern Anal. Mach. Intell. **22**(8), 747–757 (2000)

9. Piccardi, M.: Background subtraction techniques: a review. IEEE Int. Conf. Syst. Man Cybern. **4**, 3099–3104 (2004)
10. Cope, B., Cheung, P.Y.K., Luk, W., Howes, L.: Performance comparison of graphics processors to reconfigurable logic: a case study. IEEE Trans. Comput. **59**(4), 433–448 (2010)
11. Papakonstantinou, A., Gururaj, K., Stratton, J.A., Chen, D., Cong, J., Hwu, W.M.W.: FCUDA: Enabling efficient compilation of CUDA kernels onto FPGAs. In: IEEE Symposium on Application Specific Processors, pp. 35–42 (2009)
12. Amer, A., Dubois, E.: Fast and reliable structure-oriented video noise estimation. IEEE Trans. Circuits Syst. Video Technol. **15**, 113–118 (2005)
13. Achkar, F., Amer, A.: Hysteresis-based selective Gaussian mixture models for real-time background maintenance. IS T/SPIE Symp. Electron. Imaging **6508**(2), 65082J.1–65082J.11 (2007)
14. Amer, A.: Memory-based spatio-temporal real-time object segmentation. In: SPIE International Symposium on Electronic Imaging, Conference on Real-Time Imaging, vol. 5012, pp. 10–21 (2003)
15. Ratnayake, K., Amer, A.: An FPGA-based implementation of spatio-temporal object segmentation. In: IEEE International Conference on Image Processing, pp. 3265–3268 (2006)
16. Zhang, D., Lu, G.: Segmentation of moving objects in image sequence: a review. Circuits Syst. Signal Process. **20**(2), 143–183 (2001)
17. Karman, K.P., von Brandt, A.: Moving object recognition using an adaptive background memory. Time Varying Image Process. Movi. Object Recogn., pp. 297–307 (1990)
18. Toyama, K., Krumm, J., Brumitt, B., Meyers, B.: Wallflower: Principles and practice of background maintenance. IEEE Int. Conf. Comput. Vis. **1**, 255–261 (1999)
19. Zivkovic, Z.: Improved adaptive Gaussian mixture model for background subtraction. In: IEEE International Conference on the, Pattern Recognition, pp. 28–31 (2004)
20. Lee, D.S.: Effective Gaussian mixture learning for video background subtraction. IEEE Trans. Pattern Anal. Mach. Intell. **27**(5), 827–832 (2005)
21. Cucchiara, R., Grana, C., Piccardi, M., Prati, A.: Detecting moving objects, ghosts, and shadows in video streams. IEEE Trans. Pattern Anal. Mach. Intell. **25**(10), 1337–1342 (2003)
22. Elgammal, A.M., Duraiswami, R., Davis, L.S.: Efficient kernel density estimation using the fast gauss transform with applications to color modeling and tracking. IEEE Trans. Pattern Anal. Mach. Intell. **25**(11), 1499–1504 (2003)
23. Appiah, K., Hunter, A.: A Single-Chip FPGA implementation of real-time adaptive background model. In: EEE International Conference on Field-Programmable Technolog, pp. 95–102 (2005)
24. Oliveira, J., Printes, A., Freire, R.C.S., Melcher, E., Silva, I.S.S.: FPGA architecture for static background subtraction in real time. In: Annual Symposium on Integrated Circuits and Systems Design, pp. 26–31 (2006)
25. Schlessman, J., Lodato, M., Ozer, B., Wolf, W.: Heterogeneous MPSoC architectures for embedded computer vision. In IEEE International Conference on Multimedia and Expo, pp. 1870–1873 (2007)
26. Kristensen, F., Hedberg, H., Jiang, H., Nilsson, P., Öwall, V.: An embedded real-time surveillance system: implementation and evaluation. J. Signal Process. Syst. **52**(1), 75–94 (2008)
27. Jiang, H., Ardo, H., Öwall, V.: A hardware architecture for real-time video segmentation utilizing memory reduction techniques. IEEE Trans. Circuits Syst. Video Technol. **19**(2), 226–236 (2009)
28. Genovese, M., Napoli, E.: FPGA-based architecture for real time segmentation and denoising of HD video. J. Real Time Image Process. (2011) (Published online)
29. Teuhola, J.: A compression method for clustered bit-vectors. Inf. Process. Lett. **7**, 308–311 (1978)
30. Ratnayake, K., Amer, A.: An FPGA architecture of stable-sorting on a large data volume: application to video signals. In: IEEE Conference on Information Sciences and Systems, pp. 431–436 (2007)
31. Wang, T.C., Fang, H.C., Chao, W.M., Chen, H.H., Chen, L.G.: An UVLC encoder architecture for H. 26L. In: IEEE International Symposium on Circuits and Systems, vol. 2, pp. 308–311 (2002)
32. Osman, H., Mahjoup, W., Nabih, A., Aly, G.M.: JPEG encoder for low-cost FPGAs. In: International Conference on Computer Engineering Systems, pp. 406–411 (2007)
33. Yu, G., Vladimirova, T., Wu, X., Sweeting, M.N.: A new high-level reconfigurable lossless image compression system for space applications. In: NASA/ESA Conference on Adaptive Hardware and Systems, pp. 183–190 (2008)
34. Mahapatra, S., Singh, K.: An FPGA-based implementation of multi-alphabet arithmetic coding. IEEE Trans. Circuits Syst. I Regul. Papers **54**(8), 1678–1686 (2007)
35. Rosin, P.L.: Thresholding for change detection. Comput. Vis. Image Underst. **86**, 79–95 (2002)

Kumara Ratnayake received the B.Eng. degree (First Class Honors) in electronic engineering from the University of Central Lancashire, Preston, United Kingdom in 1998 and the M.A.Sc. degree in electrical and computer engineering from Concordia University, Montréal, QC, Canada, in 2007, where he is currently pursuing his Ph.D. degree in the Electrical and Computer Engineering Department. Kumara has been with Teledyne DALSA Inc., Montréal, Canada, as a senior FPGA design architect since 1999. He has several publications and his research interests include FPGA-based reconfigurable architectures for video processing, video object segmentation, video analysis, tracking, and noise reduction.

Aishy Amer received the Diploma degree in computer engineering from Dortmund University, Dortmund, Germany, in 1994 and the Ph.D. degree in telecommunications from the INRS, Université du Québec, Montréal, QC, Canada, in 2001. Currently, she is an Associate Professor at the Department of Electrical and Computer Engineering, Concordia University, Montréal, QC, Canada. From 1995 to 1997, she was at Siemens- AG/Munich and Dortmund University as a Research and Development Associate. She is particularly interested in video object segmentation and its integration into end-to-end video systems. Her research interests include Audio–Video Integration, Video Surveillance, Advanced TV-Systems, Video Analysis (Object Segmentation, Object Tracking, and Object Motion Estimation), Video Quality Enhancement (Noise Reduction and Estimation), Video Interpretation (Event and Semantic Detection). She has four patents and over 60 publications. She is serving as an Associate Editor for the Springer Journal of Real-Time Image Processing (JRTIP). She served as an Associate Editor for the Elsevier Journal for Real-Time Imaging (RTI) and as a Guest Editor for the RTI Special Issue on Video Object Processing for Surveillance Applications.