TESI DI DOTTORATO

UNIVERSITÀ DEGLI STUDI DI NAPOLI "FEDERICO II"

DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELLE TECNOLOGIE DELL' INFORMAZIONE

DOTTORATO DI RICERCA IN
INGEGNERIA ELETTRONICA E DELLE TELECOMUNICAZIONI

# HARDWARE ARCHITECTURES FOR REAL TIME PROCESSING OF HIGH DEFINITION VIDEO SEQUENCES

**MARIANGELA GENOVESE**

| Il Coordinatore del Corso di Dottorato | Il Tutore |
|---|---|
| Ch.mo Prof. Niccoló RINALDI | Ch.mo Prof. Ettore NAPOLI |

A. A. 2013–2014

# Acknowledgments

I want to express my sincere gratitude to my tutor, Professor Ettore Napoli. His experience, knowledge, and scientific methodology have been a constant example for me. I cannot imagine the completion of my thesis work without his help.

I would also like to thank Professor Antonio Stollo. His suggestions and invaluable experience have been a precious benchmark during my research activity.

I am very grateful to Davide and Nicola for their teachings and their professional and moral support. They are excellent teachers and very good friends.

Moreover, I would like to extend my gratitude to the colleagues and friends at the department for their friendship and the nice and priceless time spent together.

I am also very grateful to my special friend Salvatore for reading part of the present thesis and, above all, for giving me the joy and the serenity needed to complete my work.

Last but not least, I sincerely thank my dear family for being always on my side.

Mariangela

# Contents

# List of Figures

# List of Tables

# Introduction

Video-related applications have increased dramatically in recent years. Application fields, such as medicine, space exploration, surveillance, authentication, HDTV, and automated industry inspection, require capturing, storing and processing continuous streams of video data. Different process techniques (video enhancement, segmentation, object detection, or video compression, as examples) are involved in these applications.

The implementation of such techniques on a general purpose computer is generally simple but it is not time-efficient due to the significant number of operations required by the processing algorithms that depends on the algorithm complexity and the video resolution. The actual trend, driven by the consumer electronics market, is towards lightweight and high performance portable systems capable of processing high definition (HD) video sequences in real-time. Such demand is mainly targeted through the extensive use of integrated digital electronic systems.

Very high performances can be obtained by using full custom ASIC implementations. However, the complexity and the cost associated with the ASIC design is very high. Moreover, ASIC implementations are not reconfigurable and require a long design time. For these reasons, Field Programmable Gate Array (FPGA) devices are always more frequently being chosen as target technology for the hardware acceleration.

FPGA are reprogrammable, and are available in a wide range of performance. They allow to implement low cost dedicate architectures and provide fast time to market. Finally, rapid prototyping of complex algorithms, along with debugging and verification simplifications, are easily achievable with FPGA devices.

In this dissertation several hardware architectures for real-time video processing of high definition video sequences, that overcome the state of the art in terms of performances, are proposed.

The implemented architectures are listed in the following.

Background identification circuit

A background identification system identifies moving objects in video sequences. Several background identification algorithms have been proposed in the years. Among all, the Gaussian Mixture Model is able to provide good performance in presence of illumination changes and multimodal background. For these reasons, the Gaussian Mixture Model has been introduced in the OpenCV, an open source library for computer vision. The OpenCV version of the Gaussian Mixture Model allows a fast identification also in the first phase of the video processing.

Main drawback of the algorithm is the high computational complexity so that a hardware implementation is required for real-time processing of HD video sequences.

A new formulation of the algorithm equations, along with the implementation of ROM approximation techniques and truncated multipliers, leaded to an optimized hardware implementation able to process 1080p video sequences in real time.

The circuit has been implemented both on Field Programmable Gate Array (FPGA) devices and by using a standard cells library in UMC-90nm technology and has been experimentally validated by implementing two running on-line video systems.

When implemented on Virtex6 (xc6vlx195t) FPGA the circuit is able to process 57 HD fps (fps, frames per second) by using the 1% of the Slice of the target device.

Finally, two ASIC implementations have been proposed. The circuits have been optimized in order to allow the processing of 60 fps with reduced silicon area utilization or with low power dissipation.

The background identification algorithms and the results of the research activity are presented in the Chapters 1 and 2.

Denoising circuit for binary images

The output video of a background identification circuit is composed by binary images in which each pixel is equal to '0' if it belongs to the background or '1' otherwise (or vice versa). Some pixels could be wrongly classified and some holes could appear in the objects or close objects could be connected.

In order to enhance the binary images, a denoising circuit, based on morphological operators, can be implemented.

The mathematical morphology consists of a set of powerful tools for geometrical image analysis, image, and video compression, error correction, noise suppression, and video segmentation.

The circuit has been implemented for processing HD video sequences with reduced area utilization and memory requirements, and without introducing latency in the processing of consecutive frames.

Implemented on Virtex6 (xc6vlx195t) FPGA, the circuit runs at 405.68 MHz (the processing capability is equal to 195 HD fps) by using 133/31200 Slices. In association with the background identification circuit previously described, the circuit is able to process 57 HD fps by using 434 of the 31200 Slices of the target FPGA.

The results are shown in Chapter 3.

Spatio-temporal average filter for denoising of fluoroscopic images

Fluoroscopy is a technique massively adopted in clinical environments for image-guided surgery and therapy.

During the fluoroscopy, the dosage of X-ray is low. This generates the well-known quantum noise, which can be modelled as a signal-dependent Poisson distributed noise source.

Several techniques have been proposed to filter such quantum noise. In recent years, a spatio-temporal conditioned average filter has been proposed in the scientific literature. This filter is capable of filtering images while still preserving edges and moving objects and favourably compares with more complex filtering techniques such as BDM, BM3Dc, K-SVD. Furthermore, the proposed filter is optimally suited for a hardware implementation. The algorithms for the denoising of fluoroscopic images are described in Chapter 4.

The research activity has been oriented to the hardware implementation of the proposed spatio-temporal conditioned average filter and results, to the best of our knowledge, in the first circuit able to process fluoroscopic images in real time. The circuit has been designed aiming to the reduction of the arithmetic circuital units requested to average the pixels. Furthermore, the filtering requires the adoption of an external memory and, thus, several addressing schemes, each of them involving different trade-off between hardware complexity and maximum working frequency, have been considered and the addressing scheme that optimizes this trade-off has been implemented.

The circuit, implemented on Spartan6 FPGA, is able to process 58 fps with resolution equal to $1024 \times 1024$.

Chapter 5 shows the results of the hardware implementation.

It is worth noting that, although the implemented algorithms find application in different contexts, all the described architectures are able to process HD video sequences in real-time.

The circuits are designed by using Hardware Description Languages (HDL) and the target technologies for the implementation are mainly Xilinx and Altera FPGA devices. Area utilization, maximum working frequency, and power dissipation have been also computed and analyzed for all the described architectures and several experiments have been carried out to test the circuits behavior.

The comparison with previously proposed works shows that the circuits performance overcome the state-of-the-art architectures, highlighting the effectiveness of the proposed solutions.

## Dissertation outline

The dissertation is organized as follows:

- **Chapter 1** summarizes the background identification algorithms proposed to date;
- **Chapter 2** describes the hardware implementation of the OpenCV version of the Gaussian Mixture Model algorithm;
- **Chapter 3** deals with the denoising circuit for binary images;
- **Chapter 4** is focused on the state-of-the-art techniques used for filtering fluoroscopic images;
- **Chapter 5** describes the hardware implementation of the spatio-temporal average filter to perform the real-time processing of fluoroscopic video sequences.

## Publications

Background identification circuit

- M. Genovese, E. Napoli, N. Petra, "Hardware Performance Versus Video Quality Trade-Off for Gaussian Mixture Model Based Background Identification Systems," ICDIP., Athens, Greece, 2014 (accepted for publication);
- M. Genovese, E. Napoli, "ASIC and FPGA Implementation of the Gaussian Mixture Model Algorithm for Real-Time Segmentation of High Definition video," IEEE Trans. on Very Large Scale Integration (VLSI) Systems, vol.PP, no.99, 2013;

- M. Genovese, E. Napoli, D. De Caro, N. Petra, and A. G. M. Strollo, FPGA Implementation of Gaussian Mixture Model Algorithm for 47 fps Segmentation of 1080p Video, Journal of Electrical and Computer Engineering, vol. 2013, 2013;
- M. Genovese, E. Napoli, Processor core for real time background identification of HD video based on OpenCV Gaussian mixture model algorithm, in Proc. SPIE 8764, VLSI Circuits and Systems VI, Grenoble, France, May 2013;
- M. Genovese and E. Napoli, FPGA implementation of OpenCV compatible background identification circuit, in Proc. of the 3rd International Symposium on Computational Modeling of Objects Represented in Images: Fundamentals, Methods and Applications (CompIMAGE 12), pp. 7580, Rome, Italy, Sept.2012;
- M. Genovese, E. Napoli, "An FPGA-based Real-time Background Identification Circuit for 1080p Video," Signal Image Technology and Internet Based Systems (SITIS), 2012 Eighth International Conference on , pp.330,335, 25-29 Nov. 2012;
- M. Genovese, E. Napoli, N. Petra, "OpenCV compatible real time processor for background foreground identification," Microelectronics (ICM), 2010 International Conference on , pp. 467-470, Cairo, Egypt, Dec. 2010.

Denoising circuit for binary images

- M. Genovese, N. Petra, D. De Caro, E. Napoli, A. G. M. Strollo, "FPGA architecture for real time video segmentation and denoising", GE, Marina di Carrara, 2012;
- M. Genovese, and E. Napoli, "FPGA-based architecture for real time segmentation and denoising of HD video," Jour. of Real-Time Image Processing, pp.1-13, 2011.

Spatio-temporal average filter for denoising of fluoroscopic images

- M. Genovese, D. Decaro, E. Napoli, N. Petra, P. Bifulco, M. Romano, M. Cesarelli, A. G. M. Strollo, "Hardware implementation of a spatio-temporal average filter for real-time denoising of fluoroscopic images", VLSI, the Integration Journal. (submitted for publication )

# Chapter 1

# Motion detection

Several computer vision applications including video surveillance, tracking, gesture recognition, etc., require to identify events of interest in video sequences. These events correspond often to the motion of objects (foreground) that have to be separated from the static parts of the scene (background).

When a large amount of data has to be processed the foreground can not be identified manually but a system able to automatically segment the scene is needed.

Several algorithms have been developed during the years. They are based on the frame difference, [1]-[4], on the comparison with a background model [5]-[17], or on the estimation of the optical flow [18]-[22].

The algorithms based on the difference between consecutive frames, are often fast and simple to implement but the result of the identification depends on the speed of the moving objects. Moreover, they are not robust with respect to illumination changes of the environment. This last problem is also present in optical flow based algorithms that, on the other hand, allow the identification of moving objects with different speed and provide information on the direction of the motion.

The algorithms based on the background model are often able to provide good performances in presence of illumination changes. However, they present an high computational complexity since the update of the background model is required. The Gaussian Mixture Model (GMM) algorithm, [16],[17], belongs to this last category. It provides good performances in presence of both illumination changes and multimodal background (characterized by objects showing repetitive motions, e.g. a lake surface or flickering lights).

Due to its good performances the GMM has been selected as background

detection algorithm in the OpenCV (Open Source Computer Vision) library, [23], developed by Intel to provide a common base of computer vision instruments able to extract relevant details from the images and to process them in automatic way. The OpenCV library is becoming a widely used standard. The importance of the OpenCV library is demonstrated by the fact that many companies working on electronic systems for computer vision, are now studying or implementing the OpenCV algorithms in order to provide systems that can be easily transferred to new applications.

The OpenCV version of the GMM modifies the original algorithm proposed in [16],[17] by allowing a faster initialization phase of the background model.

This Chapter describes the main background/foreground identification methods (Section1.1) with the focus on the background subtraction algorithms and details the OpenCV version of the Gaussian Mixture Model (Section1.2).

## 1.1 Background identification algorithms

### 1.1.1 Frame difference

A simple way to detect the foreground is observing the difference of the pixels belonging to two or more consecutive frames [1]-[4]. By setting a threshold value, a pixel is identified as foreground if the difference is higher than the threshold value or background otherwise.

These algorithms are often very simple and suit well for the hardware implementation because both the computational complexity and the memory requirements are rather low. However, their simplicity comes at the cost of the segmentation quality. In general, bigger regions are detected as foreground area than the actual moving part. The threshold is generally a fixed value. Setting a global threshold value is problematic since the segmentation is sensitive to light intensity. When the threshold value is big, there is less noise in the result but some objects could be not well identified. On the contrary, if the threshold value is low, the noise increases. In addition, the identification depends on the speed of the moving objects and the illumination changes of the environment.

### 1.1.2 Optical flow

Optical flow is a commonly used method to identify and track the moving objects by estimating velocity distributions in a video sequence.

Many gradient-based methods such as the HornSchunck method [18] and LucasKanade method [19] have been developed to estimate the optical flow by calculating the brightness gradients of images [20], [21]. The accuracy of these methods decreases when the speed of the moving objects is high or when there is a large displacement between consecutive frames, [22].

Moreover, the optical flow method is based on the assumption of constant lightness across frames. That is true, if the illumination condition does not have drastic change or the frame rate is high.

### 1.1.3 Background subtraction

An accurate identification of the moving objects can be obtained by creating a reference model of the background and by comparing the frames of the video sequence with this model, [5]-[17].

The scene could be subjected to changes due to lighting variations or static objects that become dynamics or vice versa. The background model has to be updated to correctly model the changes of the scene. This entails that the background subtraction algorithms are often computationally expensive.

Figure 1.1 shows an conceptual overview of a background subtraction system.

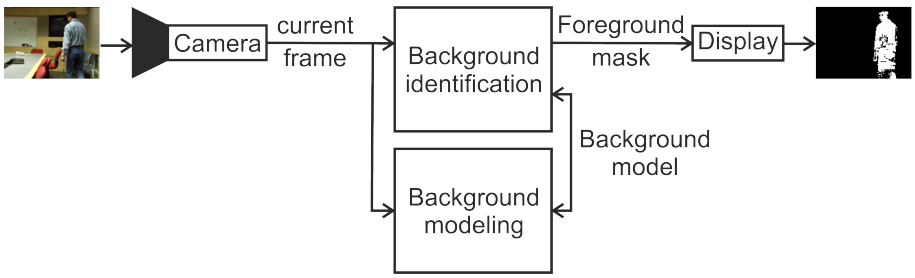


**Figure 1.1:** Conceptual overview of a background identification system.

The ”Background identification” unit is responsible for segmenting the current frame of the video sequence provided by a camera, into foreground and background regions. The identification is carried out by exploiting the model of the scene produced by the ”Background modeling” unit. The output is usually a video sequence composed by binary images in which the foreground pixels are white and the pixels belonging to the background are black (or viceversa).

The literature reports various contributions for the segmentation of the foreground pixels with background subtraction algorithms, [5]-[17].

Reference [7] builds a background model starting with the assumption that, if a pixel is stationary for a prefixed number of consecutive frames, the probability that it belongs to the *Bg* is high and, therefore, the pixel can be included in the background model.

Median filters, traditionally used in the spatial image filtering process, are also used to model background pixels with reduced noise deviation by filtering pixel values in the time domain, [8]-[10]. The median filtering process is carried out over a large number of previous frames [8]. This number grows if the objects move slowly. The frames needed for the filtering operation have to be stored while resulting in a high computational complexity and a large memory usage.

The median filter doesn't take into account how a background pixel value could drift from its mean value. To be able to estimate the current background more accurately, linear predictive filters are developed for background modeling, [11]-[14].

The authors of [11] use an one-step Wiener filter to predict a background value starting from its N previous values. Wiener filters are expensive in computation and memory requirement. N frame buffers are needed to store the N previous frames. Background pixel prediction and coefficients updating are also very costly since a set of linear functions are needed to obtain the value.

An alternative approach for linear prediction is to use a Kalman filter. Basic Kalman filter theory can be found in many literatures, [12]- [14]. Kalman filters are widely used for many background subtraction applications. They predict the current background pixel value with a recursive compute, starting from the previous estimate and the new input data.

Reference [15] proposes a model in which the fluctuations in a pixel value are modelled with a Gaussian distribution. This allows a pixel by pixel representation of the background with larger variance for pixels that experience wide change of lighting.

Predictive methods discussed before work well with background scenes with slow lighting changes but fails to deal with multi-modal background in which the pixel intensity oscillates between two o more values. In order to correctly model multi-modal backgrounds Stauffer-Grimson [16],[17] developed an algorithm based on a mixture of Gassian distribution. The algorithm of [16],[17], known as Gaussian Mixture Model, is detailed in Section 1.2.

## 1.2 Gaussian Mixture Model

The Gaussian Mixture Model (GMM) algorithm has been proposed by Stauffer and Grimson, [16],[17] with the target of efficiently dealing with lighting changes and multimodal background. A multimodal background is characterized by objects showing repetitive motion, e.g. waves, moving leaves or flickering lights. When a pixel lies in a region where a repetitive motion occurs, its brightness oscillates between two or more values. This results in false foreground detection in most algorithms.

The GMM algorithm models the intensity distribution of each pixel in a frame by using a statistical model composed by a mixture of K Gaussian distributions. Each Gaussian distribution is represented with three parameters: the mean $\mu_{p,k,t}$, the variance $\sigma^2_{p,k,t}$, and a weight $w_{p,k,t}$. The weight represents the probability that the current observation of the pixel belonging to the distribution.

Gaussian parameters differ for each Gaussian of each pixel and change for every frame of the video sequence. They are therefore defined by three indexes (p,k,t), where 'p' is the index for the pixel, 'k' is the index for the Gaussian distribution, and 't' is for the frame. In the following the pixel index is omitted since the same operations are repeated for every pixel.

Due to the good performances, the GMM algorithm has been selected as the background detection algorithm in the OpenCV [23], Open source Computer Vision software library originally developed by Intel, that provides a large number of programming functions mainly aimed at real-time computer vision. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. It has more than 47 thousand people of user community and estimated number of downloads exceeding 7 million.

The GMM algorithm proposed in the OpenCV library is an optimized version of the algorithm of [16],[17].

A detailed description of the OpenCV GMM algorithm is given in the following. The differences with respect to the algorithm of [16],[17] are indicated in the text.

### 1.2.1 OpenCV algorithm

**Parameter update**

When a frame is acquired, for each pixel, the K Gaussian distributions are sorted in decreasing order of a parameter named Fitness ($F_{k,t}$):

$$F_{k,t} = w_{k,t}/\sigma_{k,t}. \tag{1.1}$$

The incoming pixel (pixel) is checked against the K Gaussian distributions in order to verify a match condition defined as:

$$M_{k,t} = 1 \qquad if \mid (pixel - \mu_{k,t}) \mid < \lambda * \sigma_{k,t} \tag{1.2}$$

where $\lambda$ is a threshold whose value is chosen equal to $2.5$ in [16]. This threshold can be changed to adapt the algorithm to different kinds of scene. For example, shaded regions do not generally exhibit as much noise as objects in lighted regions. So different threshold values could be chosen. Equation (1.2) establishes if the pixel can be considered part of the background. A pixel can verify (1.2) for more than one Gaussian. The Gaussian that matches with the pixel ($M_{k,t} = 1$) and has the highest $F_{k,t}$ value is considered as the matched distribution and its parameters are updated as follows:

$$\begin{aligned} \mu_{k,t+1} &= \mu_{k,t} + \alpha_{k,t} \cdot (pixel - \mu_{k,t}), \\ \sigma^2_{k,t+1} &= \sigma^2_{k,t} + \alpha_{k,t} \cdot [(pixel - \mu_{k,t})^2 - \sigma^2_{k,t}], \\ w_{k,t+1} &= w_{k,t} - \alpha_w \cdot w_{k,t} + \alpha_w, \\ matchsum_{k,t+1} &= matchsum_{k,t} + 1 \end{aligned} \tag{1.3}$$

For the unmatched Gaussian distributions, mean and variance are unchanged and the weights are updated as:

$$w_{k,t+1} = w_{k,t} - \alpha_w \cdot w_{k,t}. \tag{1.4}$$

The matchsum parameter ($matchsum_{k,t}$) of (1.3) is a counter introduced in the OpenCV algorithm. The parameter $\alpha_w$ is the learning rate for the weight while $\alpha_{k,t}$ is the learning rate for mean and variance. $\alpha_w$, and $\alpha_{k,t}$ define the time constant which determine the speed at which the distributions parameters change.

$\alpha_{k,t}$ is derived from $\alpha_w$ as:

$$\alpha_{k,t} = \alpha_w/w_{k,t}. \tag{1.5}$$

Equation (1.5) is employed in the OpenCV algorithm. The equation differs from what is proposed in [16],[17] where it is calculated, being $\eta$ the Gaussian probability density function, as:

$$\alpha_{k,t} = \alpha_w \cdot \eta(pixel, \mu_{k,t}, \sigma_{k,t}). \tag{1.6}$$

Equation (1.5) and the introduction of the matchsum parameters allow to obtain an improved initial learning phase with respect to the algorithm of [16],[17], in which this phase is very slow, [24],[25].

Figure 1.2 shows a comparison between the learning phase of the conventional GMM and the learning phase of the OpenCV algorithm. Figure 1.2(a) shows a video sequence extracted from the Wallflower database of [26]. Figure 1.2(b) and Fig. 1.2(c) show the output obtained with the proposed OpenCV GMM algorithm and with the algorithm of [16],[17], respectively. Using the OpenCV GMM, the background is almost disappeared at the $2^{nd}$ frame and it is well identified at the $56^{th}$ frame. On the contrary, the conventional GMM results in a wrong identification at the $2^{nd}$ frame because all pixels of the scene are classified as foreground and some foreground pixels are still visible to the $56^{th}$ frame.

If none of the K distributions match the current pixel value, the least probable distribution (the distribution with the lowest fitness factor) is replaced with a Gaussian with the current value as its mean value, an initially high variance, and low weight:

$$\begin{aligned} \mu_{k,t+1} &= pixel \qquad & matchsum_{k,t+1} &= 1, \\ \sigma^2_{k,t+1} &= vinit \qquad & w_{k,t+1} &= 1/msumtot, \end{aligned} \tag{1.7}$$

where vinit is a fixed initialization value and msumtot is the sum of the values of the matchsum of the K-1 Gaussians with highest Fitness. The weights of the K-1 Gaussians with highest Fitness are decremented as in 1.4 while their means and variances are unchanged.

**Background identification**

Ordering the Gaussian distributions in decreasing order of the Fitness value is equivalent to order them with the most likely background distributions on the top and the less probable background distributions towards the bottom. Then the background identification is performed by using the following algorithm:

$$B = \arg\min_b (\Sigma_{k=1}^b w_{k,t} > T). \tag{1.8}$$

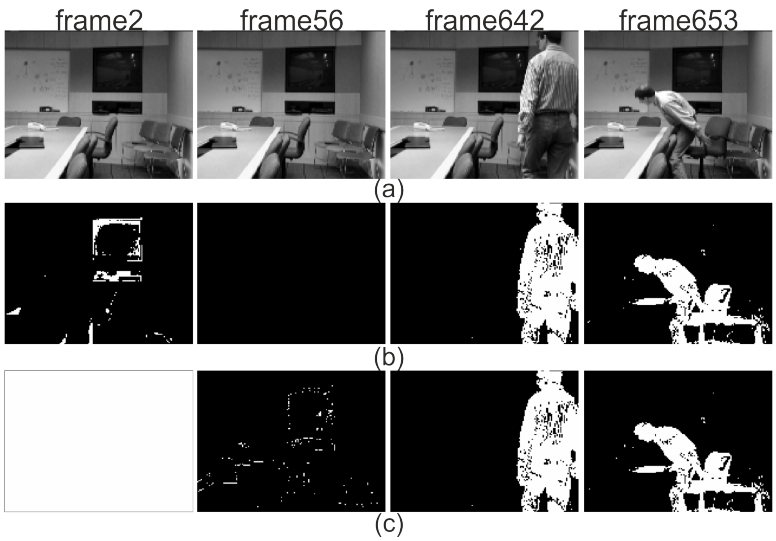


**Figure 1.2:** Comparison of the learning phase for the initial background model between the conventional GMM and the OpenCV algorithm. (a): original frames; (b): output of the OpenCV version of the GMM algorithm; (c): output of the conventional GMM algorithm.

Equation (1.7) adds in succession the weights of the first b Gaussian distributions, in Fitness order, until their sum is greater than T, a fixed threshold belonging to the [0,1] interval.

The set of the Gaussian distributions that verify (1.6) represents the background and a pixel that matches one of these Gaussians is classified as a background pixel. The algorithm entails that the pixel values that do not fit the background distributions are considered foreground.

# Chapter 2

# Hardware implementation of the OpenCV GMM

The Gaussian Mixture Model (GMM) algorithm described in Chapter 1, allows good performances in presence of both illumination changes and multimodal background.

Main drawback of the algorithm is the high computational complexity. In order to generate the updated background model, the GMM algorithm processes the video streams by computing a great number of parameters for each pixel of each frame. This entails a computational burden that grows with the frame size and result unfeasible for real time software implementation. As example, in [16] only a frame rates of 11-13 frame per second (fps) is obtained for a frame size equal to $160\times120$ on an SGI O2 workstation. The authors of [31] conducted a test on a software implementation running on an AMD4400+ processor and observed a frame rate of 4-6 fps for video sequences with $352\times288$ resolution.

Real time video applications with large frame size require dedicated hardware architectures. Hardware processors have been proposed in [27]-[32]. Papers [27],[28] present GPU implementations based on the GMM algorithm. Despite the fact that the approach described in [28] reaches 30 fps for HD (High Definition) videos, being a GPU implementation is an impediment for embedded systems and low power constraints.

Papers [29]-[32] propose FPGA implementations that are more suited to embedded and low power systems.

Reference [29] proposes a circuit able to process video sequences with frame size $1024\times1024$ at 38 fps when implemented on VirtexII FPGA plat-

form. Processing capability of [29] is, therefore, 39.8 Mega pixels per second (Mps).

In [30] the design of an automated digital surveillance system running in real-time on an embedded platform is presented. The segmentation unit of the circuit proposed in [30] is able to run at 83 MHz on VirtexII xc2pro30.

In [31], the research group of [30], improves the memory throughput with respect to [30] employing a memory reduction scheme. However, the resulting processing capability of the overall system is reduced with respect to [30] and only reaches 7.68 Mps.

The hardware implementations of [29]-[32] are not OpenCV compatible. This Chapter shows the hardware implementation of a background identification circuit based on the OpenCV GMM algorithm.

In order to obtain a hardware implementation able to process HD (frame size $1920\times1080$) video sequences in real-time, several optimization techniques have been implemented.

Using a procedure based on the computation of the Peak Signal to Noise Ratio (PSNR), it has been determined the representation of the Gaussian parameters that allows the best trade-off between hardware complexity and video quality (Section 2.1). Moreover, the equations of the OpenCV GMM have been modified in order to obtain a new formulation of the algorithm that allows an optimized hardware implementation (Section 2.2).

Experimental validations show the effectiveness of such new algorithm formulation (Section 2.3).

The circuit has been also implemented by using a UMC-90nm standard cell library (Section 2.4) that allows a frame rate equal to 180 fps. Exploiting the very high working frequency, the ASIC implementation has been modified in order to obtain two architectures with a different trade-off between area utilization and power dissipation.

Finally, truncated binary multipliers and ROM compression techniques have been implemented to further improve the circuit performances (Section 2.5). Implemented on Virtex6-vlx195t, without pipeline registers, the circuit is able to process 57 frames per second (fps) by using 301 of the 31200 Slices of the target FPGA.

The circuits have been described in VHDL and are able to process gray scale videos. When color videos need to be processed, the circuits, can be fed with the luminance channel of the YCrCb color space.

The GMM algorithm has been implemented by using three Gaussian distributions for each pixel. This number determines a trade-off between hardware

complexity and accuracy of the background model. Section 2.6 shows that a good trade-off can be obtained by implementing the GMM with three Gaussian distributions.

## 2.1 Bandwidth reduction

The OpenCV version of the GMM algorithm associates four parameters ($\mu_{k,t}$, $\sigma^2_{k,t}$,$w_{k,t}$,$matchsum_{k,t}$) to each Gaussian distribution that models a pixel. As a consequence, if the algorithm is implemented by using three Gaussian distributions per pixel, the processing of each pixel requires 12 parameters and the pixel luminance.

The software algorithm proposed in the OpenCV libraries represents the Gaussians parameters as double precision (64 bit) floating point numbers. Unfortunately, a hardware implementation using 64 bit floating point signals is not feasible. First of all the required circuitry for the calculation of double precision signals is too large and slow for the effective hardware implementation. Moreover, since as the value of the Gaussian parameters has to be stored for each pixel of a frame, an external memory is needed. An high number of bit per pixel entails an high bandwidth towards the memory that could affect the performances of the circuit. As an example, if the memory throughput is 128 bit, 6 or 7 clock cycles are needed to load the parameters for each pixel. The same number of clock cycles is required to store the updated parameters. The target of processing 41.5 Mps implies that the clock frequency towards the memory is around 550 MHz with a required bandwidth of 8.6 GBs. Such bandwidth is not feasible for a low power lightweight electronic system.

In order to reduce the logic and the bandwidth, it is necessary to reduce the number of bits that represents the pixel statistic.

The examination of the GMM algorithm reveals that the signals have a limited dynamics. Mean, variance, and weight range are in [0,255], [0,127], and [0,1], respectively. When dealing with limited dynamics signals, using a fixed point representation instead of a floating point one provides improved performances while reducing hardware complexity and the bandwidth towards the memory. This requires that the number of bits of the fixed point representation is based on both the range of the signals and the required accuracy.

In the following a specific procedure to determine the signal representation is presented. The proposed procedure guarantees a good trade-off between accuracy and hardware performances.

### 2.1.1 Word length optimization algorithm

For a fair evaluation of the performances of the circuit, five test bench videos have been selected and processed varying the number of bits that represents the Gaussian parameters. The effect of the word length reduction on the background identification has been evaluated calculating the Peak Signal to Noise Ratio (PSNR) for each frame of the test bench videos followed by the calculation of the average PSNR (AvPSNR), considered as the accuracy performance value for the processed video streams.

The PSNR is calculated as:

$$PSNR = 20 \log_{10} \frac{\max(I)}{\sqrt{MSE}} \tag{2.1}$$

$$MSE = \frac{1}{M \cdot N} \sum_{i=1}^{M} \sum_{j=1}^{N} (I(i,j) - Iref(i,j))^2 \tag{2.2}$$

where *max(I)* is the maximum intensity value of a pixel in the processed image (*I*) (equal to 1 for a one bit image). Iref is the reference image. Both *I* and *Iref* have size equal to M×N (320×240). *I(i,j)* and *Iref(i,j)* represent the values of the pixel luminance. For a binary image, the difference between *I(i,j)* and *Iref(i,j)* is '1' if the pixel has been differently classified in *I* and *Iref*; '0' otherwise. Eq.(2.2) is the ratio between the number of pixels differently classified in *I* with respect to *Iref* and the number of pixels in a frame. For each video, *Iref* has been calculated using the double precision floating point OpenCV GMM algorithm.

The word length optimization has been carried out in two phases.

In the first phase the number of bits has been simultaneously reduced for all Gaussians parameters. Table 2.1 reports, for the five test bench videos, the AvPSNR values and the corresponding number of differently classified pixels (background instead of foreground or vice versa) obtained representing mean, variance, weight and matchsum of each Gaussian distribution as a fixed point number on 52, 23, 18, 14, 12, 11, 10, 9, and 8 bits and comparing the processed binary images with the reference images.

The results of Table 2.1 are summarized in Fig. 2.1 that shows the percentage of correctly identified bits (averaged on the five test bench videos) as a function of the word length of the input signals.

Table 2.1 and Fig. 2.1 show that the 52 and 23 bit cases present the same AvPSNR value that corresponds to less than 20 differently classified pixels between the reference images and the considered images. On the other hand, the

**Table 2.1:** AvPSNR values varying the word lengths of mean, variance, weight, and matchsum signals. The table refers to the first phase of the algorithm for the optimization of the word lengths in which the same number of bits is used for every Gaussian parameter.

| **Video** | **AvPSNR (average value for 300 frames)** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **sequence** | **52 bit** | **23 bit** | **18 bit** | **14 bit** | **12 bit** | **11 bit** | **10 bit** | **9 bit** | **8 bit** |
| Video 1 | 46.22 | 44.85 | 34.31 | 19.54 | 18.25 | 17.09 | 14.95 | 8.06 | 0.96 |
| Video 2 | 44.81 | 43.46 | 34.32 | 18.13 | 16.99 | 15.69 | 14.84 | 7.80 | 0.83 |
| Video 3 | 37.45 | 36.97 | 30.17 | 15.79 | 13.81 | 12.12 | 10.73 | 6.18 | 1.41 |
| Video 4 | 37.96 | 37.82 | 32.36 | 18.49 | 16.37 | 14.94 | 13.11 | 7.54 | 0.82 |
| Video 5 | 39.33 | 38.37 | 30.73 | 15.25 | 13.78 | 12.71 | 10.02 | 6.19 | 1.24 |
| **Video** | **Average number of differently classified pixels per frame (frame size 320x200)** | | | | | | | | |
| **sequence** | **52 bit** | **23 bit** | **18 bit** | **14 bit** | **12 bit** | **11 bit** | **10 bit** | **9 bit** | **8 bit** |
| Video 1 | 2 | 3 | 28 | 854 | 1149 | 1501 | 2457 | 12005 | 61569 |
| Video 2 | 3 | 3 | 28 | 1181 | 1535 | 2072 | 2519 | 12746 | 63440 |
| Video 3 | 14 | 15 | 74 | 2025 | 3194 | 4714 | 6492 | 18508 | 55509 |
| Video 4 | 12 | 13 | 45 | 1087 | 1772 | 2462 | 3753 | 13532 | 63586 |
| Video 5 | 9 | 10 | 65 | 2293 | 3216 | 4115 | 7645 | 18466 | 57725 |

AvPSNR value is not acceptable when the signals are represented on 9 bits or 8 bits. Fig.2.1 shows that, in this case, a dramatic drop in the image quality occurs and more than 15% or 70% of the pixels are erroneously identified. The AvPSNR has a reasonable value when 10 bits or 11 bits are used for each signal. Since, for the applications of the GMM algorithm, the reliability is important and the 10 bit case is too close to the point in which the AvPSNR drops dramatically, we chose the 11 bits case as the output of this first optimization phase. Note also that using 11 bits for each signal allows, on average, to correctly identify 94% of the pixels. This can be seen averaging the number of differently classified pixels in the 11 bit column of Table 2.1 (obtaining 2973

pixel) and considering that a frame contains 64000 pixels.

In the second phase of the optimization, the word length of one signal between variance, mean, weight, and matchsum has been further reduced in order to obtain a more efficient circuit that still provides acceptable AvPSRN values. Each time a word length reduction has been attempted, it has been considered feasible if the calculated AvPSNR values are higher than the AvPSNR values obtained representing all the parameters with 10 bits (shown in Table 2.1) that is defined as the lowest performance threshold for the identification circuit.

The word length reduction has been firstly carried out on the signals for which the reduction of the number of bits provides the highest reduction of resource utilization. Analyzing the GMM equations reported in the Chapter 1, it has been determined that the variance is the parameter whose impact on resource utilization is the highest. After the variance, the signal that more heavily impacts circuit complexity is the mean, followed by the weight and the matchsum signal.

The AvPSNR values obtained for the second phase of the optimization of the word length of the signals are shown in Table 2.2.

Column A of Table 2.2 refers to the variance signal represented on 10 bits and shows that the resulting AvPSRN values are lower than the performance threshold (defined as the AvPSNR value of the 10 bit case of Table 2.1). As consequence, it is not possible to reduce to 10 bits the word length of the variance.

Column B and C refer to the reduction of the word length of the mean signal. The AvPSNR values are acceptable when the variance is represented with 11 bits and the mean with 10 bits (column B). When the mean is brought to 9 bits (column C), the AvPSRN values are too low. Consequently, the word length of the mean has been fixed to 10 bits.

Column D and E refer to the reduction of the word length of the weight signal. Keeping the variance on 11 bits and the mean on 10 bits, it has been found that is possible to reduce the word length for the weight to 8 bits without excessive deterioration of the AvPSNR values (column D). When the word length of the weight is reduced to 7 bits (column E), the AvPSNR value is very close to the performance threshold and in one case (Video 2) is lower than it. The optimal word length for the weight has therefore been fixed to 8 bits.

Column F of Table 2.2 shows that good AvPSNR values (unchanged with respect to column D) and reduced logic utilization are obtained allocating 4 bits to the matchsum signal. A further reduction of the number of bits for matchsum would provide little reduction in circuit complexity and has not been

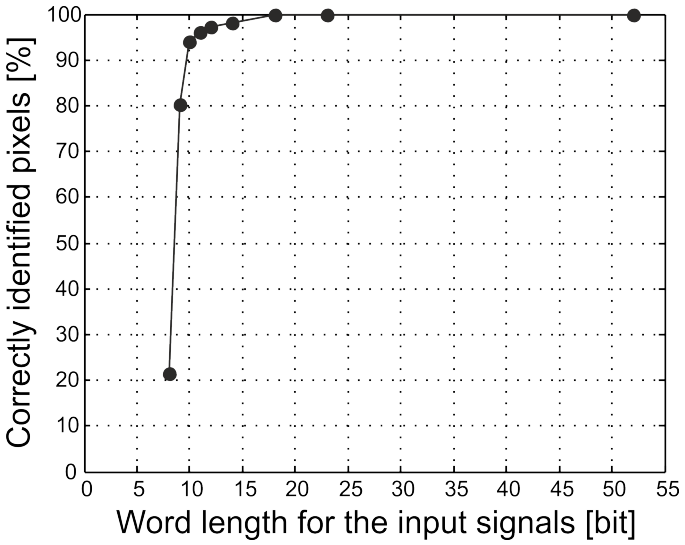


**Figure 2.1:** Percentage of correctly identified pixels for each frame as a function of the word length of the input signals. The reference frames come from the double precision floating point OpenCV GMM algorithm. The points are the results averaged on the five test bench video of Table 2.1.

**Table 2.2:** AvPSNR values for the second phase of the word length optimization algorithm. The number of bits for variance, mean, weight, and matchsum are reduced one by one starting from 11 bits that is the result of the first phase of the optimization procedure. A: variance represented with 10 bits; B: mean represented with 10 bits; C: mean represented with 9 bits; D: mean and weight represented with 10 bits and 8 bits, respectively; E: mean represented with 10 bits and weight with 7 bits; F: mean, weight, and matchsum represented with 10, 8 and 4 bits, respectively. The values that are below the performance threshold are shown in bold. They make the accuracy not acceptable.

| | **AvPSNR (average value for 300 frames)** | | | | | |
|---|---|---|---|---|---|---|
| | **Reduction of the wordlength of the** | | | | | |
| | **variance ($\sigma^2$)** | **mean ($\mu$)** | | **weight (w)** | | **matchsum** |
| | **A** | **B** | **C** | **D** | **E** | **F** |
| | $\sigma^2$ 10bit | $\sigma^2$ 11bit | $\sigma^2$ 11bit | $\sigma^2$ 11bit | $\sigma^2$ 11bit | $\sigma^2$ 11bit |
| | | $\mu$ 10bit | $\mu$ 9bit | $\mu$ 10bit | $\mu$ 10bit | $\mu$ 10bit |
| **Video** | | | | w 8bit | w 7bit | w 8bit |
| **sequence** | | | | | | matchsum 4bit |
| **Video 1** | **14.90** | 17.09 | **14.13** | 16.44 | 16.35 | 16.44 |
| **Video 2** | **14.70** | 15.69 | **14.43** | 15.05 | **14.78** | 15.05 |
| **Video 3** | **10.70** | 12.06 | 11.27 | 11.53 | 11.41 | 11.53 |
| **Video 4** | **12.49** | 14.87 | **12.53** | 14.06 | 13.25 | 14.06 |
| **Video 5** | 10.12 | 12.71 | **9.06** | 11.46 | 10.53 | 11.46 |
| **Video** | **Average number of differently classified pixels per frame (320×200)** | | | | | |
| **sequence** | **A** | **B** | **C** | **D** | **E** | **F** |
| **Video 1** | 2485 | 1501 | 2967 | 1743 | 1780 | 1743 |
| **Video 2** | 2602 | 2072 | 2769 | 2401 | 2555 | 2401 |
| **Video 3** | 6537 | 4779 | 5733 | 5400 | 5551 | 5400 |
| **Video 4** | 4329 | 2502 | 4289 | 3016 | 3634 | 3016 |
| **Video 5** | 7471 | 4115 | 9536 | 5487 | 6798 | 5487 |

**Table 2.3:** Word lengths and representations for the Gaussian parameters and the pixel luminance value.

| Signal | Representation | ♯ bit for each signal | ♯ bit for three Gaussians |
|---|---|---|---|
| pixel | $U_{7,0}$ | 8 | 8 |
| $w_{k,t}$ | $U_{-1,8}$ | 8 | 24 |
| $\mu_{k,t}$ | $U_{7,2}$ | 10 | 30 |
| $\sigma^2_{k,t}$ | $U_{13,-3}$ | 11 | 33 |
| $matchsum_{k,t}$ | $U_{3,0}$ | 4 | 12 |

addressed.

The resulting representations used for the Gaussian parameters are shown in Table 2.3 and are given with $U_{m,n}$ notation. $U_{m,n}$ indicates an unsigned fixed point number where $2^m$ is the weight of the MSB and $2^{-n}$ is the weight of the LSB. The last column of Table 2.2 allows to determine the quality of the processed videos when using the word lengths reported in Table 2.3. It can be seen that the percentage of differently classified pixels for the five test bench videos is 2.7%, 3.7%, 8.4%, 4.7%, and 8.6%. The analysis shows that the proposed fixed point representations provide little impact on the quality of the processed videos.

With the representations of Table 2.3, 107 bits of data are needed for each pixel, resulting in a required memory bandwidth for HD real time video processing of 0.99 GBs that can be compared with the results of [30] and [29]. The segmentation architecture proposed in [30], while processing 25 fps with resolution 640×480, requires a bandwidth toward the memory of 4.3 Gbit/s that can be reduced to 0.82 Gbit/s using a data compression technique. The GMM implementation for FPGA devices of [29], particularly oriented to the reduction of the bandwidth towards the memory, needs 170 MBs of bandwidth when processing 7.68 Mps. As a consequence, the architectures of [30] and [29], when processing 20 HD frames per second, require 0.60 GBs and 0.90 GBs of memory bandwidth, respectively. The bandwidths obtained in [30] and [29] are 39% and 9% lower than the bandwidth obtained with the representations reported in Table 2.3. This is however obtained, as will show in Section 2.1.2, to the expense of processing capabilities and with a large increase of

logic utilization.

The proposed analysis has been conducted by using an $\alpha_w$ (learning rate) value of $2^{-6}$ that is appropriate for most applications. Since, in principle, the number of bits of the signals is also a function of $\alpha_w$, as smaller learning rates increase the required precision, a similar analysis has been conducted for a learning rate equal to $2^{-9}$. Note that learning rates values lower than $2^{-9}$ cause a very slow adaptation of the background model that becomes unable to describe suddenly changes of the background, [30]. The result of the optimization showed no significant changes in this worst case condition. The only difference is the word length of the weight that cannot be reduced below 10 bits. This has a minimal impact on circuit complexity and memory bandwidth.

The OpenCV version of the GMM algorithm described in Section 1.2.1 has been implemented on FPGA devices by using the representations reported in Table 2.3. The circuit is indicated in the Chapter as FgBg (Foreground/Background) BW (Bandwidth) optimized circuit and its implementation is described in the following.

### 2.1.2 FgBg BW optimized circuit- FPGA implementation

Figure 2.2 shows the block diagram of the FgBg BW optimized circuit. The input data are the 8 bit luminance of the input pixel (Pixel), and the statistical model of the pixel for the given frame ($\mu_{k,t}$, $\sigma^2_{k,t}$, $w_{k,t}$, $matchsum_{k,t}$). The output data are the updated statistical model ($\mu_{k,t+1}$, $\sigma^2_{k,t+1}$, $w_{k,t+1}$, $matchsum_{k,t+1}$) and the Fg/Bg tag that indicates if the Pixel belongs to the background (logic value '0') or to the foreground (logic value '1').

The FgBg BW optimized circuit implements the GMM algorithm by using three Gaussian distributions for each pixel (k=[1,2,3]). As explained in [16], the number of Gaussian distributions per pixel depends on a trade-off between hardware complexity and accuracy of the background model. Section2.6 shows this trade-off while validating the choice of using three Gaussian distributions per pixel.

Target devices for the proposed implementation are FPGA devices, a technology largely used and of great interest for the considered applications. The proposed HW design is described in VHDL code and parameterized with respect to the word lengths of the input signals allowing a straightforward modification of the representation of the signals.

The FgBg BW optimized circuit process gray scale videos. When color videos need to be processed, the circuit, can be fed with the luminance channel of the YCrCb color space.

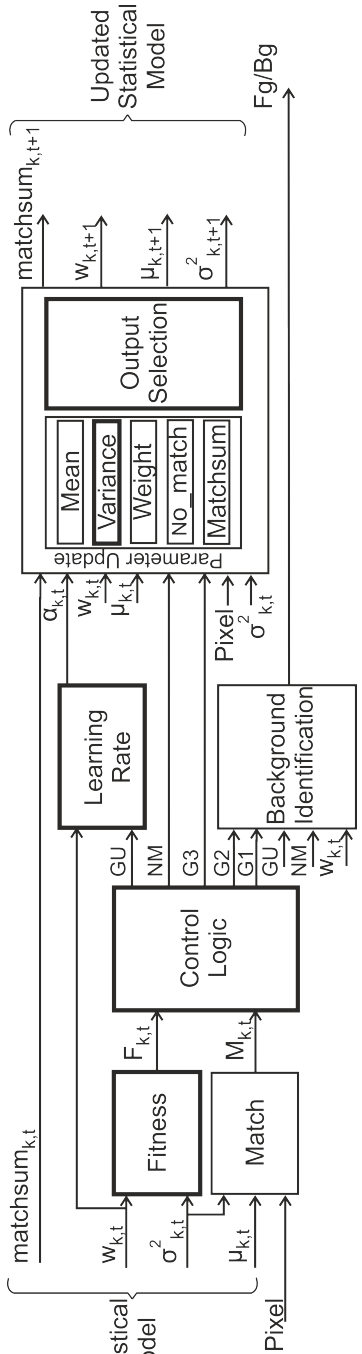


**Figure 2.2:** Block diagram of the proposed FgBg BW optimized circuit. The circuital units that are on the critical path are shown with thick boundary lines. The signals shown with subscript "k" refer to a bus of three signals (e.g. $w_{k,t}$=$w_{1,t}$,$w_{2,t}$,$w_{3,t}$).

A detailed explanation of the working principle of the circuit of Fig. 2.2 is given in the following.

**Fitness**: Computes the Fitness factors for the three Gaussian distributions. It is composed, as shown in Fig. 2.3, by three identical units that implement (1.1). Equation (1.1) requires both a binary inversion and a square root operation. Nonlinear operations such as inversion and square root require a large and slow hardware, [33]. A common technique to reduce the logic for the non linear computations is the implementation of ROM circuits that directly store the results of the non linear operations. The "Fitness" unit is implemented with a ROM, in which the precalculated inverse of the square root of the variance input are stored, and a multiplier, that multiplies the ROM output by the weight of the Gaussian. This implementation technique allows to compute the inverse of the square root in one clock cycle with a small hardware. Since the variance and the inverse of the standard deviation are on 11 and 8 bits, respectively, ROM size is $211\times8$ bits. As an example, when implemented on a Virtex5 xc5vlx50 FPGA, if the ROM is implemented by using LUT as memory elements, the Fitness unit requires 216 LUTs and 1 DSP block.

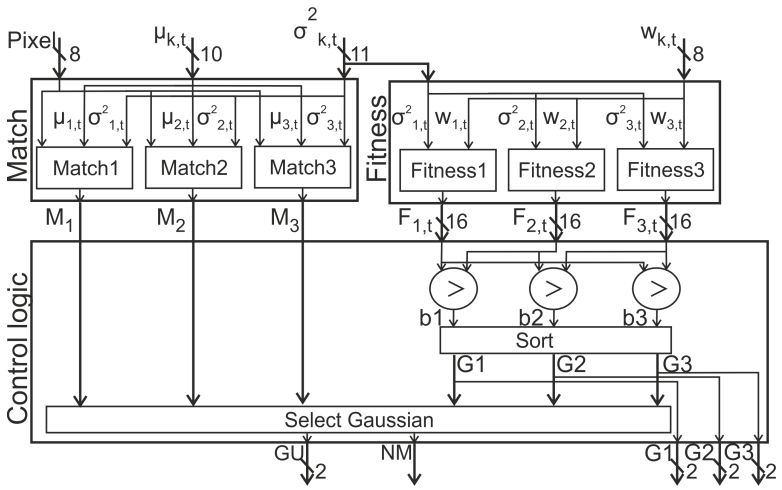


**Figure 2.3:** Detailed view and block diagram for "Fitness", "Match", and "Control Logic" circuital blocks shown in Fig. 2.2.

**Match**: Verifies the match condition for the three Gaussian distributions and is composed, as shown in Fig. 2.3, by three identical units (one for each Gaussian). Each unit implements:

$$M_{k,t} = 1 \qquad if(pixel - \mu_{k,t})^2 < \lambda^2 \cdot \sigma_{k,t}^2, \tag{2.3}$$

that is the square of (1.2). Equation (1.2) requires the standard deviation as input while (2.3) requires the variance as input. Since the input data for the circuit include the variance, using (1.2) would require the calculation of the standard deviation with a square root circuit. Using (2.3) is therefore simpler.

The resulting circuit employs a subtractor, a squarer, a multiplier and a comparator.

As an example, when implemented on Virtex5-vlx50 FPGA with the representations of Table 2.3, its logic utilization is 90 LUTs and 6 DSP blocks.

**Control Logic**: Sorts the Gaussian distributions in decreasing Fitness order and establishes which Gaussian has to be updated as in (1.3), (1.4), and (1.7).

Fig. 2.3 shows a detailed view of the circuit. The Fitness values are compared using three comparators whose binary outputs (b1, b2, b3) are used by the circuit to establish the order of the Gaussians. The circuital unit is therefore implemented by using only three comparators and few logic gates.

The output signals G1, G2 and G3 represent the first, second and third Gaussian in Fitness order while NM (No-Match) and GU (Gaussian Update) establish which Gaussian must be updated. If NM is '0' the Gaussian selected by GU is updated as in (1.3) while the weights of the remaining Gaussians are updated as in (1.4). If "NM" is '1' the Gaussians are updated as in (1.7).

**Learning Rate**: computes the learning rate $\alpha_{k,t}$ as in (1.5) to update mean and variance of the Gaussians. The proposed implementation uses a representation of $\alpha_{k,t}$ that simplifies the circuits that compute (1.3) and (1.4). In fact, the calculation of (1.3) and (1.4), as shown in Fig. 2.4, implies the use of multipliers. In the proposed implementation the $\alpha_w$ and $\alpha_{k,t}$ values are quantized as power of two ($\alpha_w = 2^{ew}$ and $\alpha_{k,t} = 2^{ekt}$), allowing the replacement of the multipliers with shifters (dashed lines in Fig. 2.4). The *ew* value is hardwired while the *ekt* values that better approximate (1.5) as a function of $w_{k,t}$ are stored in a ROM. The resulting "Learning Rate" block is composed of a single ROM that uses 11 LUT of a Virtex5-vlx50 FPGA.

The quantization of the learning rates introduces a further approximation on the GMM algorithm. Fig. 2.5 shows a selection of frames taken from a test bench video (Fig. 2.5(a)) and processed with the hardware implementations that uses the multipliers (Fig. 2.5(b)) or the shifters (Fig. 2.5(c)). The frames size is 272×176 pixels and 300 frames have been processed. The AvPSNR

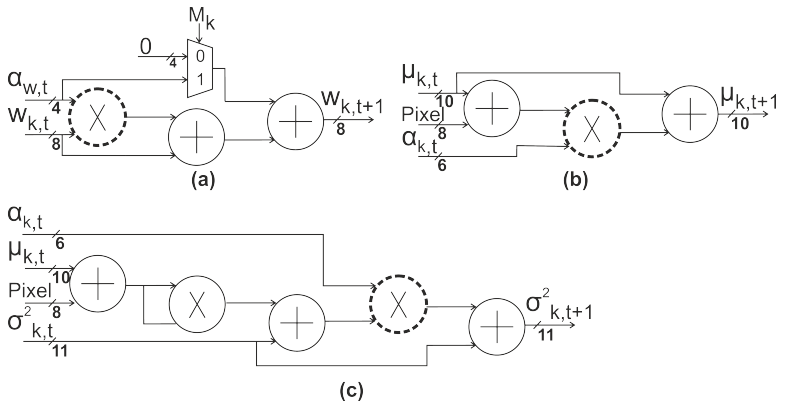


**Figure 2.4:** Proposed implementation of the updating equations for (a) the weight, (b) the mean, (c) the variance signals. The multipliers drawn with dotted lines are replaced by shifters. Circuits (a), (b), and (c) correspond, respectively, to the "Weight", "Mean" and "Variance" blocks shown in Fig .2.2.

value, calculated considering as reference the output of the FgBg BW optimized circuit implemented with the multipliers, is equal to 24.11. This means that the two video sequences differ, on average, in 186 pixels per frame (0.4% of differently classified pixels). The test demonstrates that the proposed use of shifters instead of multipliers is feasible for the considered application.

**Parameter Update**: A detailed scheme of the unit is shown in Fig. 2.6. If the match condition is verified, "Weight", "Mean", and "Variance" blocks update the parameters according to equation (1.3) using the circuits shown in Fig. 2.4 where the multipliers have been replaced by shifters. The "Mean" block, for a Virtex5-vlx50 implementation, occupies 53 LUTs. The "Weight" block uses 14 LUTs. The "Variance" block is implemented by using 69 LUTs and 1 DSP block.

If no Gaussians match the pixel, the "$No\_match$" block updates the mean, the variance and the weight of the Gaussian with smallest Fitness value according to (1.7).

The "Matchsum" block is the circuit that updates the matchsum signal according to (1.3) and (1.7). As explained in Section 1.2.1, the matchsum signal is a counter, associated to every Gaussian, and introduced in the OpenCV GMM algorithm. The matchsum signal counts the number of times that a given

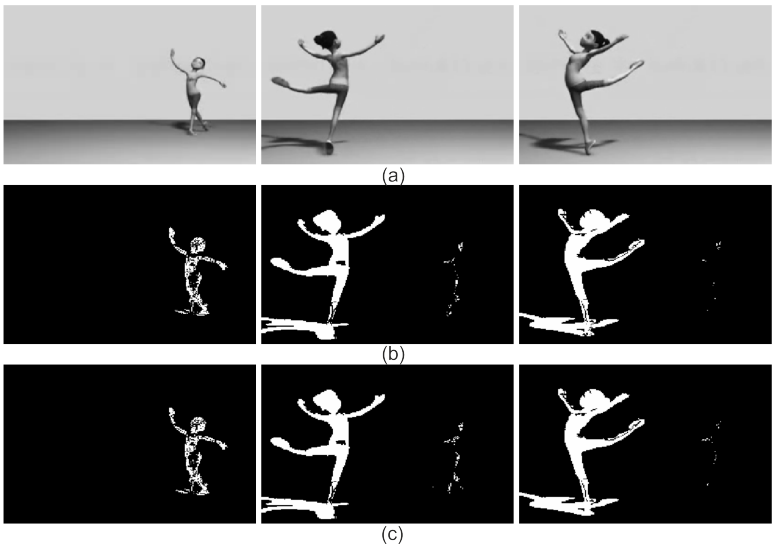


**Figure 2.5:** Background masks obtained processing the frames showed in (a) with the hardware implementations that use the binary multipliers (b), and the shifters (c).

Gaussian is matched according to (1.2). The introduction of the matchsum entails the synthesis of three counters that are not on the critical path. The only drawback is an increase of circuit area and of power dissipation.

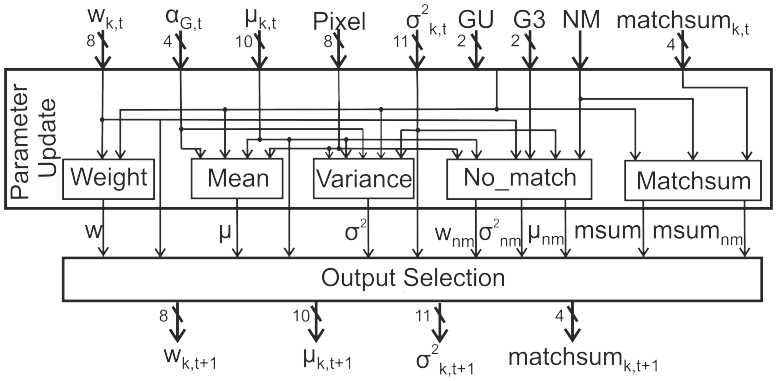


**Figure 2.6:** Block diagram of the "Parameter update" and "Output Selection" units shown in Fig. 2.2.

**Output Selection**: establishes the values of the updated parameters depending on whether the match condition is verified or not.

**Background Identification**: Verifies the background identification condition shown in (1.8) determining, also as a function of the NM and GU signals, if the input pixel belongs to the background or not. Logic occupation for Virtex5 implementation is 47 LUTs

### Results and performances

The FgBg BW optimized circuit described in Section 2.1.2 has been synthesized and implemented on Virtex6, Virtex5, and Spartan6 Xilinx FPGA. Moreover, in order to compare the circuit with previous works, the circuit has been also implemented on VirtexII, VirtexII Pro, VirtexE Xilinx FPGA devices.

The synthesis for VirtexII Pro is conducted using Synplify while XST has been used for Virtex6, Virtex5, Spartan3, VirtexII and VirtexE synthesis. Fitting and Place and Route have been carried out using ISE for Xilinx FPGA devices.

Circuit simulations uses ISim (Xilinx) that also provide the "vcd" files for the accurate determination of the power dissipation.

Power dissipation is, afterword, computed using by XPower. Only the

dynamic power dissipation, without including the power dissipation due to the I/O pads is reported.

The analyses have been conducted including input and output registers that synchronize the circuit and provide timing performances that are not dependent on the I/O pads.

Table 2.1.2 shows the performances of the FgBg BW optimized circuit as a function of the target FPGA and of the number of pipeline levels on Virtex6 (xc6vlx195t), Virtex5 (xc5vlx50), and Spartan6 (xc6slx100) Xilinx FPGA.

The input and output registers are not counted has pipeline levels. When the number of pipeline levels is not indicated or is indicated as zero only the input and output registers are included into the circuit. When the number of pipeline levels is indicated as one, a further level of registers has been inserted into the circuit, breaking the maximum combinatorial delay and improving the maximum working frequency. In this case the optimal placement of the registers has been obtained with an iterative procedure that mixes hand placement of registers and exploiting the retiming feature of the XST synthesizer.

The placing and the number of pipeline registers has been optimized using the retiming feature of Synplify and XST synthesizers.

FgBg BW optimized circuit implementations on Virtex6 and Virtex5 FPGA, without pipeline levels, are able to run at 59.20 MHz and 57.22 MHz, respectively. As a consequence, the two implementations allow the processing of 28 fps and 27 fps, respectively. When implemented on Spartan6 FPGA without exploiting the pipelining, the FgBg BW optimized circuit is able to process 17 fps (maximum working frequency equal to 35.95 MHz). The maximum working frequency (and, as a consequence, the frame rate) can be improved by implementing the circuit with some pipeline levels. Of course, the pipelining technique entails an increase of the utilization of sequential elements. However, Table 2.1.2 shows that this doesn't entail a consequent increase of the Slice's utilization on the FPGA because this depends how the synthesizer can compact the design using an higher or a lower number of resources in a Slice.

With reference to the dynamic power dissipation can be observed that using more recent FPGA devices (that use a more advanced silicon technology) provides lower power dissipation. This is expected since scaled silicon technologies provide both higher working frequency and lower power dissipation. Moreover, an interesting result regarding the power dissipation is that, for the considered circuits, the introduction of pipeline levels provides a reduction of the power dissipation. This is due to the reduction of the glitch related power dissipation that is more significant than the power dissipation increase due to

**Table 2.4:** Performances of the FgBg BW optimized circuit implemented on Virtex6, Virtex5, and Spartan6 Xilinx FPGA devices.

| Target device | Pipeline levels | LUT | Flip-Flop | Slice | DSP-Mult | Frequency (MHz) | fps (1920×1080) | Power dissipation (mW/MHz) |
|---|---|---|---|---|---|---|---|---|
| Virtex6 xc6vlx195t | 0 | 723/124800 | 0/249600 | 282/31200 | 7/640 | 59.20 | 28 | 3.95 |
| | 1 | 818/124800 | 228/249600 | 337/31200 | 7/640 | 120.78 | 58 | 1.30 |
| | 2 | 825/124800 | 436/249600 | 312/31200 | 7/640 | 141.46 | 68 | 0.67 |
| Virtex5 xc5vlx50 | 0 | 695/28800 | 0/28800 | 289/7200 | 7/48 | 57.22 | 27 | 7.28 |
| | 1 | 657/28800 | 171/28800 | 307/7200 | 7/48 | 102.31 | 49 | 2.19 |
| | 2 | 838/28800 | 416/28800 | 364/7200 | 7/48 | 110.70 | 50 | 0.98 |
| Spartan6 xc6slx100 | 0 | 948/63288 | 0/12657 | 343/15822 | 7/180 | 35.95 | 17 | 10.98 |
| | 1 | 996/63288 | 288/126576 | 472/15822 | 7/180 | 66.96 | 32 | 2.38 |
| | 2 | 860/63288 | 443/126576 | 351/15822 | 7/180 | 85.29 | 41 | 2.34 |

the presence of the flip flops. This reduction is higher for the first stage of pipeline levels and decreases when further pipeline levels are introduced in the circuit. For the Spartan6 implementation, for example, the power dissipation decreases of the 70% for the implementation with one level of pipeline with respect to the combinatorial implementation of the circuit but only a reduction of the 0.01% is obtained by introducing a further level of registers.

### Comparison with previous art

In order to compare the circuit with previous art, the FgBg BW optimized circuit has also been implemented on VirtexII, VirtexII Pro and VirtexE. The results are shown in Table 2.5.

Papers [29]-[32], propose different FPGA implementations of the GMM algorithm that can be compared with the FgBg BW optimized circuit even if they provide no information regarding pipeline levels or power dissipation of the systems, and do not comply with the OpenCV algorithm.

In [30] the design of a digital surveillance system running in real-time on an embedded platform is presented. The circuit proposed in [30] is able to process 25 fps with frame size 320x240 and hence reaches 1.9 Mps. The segmentation circuit proposed in [30] runs at 83 MHz on VirtexII xc2pro30 using 3397 of the 27932 available LUT and 13 of the 136 available BRAM. The proposed circuit with two levels of pipeline is faster than [30] and present a lower logic occupation (1832 LUT). Moreover, no BRAMs are used. However, the circuit of [30], using a memory compression technique, has a memory bandwidth that is 39% lower than the proposed circuit.

In [29], the research group of [30], proposes a circuit with improved processing capabilities that processes 1024x1024 images at 38 fps on VirtexII xc2v1000 FPGA. Processing capability of [29] is 39.8 Mps. Not many details regarding the performances but the speed are reported in [29]. In particular no information is provided regarding programmable logic occupation, the number of pipeline levels, and the power dissipation. The processing speed of [29] does not allow the real time processing of HD video. Table 2.5 shows that the circuit, without pipeline levels, overcome the processing speed of [29].

Reference [31] proposes an implementation on VirtexII xc2pro30 oriented to the reduction of the bandwidth towards the memory. The circuit employs a memory reduction scheme. The circuit of [31] has a memory throughput that is 9% lower than the proposed circuits. Table 2.5 shows that [31] is not able to process real time HD video and has a programmable logic occupation that is more than seven times larger than the proposed circuit.

**Table 2.5:** Comparison between the performances of the proposed FgBg BW optimized circuit and the performances of previously proposed FPGA implementations of the GMM algorithm.

| Target device | Circuit | Pipeline levels | LUT | Flip-Flop | BRAM | Slice | Frequency (MHz) | fps (1920×1080) | Power dissipation (mW/MHz) |
|---|---|---|---|---|---|---|---|---|---|
| VirtexII xc2pro30 | FgBg BW optimized | 2 | 1832/27392 | 566/27392 | 0/136 | 1114/13696 | 83.91 | 40 | 1.19 |
| | Ref.[30] | N.A. | 3397/27932 | N.A. | 13/136 | N.A. | 83.00 | 40 | N.A. |
| | FgBg BW optimized | 1 | 1724/27392 | 354/27392 | 0/136 | 993/13696 | 55.79 | 26 | 3.01 |
| | Ref.[31] | N.A. | N.A. | 4273/27392 | 84/136 | 6107/13696 | 7.7 | 3 | N.A. |
| VirtexII xc2v1000 | FgBg BW optimized | 0 | 1421/10240 | 0/10240 | 0/136 | 800/5120 | 51.17 | 24 | 5.59 |
| | Ref.[29] | N.A. | N.A. | N.A. | N.A. | N.A. | 40 | 19 | N.A. |
| VirtexE xcv2000E | FgBg BW optimized | 2 | 2340/4704 | 470/4704 | 0/160 | 1381/2352 | 42.6 | 20 | 13.16 |
| | Ref.[32] | N.A. | 1845/4704 | N.A. | N.A. | 1456/2352 | 27.0 | 13 | N.A |

In [32] a GMM based classifier based on distributed arithmetic is developed. The implementation is carried out on Xilinx XCV2000E FPGA. The maximum working frequency of the circuit is 27 MHz. It is able to process 27 Mps. When compared with the proposed circuit with two levels of pipeline, the circuit proposed in [32] occupies 5.4% more slices (1456 vs. 1381) while being 35% slower (HD frame rate of 13 fps vs. 20 fps). No information is provided in [28] regarding the power dissipation, the number of pipeline levels, and the bandwidth towards the memory.

## 2.2 Hardware oriented GMM algorithm

The GMM algorithm equations detailed in Section 1.2.1 require the implementation of non linear functions. Dedicated circuits for the non linear operations would be too complex for the realization of a fast and energy efficient circuit.

A common technique to reduce the logic for the non linear computations is the implementation of ROM circuits that directly store the results of the non linear operations. Taking into account that the input data of the circuit are represented on a limited number of bits, the FgBg BW optimized circuit described in Section 2.1.2 uses, for example, three look-up tables implemented with ROM in order to compute the $F_{k,t}$ factors for the three Gaussian distributions (equation (1.1)), one look-up table is used to compute the learning rate $\alpha_{k,t}$ for the matched distribution (equation (1.4)) and a further look-up table is used to obtain the inverse of the *msumtot* signal (equation (1.7)).

In order to reduce the logic occupation, a new order parameter for the Gaussian distributions can be introduced. This parameter, indicated in the following as $IF_{k,t}$, is defined as the square of the inverse of the $F_{k,t}$ factor:

$$IF_{k,t} = \left(\frac{1}{F_{k,t}}\right)^2 . \tag{2.4}$$

With simple manipulations, taking into account (1.1) and (1.5), $IF_{k,t}$ can be written as:

$$IF_{k,t} = \left(\frac{\sigma_{k,t}}{w_{k,t}}\right)^2 = \left(\frac{\sigma_{k,t}}{w_{k,t}} \cdot \frac{\alpha_{k,t}}{\alpha_{k,t}}\right)^2 = \sigma_{k,t}^2 \cdot \left(\frac{\alpha_{k,t}}{\alpha_w}\right)^2 . \tag{2.5}$$

Equation (2.5) shows that the $IF_{k,t}$ factor can be computed as a function of the learning rate $\alpha_{k,t}$.

Moreover if the learning rates $\alpha_w$ and $\alpha_{k,t}$ are quantized as power of two

$$\alpha_w = 2^{ew} \qquad \alpha_{k,t} = 2^{ekt}. \tag{2.6}$$

the $IF_{k,t}$ factor can be computed as:

$$IF_{k,t} = \sigma^2_{k,t} \cdot 2^{2(ekt-ew)}, \tag{2.7}$$

and the multiplication between $2^{2(ekt-ew)}$ and $\sigma^2_{k,t}$ can be performed using a shifter instead of a binary multiplier.

It is worth noting that ordering the Gaussian distributions as a function of $IF_{k,t}$ is equivalent, as shown in (2.8), to ordering versus the $F_{k,t}$ value.

$$\begin{aligned} F_{1,t} > F_{2,t} > F_{3,t} <=> \\ F^{-1}_{1,t} < F^{-1}_{2,t} < F^{-1}_{3,t} <=> \\ F^{-2}_{1,t} = IF_{1,t} < F^{-2}_{2,t} = IF_{2,t} < F^{-2}_{3,t} = IF_{3,t}. \end{aligned} \tag{2.8}$$

Implementing equations (1.5) and (2.7) require only a ROM and a shifter and, as an example, if implemented on Virtex5-vlx50 FPGA, require the usage of 80 (of 7200) Slices. This result can be compared with the implementation of the equations (1.5) and (1.1) (the technique used to implement the FgBg BW optimized circuit described in Section 2.1.2) that, on the same FPGA, requires 123 Slices and 3 DSP. This demonstrates that the new formulation of the GMM algorithm equations is effective in reducing circuit complexity.

The circuit that implements the equations previously described is indicated in the following as FgBg HW (Hardware) oriented circuit. Its FPGA implementation is described in the next Subsection. It is worth nothing that the representations used for the Gaussian parameters are those reported in Table 2.3. As a consequence the bandwidth toward the memory is the same of the FgBg BW optimized circuit.

### 2.2.1 FgBg HW oriented circuit- FPGA implementation

Fig. 2.7 shows the schematic of the FgBg HW oriented circuit.

With respect to the FgBg BW optimized circuit the "Fitness" unit is replaced with the "IFitness" for the computation of equation (2.7) starting from the learning rate. It is worth nothing that the FgBg BW optimized circuit of Fig. 2.2 computes the learning rate (equation (1.5)) only for the matched distribution. The new formulation of the algorithm equations requires the computation of (1.5) for all the three Gaussian distributions. As a consequence,

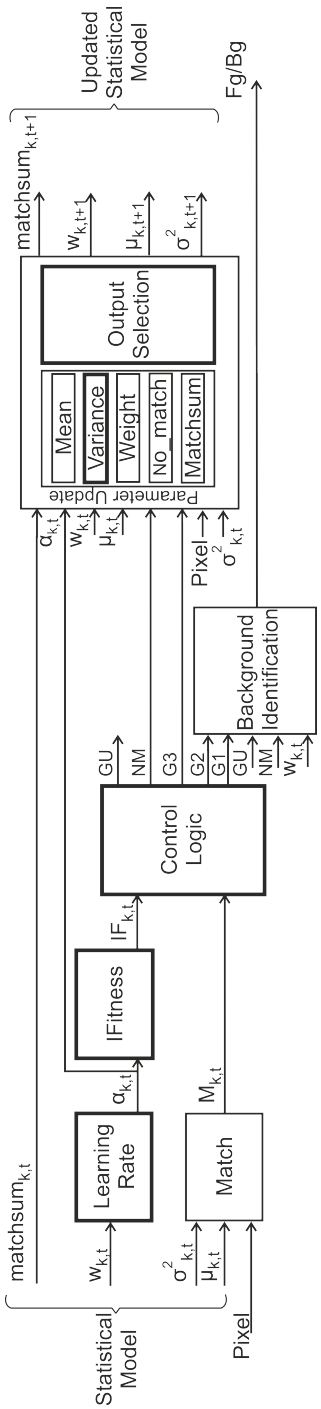


**Figure 2.7:** Block diagram of the FgBg HW oriented circuit that implements the optimized hardware oriented formulation of the GMM described in Section 2.2.

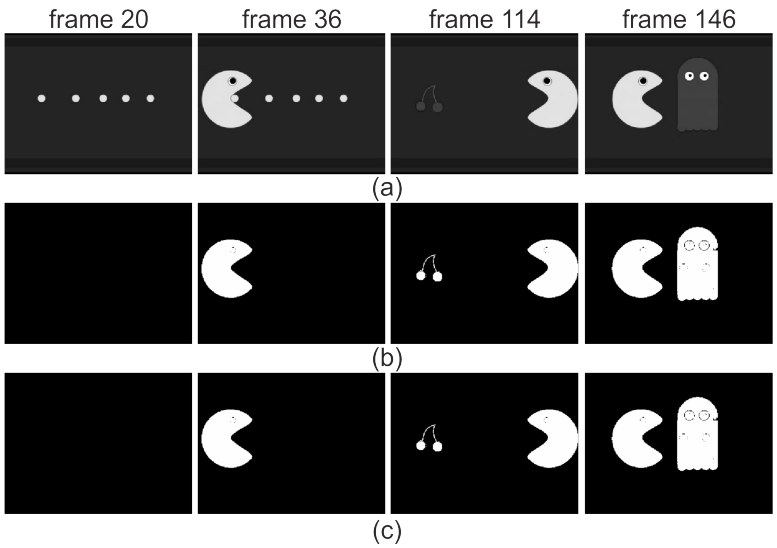


**Figure 2.8:** Comparison between the background masks obtained by using the GMM equations described in Section1.2.1 and the formulation proposed in Section 2.2 (uses (1.5) and (2.5)). (a): input video frames; (b): background masks obtained by using the formulation described in Section 2.2; (c): background masks obtained by using the equations of Section 1.2.1. The background masks in (b) and (c) are nearly identical showing that the proposed formulation does not affect the quality of the processed videos.

the ”Learning rate” unit of Fig. 2.7 is implemented three times (one for each Gaussian).

The performances of the FgBg HW oriented circuit and the comparison with the performances of the FgBg BW optimized circuit are reported in the following.

### Results and Performances

The proposed FgBg HW oriented circuit is synthesized and implemented on Virtex6 (xc6vlx195t), Virtex5 (xc5vlx50), and Spartan6 (xc6slx100) Xilinx FPGA.

The synthesis and the Place&Route have been carried out by using ISE. ISim has been used to perform the circuit simulations.

**Table 2.6:** Performances of the FgBg HW oriented circuit whose block diagram is shown in Figure 2.7. The circuit has been implemented on Virtex6, Virtex5, Spartan6 Xilinx FPGA. The table shows also a comparison with the performance of the FgBg BW optimized architecture of Fig. 2.2

| Target device | Circuit | Pipeline levels | LUT | Flip-Flop | Slice | DSP-Mult | Frequency (MHz) | fps (1920×1080) | Power dissipation (mW/MHz) |
|---|---|---|---|---|---|---|---|---|---|
| Virtex6 xc6vlx195t | FgBg HW oriented | 0 | 692/124800 | 0/249600 | 279/31200 | 3/288 | 85.70 | 41 | 3.80 |
| | FgBg BW optimized | 0 | 723/124800 | 0/249600 | 282/31200 | 7/640 | 59.20 | 28 | 3.95 |
| Virtex5 xc5vlx50 | FgBg HW oriented | 0 | 742/28800 | 0/28800 | 303/7200 | 3/48 | 78.87 | 38 | 5.66 |
| | FgBg BW optimized | 0 | 695/28800 | 0/28800 | 289/7200 | 7/48 | 57.22 | 27 | 7.28 |
| Spartan6 xc6slx100 | FgBg HW oriented | 0 | 914/63288 | 0/126576 | 344/15822 | 3/180 | 58.86 | 28 | 7.15 |
| | FgBg BW optimized | 0 | 948/63288 | 0/12657 | 343/15822 | 7/180 | 35.95 | 17 | 10.98 |

The performances analysis has been conducted including input and output registers that synchronize the circuit and provide timing performances that are not dependent on the I/O pads.

Table 2.6 shows the performance of the FgBg HW oriented circuit and compares them with the performance of the FgBg BW optimized circuit.

The circuit of Fig. 2.7, implemented on Virtex6, Virtex5, and Spartan6 FPGA without pipeline levels, is able to process 41 fps, 38 fps, and 28 fps respectively. With respect to the FgBg BW optimized circuit, the processing capability is improved of the 46%, 41%, 65% when FgBg HW oriented is implemented on Virtex6, Virtex5, and Spartan6 FPGA, respectively.

Power dissipation and resource utilization are reduced for all the implementations. The decrease of the are utilization is not always evident because, even if the number of DSP is reduced from 7 to 3 for all the implementations, the number of Slices is very close both for FgBg HW oriented and for FgBg BW optimized implementations. In order to better highlight this reduction, both circuits have been implemented on Virtex6 without using DSP slices. Table 2.7 shows that the Slice utilization is reduced of the 24% for the FgBg HW oriented circuit with respect to the FgBg BW optimized circuit.

**Table 2.7:** Performances of the FgBg HW oriented and FgBg BW optimized circuits implemented on Virtex6 FPGA without using DSP. These implementations allow a fair comparison of the logic resource utilization for the two circuits.

| **Target device** | **Pipeline levels** | **Circuit** | **LUT** | **Flip-Flop** | **Slice** | **DSP** |
|---|---|---|---|---|---|---|
| Virtex6 | 0 | FgBg BW optimized | 1815/124800 | 0/249600 | 570/31200 | 0/288 |
| xc6vlx195t | | FgBg HW oriented | 1432/124800 | 0/249600 | 431/31200 | 0/288 |

In order to verify that the proposed formulation of the GMM equations does not affect the quality of the processed video, the resulting frames obtained by using the new formulation have been compared with frames obtained with the algorithm of Section 1.2.1. The results are in Fig. 2.8 and show that the resulting videos are nearly identical and actually differ for few pixels (the average MSE value calculated on the video sequence of Fig. 2.8 is equal to 0.007; the 99.3% of the pixels is correctly classified).

## 2.3 Experimental validation

The FgBg HW oriented of Fig. 2.7 has been experimental validate by implementing it on two different platforms. The first one is equipped with low cost Altera Cyclone II FPGA while the second one is equipped with Virtex4 FPGA. In both cases the circuits work smoothly and correctly identify the background.

### On-line running video system with Cyclone II FPGA

The background identification circuit described in Section 2.5 has been implemented on the Cyclone II EP2C35 FPGA of the DE2 Altera board.

In a first phase the maximum working frequency obtained through the Timing Analysis software has been experimentally verified. Random input data have been fed to the circuit by using proper LFSR (Linear Feedback Shift Register) circuits implemented on the same FPGA. The output data have been verified by using a logic state analyzer. The experiments verified that, has reported by the Timing Analysis software, the Cyclone II implementation correctly works with 35 MHz clock frequency.

In the second phase, the circuit has been integrated in an on-line running video system composed by 5Mp digital camera (D5M camera produced by Terasic), Altera DE2 board, and VGA monitor. The RGB channels coming from the sensor are converted to 8 bit grayscale and fed to the circuit. The experiment demonstrated that the proposed circuit is able to work in an on-line running video system even if a memory bandwidth limitation allows to process only low resolution (320×240) images at 5 fps.

### On-line running video system with Virtex4 FPGA

The FgBg HW oriented circuit has been also implemented on a platform equipped with a Virtex4 VFX12 FPGA that bridges between camera, monitor and SDRAM memory. The platform is equipped with faster RAM and larger FPGA device than previously described platform and allows the design of a system with improved performances.

The color CMOS image sensor captures the video sequence and gives the pixels values in a Bayer pattern format. Video format is 720p (1280×720). A CMOS interface elaborates the CMOS sensor output values and, after a RAW to YCrCb transformation gives the luminance pixel value to the Bg identification circuit.

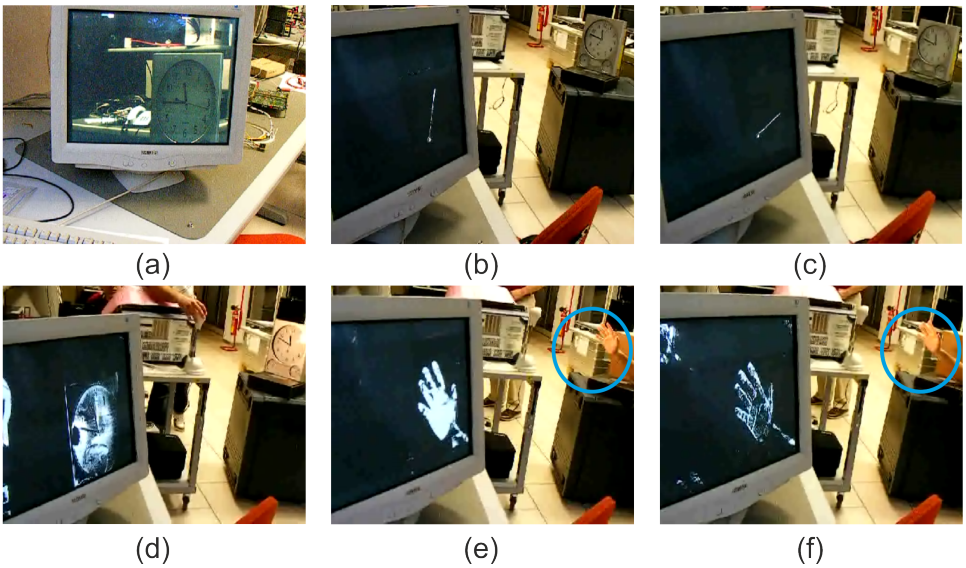


**Figure 2.9:** (a): the camera frames a clock. The picture shows the color video when the background identification circuit is not active. (b) - (c): once the background identification circuit is activated, the only visible object is the moving second hand of the clock. (d): when the illumination changes the whole clock appears on the screen for few instants. (e)-(f): the hand appearing on the scene is identified when it moves and fades when it becomes static.

For each pixel the parameters of the three corresponding Gaussian distributions are stored in off-chip memory (a 100 MHz SDRAM with a 64 bit data bus) while the output video sequences are displayed on the monitor.

Fig. 2.9 shows some frames that demonstrate the functionality of the system. The above described system is able to process 20 fps in the 720p color format.

## 2.4 ASIC implementation

When a large scale implementation or more advanced performances are required, an ASIC implementation of the circuit is needed. To the best of our knowledge, the literature does not report the description of ASIC implementations of the GMM algorithm able to identify moving objects in a video se-

quence. It is therefore very important to know the performances of the algorithm when an ASIC implementation is carried out. As a consequence, the FgBg HW oriented architecture of Fig. 2.7 has also been implemented by using the UMC-90nm standard cell CMOS technology.

The ASIC implementation has been carried out by using Cadence Encounter RTL Compiler. The design has been simulated with NCSim and the Toggle Count File (.tcf) has been generated in order to obtain an accurate estimation of the power dissipation. Similarly to the FPGA implementation, the analysis have been conducted including input and output registers that synchronize the circuits.

The circuit of Fig. 2.7, when implemented using the UMC-90nm technology, runs at 374.5 MHz allowing a processing capability of 180 HD fps. Silicon area occupation and energy consumption are 28847 $\mu m^2$ and 33.2 $pJ/pixel$, respectively.

The obtained frame rate is much higher than what is needed in the majority of the applications. Currently, a HD progressive scan format operating at 50 or 60 fps is being evaluated as a future standard for moving picture acquisition in live broadcast applications [34]. It is therefore possible to take advantage of the higher speed provided by the standard cell technology to save power or silicon area occupation.

The ASIC implementation is hence proposed in two versions. The first version, named in the paper "power aware" ASIC implementation, minimizes the power dissipation while the second one, named in the paper "silicon aware" ASIC implementation, minimizes the area occupation.

Table 2.8 shows the performance of the ASIC implementation of the circuit of Fig. 2.7 and of the proposed power aware and silicon aware ASIC implementations.

The power aware ASIC implementation uses the constant voltage scaling technique to reduce power dissipation. To this purpose, a ring oscillator has been designed and simulated at transistor level with decreasing supply voltage starting from the standard supply voltage of 1.0V. The simulation shows that the circuit of Fig. 2.7, implemented in UMC-90nm with a supply voltage of 0.56V, runs at 125.0 MHz. The obtained power aware circuit processes one pixel per clock cycle and, running at 125.0 MHz, is able to process 60 HD fps.

Table 2.8 shows that, with respect to the straightforward ASIC implementation of the circuit of Fig. 2.11, the resulting power aware ASIC implementation reduces the energy dissipation per pixel (15.3 pJ/pixel) without modifying the silicon area occupation (the constant voltage scaling technique reduces the

**Table 2.8:** Performance of the UMC-90nm ASIC implementation of the circuit of Fig. 2.7, the power aware, and the silicon aware ASIC implementations.

| **Implementation** | **Frequency (MHz)** | **Voltage (V)** | **Cell Area ($\mu m^2$)** | **Energy per pixel (pJ/pixel)** | **fps (1920×)** |
|---|---|---|---|---|---|
| Circuit of Fig. 2.7 | 374.5 | 1.00 | 28847 | 33.2 | 180 |
| Power aware | 125.0 | 0.56 | 28847 | 15.3 | 60 |
| Silicon aware | 374.5 | 1.00 | 21847 | 49.4 | 60 |

power consumption without changing the circuit).

The silicon aware ASIC implementation uses the standard supply voltage of 1.0V and modifies the circuit of Fig. 2.7 in order to process 60 fps while reducing silicon area occupation. Some circuital units of Fig. 2.7 have been modified in order to process one pixel in three clock cycles. The modified units are: *Match*, *Learning Rate*, *Weight*, and *IFitness*. Table 2.9 reports the area breakdown of the ASIC implementation of the circuit of Fig. 2.7 and of the silicon aware circuit, showing that the area occupied by the blocks that have been shrunk is more than 50% of the total area.

Table 2.9 shows also the introduction of various registers in the silicon aware circuit. The registers store the partial results of the updating equations during the first and the second clock cycle of the three clock cycles needed to process one pixel (two clock cycles are needed to process the parameters of the first and second Gaussian while a further clock cycle processes the parameters of the third Gaussian and provides the output data of the circuit). The silicon aware circuit also includes an internal enable signal that freezes the circuital units that, during the first two clock cycles, are not ready to produce useful results. The effect of the enable signal is a reduction of power dissipation.

With the above described modifications, the area utilization of the silicon aware ASIC implementation, including the synchronization registers, is reduced by 24% (21847 $\mu m^2$ versus 28847 $\mu m^2$).

The energy consumption of the proposed silicon aware ASIC implementation is 49.4 $pJ/pixel$ (Table 2.8). The increase in energy consumption of the silicon aware circuit with respect to the ASIC implementation of the circuit of Fig. 2.7 is mainly due to the additional registers that store the partial results of the updating equations.

**Table 2.9:** Comparison between the area utilization of the UMC-90nm ASIC implementation of the circuit of Fig. 2.7 and of the silicon aware ASIC implementation.

| | **Circuital Unit, Area ($\mu m^2$)** | | | | | | |
|---|---|---|---|---|---|---|---|
| | **Match** | **Weight** | **Learning Rate** | **IFitness** | **FF** | **Others (Mean, Variance, ... )** | **Tot** |
| **Circuit of Fig. 2.7** | 10966 | 811 | 647 | 2637 | 4316 (Synch.) | 9470 | 28847 |
| **Silicon aware** | 3515 | 270 | 248 | 873 | 6663 (Synch. and Folding) | 10278 | 21847 |
| **Comparison** | -68% | -67% | -62% | -67% | +54% | +8% | -24% |

Table 2.8 shows no comparisons with others works since, to the best of our knowledge, in the literature there are no ASIC implementations of the GMM algorithm able to identify moving objects in a video sequence.

## 2.5 Optimized architecture

The performances of the FgBg HW oriented circuit can be further improved by:

- replacing some full-width binary multipliers with truncated binary multipliers, [35]-[38];
- approximating non linear functions with piecewise linear functions, [39]-[43];

FgBg HW oriented and FgBg BW optimized circuits implement the match condition by using equation (2.3). This allows to avoid the implementation of a circuit for the computation of the square root of the variance signal but entails the implementation of a multiplier (for the left side of the equation). It is possible to obtain an improved implementation of the match condition by using equation (1.2) and by implementing the square root function with a piecewise linear approximation technique. This technique has been previously proposed as an effective method to reduce hardware complexity, [44]-[45].

The range of variation of $\sigma^2_{k,t}$, [8:16376], is divided, as shown in Fig. 2.10, into four non uniform intervals, $I_1, \cdots, I_4$. In each interval, a linear function $y_i$ approximates $\sigma_{k,t}$ as:

$$y_i = q_i - m_i \cdot \sigma^2_{k,t} \qquad \sigma^2_{k,t} \in I_i \tag{2.9}$$

with i=1,...,4 and $q_i$, $m_i$>0. In order to simplify the hardware, the $m_i$ coefficients are quantized as powers of two and the multiplication between $m_i$ and $\sigma^2_{k,t}$ is performed by using a shifter resulting in lower area utilization and higher working frequency.

A similar approach can be used also to implement the inverse function needed to update the weight when no Gaussians match the pixel (see equation (1.7)). Both FgBg HW oriented and FgBg BW optimized circuit uses a ROM-based approach to implement the required function.

In the proposed optimized implementation, named in the following FgBg optimized circuit and whose block diagram is shown in Fig. 2.11, the ROM

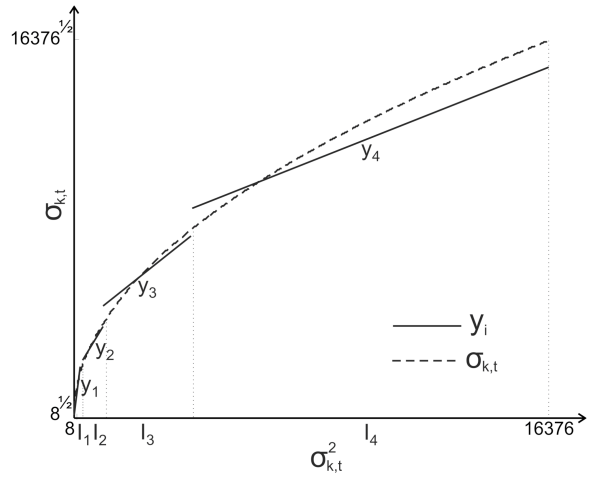


**Figure 2.10:** Piecewise linear polynomial approximation of the square root of the $\sigma^2_{k,t}$ signal.

is eliminated by using a piecewise linear approximation technique similar to those used for the calculation of the $\sigma_{k,t}$ signals. The *msumtot* signal range [1:63] is divided in four non uniform intervals $J_i$ and in each interval, a function $z_i$ approximates *1/msumtot* as:

$$z_i = \lambda_i - \eta_i \cdot msumtot \qquad msumtot \in J_i \tag{2.10}$$

with i=1,...,4, $\lambda_i$, $\eta_i$>0, and $\eta_i$ quantized as power of two so that the multiplication between $\eta_i$ and *msumtot* is performed using a shifter.

A truncated multiplier has been instead used to implement the left-hand of the variance update equation of (1.3).

Truncated binary multipliers, for which the optimality in terms of mean square error is analytically demonstrated, have been proposed in [33]-[38]. Truncated multipliers have been used with good results in a variety of applications, [44]-[45]. A full-width digital $n \times n$ bits multiplier computes the $2n$ bits output as a weighted sum of partial products. A truncated multiplier is an $n \times n$ multiplier with an $m < 2n$ bits output.

In the FgBg optimized circuit the inputs of the multipliers are represented on 12 bits. The output of a 12×12 full-width multiplier is hence on 24 bits. In order to reduce HW complexity and improve circuit speed, the circuit implements a truncated binary multiplier, based on the architecture proposed in

**Figure 2.11:** Block diagram of the FgBg optimized circuit that implements the optimized hardware oriented formulation of the GMM described in Section 2.2 and uses truncated binary multipliers and ROM approximation techniques.

[38], in which the output is directly produced on 15 bits (9 less-significant bits are discarded with an almost 40% reduction of the required HW for the multiplier).

### 2.5.1 FgBg optimized circuit - FPGA implementation

The FgBg optimized circuit has been implemented on Virtex6, Virtex5, and Spartan6 FPGA. The performances are shown in Table 2.7.

The utilization of truncated multipliers, and the implementation of ROM approximation techniques, could worsen the accuracy of the processed video streams. Figure 2.12 shows a selection of frames with resolution $320\times240$ extracted from two video sequences. The original frames of Figure 2.12(a) and Figure 2.12(d) have been processed by using the FgBg optimized implementation (Figure 2.12(b) and Figure 2.12(e)) or by using the FgBg HW orientede circuit that uses full-width multipliers and no ROM compression techniques (Figure 2.12(c) and Figure 2.12(f)). The background masks of Figure 2.12(b),(e) and Figure 2.12(c),(f) differ in few pixels (in both the video sequences less than the 1% of the total number of the pixels is differently classified by the two circuits). This demonstrates that the proposed implementation provides negligible effect on the quality of the processed video streams.

## 2.6 Hardware resources versus video quality

The GMM algorithm models the statistic of each pixel of the video sequence with a mixture of K Gaussian distributions. The number of Gaussians per pixel determines the accuracy of the background model but influences also the computational complexity of the algorithm and the performances of the HW implementation. The accuracy of the model and the computational complexity grow with K while the circuit performances are worsened. As a matter of fact power dissipation, logic resource occupation, and the required bandwidth to the memory are increased while circuit speed decreases.

Stauffer and Grimson, in [16], suggest a model that employs from three to five Gaussians. The FgBg BW optimized, FgBg HW oriented, and FgBg optimized circuits implement the Gaussian Mixture Model algorithm by using three Gaussian distributions for each pixel (k=[1,2,3]). The choice of three Gaussians provides good quality of the processed images while limiting circuit complexity and memory requirements.

In order to verify that implementing the algorithm by using three Gaussians

**Table 2.10:** Performances of the FgBg optimized circuit implemented on Virtex6, Virtex5, and Spartan6 FPGA.

| **Target device** | **Pipeline levels** | **LUT** | **Flip-Flop** | **Slice** | **DSP-Mult** | **Frequency (MHz)** | **fps (1920×1080)** | **Power dissipation (mW/MHz)** |
|---|---|---|---|---|---|---|---|---|
| Virtex6 xc6vlx195t | 0 | 1094/124800 | 0/249600 | 301/31200 | 0/640 | 119.16 | 57 | 0.77 |
| Virtex5 xc5vlx50 | 0 | 844/28800 | 0/28800 | 301/7200 | 0/48 | 91.03 | 43 | 1.01 |
| Spartan6 xc6slx100 | 0 | 1188/63288 | 0/126576 | 442/15822 | 0/180 | 60.85 | 29 | 1.30 |

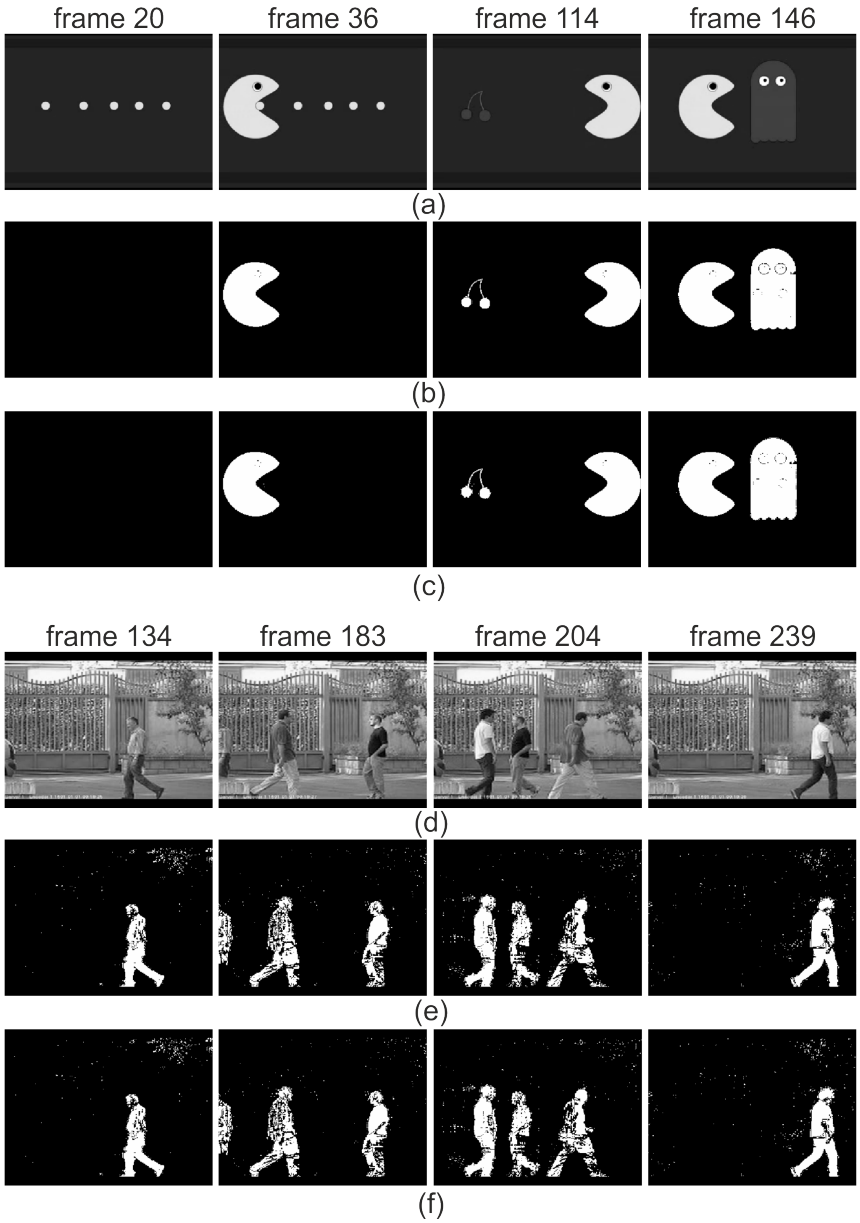


**Figure 2.12:** Comparison between the background masks obtained with the FgBg optimized circuit and with the FgBg HW oriented implementation. (a),(d): unprocessed video frames; (b),(e): output of the FgBg optimized circuit; (c),(f): output of the FgBg HW oriented circuit.

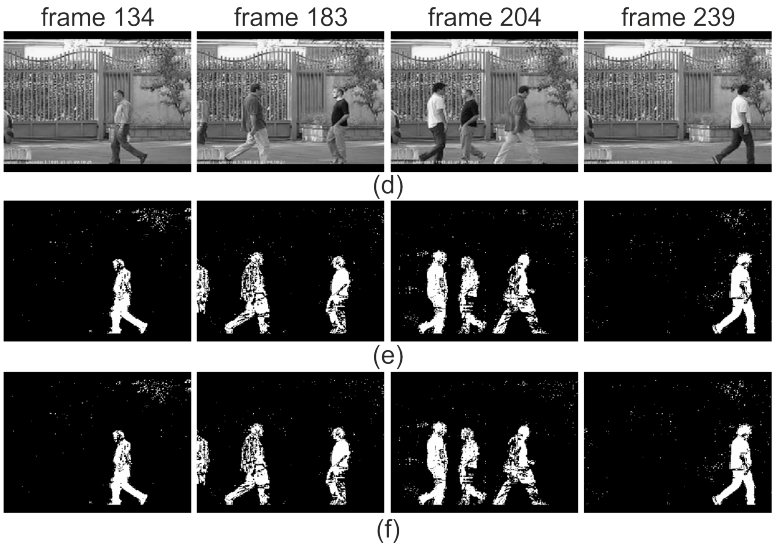


**Figure 2.13:** Examples of video sequences processed with the proposed circuits: (a),(d) original frames; (b),(e) frames obtained with the GMM HW implementation that uses three Gaussian distributions per pixel; (c),(f) frames obtained with the GMM HW implementation that uses five Gaussian distributions per pixel

per pixel allows a good trade-off between hardware complexity a video quality, the FgBg optimized circuit has been implemented on Virtex6 (xc6vlx195t) Xilinx FPGA by using three and five Gaussians per pixel. The synthesis have been carried out by using XST and fitting and Place&Route have been carried out by using ISE. ModelSim PE has been used to perform the circuits simulations.

The performances of the proposed circuits are in Table 2.11 while Fig. 2.13 compares the outputs of the two circuits.

The circuit that uses three Gaussians per pixel is able to process 38 HD frames per second (fps) by using less than the 3% of FPGA resources. When five Gaussians distributions are used the maximum working frequency is decreased by 13% (the maximum frame rate for 1080p video is 34 fps). The logic resource utilization is increased by 79%. With reference to the bandwidth towards the memory, being the Gaussian data the biggest load on data bandwidth, the circuit that uses five Gaussians increases the bandwidth by 66%.

Figure 2.13 reports some frames of a video sequence processed with the

**Table 2.11:** Performances of the FgBg optimized circuit implemented on Virtex6 (xc6vlx195t) FPGA by using three and five Gaussian distributions per pixel.

| Gaussians per pixel | LUT | Slice | DSP | Frequency (MHz) | fps (1920×1080) | Bandwidth Gbit/s |
|---|---|---|---|---|---|---|
| 3 | 1094/124800 | 346/31200 | 0/640 | 119.16 | 57 | 0.40 |
| 5 | 2070/124800 | 700/31200 | 0/640 | 90.33 | 43 | 0.65 |

circuit that use three Gaussians (Fig. 2.13(b)) and with the circuit that use five Gaussians per pixel (Fig. 2.13(c)). People called to provide a subjective assessment of video quality refer of a similar video quality and a reduced difference between the binary images obtained with the two circuits. A more accurate comparison between the frames can be obtained by computing the Peak Signal to Noise Ratio (PSNR) defined as in equation (2.1), where, for each video, *Iref*, that in this case is considered as the ground truth, has been calculated using a software implementation of the OpenCV GMM algorithm using double precision floating point numbers.

The second video sequence (Fig. 2.13(a)) has resolution and lenght equal to 120×160 pixels and 800 frames. The PSNR values are 31.4 and 31.7 when three or five Gaussians per pixel are used. The percentage of pixel differently classified is 0.05%.

The analysis of the video quality and of the HW performances shows that the circuit that uses three Gaussian distributions per pixel, provides a substantial improvement of circuit performances while not showing significant variations in the identification accuracy. The preliminary results shown here suggest that an accurate evaluation of the precision of the circuit and of the equation is more effective than the increase of the number of Gaussians when the quality of the processed video is the target.

# Chapter 3

# Denoising

The background identification algorithms described in the Chapter 1 process video sequences in order to detect objects supposedly belonging to the foreground. The input data of these algorithms is a grayscale or RGB video sequence. The output is a video sequence composed by binary images in which a pixel is represented with one bit whose value is equal to '0' if the pixel is classified as background or '1' if the pixel is classified as foreground (or viceversa).

The binary mask provided by the segmentation circuit is usually noisy with foreground objects that are often split into several parts. In order to remove the noise and enhance the appearance of the binary images, the identification is usually followed, as shown in Fig. 3.1, by a denoising phase that can be performed by using the operators of the mathematical morphology, [48].

The mathematical morphology, [49], [50], is derived from the set theory and provides powerful tools for geometrical image analysis, image and video compression, error correction, noise suppression and video segmentation.

Morphology operators can be employed to process binary, gray-scale and color images, [51]-[52].

The architectures that implement morphological operators can be divided into two large classes: systolic arrays, [53], [54] and pipelined systems, [55].

This Chapter proposes a denoising circuit belonging to the second class and based on morphological operators.

The circuit processes the images pixel by pixel: it acquires the input and produces the output in raster scan order. Then, each pixel is processed by using the morphological operators. The output value depends on the pixel to be processed and the pixels in a neighborhood. A delay-line based architecture

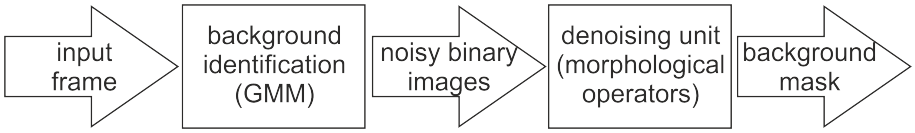


**Figure 3.1:** Schematic view of a system that performs background identification and denoising of video sequences. The input video is processed by the background identification unit that produces a video sequence composed of binary images. The output video sequence is usually noisy and is processed by the denoising unit.

correctly aligns the pixels in order to process them. Its hardware implementation has been designed in order to minimize the hardware resources utilization. Moreover, a particular choice for the processing of the boundary pixels has allowed to obtain a stall-free hardware architecture (the circuit processes one pixel in each clock cycle) able to process also the boundary pixels.

The overall system that performs both the identification phase and the denoising, when implemented on Virtex6-vlx195t FPGA, is able to process 57 fps by using the 1.4% of the logic resources of the FPGA.

This Chapter is organized as follows. In Section 3.1 the morphological operators are described while Section 3.2 details the proposed hardware implementation and its performance.

## 3.1 Morphology

Mathematical morphology is a set of mathematical operators, based on the set-theory, used to manipulate the shape or analyze the structure of connected clusters of pixels [49], [50]. The common usage includes edge detection, noise removal, image reconstruction, and image segmentation. The operators were originally developed for the analysis of binary images but they have been extended and applied to gray scale, [51], [56], and color images, [52].

Erosion and dilation are the two most basic operators. They take as input the image ($I$) to be eroded or dilated and a structuring element ($SE$) that determines the effect of the operators on $I$. The $SE$ can be viewed as a small binary image, represented as a set of pixels on a grid, assuming the values 1 if the pixel belongs to $SE$ and 0 otherwise (Fig. 3.2).

An origin of the $SE$ must be identified. When a morphological operation is carried out, the $SE$ is superimposed on $I$ so that its origin coincides with

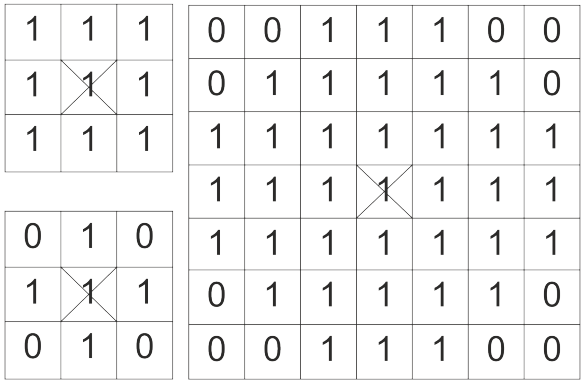


**Figure 3.2:** Examples of SE. The origin (crossed element) is in the center of the SE.

the input pixel. The origin of the *SE* is shifted on *I*, the points within *SE* are compared with the underlying image and a decision is taken based on the comparison.

The effect of erosion on *I* is to erode the boundaries of foreground regions. Thus the area of the foreground regions decreases and holes within the foreground objects become larger. To compute the erosion, the origin of *SE* is superimposed on each pixel in *I*. If, for every pixel in *SE*, any of the corresponding pixels in *I* belongs to the background, the input pixel is also set to the background value, otherwise it remains unchanged.

The effect of the dilation on *I* is to enlarge the boundaries of foreground objects. Thus the area of foreground regions grows while the holes within those objects become smaller. To compute the dilation, the origin of *SE* is superimposed on each pixel in *I*. If at least one pixel in *SE* coincides with a foreground pixel in the image underneath, the input pixel is set to the foreground value, otherwise it remains unchanged.

Fig. 3.3 illustrates the effect of the erosion (Fig. 3.3(c)) and of the dilation (Fig. 3.3(d)) on a binary image (Fig. 3.3(a)) with a cross shaped $3\times3$ *SE* (Fig. 3.3(b)). In Fig. 3.3 the foreground regions are represented with white pixels (binary value 1), while black pixels (binary value 0) denote the background. The effect of the erosion on a single foreground pixel surrounded by the background pixels, is the removal of the foreground pixel. In Fig. 3.3(c) one row and one column are removed from each side of the square of Fig. 3.3(a). The effect of the dilation on a single foreground pixel surrounded by the back-

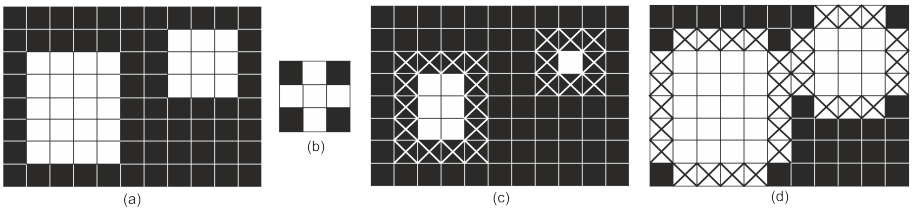


**Figure 3.3:** (a) Original foreground mask; (b) structuring element with a cross shape in which the origin element is at the center of the cross; (c) eroded mask; (d) dilated mask. Crossed pixels in (c) are the eroded pixels. Crossed pixels in (d) are the dilated pixels.

ground pixels, is the transformation of the foreground pixel in a cross equal to the *SE*. In Fig. 3.3(d) one row and one column are added on each side of the square of Fig. 3.3(a).

It is worth highlighting that erosion and dilation are dual operators and eroding foreground pixels is equivalent to dilating the background pixels. Dilation and erosion operators are indicated with the $\oplus$ and $\ominus$, respectively. The combination of dilation and erosion forms other morphological operations such as the opening, denoted with $\circ$, and the closing, denoted with $\bullet$:

$$\begin{aligned} I \circ SE &= (I \ominus SE) \oplus SE \\ I \bullet SE &= (I \oplus SE) \ominus SE \end{aligned} \tag{3.1}$$

The basic effect of an opening is somewhat like erosion in that it tends to remove some of the foreground pixels from the edges of regions of foreground pixels. However it is less destructive than erosion in general. The effect of the operator is to preserve foreground regions that have a similar shape to this structuring element, or that can completely contain the structuring element, while eliminating all other regions of foreground pixels.

Closing is similar in some ways to dilation in that it tends to enlarge the boundaries of foreground regions in an image. Its effect is to preserve background regions that have a similar shape to this structuring element, or that can completely contain the structuring element, while eliminating all other regions of background pixels.

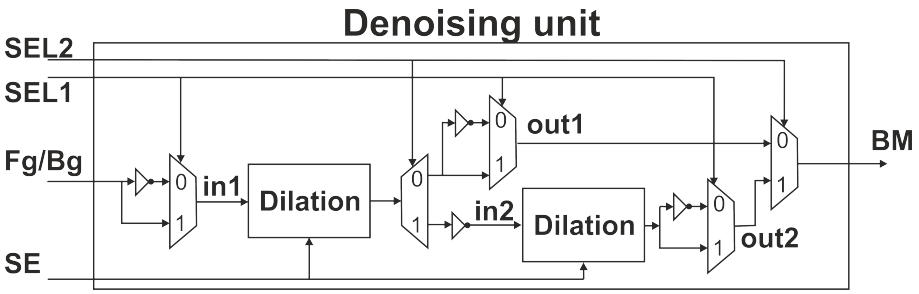


**Figure 3.4:** Circuit that implements the denoising unit of Fig. 3.1 by using the morphological operators.

**Table 3.1:** Operators performed by the denoising unit of Fig.3.4 as a function of the signals SEL1 and SEL2.

| SEL1 | SEL2 | Operation | In1 |
|---|---|---|---|
| 0 | 0 | Erosion | Not(Fg/Bg) |
| 1 | 0 | Dilation | Fg/Bg |
| 0 | 1 | Opening | Not(Fg/Bg) |
| 1 | 1 | Closing | Fg/Bg |

## 3.2 FPGA implementation of morphological operators

The denoising unit of Fig. 3.1 has been designed in order to implement erosion, dilation, opening and closing operations. An expanded view of this unit is shown in Fig. 3.4.

The noisy binary images obtained with the identification circuit are processed by the denoising unit pixel by pixel. As a consequence, the input signal of the circuit are the *Fg/Bg* tag (see Fig. 2.7), the *SE* and two control signals (*SEL1* and *SEL2*). The output is the background mask. *SEL1* and *SEL2* are two binary signals that establish the operation to be performed according to Table 3.1. Opening and closing can be derived by erosion and dilation as in (3.1). Moreover, the erosion and dilation operations are one the dual of the other. It is therefore possible to obtain the erosion of the image $I$ by using the dilation operator if the input image $I$ and the result of the dilation are inverted. As a consequence all the considered morphological operators can be implemented using only the dilation operator.

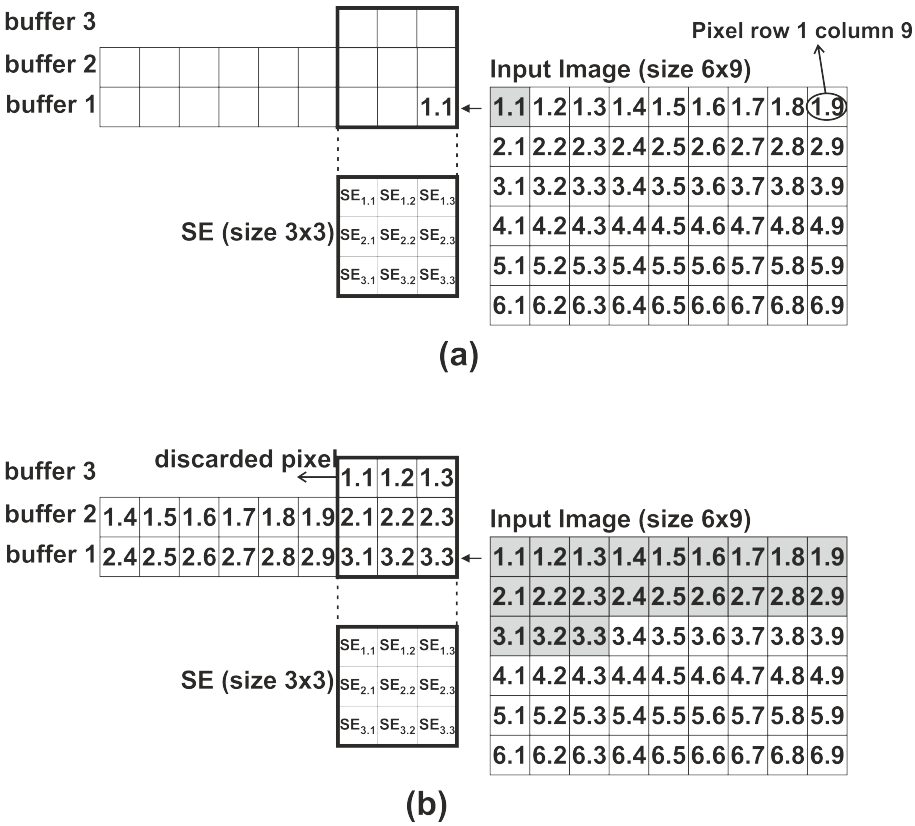


**Figure 3.5:** Example of delay lines used to perform morphological operations. In (a) the Input image is shifted into three buffers (SE size $3\times3$) whose width depends on the image size (6x9). When the buffers are full (b), the first pixel is discarded.

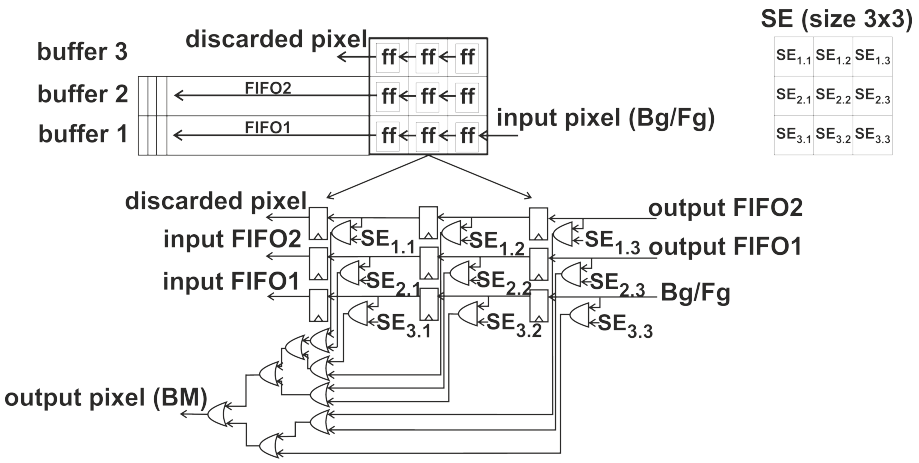


**Figure 3.6:** Delay lines implementation with a 3×3 SE.

$$\begin{aligned} I \ominus SE &= (I' \oplus SE)' \\ I \circ SE &= (I' \oplus SE)' \oplus SE \\ I \bullet SE &= ((I \oplus SE)' \oplus SE)' \end{aligned} \tag{3.2}$$

The circuital unit that implements the operator dilation is indicate as "Dilation" in the block diagram of Fig.3.4. The figure shows that the Dilation unit is duplicated. This allows to implement closing and opening operations without storing the intermediate results.

Depending on the *SEL1* value, *Fg/Bg* tag or its binary inversion are sent as input to the Dilation unit (Table 3.1). The *Fg/Bg* tag is a binary value: '0' if the pixel has been classified as background; '1' otherwise.

The Dilation unit is based on a delay line architecture. The input binary values *Fg/Bg* are stored in some buffers as shown in Fig. 3.5. If *SE* size is equal to m×n, m buffers are employed. The length of the buffers depends on the image size. If the *I* size is i×j, 1 buffer n bit long and m-1 buffers j bit long, are needed. When all the shift registers are full (Fig. 3.5(b)), as an incoming pixel is shifted in, the oldest pixel is shifted out.

The delay lines have been designed in order to minimize the hardware resources utilization. The pixels covered by the *SE* are stored using flip flops while the other pixels are stored in FIFOs (Fig. 3.6). The FIFO can be implemented by using sequential elements or LUT. Table 3.2 shows the resources

**Table 3.2:** Performances of Denoising unit implemented on Virtex5 FPGA. Frame size 1920×1080. The FIFO are implemented by using Flip Flop or LUT.

| Circuit | FIFO implementation | Frequency (MHz) | Slice Register | Slice LUT | Slice |
|---|---|---|---|---|---|
| Dilation | Flip Flop | 374.25 | 1.946 | 227 | 542 |
| Dilation | LUT | 432.71 | 30 | 173 | 51 |
| Denoising unit | Flip Flop | 263.37 | 3898 | 494 | 1078 |
| Denoising unit | LUT | 341.53 | 56 | 361 | 106 |

utilization and the maximum working frequency of the Dilation unit and of the overall denoising circuit when flip-flop or LUT are used to implement the FIFO. It is worth noting that the number of Slices used by implementing the FIFO with LUT is very lower than the number needed to implement the FIFO as shift-registers. As a consequence the FIFO has been implemented by using the LUT.

The dilation is performed on the pixels stored in the flip flops using the logic shown in Fig. 3.6.

When designing a morphological hardware unit, an obstacle is to process the boundary pixels of the frames. The implementation of the morphological operators at the boundary of a frame is different from those in the interior of the image, Fig. 3.7(a). Usually, two solutions are adopted. The first one is to add extra control logic to avoid the boundary pixel processing. The second one is the padding technique, Fig. 3.7(b) that increases the frame size. Additional pixel are inserted outside the boundary of $I$. Since they should not affect the result of the operations, the padding area is filled with '1' for the erosion and '0' for the dilation. Padding increments the output delay and, more important, fragments the dataflow. In this paper a technique that combines the two methods is adopted. It allows the processing of the boundary pixels without stopping the dataflow. In the proposed technique a control logic is activated when a boundary pixel has to be processed and fixes some nodes, originally connected to the output of registers, at logic value '0'. The required logic consist in two counters and some comparators to identify the boundary pixel. This entails an increase of circuit area but does not increase the output delay that can be very high when HD videos are processed. The same counters are used to synchronize the two Dilation units when opening and closing are performed.

It is worth noting that the SE is an input for the proposed circuit. This allows to change its shape of up to 3×3 pixels in size.

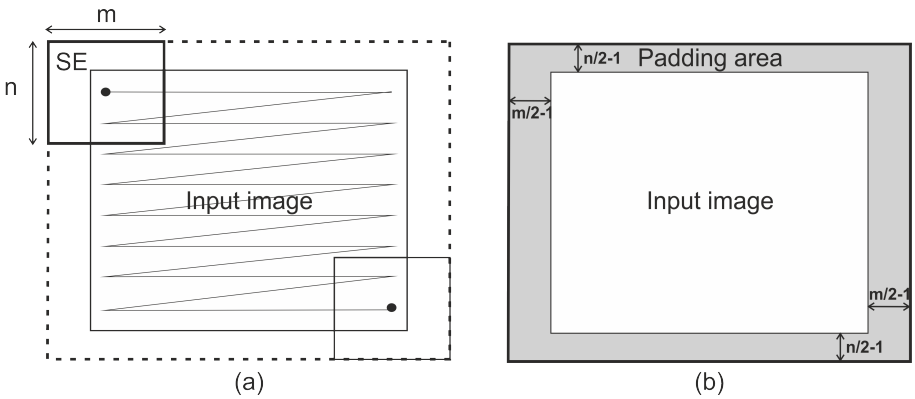


**Figure 3.7:** (a) Boundary problem. SE stretches outside the image borders. (b) Frame padding necessary for an SE whose size is m×n.

## 3.3 Results and performances

The proposed circuit has been implemented on Virtex6 and Virtex5 Xilinx FPGA. The synthesis has been conducted by using XST. Circuit simulations use ModelSim XE. The simulation tools also generate the 'vcd' files necessary for the accurate determination of the power dissipation. Power dissipation has then been computed using XPower Analyzer software.

The analyses have been conducted including input and output registers that synchronize the circuit and provide timing performances that are not dependent on the I/O pads. The circuit has been tested using artificial videos, computer animated videos with simple backgrounds and through video sequences taken from real surveillance cameras. The circuit performs optimally and run smoothly without showing reliability problems.

Table 3.3 shows the performances of the proposed circuit as a function of the target FPGA.

As shown in Table 3.3, the circuit that performs both the identification and the denoising, when implemented on Virtex6 and Virtex5 FPGA, without pipeline, has a maximum working frequency of 119.16 MHz and 91.03 MHz, respectively. The maximum working frequency is imposed in both cases by the FgBg circuit. The overall system uses the 1.4% and the 5.7% of the Slices on Virtex6 and Virtex5, respectively. In both cases only a low number of Slices is used from the denoising unit (the 31% on Virtex6, and the 26% on Virtex5).

In order to test the correct behaviour of the system, several video sequences

**Table 3.3:** Performances of the identification and the denoising circuits implemented on Virtex6 (xc6vlx195t) and Virtex5 (xc5vlx50) FPGA. The identification phase is carried out by using the FgBg optimized circuit described in the Chapter 2. The denoising is performed by using the circuit of Fig. 3.4. The table shows a breakdown of the logic resources of the two circuital units.

| FPGA | Circuital unit | Pipeline levels | LUT | Flip-Flop | Slice | DSP | Frequency (MHz) |
|---|---|---|---|---|---|---|---|
| Virtex6-vlx195t | FgBg | 0 | 1094/124800 | 0/249600 | 301/31200 | 0/640 | 119.16 |
| | Denoising | 0 | 216/124800 | 58/249600 | 133/31200 | 0/640 | 405.68 |
| | FgBg+Denoising | 0 | 1310/124800 | 58/249600 | 434/31200 | 0/640 | 119.16 |
| Virtex5-vlx50 | FgBg | 0 | 844/28800 | 0/28800 | 301/7200 | 0/48 | 91.03 |
| | Denoising | 0 | 361/28800 | 56/28800 | 106/7200 | 0/48 | 341.53 |
| | FgBg+Denoising | 0 | 1205/28800 | 56/28800 | 407/7200 | 0/48 | 91.03 |

have been processed. Fig. 3.8 shows an example of frames on which Erosion, Dilation, Opening and Closing operations have been performed.

The frames of Fig. 3.8(a) are the output of the FgBg optimized circuit described in Chapther 2. Figure 3.8(b) shows the effect of the erosion. It removes the noise but also some pixels belonging to the moving objects. The dilation (Fig. 3.8(c)) can be used to obtain more compact objects. It can be used, for example, in order to obtain the blobs in detection and tracking applications, [57]. However, the dilation highlights the noise. Opening and closing (Fig. 3.8(d) and Fig. 3.8(e), respectively) allow, in this case, to obtain better results both for the filtering (Fig. 3.8(d)) and for the blob creation (Fig. 3.8(e)).

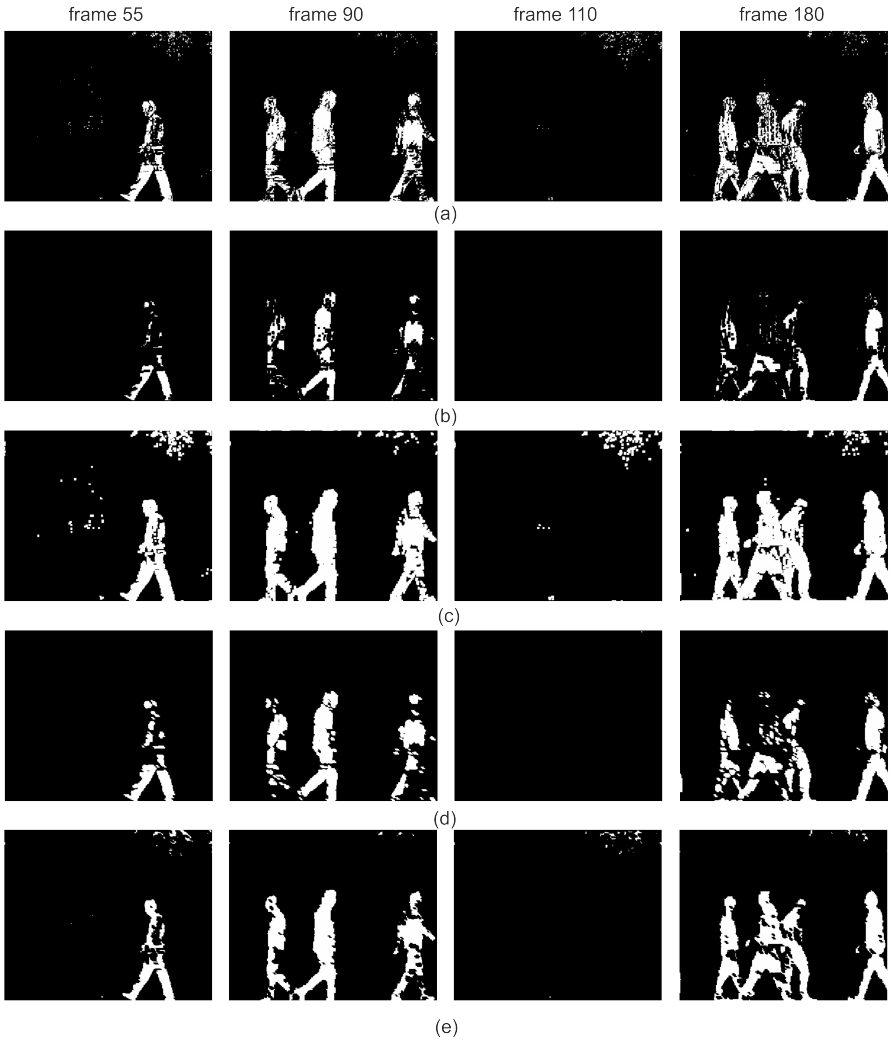


**Figure 3.8:** Results of (b) erosion, (c) dilation, (d) opening, (e) closing on the frames shown in (a) that are the output of the identification circuit of Fig. 2.7.

# Chapter 4

# X-ray fluoroscopy: noise modeling and denoising

Fluoroscopy is a widely used technique in clinical environment for image-guided surgery and therapy.

Real-time X-ray screening of patients allows to support specific surgical procedures, such as angiography, angioplasty, pacemaker and defibrillator implantation, orthopedic surgery, etc ([58], [59]), by tracking in real-time surgical instruments, catheters, and wire guides inside patient's body.

The fluoroscopy also helps as diagnostic tool in investigations of the gastrointestinal tract, blood vessels, assessment of joint, [60], implanted prosthesis [61] and [62].

The duration of the fluoroscopy application obviously depends on the clinical requirements of the specific procedure but, in general, the X-ray exposure may be protracted for a long time. To keep patient's radiation dose acceptably low, the number of X-ray photons is strongly reduced and image intensifiers or flat panel detectors have to be used to amplify radiations and form the fluoroscopic image, Fig. 4.1.

The limited availability of photons per pixel generates the so called quantum noise. Quantum noise is by far the most dominant noise in fluoroscopic images whereas other noise sources (e.g. thermal noise, video-system noise, quantization error, ecc) can be generally neglected, [63]-[66]. The recent progresses in flat panel technology [67],[68] have improved sensitivity, resolution, and system lag with respect to image intensifiers. However, the limitation of the quantum noise still remains, because it is inherent to the image formation process and not to the specific sensor. Because of the relatively high amount of

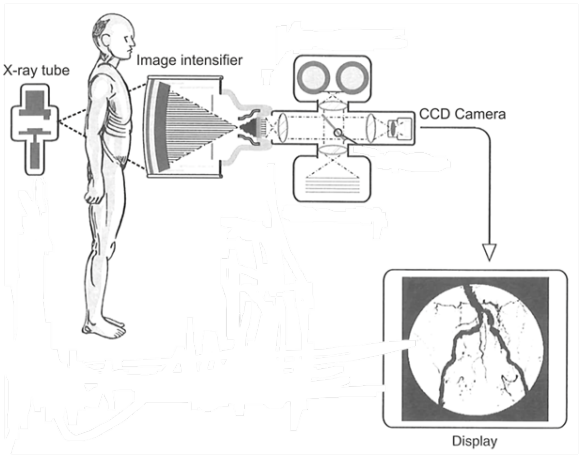


**Figure 4.1:** Scheme of a fluoroscopic system. The X-ray tube generates the radiation needed to obtain the image of the body structure. The radiation is amplified by using an image intensifier. The obtained images are displayed on a monitor.

noise, fluoroscopic image restoration is essential to recover poor constrained details and to improve image quality and in turn the visual perception of the doctors during the operation.

Quantum noise is a signal dependent Poisson distributed noise source [69]-[77], its strength varies over the image depending on the local grey-level intensity.

Different denoising methods have been proposed to filter Poisson noise such as total variation regularization and/or wavelets [78]-[85] and patch-based sparse priors such [86] or the remarkable BM3D [87] algorithm. Unfortunately, these denoising methods are computationally complex and therefore hardly suited for real-time processing of fluoroscopic images (sequence of large images at a frame rate of 30 fps or more). Commercial fluoroscopic devices usually only involve simple temporal and/or spatial average filtering [69],[70], which nevertheless generates motion blur and smears edges. More advanced algorithms include object or motion detection [88]-[90].

Very recent techniques aim to stabilize the Poisson's noise variance (e.g. via the Anscombe transformation), then filter the noise as additive, white and Gaussian and finally antitransform the filtered image [91]-[93]. These methods do not allow an easy real-time implementation, too.

Recently, a simple (suitable for a real-time implementation) and efficient filter technique [94], [95] was proposed for quantum noise suppression in fluoroscopic image sequences. The algorithm of [94], [95] considers a preliminary estimate of the relationship between the pixel gray-level and the correspondent standard deviation of the noise at that luminance, relatively to the actual fluoroscopy device setup.

It is worth noting that if the variance is large enough, as largely verified in actual fluoroscopic application, the Poisson distribution can be locally approximated by a Gaussian distribution N(0,$\sigma^2$), with its variance $\sigma^2$ proportional to the local luminance. By holding this information, noise suppression can be exclusively performed by averaging the only local data that have high probability to be included in the noise statistics (luminance-dependent Gaussian model). Taking into account the local intensity of the noise can be regarded as a sort of equalization or stabilization of noise variance.

The filter operates both in the space and in the time, preserving edges and motion. Despite its simplicity, the algorithm offers performance comparable to those of much more complex algorithms (e.g. BM3D) in terms of peak-signal-to-noise ratio (PSNR), signal-to-noise ratio (SNR), mean square error (MSE), structural similarity index (SSIM) [95].

It is important to underline that usually fluoroscopic devices apply a non-linear transformation of the image gray levels [94] in order to compensate the exponential attenuation of X-rays (e.g white compression, gamma correction). These transformations, generally determine an expansion of contrast for darker pixels and a compression for those brigther. This results in a modification of the noise statistic.

Section 4.1 describes the fluoroscopic noise model while the main fluoroscopic filtering techniques are described in Section 4.2

## 4.1 Fluoroscopic noise modeling

Fluoroscopy is commonly used during clinical applications to support many surgical interventions and a variety of diagnostic procedures. In order to avoid risks for the patient health, X-ray intensity has to be kept acceptably low during the clinical applications. At low exposure levels, the number N of photons emerging from a patient under a fluoroscopy system and detected at the position $r = [x, y]^T$ can be described by a Poisson distribution, with probability mass function given by:

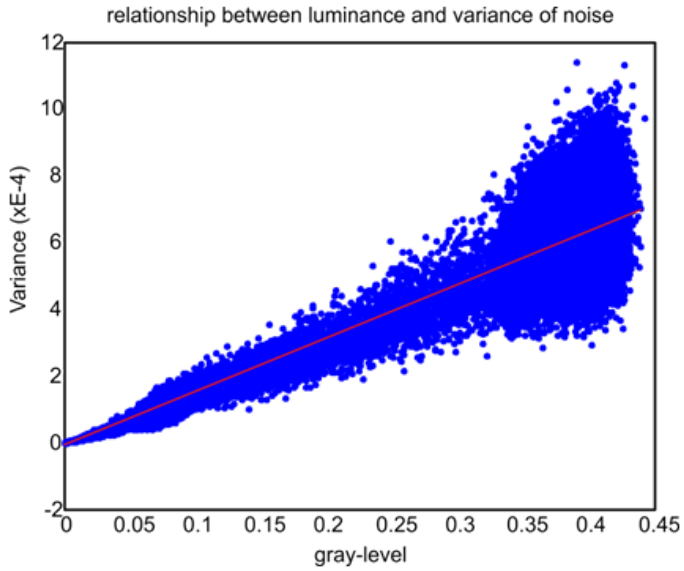


**Figure 4.2:** Relationship between luminance and variance noise for a fluoroscopic sequence obtained with a step phantom. The solid red line represents the estimated linear mean-variance characteristic.

$$P_N(N(r)) = \frac{\lambda(r)^{N(r)} e^{-\lambda(r)}}{N(r)!} \tag{4.1}$$

where $\lambda >> 1$ is the expected number of photons in an interval of time that depends on the fluoroscope frame rate. The mean and variance of the Poisson distribution are equal to:

$$E[N(r)] = var[N(r)] = \lambda(r) \tag{4.2}$$

Generally, pixel intensity ($I$) is linearly dependent on the number of detected photons:

$$I(r) = g_d \dot{N}(r) \tag{4.3}$$

where $g_d$ is a positive constant representing the detector gain that depends on the characteristics of the fluoroscope. As a result, the pixel intensity can be modeled as Poisson-distributed. Reference [71] shows that for $\lambda > 10$,

the Poisson distribution can be locally approximated with an additive Gaussian noise with zero-mean and signal-dependent variance. Therefore, pixel intensity can be locally decomposed as the summation of the expected pixel intensity (*pix*) plus a zero-mean signal-dependent noise component:

$$I(r) = pix(r) + G(r) \tag{4.4}$$

where G is:

$$G \sim N(0, \sigma_G^2(pix)) \tag{4.5}$$

As a consequence, the mean of the image intensity *I* results equal to

$$E[I(r)] = pix(r) = g_d E[N(r)] \tag{4.6}$$

and the variance is equal to

$$var[I(r)] = var[G(r)] = g_d^2 var[N(r)] = g_d^2 E[N(r)] = g_d E[I(r)] \tag{4.7}$$

As a consequence, the variance noise linearly depends on the expected pixel intensity and is strongly signal-dependent.

Figure 4.2 shows the noise variance as a function of the pixel intensity (gray-level) for a video sequence obtained with the fluoroscopy of a step phantom. The video sequence is composed of about 100 frames with resolution $1024 \times 1024$.

## 4.2 Denoising methods for fluoroscopic images

In order to improve the quality of fluoroscopic images, several denoising algorithms for Poisson noise have been developed during the years, [95].

The denoising is generally carried out by averaging the pixel values both in time and space. Linear filters usually assume noise to be additive, white and Gaussian (AWGN) and the noise is supposed not to be signal-dependent. These filters are usually very fast and allow the real-time processing of fluoroscopic video sequences. However, they exhibit the undesirable effect of degrading edges and tiny structures that make them not suitable for applications in which a good trade-off between noise reduction and signal preservation is required (e.g. image segmentation, object recognition or image registration).

In order to overcome the limitations of the average filters, more complex digital processing methods and denoising strategies have been proposed, [94]-[93].

Reference [95] analyzes and compares several denoising methods in terms of video quality and computational time. In particular the following algorithms have been considered:

- The spatio-temporal average filter proposed in [94], and designed for signal-dependent noise. It performs the average of the only neighbouring pixels that differ less than a selected threshold from the gray level of the central pixel of the filter mask. The threshold is set to two times the estimated standard deviation of the noise associated with the local gray level. This intrinsically permits to preserve the edges with a luminance gradient greater than the local noise intensity.

- The adaptive variational denoising proposed in [80]. This method is able to perform the denoising by preserving fine scale features.

- The BM3D, [87]. It performs an image collaborative denoising strategy based on an enhanced sparse representation in transform domain obtained by grouping similar image regions (e.g. blocks) into 3-D data arrays.

- A denoising algorithm for signal-dependent clipped noisy observations (BM3Dc), [87]. The approach involves a BM3D filter designed for AWGN and derives specific homomorphic transformations to adapt the estimated noise variance to the actual signal-dependent noise model, to compensate the bias due to the clipped distribution in the variance-stabilized domain and to compensate the estimation bias between the denoised clipped variables and the non-clipped true variables.

- Wavelet-domain denoising algorithms, [79]-[85]. A framework for statistical signal processing based on wavelet-domain hidden Markov models that concisely models the statistical dependencies and non-Gaussian statistics encountered in real signals was assumed . The method involves an efficient expectation maximization algorithm for fitting the HMM to observational signal data. This approach can be very useful for reconstructing images affected by non-Gaussian noise.

- The K-SVD [86], an image denoising algorithm for AWGN based on sparse and redundant representations over trained dictionaries.

Reference [95] shows that collaborative denoising strategy (BM3D, BM3Dc) provide the most effective image denoising. In particular, BM3Dc filter presents the highest PSNR and SNR values being also the fastest algorithms.

However, the majority of these denoising methods is hardly suited for hardware implementations.

The filter proposed in [94], despite its simplicity, offers performance comparable to those of much more complex algorithms (e.g. BM3D) in terms of peak-signal-to-noise ratio (PSNR), signal-to-noise ratio (SNR), mean square error (MSE), structural similarity index (SSIM) and is feasible for an hardware implementation that allows real-time processing of fluoroscopic images.

Next chapter shows hardware implementation of the algorithm proposed in the [94] whose description is given in the following paragraph.

## 4.3 Spatio-temporal average filter

The spatio-temporal average filter proposed and tested in software in [94], if compared to others available in literature, filters the fluoroscopic images while preserving edges and motion and results feasible for an FPGA-based implementation that allows to process fluoroscopic video sequences in real-time.

In order to perform a better noise reduction, the implemented filter operates both in the space and in the time. For each pixel of each new acquired frame, the filter performs a conditioned average of the pixel luminance values in a spatial surrounding of the current pixel ($Pix_{cur}$ in the following) and between the corresponding pixels in K-1 previously acquired frames.

The filtering operation is therefore performed on a (2X+1)(2Y+1)K spatio-temporal window centered, as shown in Fig. 4.3 , in $Pix_{cur}$. The spatio-temporal window is indicated as $Pix_{win}$ in the following. Each pixel of $Pix_{win}$ (indicated as $Pix_{ref}$ in the following) is compared with $Pix_{cur}$. The $Pix_{ref}$ whose luminance value differs from $Pix_{cur}$ more than a luminance dependent threshold (T($Pix_{cur}$) in the following) are excluded from the computation of the average value. The equation for the average luminance value ($Pix_{out}$) is:

$$Pix_{out}(m,n,K) = \frac{\sum_{k=1}^{K}\sum_{x=m-X}^{m+X}\sum_{y=n-X}^{n+X} Pix(x,y,k)}{\sum_{k=1}^{K}\sum_{x=m-X}^{m+X}\sum_{y=n-X}^{n+X} C(x,y,k)}$$
$$k = 1,...,K \quad m = 1,...,M \quad n = 1,...,N \tag{4.8}$$
$$x = m-X,...,m+X \quad y = n-Y,...,n+Y$$

where:

$$\begin{aligned} Pix(x,y,k) \quad = \quad & Pix_{ref}(x,y,k) \\ & if|Pix_{ref}(x,y,k) - Pix_{cur}(m,n,K)| \leq T(Pix_{cur}) \\ & 0otherwise \end{aligned} \tag{4.9}$$

$$\begin{aligned} C(x,y,k) \quad = \quad & Pix_{ref}(x,y,k) \\ & if|Pix_{ref}(x,y,k) - Pix_{cur}(m,n,K)| \leq T(Pix_{cur}) \\ & 0otherwise \end{aligned} \tag{4.10}$$

In (4.8),(4.9),(4.10) it is assumed that the K frames have size equal to M×N, k is the index for the frame, x,y, and m,n indicate the position of the pixels in the frame k. The threshold T($Pix_{cur}$) was assumed to be a multiple (e.g. the double) of the standard deviation value of the Poisson noise which depends on the $Pix_{cur}$ luminance value. It is worth noting that the particular choice of the filter allows to immediately display a newly arrived pixel in the image independently on the number of considered frames in the spatio-temporal average filter.

The filter operation is based on the knowledge of the relationship between the variance of noise and the luminance of the pixel. The condition:

$$|Pix_{ref}(x,y,k) - Pix_{cur}(m,n,K)| \leqslant T(Pix_{cur}) \tag{4.11}$$

allows to exclude by the average calculation the pixels having a luminance value that most likely does not belong to the local statistical distribution of noise. Over time this means that most likely there was a motion and another object has entered or left the area. In space, the same event indicates that the

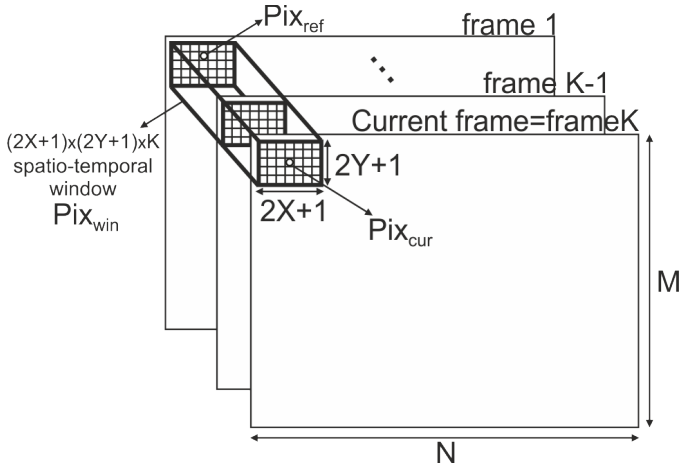


**Figure 4.3:** Example of filtering operation. The current frame (frame K) is filtered by using K-1 previously acquired frames. The frame size is M×N. The filtering is performed by using a window with size (2X+1)(2Y+1)K.

pixels lie on an edge between two objects. Therefore, the condition (4.11) allows to remove the noise while preserving objects edges even for moving objects, [94].

Due to the nature of the Poisson noise the relationship between pixel luminance and local noise variance is linear (at the sensor level) with a slope depending on detector gain. Hence, the relationship between luminance and noise standard deviation (SD) is non-linear: it follows a square-root law or a more complex equation when white compression or gamma correction are applied, [94].

The actual noise statistic can be either estimated at the beginning of a session of X-ray fluoroscopy, or pre characterized as a function of the X-ray tube parameters. In both cases, the relationship between noise standard deviation and image gray-level (luminance) can be estimated by recording few images of a static scene and making their differences (difference of two Poissons distributions gives rise to a Skellam distribution [96]- [97]). Finally, by fitting analytical derived model (see ref. [94] for practical examples) with the actual data, the relationship can be estimated. In particular, for a given fluoroscopic device the luminance- noise SD relationship can be parameterized with respect to all possible settings of the device (e.g. KVp, mA, pulsed mode).

# Chapter 5

# Real-time denoising of fluoroscopic images

Fluoroscopic devices use X-rays to obtain real-time moving images of patients and support many surgical interventions and a variety of diagnostic procedures. In order to avoid risks for the patient health, X-ray intensity has to be kept acceptably low during the clinical applications. This implies that fluoroscopic images are corrupted by large quantum noise (Poisson-distributed). Real-time noise reduction can offer a better visual perception to doctors and makes possible further reductions of the radiation dose.

In order to remove the Poisson noise, several algorithms have been proposed (Chapter 4). Among these, the algorithm proposed in [94] allows good performances in terms of PSNR, SNR, MSE, and SSIM while preserving edges and moving objects. The filter incorporates information on the dependence of the standard deviation of the noise on the local brightness of the image and performs a conditioned average operation. Moreover, the filter is suited for an hardware implementation that allows real-time processing of fluoroscopic video sequences.

This Chapter describes the hardware implementation of the algorithm proposed in [94].

The proposed circuit is implemented on FPGA (Field Programmable Gate Array) devices allowing the real time elaboration of video streams composed by frames with $1024 \times 1024$ pixels and uses an external DDR2 (Double Data Rate 2) memory for the storage and the reuse of the fluoroscopic frames needed by the filter.

When implemented on StratixIV-GX230 FPGA the circuit is able to pro-

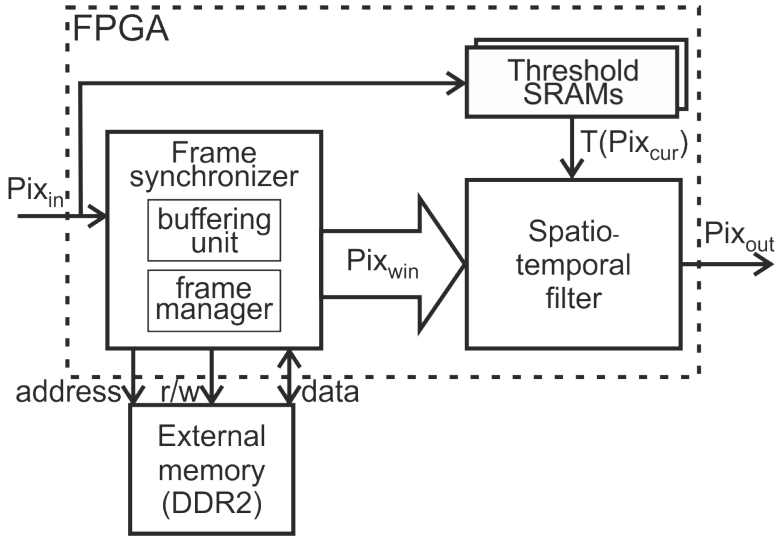


**Figure 5.1:** Conceptual overview of the proposed circuit for the fluoroscopic images filtering. The Frame synchronizer unit synchronizes the input pixel $Pix_{in}$ with the pixels belonging to K-1 frames previously acquired and stored in the External memory. The pixel luminance values of the whole spatio-temporal window ($Pix_{win}$ in Fig.4.3) are processed from the Spatio temporal filter that provides as output the average value $Pix_{out}$. Two SRAMs are also implemented to store the relationship between noise standard deviation and pixel luminance and to allow the real-time modification of this relation.

cess up to 58 fps (frames per second) while using 24% of the logic resources of the FPGA.

## 5.1 Hardware implementation

Figure 5.1 shows a conceptual overview of the proposed circuit. The circuit is able to process the fluoroscopic images in real time.

The images acquired from the fluoroscope are transmitted to the filtering circuit pixel by pixel. The luminance value of the input pixel $Pix_{in}$ is synchronized by the 'Frame synchronizer' unit with the luminance values of the pixels belonging to K-1 frames previously acquired and stored in the 'External memory'. The pixel luminance values of the whole spatio-temporal window

($Pix_{win}$ in Fig. 1) feed the 'Spatio temporal filter' that provides as output the average value '$Pix_{out}$' computed as described in the Chapter 4.

The threshold value T($Pix_{cur}$) needed to the filtering operation is obtained by using a SRAM (Threshold SRAM unit of Fig. 5.1) that stores the relationship between noise SD and the pixel luminance. Figure 5.1 shows that two 'Threshold SRAMs' are implemented on the FPGA. This ensures that, as explained in Section 5.1.3, this relationship can be changed in real time, without suspending the processing of the current image.

The 'Frame synchronizer', the 'Spatio-temporal filter', and the 'Threshold SRAMs' are designed and implemented on a FPGA based system composed by a StratixIV (EP4SGX230KF40C2) Altera FPGA and an external DDR2 SDRAM memory with 1GB of storage capacity (DDR2 800 1GB).

A detailed explanation of the hardware implementation of the three circuital units is given in Section 5.1.1, Section 5.1.2, and Section 5.1.3, respectively. While most design considerations are conducted for the general case in terms of size of the frame (M×N), and size of the filtering window (X,Y,K) (see Fig.4.3), when the sake of clarity requires it, the actual numbers used for the implementation (M=N=1024, X=Y=3, K=5) are used in the text and in the figures.

### 5.1.1 Frame synchronizer

In order to correctly filter the input fluoroscopic images, the implemented filter needs K-1 previously acquired frames. The most common fluoroscopic images are composed by 1024×1024 pixels. As a consequence, also for low K values, a significant amount of memory is needed to store the frames. As an example, for K=5 and with the pixel luminance values represented on 8 bits, 32Mb of memory are required. Such amount of memory is generally unavailable in FPGA devices. As an example, the StratixIV-GX230 FPGA can store up to 14 Mb of data. As a consequence, the electronic system that has been designed relies on the use of an external memory that stores the K-1 previously acquired frames.

As shown in Fig. 5.1, the 'Frame synchronizer' is composed by two subunits: 'frame manager' and 'buffering unit'. A more detailed block diagram of the Frame synchronizer is given in Fig.5.2. The 'frame manager' receives, in streaming fashion, the pixels of the frame that is being filtered and, fetches from the DDR2 memory the additional K-1 streams of the K-1 previous frames. The data, synchronized with the input stream are then provided to the 'buffering unit'. If, as in the implemented circuit, K=5 and the pixels

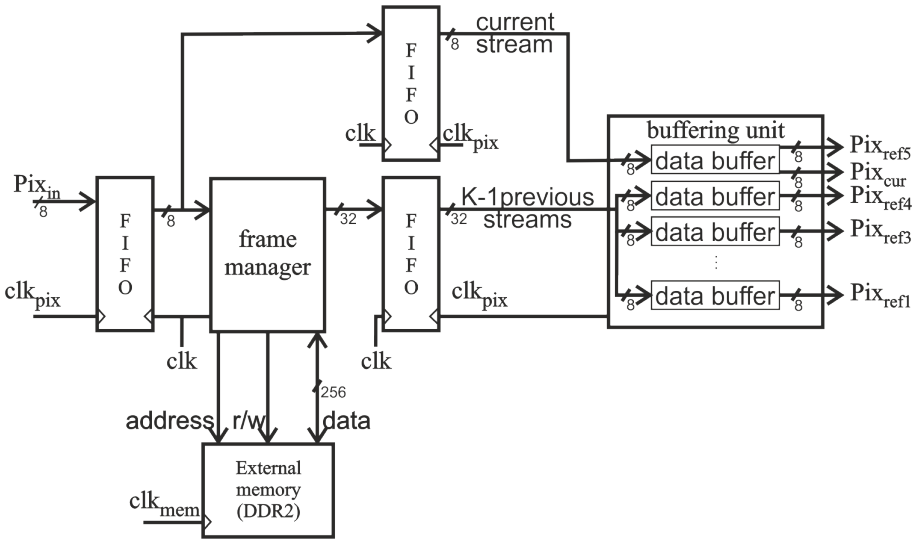


**Figure 5.2:** Detail of the Frame synchronizer of Fig. 5.1. Frame manager and buffering unit are synchronous with clk and $clk_{pix}$, respectively. The two different clock domains are synchronized by using three asynchronous FIFO.

are represented on 8 bits, the 'frame manager' provides 40 bits (5 pixels) to the 'buffering unit' at every cycle of the $clk_{pix}$ signal. Starting from these K streams, the 'buffering unit' is in charge of providing to the Spatio temporal filter the pixels belonging to the spatio-temporal window $Pix_{win}$ of Fig. 4.3.

The 'frame manager' and the 'buffering unit' of Fig. 5.2 are synchronous with two different clock signals that run at two different working frequencies. The 'frame manager' is synchronized by the clk signal (with a frequency of 200 MHz in this case), derived from DDR2 clock signal $clk_{mem}$. The clock pixel signal ($clk_{pix}$), which synchronizes the input stream, is used as clock signal for the buffering unit and the following 'Spatio-temporal filter'. Asynchronous FIFOs, sized to avoid overflow conditions, are used to interface the two different clock domains.

### Frame manager

The 'frame manager' unit of Fig.5.2 manages, the flow of data going to and from the external memory by generating the read and write requests and by

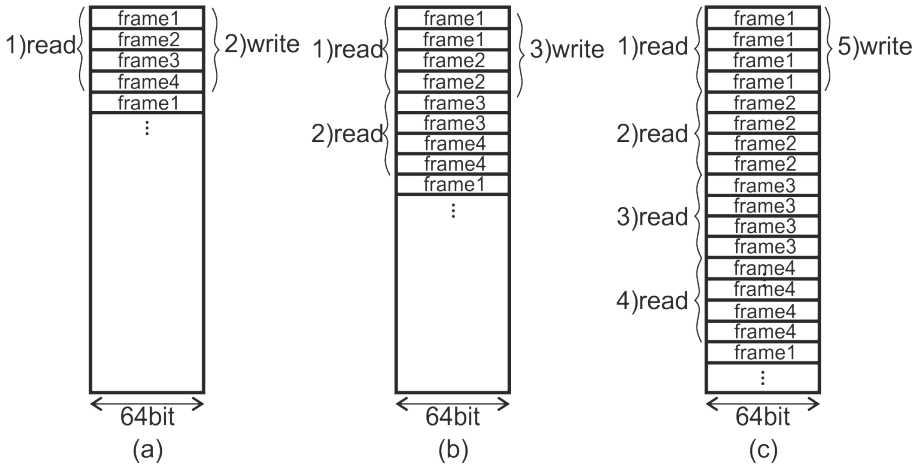


**Figure 5.3:** Example of DDR2 addressing schemes that entail a different trade-off between maximum frame rate and hardware complexity. Read operations are executed to read an equal number of pixel for each frame. The read operations are followed by a write operation to store in the memory the pixels of the new input frame. In the figure, the numbers near read/write indicate the order in which the operations are executed.

determining the memory addresses in which the data must be stored.

The external memory that we will consider is a generic SDRAM widespread in systems and rapid prototyping boards. SDRAM read and write operations are synchronous with a clock signal and are burst oriented: the access starts at a selected memory location and continues for a burst length. Typical burst length values are 2, 4 or 8 memory locations. Write operations can be executed by enabling the data masking: input data are masked so that only some byte of the burst are written.

The memory locations are organized in banks, rows and columns. When a read or write request is issued, a row of a memory bank is activated and the data are read or written in consecutive columns of the active row.

The execution of read/write operations requires several clock cycles. The number of clock cycles also depends on the sequence of the read/write operations and on the memory location addresses in which these operations are executed. As an example, on a DDR2 800 the execution of a read operation followed by a write operation requires 14 clock cycles at 400 MHz (corre-

sponding to a delay of 35 ns) when read and write operations are carried out on the same row of the same memory bank. This delay increases when the data are stored on different memory banks or on different rows of the same memory bank. As an example, 20 clock cycles are needed when data are stored on the same row of different memory banks.

The sequence with which read/write operations are executed depends on the implemented memory addressing scheme and on the required application. The chosen addressing scheme, largely determines the average time required from the memory operations with an impact on overall circuit performances.

In the proposed circuit, the Spatio-temporal filter of Fig.5.2 processes the input frame and K-1 previously acquired frames. The read/write operations must guarantee that the pixels of the K-1 previously acquired frames are read from the external memory and the pixels of the new input frame are stored in the memory. Figure 5.3 shows three possible memory addressing schemes that correspond to three different sequences of read/write operations. The figure assumes 64 bit memory locations, burst length equal to 4 (256 bit can be written/read in a single operation), and K equal to 5 (4 frames are stored in the memory).

When the data are stored as in Fig. 5.3(a), if the pixels are represented on 8 bit, a single read operation allows to read 8 pixels for each one of the 4 stored frames. A write operation is needed to overwrite 8 pixels of the oldest frame. The write operation is masked in order to overwrite one only memory location. The scheme of Fig. 5.3(b) allows to read, with a single read operation, 16 pixels from two frames. Two consecutive read operations are then needed to fetch the data from the four stored frames. A single write operation, masked as in the previous case, is executed to store the 16 new input pixels.

Finally, when the data are stored as shown in Fig. 5.3(c) four consecutive burst read operations are required to fetch 32 pixels from each frame and a single, not masked, write operation is executed to store 32 pixels of the new input frame.

The maximum number of processed frame per second (frame rate) can be obtained calculating the the number of read write bursts that are needed to store the new frame in the memory (being this time equal to the time needed to fetch and store the required K frames in the memory). The parameters that determine the frame rate are:

- *bit per burst* the number of bit read in each burst (see Fig. 5.3). In the considered application a memory location contains 64 bits and a burst reads 4 consecutive memory locations. As a consequence bit per burst

**Table 5.1:** AvPSNR values varying the word lengths of mean, variance, weight, and matchsum signals. The table refers to the first phase of the algorithm for the optimization of the word lengths in which the same number of bits is used for every Gaussian parameter.

| K | frame per burst | $N_{read}$ | $N_{write}$ | $N_{overhead}$ | | | | | fps |
|---|---|---|---|---|---|---|---|---|---|
| | | | | read/write overhead | refresh | precharge | activate | tot | |
| 5 | 4 | 2 | 2 | 10 | 0.22 | 0.03 | 0.05 | 10.30 | 213.4 |
| | 2 | | | | 0.25 | 0.03 | 0.05 | 10.33 | 373.5 |
| | 1 | | | | 0.32 | 0.04 | 0.06 | 10.42 | 597.9 |
| 9 | 8 | 2 | 2 | 10 | 0.22 | 0.03 | 0.05 | 10.30 | 106.7 |
| | 4 | | | | 0.25 | 0.03 | 0.05 | 10.33 | 186.8 |
| | 2 | | | | 0.32 | 0.04 | 0.06 | 10.42 | 298.9 |
| | 1 | | | | 0.44 | 0.06 | 0.08 | 10.58 | 427.2 |
| 17 | 16 | 2 | 2 | 10 | 0.22 | 0.03 | 0.05 | 10.30 | 53.4 |
| | 8 | | | | 0.25 | 0.03 | 0.05 | 10.33 | 93.4 |
| | 4 | | | | 0.32 | 0.04 | 0.06 | 10.42 | 149.5 |
| | 2 | | | | 0.44 | 0.06 | 0.08 | 10.58 | 213.6 |
| | 1 | | | | 0.69 | 0.09 | 0.11 | 10.89 | 271.9 |
| 17 | 32 | 2 | 2 | 10 | 0.22 | 0.03 | 0.05 | 10.30 | 26.7 |
| | 16 | | | | 0.25 | 0.03 | 0.05 | 10.33 | 46.7 |
| | 8 | | | | 0.32 | 0.04 | 0.06 | 10.42 | 74.73 |
| | 4 | | | | 0.44 | 0.06 | 0.08 | 10.58 | 106.8 |
| | 2 | | | | 0.69 | 0.09 | 0.11 | 10.89 | 136.0 |
| | 1 | | | | 1.19 | 0.15 | 0.17 | 11.51 | 157.5 |

is equal to 256.

- *bit per pixel* is 8 bit in the considered application.
- *frame size* equal to 1024×1024 that is 1Mpx in the considered application.
- *frame per burst* the number of frames that are interested by every burst operation. In Fig. 5.3(a), frame per burst is equal to 4, in Fig. 5.3(b) it is 2, while in Fig. 5.3(c) is 1.
- *average cycle time*- is the average time needed to complete a read write cycle. In Fig. 5.3(a) a read write cycle is composed by one read burst and one write burst. In Fig. 5.3(b) a read write cycle is composed by two read bursts and one write burst. In Fig. 5.3(c) a read write cycle is composed by four read bursts and one write burst.

The number of processed frames per second is then:

$$fps = \frac{bit\ per\ burst/(frame\ per\ burst \cdot bit\ per\ pixel)}{(frame\ size) \cdot (average\ cycle\ time)} \tag{5.1}$$

Where the term *bit per burst*/(*frame per burstbit per pixel*) takes into account the fact that, as an example, if the addressing scheme of Fig. 5.3(a) is used, only 8 pixels of the frame are written during the write burst while, with the addressing scheme of Fig. 5.3(c), 32 pixels are written during the write burst.

The average cycle time is the average time needed to complete a read write cycle for the considered addressing scheme. It can be expressed in terms of number of clock cycles of the memory clock signal, indicated in (5.1) as $Tclk_{mem}$.

The average cycle time can be further divided into three components: *Noverhead*, the average number of clock cycles needed to precharge, and activate rows and banks, to refresh the memory locations, and also includes the fixed overhead for read and write operations; *Nread*, the number of additional clock cycles for each read operation in a cycle; *Nwrite*, the number of additional clock cycles for each write operation in a cycle. The average cycle time (avg cycle time) can be written as

$$avg\ cycle\ time = [N_{read}\left(\frac{K-1}{frame\ per\ burst}\right) + N_{write} + \\ + N_{overhead}]T_{clk\_mem} \tag{5.2}$$

*Noverhead* , *Nread* , and *Nwrite* change with the type of external memory, with the number of read/write operations in a cycle, and with the order in which these operations are executed.

The parameter *Noverhead* is minimized by reducing the number of bank and row changes. For this purpose the physical addressing scheme shown in Fig. 5.4 can be employed. Equation (5.2) can be rewritten as:

$$fps = \frac{bit\ per\ burst/bit\ per\ pixel}{(frame\ size)} \cdot \\ \cdot \frac{1}{[(K-1)N_{read} + (frame\ per\ burst)(N_{overhead} + N_{write})]T_{clk\_mem}} \tag{5.3}$$

In the proposed circuit a DDR2 800 1GB is used with maximum working frequency of 400MHz ($Tclk_{mem}$=2.5ns).

Table 5.1 shows the maximum frame rate (fps column) as a function of K, frame per burst, *Nread*, *Nwrite*, and *Noverhead*. The parameters *Nread*,

*Nwrite*, and *Noverhead* have been obtained from the DDR2 SDRAM specification [24]. Table 5.1 shows that *Noverhead* (column tot in Table 5.1) is about $10Tclk_{mem}$ for any value of K and frame per burst. As a consequence, the maximum frame rate variations are mainly due to the frame per burst parameter. Low values of frame per burst provide the highest frame rate towards the memory.

The frame per burst parameter is also important for the determination of the complexity of the circuit. As a matter of fact a reduced number of frames addressed in each burst increases the number of pixels fetched from the memory in a single read write cycle. These pixels have to be locally stored in the FPGA device so that the decreasing of frame per burst determines an increase of the hardware resources utilization. Therefore, there is a trade off between the frame rate towards the memory and the hardware complexity. For a given K value the hardware complexity is minimum when the number of frame per burst is equal to K-1.

In this work a K value equal to 5 has been chosen. In this case all the analyzed addressing schemes allow frame rate values higher than the frame rate required for real-time applications. The choice of the addressing scheme can be solely based on the hardware complexity of the resulting circuit that is minimum when the addressing scheme of Fig. 5.3(a) is implemented (number of frames per burst equal to 4).

### Buffering unit

The buffering unit of Fig. 5.2 receives the K pixel streams of the K frames from the 'frame manager' and provides to the filter the set of pixels belonging to the $Pix_{win}$ of Fig. 4.3. The buffering unit is composed by K data-buffers, each in charge of managing the pixel of a single frame. Figure 5.5 shows an example of data buffer. It works as a delay line that stores 2Y+1 rows of the image (Fig.5.5(a)). When the buffer is filled, the first pixel is discarded. The pixels in the highlighted region of size (2X+1)x(2Y+1) are involved in the filtering operation.

When implemented by using only flip-flops (FF), the architecture shown in Fig. 5.5 would consume too many logic resources. In fact, it would need K·N(2Y+1) flip-flops (286720 FF in the proposed architecture). In order to reduce the number of used FF, the hardware implementation of the data buffers uses a number of RAM components yet available in the FPGA device (both BRAM, Block RAM components, and MLAB, Memory Logic Array Blocks, are used) that store the section of the data buffer that does not contain the

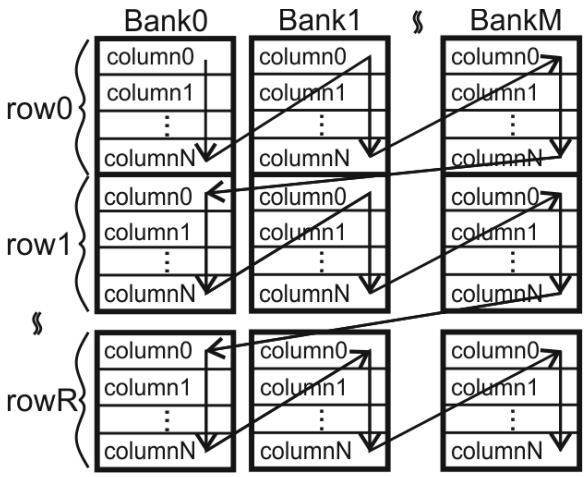


**Figure 5.4:** Physical addressing scheme which minimizes the average cycle time of (5.2). The maximum delay in read/write operations is reached when the row address changes. Therefore, the data are stored in corresponding rows of different banks.

$Pix_{win}$ data. In this way only K·(2X+1)·(2Y+1) FF are needed (245 FF in the proposed architecture). In particular, as shown in Fig. 5.5(b), the pixels in the region highlighted with the thick solid line are stored by using flip-flops while the other pixels are stored in FIFO (Firt In First Out circuits) implemented with BRAMs and MLABs. The pixels stored in the flip flops are used by the Spatio-temporal filter of Fig. 4.3 to perform the filtering operation.

### 5.1.2 Filter implementation

The Spatio-temporal filter of Fig. 5.1, whose block diagram is shown in Fig. 5.6, receives as inputs the $Pix_{ref}$ values needed to perform the filtering operation on the K consecutively acquired frames. As explained in the Chapter 4 the filtering algorithm computes the conditional average value of the $Pix_{ref}$ that satisfy (4.11). The hardware implementation of (4.11) requires 3(2X+1)(2Y+1)K adders. In order to simplify the hardware implementation, the condition (4.11) has been rewritten as:

$$Pix_{cur}(m,n) - T(Pix_{cur}) \leq Pix_{ref_k}(x,y) \leq Pix_{cur}(m,n) + T(Pix_{cur}) \tag{5.4}$$

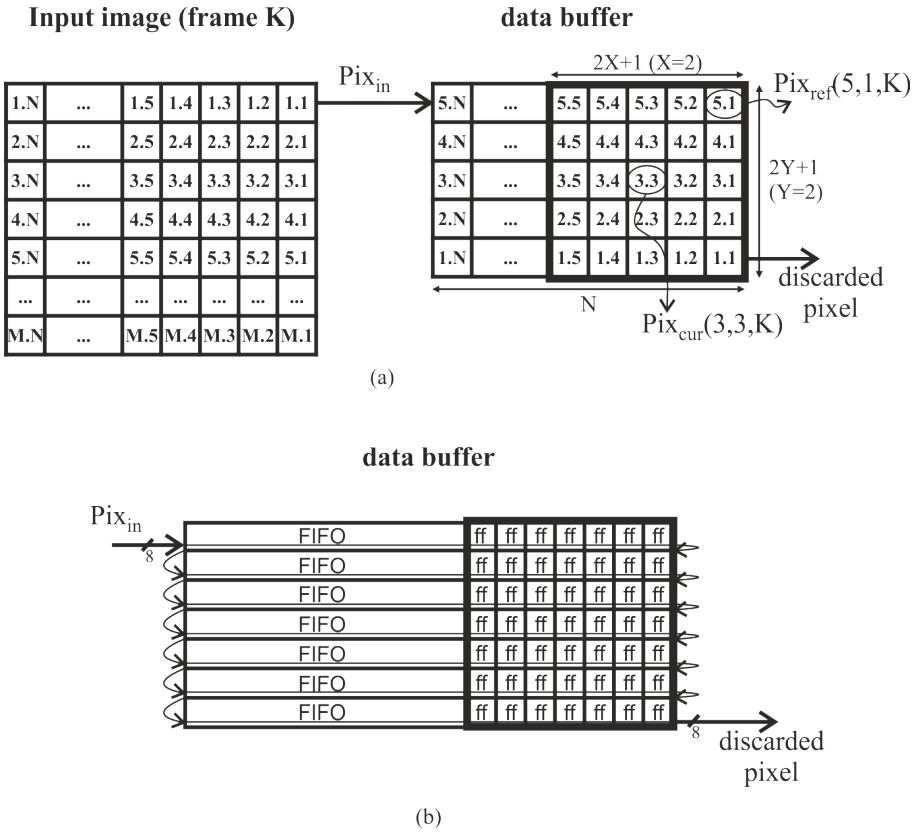


**Figure 5.5:** Example of data buffer used to perform the image filtering. (a) The Input image is shifted into a (2Y+1)×N buffer (with Y= 2). The width of the data buffer depends on the image size (M×N) while the height depends on the filter mask size. When the buffer is full the first pixel is discarded. (b) Hardware implementation of the data buffer. The data in the window highlighted with the thick solid line are stored in flip flop (ff) while, in order to reduce the ff utilization, the other data are stored in FIFO implemented by using both BRAM and MLAB. The pixels in the window of size (2X+1)×(2Y+1) are processed by the Spatio-temporal filter.

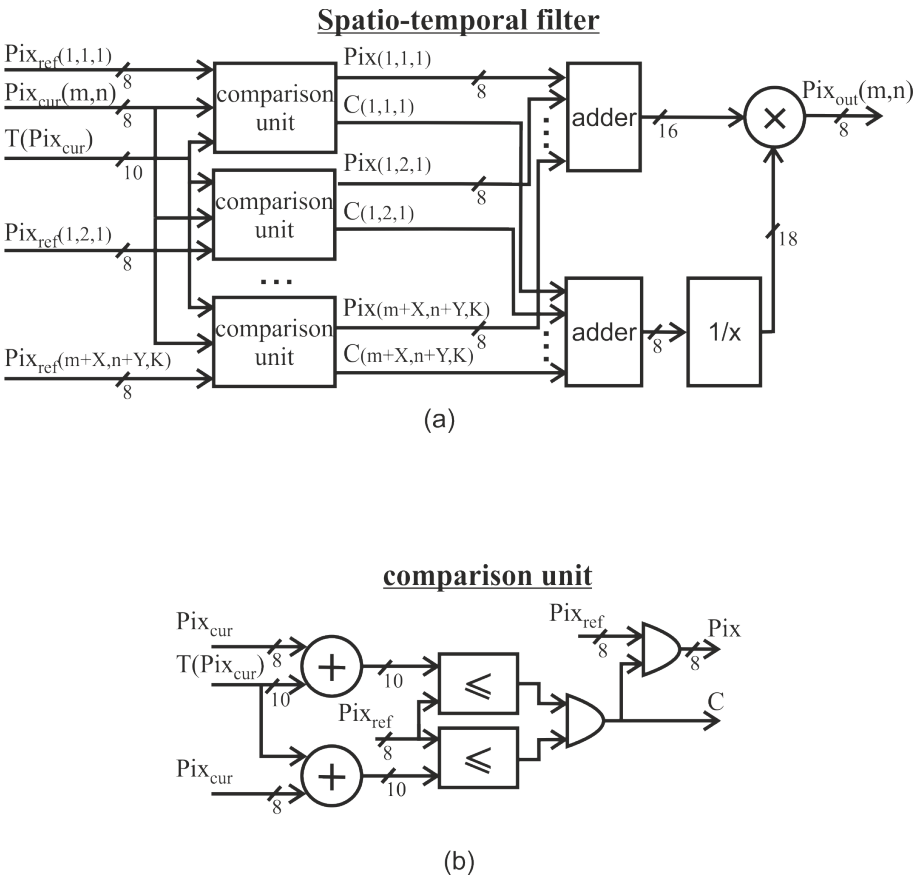


**Figure 5.6:** (a) Block diagram of the Spatio-temporal filter of Fig. 5.1. (b) Detailed view of the comparison unit that implements equation (5.4).

Equation (5.4), exploits the fact that $Pix_{cur}$ is fixed during the filtering of a certain $Pix_{win}$ and hence $Pix_{cur}$ - T($Pix_{cur}$) can be calculated only once. In this way equation (5.4) is implemented by using 2(2X+1)(2Y+1)K+2 adders. If the filter mask has size 7×7 (X=Y=3), K=5, and the pixels are represented on 8 bit, the condition (5.4) entails the implementation of 492 adders on 8 bit versus the 735 adders on 8 bit needed to implement (4.11) (the hardware utilization is reduced by the 33%). Equations (4.8), (4.9), (4.10) are implemented by the comparison unit of Fig. 5.6(a) whose expanded view is given in Fig. 5.6(b). The outputs of the comparison units feed two adders, a circuit that calculates the inverse of the input signal (1/x), and a multiplier, as shown in Fig. 5.6(a). These circuital units calculate the $Pix_{out}$ value following (4.8). The implementation of a circuit that calculates the inverse of the input (1/x) is, in general, a challenging task. In this case, however, since the input of the circuit is on 8 bits, the unit (1/x) can be implemented by using a look up table realized with a ROM that contains 256 memory locations each one on 18 bits (see Fig. 5.6(a)) for a total of 4608 bit.

### 5.1.3 Threshold SRAMs

The threshold value T($Pix_{cur}$) depends on the noise model and is a function of the $Pix_{cur}$ luminance value. The relationship between the noise variance and the luminance value of $Pix_{cur}$ could change with the fluoroscopy device setup and, in principle, also while the circuit is working. In the proposed implementation, in order to obtain the maximum possible flexibility, the relation between the noise SD and the luminance value is stored in a SRAM internal to the FPGA. Since the pixel luminance values are represented by using 8 bits the SRAM only contains 256 memory locations. This allows to easily compute the threshold value T($Pix_{cur}$) by using a look up table realized with a small SRAM. In this way, in the proposed implementation, the reliance of the noise variance on the luminance value can be defined by the user, allowing to elaborate images acquired with different fluoroscopic equipments and in different operating conditions.

In order to address the case in which the fluoroscope operating conditions change during the surgery two 'Threshold SRAMs' units are implemented as shown in Fig. 5.1. The first one stores the threshold values used for the current filtering operation while the second one is used to store the updated threshold values in real time. This implementation allows to change the filter parameters without suspending the processing of the current image.

## 5.2 Results and performances

The circuit described in Section 5.1 has been designed and implemented on FPGA. In the proposed work the target device is the StratixIV EP4SGX230KF40C2 Altera FPGA.

The synthesis and place and route of the circuit have been conducted by using the Quartus II Altera tool. Behavioral and gate level simulations have been carried out with ModelSim PE.

The software implementation, [94], of the implemented algorithm represents the pixel luminance values by using 16 bit per pixel and by normalizing the gray levels to the range 0(black)-1(white). Since the images are noisy the precision with which the pixels are represented can be reduced without degrade the filtered images. The reduction of the signal word length allows to reduce the hardware resources utilization. In the proposed work the pixels are represented on 8 bit.

In order to evaluate the quality of the filtered images and verify that the reduction of the pixel word length does not affect the filtering result, the Peak Signal to Noise Ratio (PSNR) has been calculated on several video sequences.

The PSNR is defined as:

$$PSNR = 10\log_{10}\left(\frac{\max(\text{Pixel luminance value})^2}{MSE}\right) \tag{5.5}$$

$$MSE = \frac{1}{M \cdot N}\sum_{m=1}^{M}\sum_{n=1}^{N}\left(Pix_{out}(m,n) - Pix_{nf}(m,n)\right)^2 \tag{5.6}$$

Where max(Pixel luminance value) is the maximum pixel intensity and Pixnf is the luminance value of the noise-free image used as reference to estimate the PSNR value.

In the proposed work the PSNR values are calculated on sequences of fluoroscopic static images. For each video sequence, the reference image is obtained by averaging over time all the fluoroscopic images (the hypothesis of ergodicity was assumed).

Figure 5.7(a) shows an image extracted from a fluoroscopic video sequence composed by 70 frames with resolution 1024x1024. The zoomed view of a region composed by 120×160 pixels is shown in Fig. 5.7(b). Fig. 5.7(c) shows the result, for the same region, of the spatio temporal filtering operation, processed in software, obtained quantizing the pixel values on 16 bit. On the other

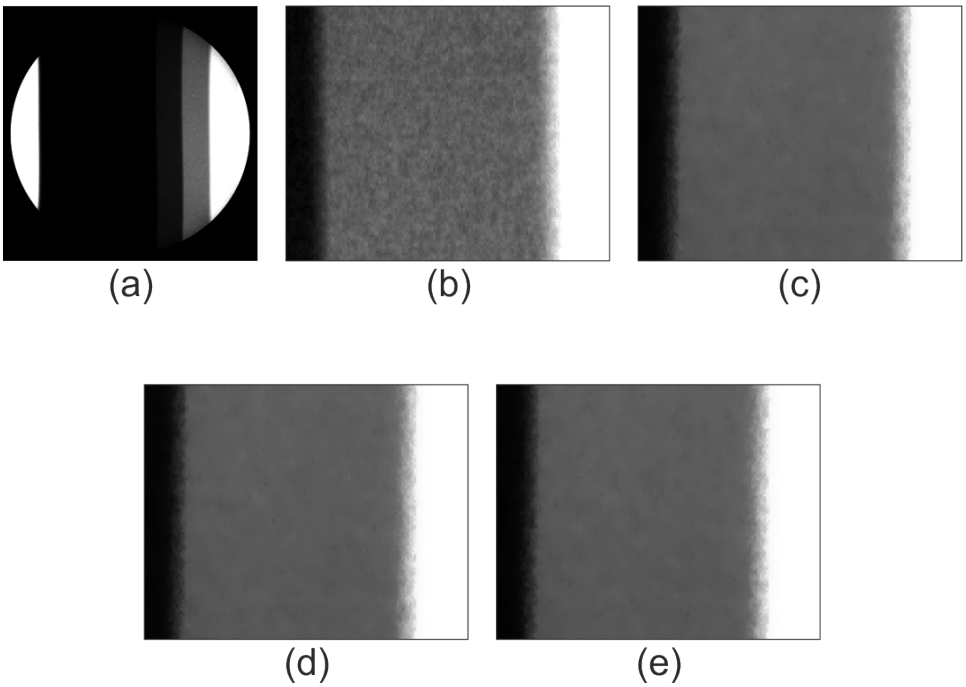


**Figure 5.7:** (a) Example of fluoroscopic image extracted from a video sequence with frame size 1024×1024. (b) image detail with size 120×160 processed with: (c),(d) software implementations of the spatio-temporal filter in which pixel luminance values are quantized with 16 and 8 bit, respectively; (e) proposed circuit in which pixels are represented on 8 bit.

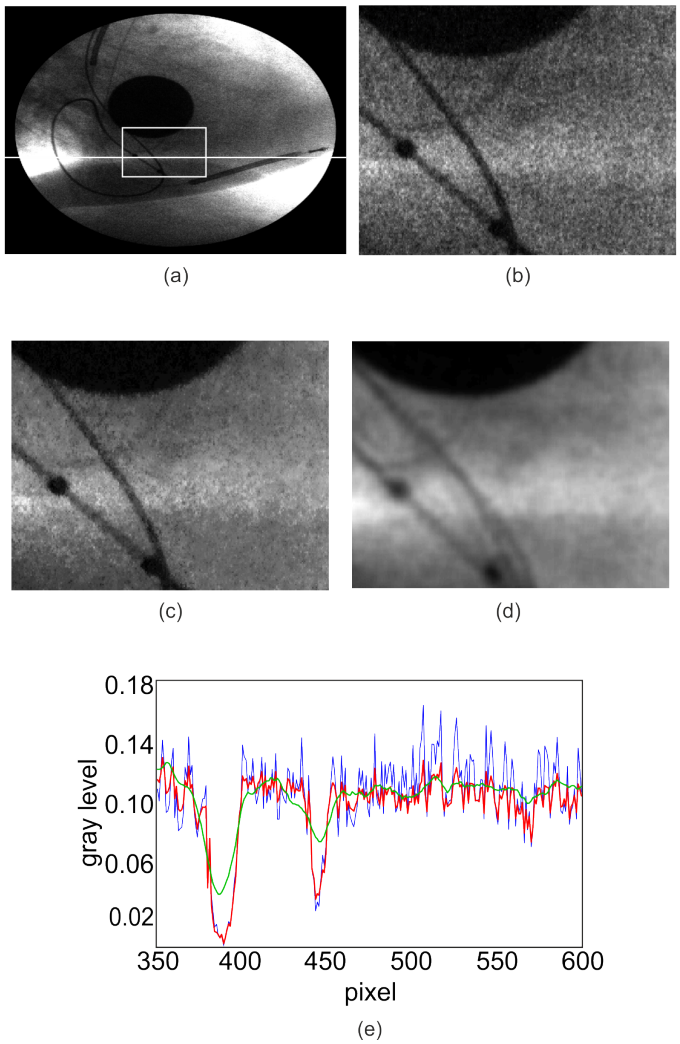


**Figure 5.8:** (a) original fluoroscopic image showing electric leads of a pacemaker during device implantation. (b) expanded view of the 250x200 image section highlighted in a. (c) 250x200 image region in which the Poisson noise has been removed by using the proposed circuit. (d) 250x200 image region in which the noise has been removed by using an average filter. (e) gray-level profile along the horizontal segment (shown in (a)) before (blue line) and after applying the implemented noise suppression filter (red line) and the average filter (green line).

hand, in Fig. 5.7(d) the image has been filtered quantizing the pixel values on 8bit. Fig. 5.7(e) is obtained by processing the image of Fig. 5.7(b) with the proposed circuit. The average PSNR value (obtained by calculating (5.5) for each frame of the video sequence and by averaging over the 70 frames) is equal to 38.99, 38.63, and 38.35 dB by using 16 bit per pixel, 8 bit per pixel and with the hardware implementation, respectively. Therefore, the reduction of the word length to 8 bit has determined a decrease of the PSNR of 0.36 dB while the hardware implementation has determined a further reduction of the 0.28 dB.

Figure 5.8(a) shows a further example of fluoroscopic image. The highlighted region of the image of Fig. 5.8(a), whose expanded view is given in Fig. 5.8(b), is filtered with the proposed circuit (Fig. 5.8(c)) and with an avarage filter that performs the mean of the pixels in Pixwin without considering the noise statistic. Fig. 5.8(e) shows a profile of the gray levels along a horizontal image segment (white line in Fig. 5.8(a)) in the $250\times200$ image section of figures 5.8(b), 5.8(c), 5.8(d). The blue line of Fig. 5.8(e) represents the gray levels of the pixels in the noisy image. It is worth noting that the output of the proposed circuit (red line) provides a good noise suppression while preserving the edges that are excessively smoothed by applying the avarage filter (green line). Table 5.2 shows the performances and the logic resource occupation for the proposed circuit implemented on StratixIV EP4SGX230KF40C2. A breakdown of the the logic resource occupation of the main circuital units of the proposed circuit is also reported in Table 5.2.

The maximum working frequency of the proposed circuit is 61.0 MHz and is limited by the 'Spatio-temporal filter'. As a consequence the maximum frame rate is equal to 58 fps if the frame size is $1024\times1024$.

Modern fluoroscopic devices require to process $1024\times1024$ grey-scale images with a frame rate up to 30 frame per seconds. As a consequence, the proposed circuit is able to filter the images in real time.

The circuit uses the 24% of the logic resources of the FPGA (composed by 91200 ALM, Adaptive Logic Module) and the 9% of the BRAM. Only two DSP (Digital Signal Processing) slices are used to implement the multiplier of Fig.5.6(a) while the usage of flip flop is equal to the 12% of the total number of flip flops in the target FPGA.

**Table 5.2:** Performances of the proposed circuit implemented on StratixIV-EP4SGX230KF40C2 FPGA. The table shows also a breakdown of the resource utilization of the filter and of the Frame synchronizer unit of Fig. 5.1.

| Circuit | | ALM | FF | BRAM (M9K) | MLAB (kb) | DSP | Frequency (MHz) | fps (1024×1024) |
|---|---|---|---|---|---|---|---|---|
| Filter | | 2935/91200 | 4034/182400 | 0/1235 | 0/2850 | 2/1288 | 61.0 | 58 |
| Frame synchronizer | frame manager and buffering unit | 19048/91200 | 17338/182400 | 117/1235 | 41/2850 | 0/1288 | 200.0 | 138 |
| | FIFO | 113/91200 | 932/182400 | 0/1235 | 56/2850 | 0/1288 | 200.0 and 61.0 | - |
| Total | | 22096/91200 | 22736/182400 | 117/1235 | 97/2850 | 2/1288 | 61.0 | 58 |

# Conclusion

This dissertation has presented the hardware implementation of digital circuits for real-time processing of High-Definition (HD) video sequences.

Chapter 1 illustrates an overview of state-of-the-art background identification algorithms typically implemented to identify moving objects in video sequences. These algorithms (named, background subtraction algorithms), based on the creation of a background model, perform a fast background identification while correctly modeling scene variations. However, the main drawback of such approaches is a high computational complexity that involves their hardware implementation for real-time processing of HD video sequences.

Among the proposed background subtraction algorithms, Gaussian Model Mixture (GMM) provides good performances in different lighting conditions and in presence of multimodal background. GMM version realized in OpenCV performs a fast background modelling also in the first phase of the identification process. Due to its good performances, OpenCV GMM has been chosen for hardware implementation in the present dissertation. GMM equations adopted in the OpenCV version have been also reported in Chapter 1.

Chapter 2 describes the hardware implementation of OpenCV GMM. The algorithm has been realized both on FPGA devices and by using a standard cells library in UMC-90nm technology. The main contributions of this work can be summarized as follow:

- the representation providing the best trade-off between circuit complexity and identification accuracy has been determined for each input parameter;

- an innovative, hardware-oriented, formulation of the GMM equations, capable of guaranteeing hardware saving and speed improvement without affecting the GMM algorithm results, has been used;

- non-linear functions have been implemented by using piecewise linear approximations;
- shifters and truncated multipliers have been used to implement full-width multipliers;
- a depth analysis of the performance when the circuit is implemented on several FPGA devices and varying the number of pipeline levels is provided;
- the experimental demonstration of the proposed FPGA circuit in running on-line video systems;
- The circuit ASIC standard cell implementation in two versions (capable of optimizing power or silicon area occupation, respectively), in order to provide a performance reference for ASIC designers currently not available in literature.

As a result of the above mentioned contributions, the circuit, when implemented on Virtex6 (xc6vlx195t) FPGA, is able to process 57 HD fps by only using the 1% of the target device Slices.

Furthermore, the two ASIC implementations have been optimized in order to process 60 fps either with a reduced silicon area utilization (equal to 21847 $\mu m^2$) or with a low-energy consumption (15.3 pJ/pixel).

In Chapter 3, a denoising circuit specifically addressed for binary images has been described and implemented. The circuit is based on morphological operators and has been adopted as second stage in a background identification circuit. The identification phase is carried out by using the circuit described in Chapeter 2.

The denoising circuit can be configured to implement the four basic operators of mathematical morphology (erosion, dilation, opening, and closing) by using a structuring element with variable shape up to a $3\times3$ square. Additionally, a particular choice for boundary pixels guarantees the processing of consecutive frames without introducing latency. The circuit implementation is based on delay lines which correctly synchronize the input pixels. Such delay lines have been designed in order to minimize the use of the hardware resources on FPGA.

Implemented on Virtex6 (xc6vlx195t) FPGA, the circuit allows a processing capability equal to 195 HD fps by using 133/31200 Slices.

Chapter 4 deals with the algorithms for denoising of fluoroscopic images, mainly affected from quantum noise (typically modeled as a Poisson distribution). In particular, a spatio-temporal conditioned average filter (able to filter fluoroscopic images while preserving edges and moving objects) has been described in Chapter 4. The filter is optimally suited for hardware implementation.

The FPGA implementation of the spatio-temporal conditioned average filter has been described in Chapter 5.

The circuit has been designed aiming to the reduction of the arithmetic circuital units requested to average the pixels. Furthermore, the filtering requires the adoption of an external memory and, consequently, several addressing schemes, each of them involving different trade-off between hardware complexity and maximum working frequency. All these addressing schemes have been analysed and the one which optimizes the desired trade-off has been finally chosen for employment.

When implemented on Spartan6 FPGA, it is able to process 58 fps (frame resolution equal to $1024 \times 1024$).

The proposed implementation represents, to the best of authors knowledge, the first circuit able to process fluoroscopic images in real time.

# Bibliography


[1] Z. Chaohui, D. Xiaohui, X. Shuoyu, S. Zheng, and L. Min, "An improved moving object detection algorithm based on frame difference and edge detection," in *Fourth International Conference on Image and Graphics (ICIG)*, Aug 2007, pp. 519–523.

[2] H. Liu and X. Hou, "Moving detection research of background frame difference based on gaussian model," in *International Conference on Computer Science Service System (CSSS)*, 2012, pp. 258–261.

[3] Y. Zhang, L. Zize, L. En, and T. Min, "A background reconstruction algorithm based on c-means clustering for video surveillance," vol. 14, pp. 45–47, 2006.

[4] X. Benxian, L. Cheng, C. Hao, Y. Yanfeng, and C. Rongbao, "Moving object detection and recognition based on the frame difference algorithm and moment invariant features," in *27th Chinese Control Conference (CCC)*, 2008, pp. 578–581.

[5] S. Mohamed, N. Tahir, , and R. Adnan, "Background modelling and background subtraction performance for object detection," in *International Colloquium on Signal Processing and Its Applications (CSPA)*, 2010, pp. 1–6.

[6] T. Intachak and W. Kaewapichai, "Real-time illumination feedback system for adaptive background subtraction working in traffic video monitoring," in *International Symposium on Intelligent Signal Processing and Communications Systems (ISPACS)*, 2011, pp. 1–5.

[7] S. Chien, S. Ma, and L. Chen, "Efficient moving object segmentation algorithm using background registration technique," *IEEE Transactions*

*on Circuits and Systems for Video Technology*, vol. 12, no. 7, pp. 577–586, 2002.

[8] I. Haritaoglu, D. Harwood, and L. Davis, "W4: real-time surveillance of people and their activities," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 809–830, 2000.

[9] M. Djalalov, H. Nisar, Y. Salih, and A. Malik, "An algorithm for vehicle detection and tracking," in *International Conference on Intelligent and Advanced Systems (ICIAS)*, 2010, pp. 1–5.

[10] F. Porikli, "Multiplicative background-foreground estimation under uncontrolled illumination using intrinsic images," in *Seventh IEEE Workshops on Application of Computer Vision*, vol. 2, 2005, pp. 20–27.

[11] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers, "Wallflower: principles and practice of background maintenance," in *Proc. of Int. Conf. on Computer Vision*, vol. 1, 1999, pp. 255–261.

[12] S. Messelodi, C. Modena, N. Segata, and M. Zanin, "A kalman filter based background updating algorithm robust to sharp illumination changes," in *Image Analysis and Processing (ICIAP)*, vol. 3617, 2005, pp. 163–170.

[13] Z. Jing and S. Sclaroff, "Segmenting foreground objects from a dynamic textured background via a robust kalman filter," in *IEEE International Conference on Computer Vision*, vol. 1, 2003, pp. 44–50.

[14] C. Ridder, O. Munkelt, and H. Kirchner, "Adaptive background estimation and foreground detection using kalman-filtering," in *Proc. Int. Conf. Recent Advances in Mechatronics (ICRAM)*, 1995, pp. 193–199.

[15] C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland, "Pfinder: real-time tracking of the human body," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 780–785, 1997.

[16] C. Stauffer and W. Grimson, "Adaptive background mixture models for real-time tracking," vol. 2, p. 252, 1999.

[17] ——, "Learning patterns of activity using real-time tracking," vol. 22, pp. 747–757, 2000.

[18] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, pp. 185 – 203, 1981.

[19] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," 1981, pp. 674–679.

[20] J. Barron, D. Fleet, and S. Beauchemin, "Performance of optical flow techniques," *International Journal of Computer Vision*, vol. 12, no. 1, pp. 43–77, 1994.

[21] L. Hongche, T.-H. Hong, and M. Herman, "Accuracy vs. efficiency trade-offs in optical flow algorithms," in *Computer Vision and Image Understanding*, 1996, pp. 271–286.

[22] I. Ishii, T. Taniguchi, K. Yamamoto, and T. Takaki, "High-frame-rate optical flow system," *IEEE Transactions on Circuits and Systems for Video Technology,*, vol. 22, no. 1, pp. 105–112, Jan 2012.

[23] "Opencv library on source forge:," http://sourceforge.net/projects/opencvlibrary.

[24] P. KaewTraKulPong and R. Bowden, "An improved adaptive background mixture model..." in *In Proc. of AVBS*, 2001.

[25] D.-S. Lee, "Effective gaussian mixture learning for video background subtraction," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 27, pp. 827–832, 2005.

[26] Wallflower, http://research.microsoft.com/enus/um/people/jckrumm/WallFlower/TestImages.htm.

[27] P. Carr, "Gpu accelerated multimodal background subtraction," pp. 279–286, 2008.

[28] V. Pham, P. Vo, V. T. Hung, and L. H. Bac, "Gpu implementation of extended gaussian mixture model for background subtraction," in *International Conference on Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF)*, 2010, pp. 1–4.

[29] J. Hongtu, H. Ardo, and V. Öwall, "Hardware accelerator design for video segmentation with multi-modal background modelling," vol. 2, pp. 1142–1145, 2005.

[30] F. Kristensen, H. Hedberg, J. Hongtu, P. Nilsson, and V. Öwall, "An embedded real-time surveillance system: Implementation and evaluation," *Journal of Signal Processing Systems*, vol. 52, no. 1, pp. 75–94, 2008.

[31] J. Hongtu, H. Ardo, and V. Öwall, "A hardware architecture for real-time video segmentation utilizing memory reduction techniques," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, no. 2, pp. 226–236, 2009.

[32] S. Minghua, A. Bermak, S. Chandrasekaran, and A. A. Amira, "An efficient fpga implementation of gaussian mixture models-based classifier using distributed arithmetic," in *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2006, pp. 1276–1279.

[33] M. Ercegovac, T. Lang, J. M. Muller, and A. Tisserand, "Reciprocation, square root, inverse square root, and some elementary functions using small multipliers," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 628–637, 2000.

[34] "Ebu, future high definition television systems," *Technical recommendation*, May 2005.

[35] A. Strollo, N. Petra, and D. D. Caro, "Dual-tree error compensation for high performance fixed-width multipliers," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 52, no. 8, pp. 501–507, 2005.

[36] N. Petra, D. D. Caro, V. Garofalo, E. Napoli, and A. Strollo, "Truncated binary multipliers with variable correction and minimum mean square error," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 6, pp. 1312–1325, 2010.

[37] ——, "Design of fixed-width multipliers with linear compensation function," *IEEE Transactions on Circuits and Systems I: Regular Papers,*, vol. 58, no. 5, pp. 947–960, 2011.

[38] V. Garofalo, N. Petra, and E. Napoli, "Analytical calculation of the maximum error for a family of truncated multipliers providing minimum mean square error," *IEEE Transactions on Computers*, vol. 60, no. 9, pp. 1366–1371, 2011.

[39] D. D. Caro, E. Napoli, and A. Strollo, "Direct digital frequency synthesizers using high-order polynomial approximation," vol. 1, pp. 134–135, 2002.

[40] ——, "Direct digital frequency synthesizers with polynomial hyperfolding technique," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 51, no. 7, pp. 337–344, 2004.

[41] A. Strollo, D. D. Caro, E. Napoli, and N. Petra, "A novel high-speed sense-amplifier-based flip-flop," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 11, pp. 1266–1274, 2005.

[42] D. D. Caro, N. Petra, and A. Strollo, "High-performance special function unit for programmable 3-d graphics processors," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 9, pp. 1968–1978, 2009.

[43] D. D. Caro, N. Petra, A. Strollo, F. Tessitore, and E. Napoli, "Fixed-width multipliers and multipliers-accumulators with min-max approximation error," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 9, pp. 2375–2388, 2013.

[44] V. Garofalo, N. Petra, D. D. Caro, A. Strollo, and E. Napoli, "Low error truncated multipliers for dsp applications," in *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2008, pp. 29–32.

[45] V. Garofalo, "Fixed-width multipliers for the implementation of efficient digital {FIR} filters," *Microelectronics Journal*, vol. 39, no. 12, pp. 1491–1498, 2008.

[46] V. Garofalo, M. Coppola, D. D. Caro, E. Napoli, N. Petra, and A. Strollo, "A novel truncated squarer with linear compensation function," pp. 4157–4160, 2010.

[47] S. Kidambi, F. El-Guibaly, and A. Antoniou, "Area-efficient multipliers for digital signal processing applications," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 43, no. 2, pp. 90–95, 1996.

[48] N. Jamil, M. T. S. Tengku, and Z. Bakar, "Noise removal and enhancement of binary images using morphological operations," in *International Symposium on Information Technology (ITSim)*, vol. 4, 2008, pp. 1–6.

[49] J. Serra, *Image Analysis and Mathematical Morphology*. Orlando, FL, USA: Academic Press, Inc., 1983.

[50] P. Maragos, "A representation theory for morphological image and signal processing," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 11, no. 6, pp. 586–599, 1989.

[51] C. Huang and O. Mitchell, "A euclidean distance transform using grayscale morphology decomposition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 4, pp. 443–448, Apr 1994.

[52] E. Zaharescu, M. Zamfir, and C. Vertan, "Color morphology-like operators based on color geometric shape characteristics," in *International Symposium on Signals, Circuits and Systems (SCS)*, vol. 1, 2003, pp. 145–148.

[53] S. Fejes and F. Vajda, "A data-driven algorithm and systolic architecture for image morphology," in *IEEE International Conference on Image Processing (ICIP)*, vol. 2, Nov 1994, pp. 550–554.

[54] E. Malamas, A. Malamos, and T. Varvarigou, "Fast implementation of binary morphological operations on hardware-efficient systolic architectures," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 25, no. 1, pp. 79–93, 2000.

[55] J. Velten and A. Kummert, "Fpga-based implementation of variable sized structuring elements for 2d binary morphological operations," in *The First IEEE International Workshop on Electronic Design, Test and Applications*, 2002, pp. 309–312.

[56] A. Zarandy, A. Stoffels, T. Roska, and L. Chua, "Implementation of binary and gray-scale mathematical morphology on the cnn universal machine," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 45, no. 2, pp. 163–168, 1998.

[57] P. Borges, "Pedestrian detection based on blob motion statistics," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 23, no. 2, pp. 224–235, Feb 2013.

[58] M. Moradi and et al., "Seed localization in ultrasound and registration to c-arm fluoroscopy using matched needle tracks for prostate brachytherapy," *IEEE Transactions on Biomedical Engineering*, vol. 59, no. 9, pp. 2558–2567, 2012.

[59] J. Weese and et al., "Voxel-based 2-d/3-d registration of fluoroscopy images and ct scans for image-guided surgery," *IEEE Transactions on Information Technology in Biomedicine*, vol. 1, no. 4, pp. 284–293, 1997.

[60] P. Bifulco, M. Cesarelli, T. Cerciello, and M. Romano, "A continuous description of intervertebral motion by means of spline interpolation of kinematic data extracted by videofluoroscopy," *Journal of Biomechanics*, vol. 45, no. 4, pp. 634–641, 2012.

[61] T. Yamazaki and et al., "Improvement of depth position in 2-d/3-d registration of knee implants using single-plane fluoroscopy," *IEEE Trans. on Medical Imaging*, vol. 23, no. 5, pp. 602–612, 2004.

[62] J. Wang, L. Zhu, and L. Xing, "Noise reduction in low-dose x-ray fluoroscopy for image-guided radiation therapy," vol. 74, no. 2, 2009, pp. 637 – 643.

[63] "A dictionary learning approach for poisson image deblurring," *IEEE Trans Med Imaging*, vol. 32, no. 7, pp. 1277–89, 2013.

[64] S. Lefkimmiatis, P. Maragos, and G. Papandreou, "Bayesian inference on multiscale models for poisson intensity estimation: Applications to photon-limited image denoising," *IEEE Transactions on Image Processing*, vol. 18, no. 8, pp. 1724–1741, 2009.

[65] M. J. Tapiovaara, "Snr and noise measurements for medical imaging. ii. application to fluoroscopic x-ray equipment," *Physics in Medicine and Biology*, vol. 38, no. 12, p. 1761, 1993.

[66] R. Aufrichtig and D. Wilson, "X-ray fluoroscopy spatio-temporal filtering with object detection," *IEEE Transactions on Medical Imaging*, vol. 14, no. 4, pp. 733–746, 1995.

[67] J. Wang and T. Blackburn, "The aapm/rsna physics tutorial for residents," *RadioGraphics*, vol. 20, no. 5, pp. 1471–1477, 2000.

[68] M. Izadi and K. Karim, "Noise optimisation analysis of an active pixel sensor for low-noise real-time x-ray fluoroscopy," *Circuits, Devices Systems, IET*, vol. 1, no. 3, pp. 251–256, 2007.

[69] *Nonlinear Restoration Of Filtered Images With Poisson Noise*, vol. 0207, 1979.

[70] R. M. Harrison and C. J. Kotre, "Noise and threshold contrast characteristics of a digital fluorographic system," *Physics in Medicine and Biology*, vol. 31, no. 5, p. 515, 1986.

[71] M. Hensel, T. Pralow, and R.-R. Grigat, "Modeling and real-time estimation of signal-dependent noise in quantum-limited imaging," in *Proceedings of the 6th WSEAS International Conference on Signal Processing, Robotics and Automation*, 2007, pp. 183–191.

[72] T. Cerciello, P. Bifulco, M. Cesarelli, L. Paura, M. Romano, G. Pasquariello, and R. Allen, "Noise reduction in fluoroscopic image sequences for joint kinematics analysis," in *XII Mediterranean Conference on Medical and Biological Engineering and Computing 2010*, ser. IFMBE Proceedings, P. Bamidis and N. Pallikarakis, Eds. Springer Berlin Heidelberg, 2010, vol. 29, pp. 323–326. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-13039-7_81

[73] T. Cerciello, M. Romano, P. Bifulco, M. Cesarelli, and R. Allen, "Advanced template matching method for estimation of intervertebral kinematics of lumbar spine," 2011, vol. 33, no. 10, pp. 1293 – 1302.

[74] P. Bifulco, M. Cesarelli, R. Allen, M. Sansone, and M. Bracale, "Automatic recognition of vertebral landmarks in fluoroscopic sequences for analysis of intervertebral kinematics," *Medical and Biological Engineering and Computing*, vol. 39, no. 1, pp. 65–75, 2001.

[75] P. Bifulco, M. Sansone, M. Cesarelli, R. Allen, and M. Bracale, "Estimation of out-of-plane vertebra rotations on radiographic projections using {CT} data: a simulation study," *Medical Engineering and Physics*, vol. 24, no. 4, pp. 295 – 300, 2002.

[76] P. Bifulco, M. Cesarelli, R. Allen, M. Romano, A. Fratini, and G. Pasquariello, "2d-3d registration of ct vertebra volume to fluoroscopy projection: A calibrationmodel assessment," vol. 2010, 2010, pp. 1–8.

[77] T. Cerciello, M. Cesarelli, L. Paura, P. Bifulco, M. Romano, and R. Allen, "Noise-parameter modeling and estimation for x-ray fluoroscopy," in *Proceedings of the 4th International Symposium on Applied Sciences in Biomedical and Communication Technologies*, 2011, pp. 1–5.

[78] M. A. T. Figueiredo and J. Bioucas-Dias, "Restoration of poissonian images using alternating direction optimization," *IEEE Transactions on Image Processing*, vol. 19, no. 12, pp. 3133–3145, Dec 2010.

[79] J. Cai, B. Dong, S. Osher, and Z. Shen, "Image restoration: Total variation, wavelet frames, and beyond," *J. Amer. Math. Soc.*, vol. 25, p. 10331089, 2012.

[80] G. Gilboa, N. Sochen, and Y. Zeevi, "Variational denoising of partly textured images by spatially varying constraints," *Trans. Img. Proc.*, vol. 15, no. 8, pp. 2281–2289, 2006.

[81] M. Crouse, R. Nowak, and R. Baraniuk, "Wavelet-based statistical signal processing using hidden markov models," *IEEE Transactions on Signal Processing*, vol. 46, no. 4, pp. 886–902, 1998.

[82] G. Steidl, J. Weickert, T. Brox, P. Mrzek, and M. Welk, "On the equivalence of soft wavelet shrinkage, total variation diffusion, total variation regularization, and sides," *SIAM J. NUMER. ANAL*, vol. 42, pp. 686–713, 2004.

[83] S. Bonettini and V. Ruggiero, "An alternating extragradient method for total variation-based image restoration from poisson data," *Inverse Problems*, vol. 27, no. 9, 2011.

[84] F. Luisier, T. Blu, and M. Unser, "Image denoising in mixed poissongaussian noise," 2011.

[85] B. Zhang, J. Fadili, and J.-L. Starck, "Wavelets, ridgelets, and curvelets for poisson noise removal," *IEEE Transactions on Image Processing*, vol. 17, no. 7, pp. 1093–1108, July 2008.

[86] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," *IEEE Transactions on Image Processing*, vol. 15, no. 12, pp. 3736–3745, Dec 2006.

[87] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-d transform-domain collaborative filtering," *IEEE Transactions on Image Processing*, vol. 16, no. 8, pp. 2080–2095, Aug 2007.

[88] C. Chan, A. Katsaggelos, and A. Sahakian, "Image sequence filtering in quantum-limited noise with applications to low-dose fluoroscopy," *IEEE Transactions on Medical Imaging*, vol. 12, no. 3, pp. 610–621, 1993.

[89] R. Aufrichtig and D. Wilson, "X-ray fluoroscopy spatio-temporal filtering with object detection," *IEEE Transactions on Medical Imaging*, vol. 14, no. 4, pp. 733–746, Dec 1995.

[90] M. Panayiotou and et al., "A statistical model of catheter motion from interventional x-ray images: application to image-based gating," *Physics in Medicine and Biology*, vol. 58, no. 21, pp. 7543–7562, 2013.

[91] M. Makitalo and A. Foi, "Optimal inversion of the anscombe transformation in low-count poisson image denoising," *IEEE Transactions on Image Processing*, vol. 20, no. 1, pp. 99–109, Jan 2011.

[92] ——, "Optimal inversion of the generalized anscombe transformation for poisson-gaussian noise," *IEEE Transactions on Image Processing*, vol. 22, no. 1, pp. 91–103, Jan 2013.

[93] A. Bindilatti and N. Mascarenhas, "A nonlocal poisson denoising algorithm based on stochastic distances," *IEEE Signal Processing Letters*, vol. 20, no. 11, pp. 1010–1013, Nov 2013.

[94] M. Cesarelli, P. Bifulco, T. Cerciello, M. Romano, and L. Paura, "X-ray fluoroscopy noise modeling for filter design," *International Journal of Computer Assisted Radiology and Surgery*, vol. 8, no. 2, pp. 269–278, 2013.

[95] T. Cerciello, P. Bifulco, M. Cesarelli, and A. Fratini, "A comparison of denoising methods for x-ray fluoroscopic images," *Biomedical Signal Processing and Control*, vol. 7, no. 6, pp. 550 – 559, 2012, biomedical Image Restoration and Enhancement.

[96] J. G. Skellam, "The frequency distribution of the difference between two poisson variates belonging to different populations," *J. R. Stat. Soc. Ser. A.*, vol. 109, no. 3, p. 296, 1946.

[97] Y. Hwang, J.-S. Kim, and I. S. Kweon, "Difference-based image noise modeling using skellam distribution," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 7, pp. 1329–1341, 2012.