Journal of Real-Time Image Processing manuscript No.
(will be inserted by the editor)

An Adaptive High-Throughput Edge Detection Filtering System using Dynamic Partial Reconfiguration

Milica Orlandić · Kjetil Svarstad

Received: date / Accepted: date

Abstract This paper proposes an architecture consisting of various edge detection filters implemented on modern FPGA platforms exploiting a feature of Dynamic Partial Reconfiguration (DPR). The developed system targets small scale systems, and its use in the educational setting can be of great interest. Two dimensional convolution is the most common operation in digital video/image processing and its implementation is highly demanding in terms of computational intensity, high-throughput and hardware resources. In the case of a variety of filtering techniques used for edge detection, the hardware resources become a constraint, in particular when using convolution kernels with varying parameters and sizes. DPR introduces significant functional density and increased flexibility by providing logic switching within a constrained hardware area. Furthermore, parallel and pipelined hardware solutions for filter implementation overcome computational performance of software solutions and increase effectiveness compared to static hardware solution. The advantages of accommodating a number of various algorithms within the same datapath at low cost and considerable time is exploited in the proposed work. The effectiveness of the DPR feature for edge detection application is tested on the filter scenarios varying in sizes, complexity and intensity of computation, where the resource utilization and timing are evaluated. Experimental results are proposed through comparisons between different configurations (with DPR and without DPR) and detailed performance analysis.

Keywords Edge detection · Canny Edge Detector · Hardware implementation · Field Programmable Gate Arrays (FPGAs) · Dynamic partial reconfiguration (DPR)

M. Orlandić
Department of Electronic Systems,
Norwegian University of Science and Technology (NTNU),
Trondheim, Norway
E-mail: milica.orlandic@ntnu.no

K. Svarstad
Department of Electronic Systems,
Norwegian University of Science and Technology (NTNU),
Trondheim, Norway
E-mail: kjetil.svarstad@ntnu.no

1 Introduction

Edge detection algorithms are widely used in the preprocessing stage in many image/video processing algorithms. Most of edge detection filters are convolution-based, and these systems are characterized by high computational load in particular with increasing resolution requirements. Increased performance and throughput are requested accordingly. The convolution operation traditionally implemented as software running on CPU is characterized by a sequential execution of source code. The trend of increasing processor speed has reached the limit and alternative ways of increasing the performance are required. Software oriented architecture are discarded in time-critical scenarios due to its increased latency and low throughput. On the other side, intensive task-level pipeline and data-level parallel processing can enhance the convolution performance. The General Purpose Graphic Processing Unit (GPGPU) has emerged as a powerful computing platform for intensive data applications. The number of ASIC [1] and GPU solutions [2, 3] for convolution algorithm processing are proposed. Canny edge detection implementations [4, 5] have been tested on a number of GPU configurations (G80, GT200, Fermi). However, there has been a considerable amount of research into reconfigurable tech-

niques in order to design image processors with high flexibility and performance, thus within limited usage of hardware resources and low power consumption. The flexibility arises as a mandatory characteristic within the embedded real-time applications. In particular case, targeting edge detection filtering focuses on efficient matrix computation solutions. For example, FPGA implementation [6] proposes a fast 1D convolution with large kernels up to sizes 24×24, whereas Canny edge detection FPGA implementations for real time applications have been presented by several works [7, 8, 9, 10].

The increasing resolutions represent a requirements which limits hardware resources in choice of available processing functions. By use of Dynamic Partial Reconfiguration (DPR) functionality on modern FPGAs, reconfigurable solutions get additional flexibility. Programmable Logic (PL) resources are saved by storing mutually time-exclusive algorithms within the defined hardware area and by changing functionality/logic as the application requires. This is performed by storing partial bitstreams within the memory and programming a piece of PL with partial bitstream of chosen functionality when requested. DPR has been increasingly used in architectures for video processing. A scalable architecture for DCT unit is implemented in [11], whereas [12] use partial reconfiguration to implement 2-D Haar wavelet transform of various sizes for different image resolutions in face recognition system. Self-reconfigurable motion estimation algorithm with different level of parallelism (from 1 to 4 reconfiguration partitions) is proposed in [13]. In [14] the DPR is used to switch between the MPEG-2 and H.263 inverse quantization module. Biometric personal recognition composed of a number of image processing tasks [15] uses DPR for switching tasks such as segmentation, image normalization, isotropic filtering based on Gaussian filters for noise reduction, image binarization and smoothing and feature extraction executed in sequential order. [16] implement a test-platform where a VGA-camera stream is fed into the FPGA, filtered by a reconfigurable filter and then outputted onto a monitor.

In this paper, we present a System-on-Chip (SoC) platform providing a variety of edge detection filters. The edge detection filters of various sizes, complexity and sensitivity are described, developed, tested and implemented on FPGA. The complete SoC development includes filter architecture design targeting high throughput by exploiting techniques such as 1D convolution and merged kernel convolution, intensive filter testing with input reference images, comparison with software models, software development of supporting processor system for communication and partial reconfiguration, and finally algorithm/module switching by DPR on FPGA chip.

The paper is structured as follows: in Section 2 edge detector filters and their kernels are presented. Canny edge detector filter is described in detail in Section 3. Section 4 presents details of dynamic partial reconfiguration and, in particular, its features, limitations and level of abstraction. The details of filter implementations and overall system architecture are presented in Section 6, whereas in Section 7 synthesis, implementation, performance analysis and test results are summarized. Finally, the concluding remarks are given in Section 8.

2 Video/Image Edge Detection

The video algorithms can be spatial, temporal or spatio-temporal whereas spatial filtering techniques process pixels within the same frame. Most of the image filters use a convolution of the kernel H to a group of pixels. The number of pixels used for the transformation is dependent on the kernel size $N \times M$ where $N = 2n+1$ and $M = 2m + 1$ and the convolution process is denoted:

$$Y(x, y) = \sum_{i=-n}^{n} \sum_{j=-m}^{m} H(i, j)p(x - i, y - j) \quad (1)$$

where p is a generic pixel of the input image, H represents filter weights applied to $N \times M$ block of the neighbor pixels centered at pixel p, $Y(i, j)$ is a pixel of the output image.

The edge detection process identifies and locates sharp discontinuities in the image. Thus, an edge is defined as a set of connected pixels which lie on the boundary of two regions within the image. In practice the image acquisition imperfections produce blurred edges with transition region. The quality of the image acquisition and the illumination conditions are factors which determine the degree of blurring. The edge detection process contains noise removal as pre-processing stage, leading to edge detection itself as core process and post-processing thresholding as last stage which improves the complete edge detection process as presented in Fig. 1. Several methods for edge detection are proposed in the literature. However, convolution methods with defined filter mask are the most common.

The geometry of the kernel determines the direction of the edge detection. Depending on the order of the derivative employed in the method, there are two categories: gradient searching for the extremes in the first derivative of the image, and Laplacian which finds the zero crossings in the second derivative. The gradient of an image $f(x, y)$ at location (x, y), is defined

as a vector with magnitude and the direction angle as follows:

$$\nabla g = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{2}$$

where G_x and G_y are gradients in x and y direction. The edge strength is measured by absolute magnitude of the gradient ∇f defined as a distance measure as follows:

$$|G| = \sqrt{G_x^2 + G_y^2}, \tag{3}$$

which can be further simplified in order to reduce the computational complexity as presented:

$$|G| \approx |G_x| + |G_y|. \tag{4}$$

The angle θ of gradient ∇f at image location (x, y) determines the direction of the edge and is given by:

$$\theta(x, y) = \arctan(\frac{G_y}{G_x}). \tag{5}$$

Fig. 1 General Edge Detecton Dataflow

2.1 Smoothing Filter

Images are often corrupted by random intensity variations. The edge detection in noisy environment can be challenging since both noise and edges are a high-frequency content. The noise increases false edge detection and removal of real edges. The sensitivity to the noise can be minimized by filtering the input image by low-pass filter.

2.1.1 Mean Filter

Linear smoothing filters use weighted sum of the pixels in successive windows. The size of the defined neighborhood can differ. For example, a two-dimensional 3×3 filter requires all the pixels in the range $[(x-1, y-1), (x+1, y+1)]$ and the weight for each pixel is reciprocal to the total number of pixels in the neighborhood, in this case $w = \frac{1}{9}$. A weighted average filter hardware implementation for size 3×3 requires less computation compared to regular mean filter since division by 16 can be performed by shifting operation instead of using division. The convolution kernel for weighted average filter is:

$$H_{WA}^{2D} = \frac{1}{16} \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}. \tag{6}$$

2.1.2 Gaussian Smoothing Filter

Gaussian filters are class of linear smoothing filters with the weights fitting the shape of a Gaussian function. The two-dimensional Gaussian function is defined as:

$$G_{2D}(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{7}$$

where σ is the standard deviation of Gaussian function. It determines the level of smoothing in the image i.e. the width of Gaussian filter. The Gaussian kernels are rotationally symmetric and the amount of smoothing performed by the filter is the same in all directions.

The limiting factors for implementation of convolution functions of large sizes in some cases can be solved by two one-dimensional convolution. However, only 2D filters characterized by property of separability may be decomposed into 1D convolutions

$$H(x, y) = H(x) * H(y). \tag{8}$$

and their execution by means of a couple of cascaded 1D convolutions is presented as:

$$\begin{aligned} Y(x, y) &= H(x) * H(y) * I(x, y) = H(x) * [H(y) * I(x, y)] \\ &= \sum_k H_x(k) \sum_l H_y(l) I(x-k, y-l). \end{aligned} \tag{9}$$

Large Gaussian filters can be implemented very efficiently due to the property of separability of Gaussian functions. Two-dimensional convolution kernel can be presented by product of two one-dimensional vectors as given:

$$G_{2D}(x, y, \sigma) = G(x, \sigma) * G(y, \sigma) \tag{10}$$

$$= \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{\sqrt{2}\sigma}} * \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{\sqrt{2}\sigma}} \tag{11}$$

In this case image is convolved with a one-dimensional Gaussian vector and then the result is convolved with the same vector oriented orthogonally to the vector used in the first convolution phase. The convolution of an image with a filter kernel of size $N \times N$ implies

N^2 operations. When filter is presented as the outer product of two $1 \times N$ vectors, number of operations is decreased from N^2 to $2N$ operations. For example, in the case of 5×5 Gaussian filter the convolution by 1×5 separable horizontal and vertical filters requires 10 computations versus 25 for the 2D kernel. The computational saving increases with the increasing kernel sizes as presented in Table 1.

Table 1 Number of multipliers and adders required for 2D and 1D convolution

	2D convolution		1D convolution	
Kernel Size	N_{mult}	N_{adders}	N_{mult}	N_{adders}
5×5	25	24	10	8
7×7	49	48	14	12
9×9	81	80	18	16
11×11	121	120	22	20
13×13	169	168	26	24
15×15	225	224	30	28

2.2 Edge Detector Differentiation Phase

2.2.1 Sobel Edge Detector

The Sobel Edge Detector [17] defines convolution kernels for creating corresponding gradient magnitudes, and 5×5 Sobel horizontal kernel is given by:

$$H_x = \begin{pmatrix} 1 & 1 & 0 & -1 & -1 \\ 1 & 1 & 0 & -1 & -1 \\ 2 & 2 & 0 & -2 & -2 \\ 1 & 1 & 0 & -1 & -1 \\ 1 & 1 & 0 & -1 & -1 \end{pmatrix} \tag{12}$$

whereas Sobel vertical kernel is transposed matrix of H_x:

$$H_y = H_x^T. \tag{13}$$

2.2.2 Laplace Edge Detector

Laplacian represents a 2-D measure for computing the second derivative:

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}. \tag{14}$$

This filter determines regions of the image with pixel intensity change. The Laplace edge detector uses the zero-crossing of second derivative of the pixel in order to identify an edge in the process of edge detection.

2.2.3 LoG Edge Detector

The convolution operation is commutative, thus, smoothing and differentiation in sequence can be combined into a single operator $\nabla^2 G_\sigma$. The combination of Gaussian smoothing filter and Laplacian edge detection filter known as the Laplacian of Gaussian (LoG) is presented in the form:

$$LoG(x, y, \sigma) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2 + y^2}{2\sigma^2}\right] e^{-\frac{x^2+y^2}{2\sigma^2}}. \tag{15}$$

2.2.4 Differences of two Gaussian Filters - DoG Filter

The separability property of a 2-D function is determined by the rank of the kernel and for Gaussian and Sobel functions, the separability property is fulfilled. In the case of LoG function, decomposition into two 1-D functions cannot be performed and thus the LoG operator is not separable. Due to the lack of separability property, computational inefficiency is the limitation of the LoG filtering in particular in the case of large kernels. An approximation of the LoG operator is achieved by the use of differences of two Gaussian functions (DoG), which are themselves separable functions. DoG kernel operator is defined by:

$$\begin{aligned} DoG_{2D}(x, y, \sigma) &= G(x, y, \sigma_1) - G(x, y, \sigma_2) \\ &= \frac{1}{2\pi}\left(\frac{1}{\sigma_1^2} e^{-\frac{x^2+y^2}{2\sigma_1^2}} - \frac{1}{\sigma_2^2} e^{-\frac{x^2+y^2}{2\sigma_2^2}}\right) \end{aligned} \tag{16}$$

The subtraction of two Gaussian functions with different standard deviations is a bandpass filter allowing extraction of image features in the frequency band of interest which tend to correspond to edges.

2.3 Edge detection Enhancement - Binarization

Image binarization, or thresholding, represents a pixel-based decision function. The gray value of a pixel gets black or white value depending on the defined threshold:

$$I_{threshold} = \begin{cases} 0 & I < Threshold \\ 2^L - 1 & I > Threshold \end{cases} \tag{17}$$

where L represents number of bits per pixel.

3 Canny Edge Detection Algorithm

The Canny edge detector [18] is among the most widely used edge detection algorithms. Among existing algorithms, Canny edge detection filter is optimal in addressing detection, localization and number of responses. The criteria of localization assumes detected edges and real edges to be close to each other. It is computationally more insensitive when compared with other algorithms and as it is based on frame-level statistics, it is characterized by high latency. The algorithm runs in five separate steps:

- Smoothing
- Computation of horizontal and vertical gradients, gradient magnitude and orientation in each pixel location
- Non-Maximal gradient Suppression (NMS)
- Double Thresholding
- Edge Tracking

and its dataflow is presented in Figure 2. Many implementations of the Canny algorithm have been proposed on a variety of hardware platforms. As aforementioned, the implementations [7, 8, 9, 10] are FPGA-based. The 4-pixel parallel implementation [8] results in a speed-up in the processing time, however it also results in an increase of memory organization and control resources. A distributed canny edge algorithm proposed by [9] adaptively computes the edge detection threshold based on the block type and the local distribution of the gradients within the image block. Canny filtering is determined by the width in the smoothing phase and thresholds in the edge decision phase.

3.1 Smoothing

Gaussian smoothing filters are commonly used in Canny edge detection filters to reduce the noise level in an image and to improve the edge detection performance by smoothing false edges. In order to implement the Gaussian smoothing filter on the FPGA platform, the smoothing kernel is rounded to fixed-point data format.

3.2 Computation of horizontal and vertical gradients

In this phase, there can be used a number of gradient functions, such as Sobel filter. However, the gradient computation can be performed using gradient masks designed to approximate 2D partial derivatives of the Gaussian function as follow:

$$Grad_x(x, y, \sigma_1) = -\frac{x}{\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{18}$$

$$Grad_y(x, y, \sigma_1) = -\frac{y}{\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{19}$$

The choice of σ affects the performance of the Canny detector and its value can be adapted to the noise or blur in the image. Larger values of σ increases noise resilience, whereas lower values provide better edge detection. The choice depends on the present noise characteristics and image content and statistics.

3.3 Gradient Magnitude and Angle Computation

The gradient magnitude computation sums the absolute values of horizontal gradient G_x and vertical gradient G_y at each pixel location as in Sobel edge detection algorithm. However, Canny detector uses also information about the gradient direction $\alpha(G)$ for edge map computation. The gradient direction falls in one of 4 possible angle ranges (0°, 45°, 90°, 135°).

3.4 Non-Maximal Suppression - NMS

Both magnitude and direction are used in order to determine one-pixel wide edges. For the gradient directions in between the main directions, an interpolation is performed and the direction is set to one of four principle directions. Then, NMS performs comparison of the gradient magnitude of the active pixel with gradient magnitudes of neighboring pixels along the chosen principle gradient direction. The gradient magnitude is set to zero in the case it is not local maximum.

3.5 Double Thresholding

After the search for local maximum on the edge direction, double thresholding is proposed. It defines two thresholds, T_{LOW} and T_{HIGH}, and their computation is based on the histogram of the gradient magnitude for the entire input images. The high threshold T_{HIGH} is computed by defining a percentage of the total pixels to be classified as strong edges, whereas the low threshold is computed as a certain percentage of the high threshold. The typical values of these percentages are 20% and 40% for high threshold percentage and low threshold percentage, respectively. However, most of the existing implementations perform threshold computation offline and use the same fixed pair of threshold values for all images. Even if this results in a decreased edge detection performance, the latency is considerably lower.

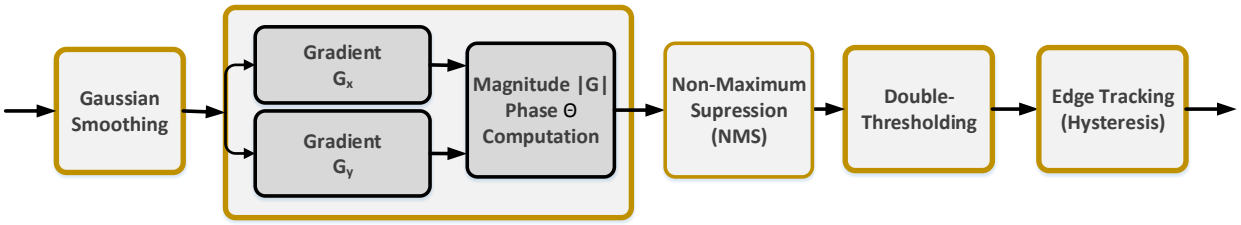

Fig. 2 Canny filter dataflow

In order to create the edge map, a double thresholding is performed to gradients for classifying pixels as definitive, possible or non- edges as follows:

$$G_{edge} = \begin{cases} 0 & |G| < T_{LOW} \\ 2^{L-1} & T_{LOW} < |G| < T_{HIGH} \\ 2^{L} - 1 & |G| > T_{HIGH} \end{cases} \quad (20)$$

If the gradient magnitude of a pixel is greater than the high threshold, the pixel is set to be a strong edge and it can be included in the final edge map. Weak edges, pixels whose gradient magnitude is in between low and high threshold, are included in the edge map only if they are connected to the strong edges.

3.6 Edge Tracking

The search for strong edge pixel candidates among weak edges is performed in the last Canny detector phase. It consists of strong edge pixel search in the 4-neighborhood, i.e. every pixel that touches one of the current pixel edges or 8-neighborhood i.e. every pixel surrounding the current pixel in various implementations. Both edge tracking algorithms require caches for storing data computed after thresholding phase. A procedure for on-fly comparison of a current pixel with the values of three neighboring pixels above the current pixel and left neighboring pixel presented by Gentsos *at al.* [8] does not require additional cache resources. If any of 4-neighborhood pixels are strong edges, the candidate pixel is included in the edge map.

4 Run-time Partial Reconfigurable Computing Technology

Nowadays, FPGAs contain a large amount of hardware logic such as look-up tables, flip-flops and build-in components as memories, processors and multipliers. Large variety of components gives possibility for intensive SoC development within a single FPGA. In addition, the DPR feature which allows the reconfiguration of a portion of FPGA, while the rest of the FPGA logic (static logic) retains the old functionality and keep running. It relies on the assumption that time-multiplexing of independent operations might significantly reduce the FPGA resource demands and increase the flexibility in choices of algorithms or protocols by loading a partial configuration bitstream of the algorithm of interest in the requested time. DPR improves FPGA fault tolerance and may reduce the resources required for implementing an algorithm. It leads to selective load of the processing core within the period in which a particular function is requested and to replacement of active functionality with other functions on the same chip.

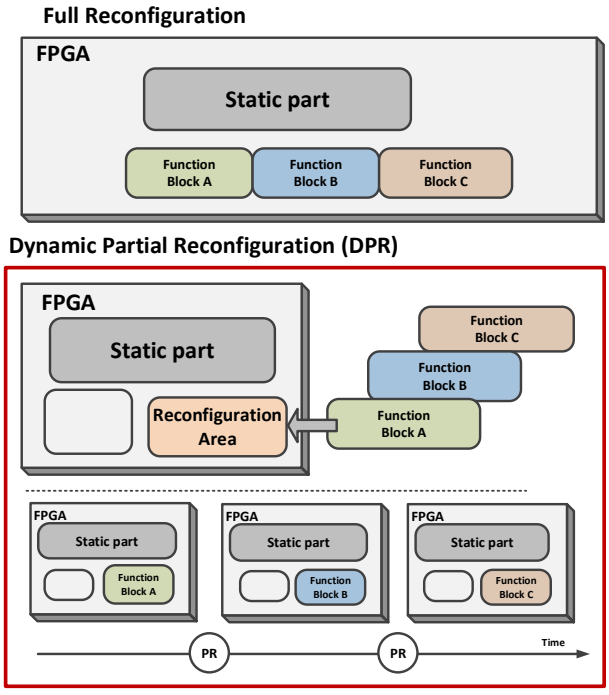

Fig. 3 Basic mechanism of Partial Reconfiguration

The techniques for placement and interaction of reconfiguration modules with reconfigurable partitions dif-

fer between manufactures. The partially reconfigurable region (PRR) on FPGA is a block where reconfigurable modules (RM) are deployed at run time. A DPR system can have multiple PRRs, but only one RM can be active within a PRR. Furthermore, DPR requires static region around the reconfigurable partition in order to provide a safe and efficient routing. Fig. 3 illustrates a DPR process with one PRR and three RMs. Earlier most common communication link between static and dynamic logic was established by Bus Macros (BM) which were manually added by the user during the design phase. BM are pre-routed slice based elements made of LUTs. Nowadays, Xilinx Partial Reconfiguration [19] uses proxy logic insertion during the synthesis, whereas GoAhead tool [20] avoids additional logic by using blockers in the reconfigurable partition.

DPR process defines two phases: the first phase includes fetching and storing the required partial bitstreams in the local memory such as Flash memory, SRAM or BRAM memory. This phase is not time-critical. The second phase loads bitstreams into reconfiguration regions including the transfer of partial bitstreams from the on-board memory to the ICAP controller. The key consideration in the second phase is the reconfiguration time. The reconfiguration time, measured by a timer peripheral, is defined as the time that elapses between the request for loading a new functionality of the co-processor and the time point at which the co-processor's new functionality gets active. The fundamental limiting factor that impacts the reconfiguration time is the speed of writing data to the configuration memory. The hard macro in FPGAs that serves the purpose of controlling reconfiguration by writing to the configuration memory is the Internal Configuration Access Port (ICAP). Xilinx's proposed partial reconfiguration flow uses the OPB-HWICAP [21], XPS-HWICAP [22] or the fastest AXI-HWICAP [23] cores to control the ICAP. Theoretical throughput for ICAP is achieved if partial bitstream data comes in every clock cycle. The ICAP with a 32-bit, 100 MHz streaming interface provides up to 400 MB/s reconfiguration throughput. However, a number of alternative reconfiguration controllers [24, 16, 25] have been implemented in order to speed up the reconfiguration process.

5 Core Development

In general, the motivation for using hardware implementation in general is to reduce the execution time of algorithms compared to the software execution. By involving additional hardware resources, the computational power of the complete system is increased. The 2D convolution modules within the proposed adaptive video filtering system are fully described in VHDL hardware description language. Structural features such as kernel size in both dimensions, pixel depth and kernel coefficients for this system can vary. The number of additions and products grows exponentially depending on the kernel dimensions as presented in the previous sections. In the case of some filters, separable vectors are used to reduce critical path and, thus, increase the operating frequency. Different levels of computation granularity such as pixel-, block- or frame-level can be required for implementing various video processing algorithms. These applications require a wide use of shift registers, memory windows and line buffers providing data context within the certain application. Adjacent pixels require memory structure consisting of only a few flip-flops. Pixel-level processing manipulates only one pixel or one array of pixels within an image/frame scan line, whereas shift registers provide a one-dimensional temporary data buffer for storing incoming pixels. A typical signal processing application which uses shift register for storing is the one dimensional finite impulse response (FIR) filter. The one-dimensional average filter uses a shift register presented in Fig. 4. The size of the FIR filter determines the size of the shift registers and the number of multiplications and additions. For two pixels at the same location in adjacent scan lines, an entire line of storage (line buffer) is required. For two pixels at the same location in two different frames, frame buffers are used.

Typically, line buffers are implemented as block RAMs to avoid the communication latency introduced by communication with off-chip DRAM memory. The depth of the RAM blocks is constrained to the width of the input image, whereas the number of line buffers is set to $M-1$ depending on the kernel height M and together with the current incoming video data signal compose an n-parallel input into convolution kernel. In this particular realization, the depth of line buffers is set to 512 8-bit pixels for testing purposes, whereas there is no restriction of the maximum height of the image. In the case of 3×3 kernel, two line buffers are used for storing two input image lines. The stored pixels are, then, used for convolution operation as central block pixel and as a neighbor pixel in convolution process of the surrounding pixels. The duration of the usage of the pixel is determined by the kernel size. The processing of the image borders is performed by zero padding to guarantee the same output image size as input image size. The intensity computation of central pixel is done by multiplying the original pixels block $N \times M$ from the image window with the kernel coefficients. In order to perform the computation, a memory window of kernel size is implemented. The memory window is a two di-

mensional grid of shift registers. In each clock cycle, the data from line buffers and input register are shifted into the grid, the data from each register in the memory window are shifted to the right neighbor register and the data from the grid are used for computing convolution for the active central input pixel. In the proposed implementation, the multiplication is performed only by use of adding and shifting operations. The computed partial products are then summed by the use of an adder tree and the total sum is delivered as output sample in each clock cycle. The block diagram of the Laplace and LoG edge detection filters are presented in Fig. 5 and Fig. 6, respectively.

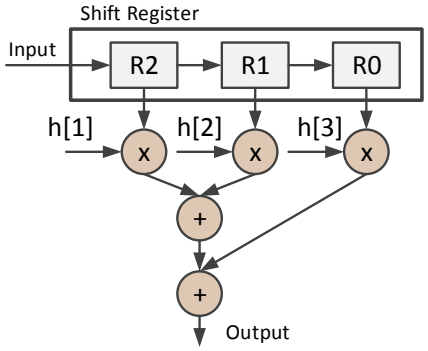

Fig. 4 Use of shift register in one-dimensional averaging filter

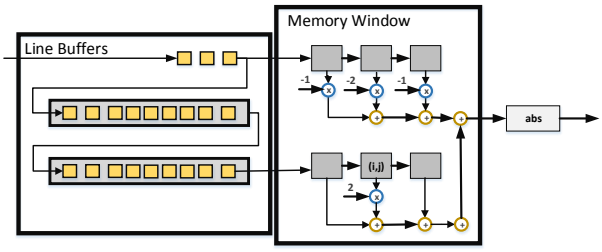

Fig. 5 Sobel Edge Detector realization (y direction)

In the caseof a separable function, the following property of convolution is used:

$$\begin{aligned} Y(x,y) &= [I(x,y) * H_1(x,y)] * H_2(x,y) \\ &= I(x,y) * [H_1(x,y) * H_2(x,y)] \,. \end{aligned} \tag{21}$$

Separable filter hardware realization, when applicable, provides lower latency and buffer size. For example, 2-D Gaussian kernel G, mean kernel and 2-D Sobel kernel are separable and therefore they can be implemented by two successive one-dimensional operators where one is applied horizontally and the other vertically as presented in Fig. 7. The results of column-wise 1-D convolutions for both horizontal and vertical computations are stored in local cache of the size $1 \times N$, and then are convoluted with row-wise vectors in order to produce final gradients. The gradient magnitude is computed as the sum of absolute values of horizontal and vertical gradient (Eq. 4). Separability property is used for building filters with large kernels whereas 2-D convolution is used for designing filters with kernels of sizes $N < 5$ due to faster memory access at the considerable low increase of arithmetic operations. The coefficient values for a number of separable functions which are used in implementing edge detection filters are given in Table 2.

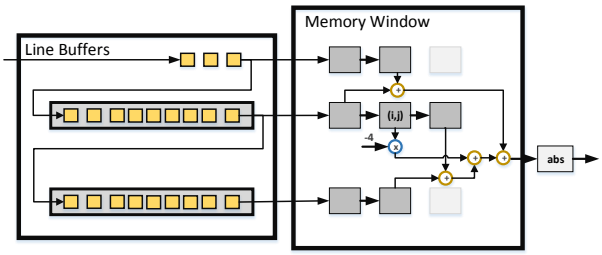

Fig. 6 LoG Filter realization

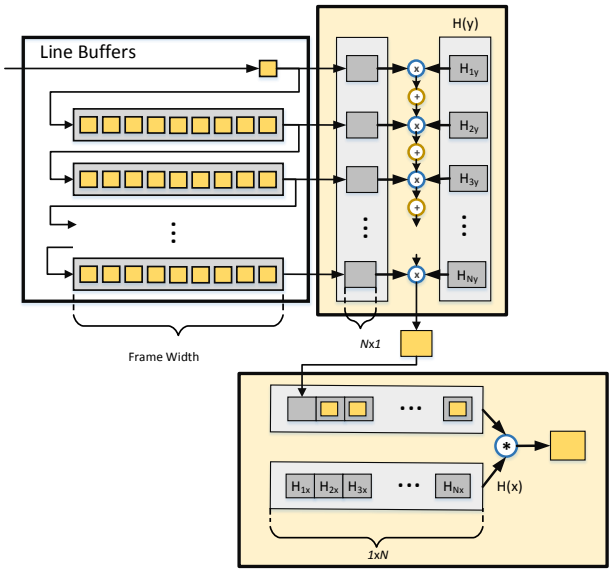

Fig. 7 Separable 2-D filter $H(x,y)$ realization

The separability property does not hold for LoG nor DoG operator or their extensions, and computational inefficiency for these operators is introduced for larger

kernel sizes. For improving the efficiency of DoG operator, the separability of the Gaussian functions is used by performing convolution of an input image I with two Gaussian kernels with standard deviations σ_1 and σ_2 using 1D Gaussian operators in both directions as presented in Fig. 8. The resulting two smoothed images I_{G1} and I_{G2} are used for computation of DoG $D_G = I_{G1} - I_{G2}$.

Table 2 Coefficients for Separable Vectors in Large Filter Kernels

Gaussian, 5×5, $\sigma = 1$
-0.0625 -0.25 -0.375 -0.25 -0.0625
Gaussian, 7×7, $\sigma = \sqrt{2}$
-0.030 -0.105 -0.222 -0.258 -0.222 -0.150 -0.030
Gaussian, 7×7, $\sigma = 2.3$
-0.085 -0.136 -0.180 -0.198 -0.180 -0.136 -0.085
Gaussian, 9×9, $\sigma = \sqrt{2}$
-0.005 -0.030 -0.104 -0.22 -0.282 -0.22 -0.104 -0.030 -0.005
Gaussian, 9×9, $\sigma = 2.3$
-0.040 -0.078 -0.125 -0.165 -0.182 -0.165 -0.125 -0.078 -0.040
Grad(Gaussian), 7×7, $\sigma = \sqrt{2}$
-0.109 -0.140 -0.131 0 0.131 0.139 0.109 -0.021 -0.072 -0.153 -0.197 -0.153 -0.072 -0.0207
Grad(Gaussian), 9×9, $\sigma = \sqrt{2}$
-0.037 -0.074 -0.142 -0.133 0 0.133 0.142 0.074 0.037 -0.003 -0.020 -0.069 -0.147 -0.189 -0.147 -0.070 -0.020 -0.003

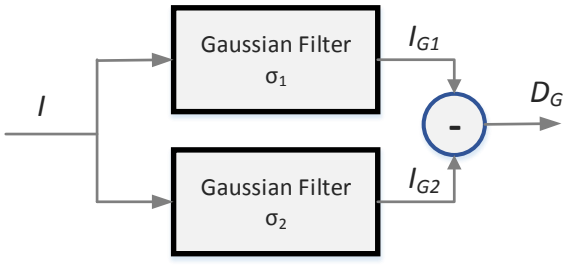

Fig. 8 DoG Filter realization

5.1 Canny Edge Detector Implementation

The proposed architecture for Canny edge detector consists of three stages as shown in Fig. 9. The stages within the architecture are convolution stages performing operations such as smoothing and computation of image gradient, followed by the NMS stage and double thresholding and a final edge tracking stage. Each of the stages is equipped with line buffers required for adapting the input data to block processing.

5.1.1 Convolution Stage

The convolution computing core processes an $N \times N$ overlapping image blocks by performing separate or combined smoothing and gradient functions. In this work both scenarios are implemented - Canny edge detector (ED Filter V) computes "Gradient of Gaussian" of the original image, where partial derivatives of smoothing Gaussian function with standard deviation $\sigma = \sqrt{2}$ are computed and approximated by 2D masks further split into two 1-D vectors. The second convolution phase implementation of Canny edge Detector (ED Filter VI) consists of separate Mean filter and Sobel filter cores.

The horizontal and vertical gradients are required for computation of the gradient magnitude and angle further used in the directional NMS module. In order to avoid the realization of the *arctan* function in hardware, the following approximation for gradient angle is used:

$$\alpha \approx \frac{G_y}{G_x}. \tag{22}$$

5.1.2 NMS Realization

In NMS phase, the directions of interest are limited to four ranges as shown in Fig. 10. The edge angles defining ranges of interest are $\{\pm\frac{\pi}{8}, \pm\frac{3\pi}{8}, \pm\frac{5\pi}{8}, \pm\frac{7\pi}{8}\}$. The values of these edge angles and their fixed-point approximation are given in Table 3. The direction of the maximum gradient is defined in Table 4 based on three parameters: the angle range of interest, the ratio of G_y and G_x and their signs S_{G_y} and S_{G_x}. NMS uses 8-neighbors window for determination of the local maximum. Thus, a three-line buffer for storing gradient magnitudes from the three frame/image rows together with a buffer which keeps track of computed direction (2-bit per pixel) are used. If the gradient of the current pixel is the largest among neighbors on the chosen direction, it is taken as an edge pixel, otherwise it is suppressed.

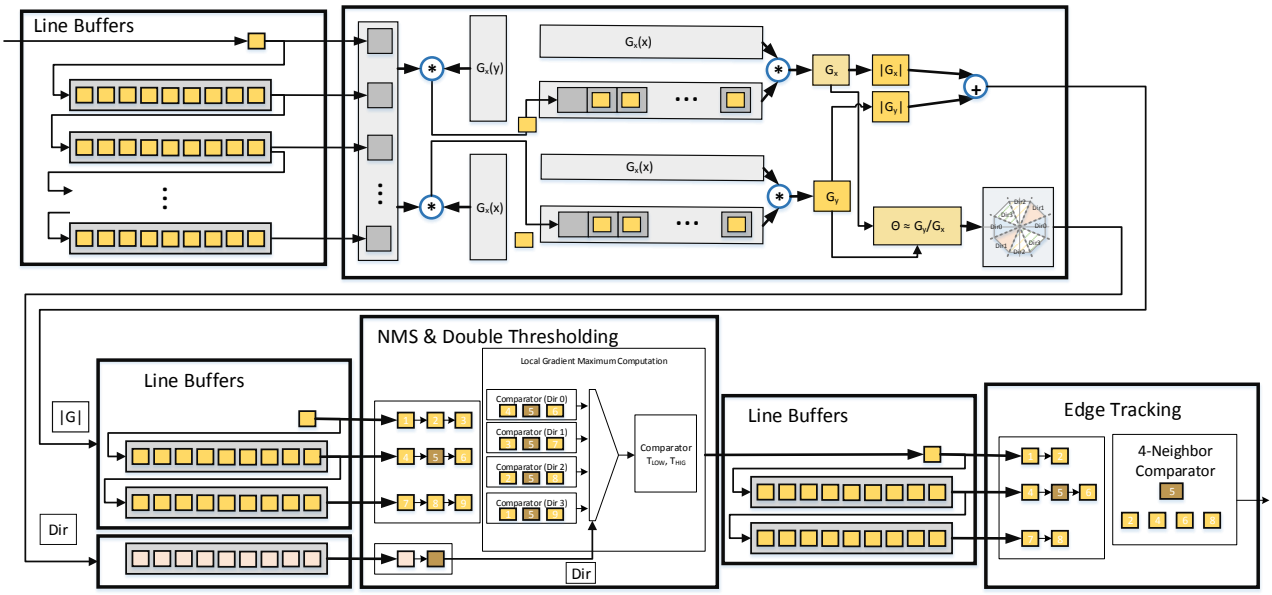

Fig. 9 The architecture of the proposed Canny edge detector algorithm

Table 3 Angles of interest for NMS function

Angle of Interest	Value	Fixed point value
$\tan(\pi/8)$	0.41	$1/2 - 1/16 - 1/32 + 1/256$
$\tan(3\pi/8)$	2.41	$2 + 1/2 - 1/16 - 1/32 + 1/256$

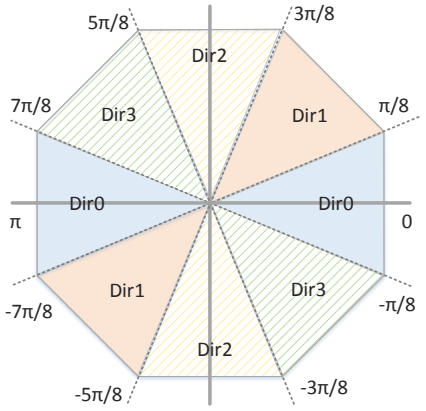

Fig. 10 Selection of Edge Direction based on the angle values

5.2 Thresholding

The edges affected by noise are preserved after using NMS stage. Thus, the edges created by the noise are further examined by double thresholding. This stage is in pipeline with NMS stage. In this particular implementation, thresholds T_H and T_L are fixed to predetermined values.

Table 4 Selection of Edge Direction based on the angle values

Direction	Range	Magnitude Sign
Dir0	$\lvert G_y\rvert < 0.41\lvert G_x\rvert$	–
Dir1	$0.41\lvert G_x\rvert < \lvert G_y\rvert < 2.41\lvert G_x\rvert$	$S_{G_y} = S_{G_x}$
Dir2	$\lvert G_y\rvert > 2.41\lvert G_x\rvert$	–
Dir3	$0.41\lvert G_x\rvert < \lvert G_y\rvert < 2.41\lvert G_x\rvert$	$S_{G_y} \neq S_{G_x}$

5.3 Edge Tracking

Comparison of the current candidate for a strong edge with its neighboring pixels is performed in the final phase. In this work, 4-pixel edge tracking is chosen for implementation, where upper, lower, left and right neighbors are considered for final decision to define a pixel as the edge pixel. If any of these pixels is strong edge, current pixel is also marked as strong edge. The proposed 4-pixel edge tracking requires data from three image lines, thus usage of two line buffer.

6 System Architecture

The proposed FPGA SoC architecture contains a set of IP cores, some of which are adaptive, connected through a common bus interface. The developed Edge Detection

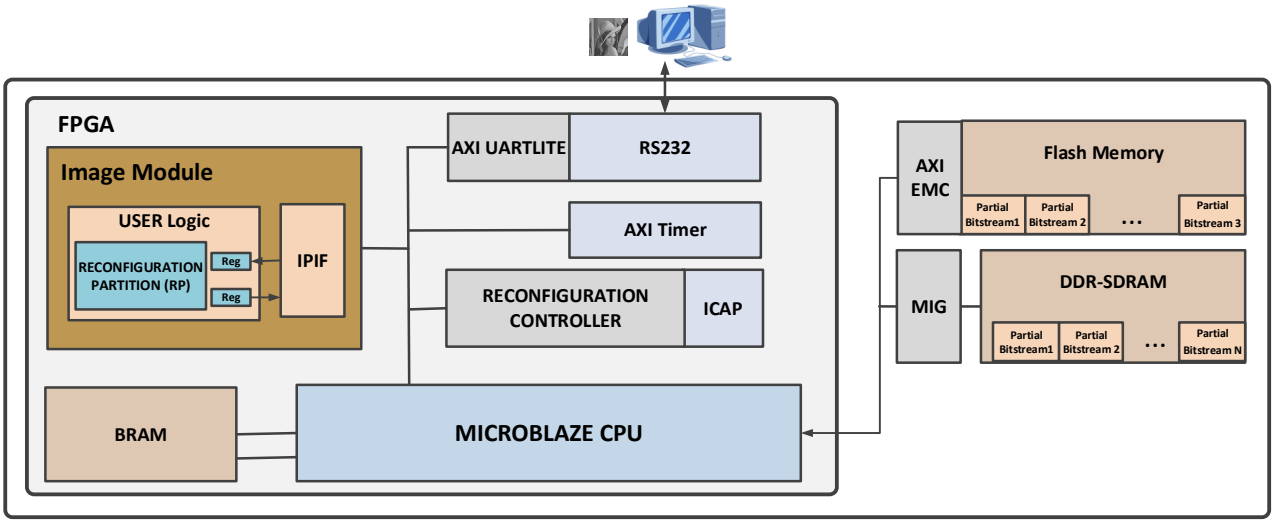

Fig. 11 System on Chip Communication with User Peripheral

filter core is with the capability of adjusting processing features as commanded by the user. This adaptation acts upon the measured component performance, in this case quality of edge detection, trying to fulfill the requirements given by the user. The flexibility introduced by FPGA implementation featured with DPR is based on the concept that mutually exclusive tasks can execute on a partition within a single device. The proposed self-reconfigurable adaptive video filtering system is presented in Fig. 11. The main components of the reconfiguration process are a reconfigurable core (RC) and a reconfiguration engine (embedded Microblaze). The RC is placed on a well-defined partially reconfigurable region within the programmable logic. The RC logic is wrapped by an additional logic with interfaces required to communicate with the processor. The self-reconfiguration using the internal ICAP configuration port provides the system with the required adaptation capabilities. The AXI-HWICAP controller is used for establishing communication with ICAP. The static region is composed of a 32-bit soft-core Microblaze processor representing system CPU. The processor controls the reconfiguration process by issuing the required commands to configure the core to the candidate reconfiguration modules. The additional peripherals such as timer for computing the reconfiguration time and serial communication for communication with the external data source are included in the system. External memories, both volatile and non-volatile, are required for storing both processing data and configuration information (complete and partial bitstreams). The communication between peripherals and memories is mediated by use of the processor unit. A serial RS232 communication interface, the link between the system and external world, transfers the original and processed images/frames from and to a PC. The timer, UART and interrupt controller, standard peripherals for Microblaze core, are interconnected through AXI-Lite processor bus, industrial standard interface adopted by Xilinx.

The dataflow chart in Fig. 12 shows the communication for performing self-reconfiguration. There are presented the phases of the reconfiguration process and algorithm choices with respect to the choice of parameters' values within different phases of edge detector algorithms. Six edge detection algorithms differing in size of convolution matrices, complexity of detection process and thresholding techniques are proposed for testing purposes. Scenario includes dynamic on-fly reconfiguration between these edge detection algorithms. The kernel sizes of proposed algorithms are fixed and predetermined varying from 3×3 - 9×9. The system is connected to the PC for sending the partial bitstream to the memory, for communication with the processor and for sending system specification including the reconfiguration commands. After establishing the communication with the memory to store bitstreams, original image data are transmitted to the processor by serial communication and are further sent to the RC. Data processed by the active reconfigurable core is sent back through the processor to the external PC.

Both Linear flash and SRAM memories are used and tested for storing full and partial bitstreams. The partial bitstreams are placed in the independent addresses of the memory and their reading is performed by ICAP programming when the reconfiguration phase starts. The full bitstream is stored on flash memory to

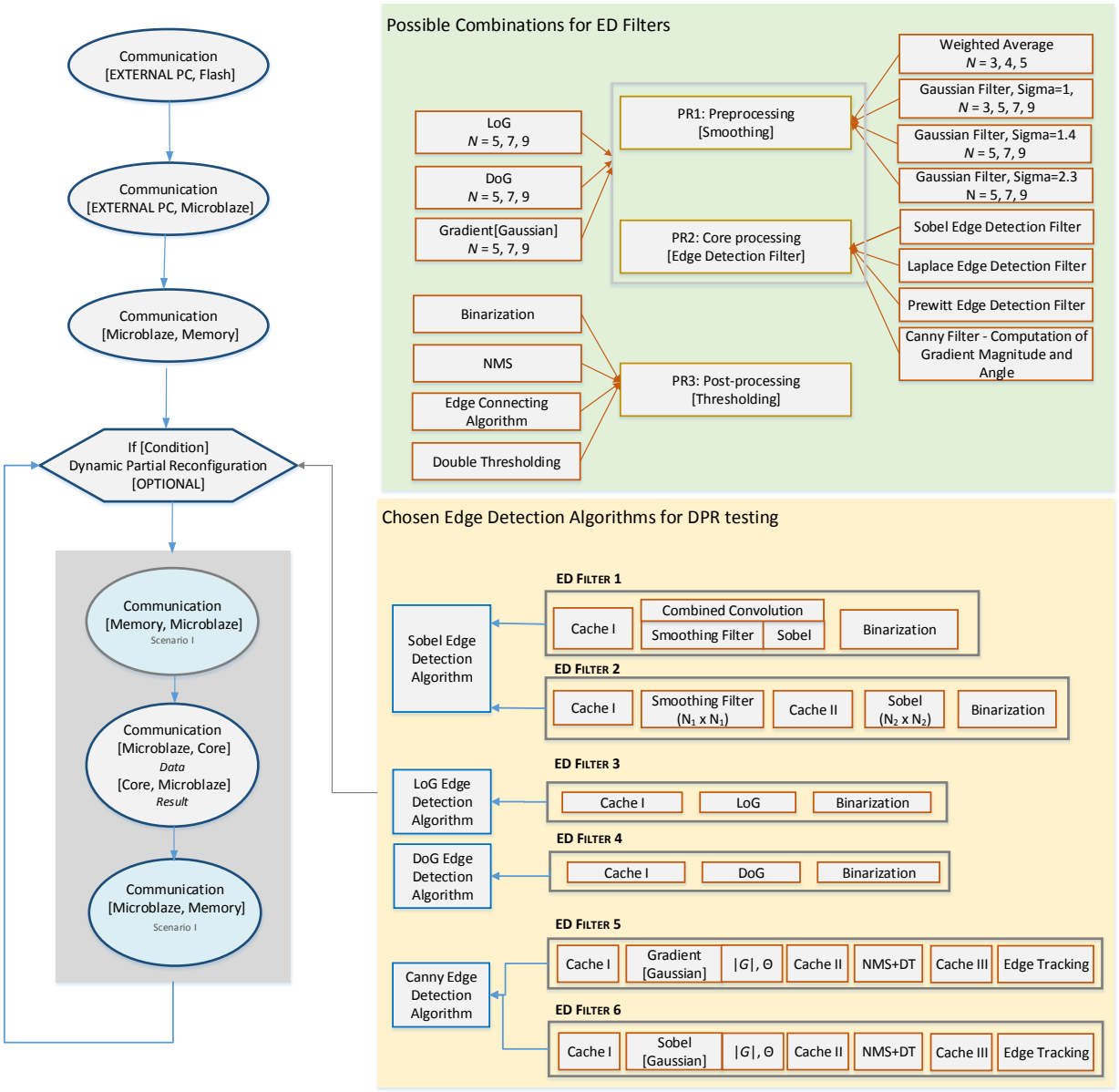

Fig. 12 Dataflow in the proposed self-reconfigurable system

be booted up into the FPGA at the power up. The reconfiguration time is measured from the start of the bitstream reading until the AXI HWICAP finishes the last configuration write. This time represents theintroduced latency of the complete reconfiguration process relevant for the possibility of running application in real-time.

7 Results

This section details the hardware resource usage, the optimization of the reconfiguration space and the analysis of the results used for testing. Concerning the area performance of implemented core functionality, Table 5 shows the resources required to build each stage of the edge detector filter algorithms, in particular convolution processors with various kernel sizes and proposed complete edge detection variants. First, the SoC with six edge detection cores is placed onto Kintex 7 board, and the functionality of each of the edge detection filter is tested. Testing was performed by comparing fixed-point FPGA results with corresponding Matlab implementation. The proposed architectures of edge detec-

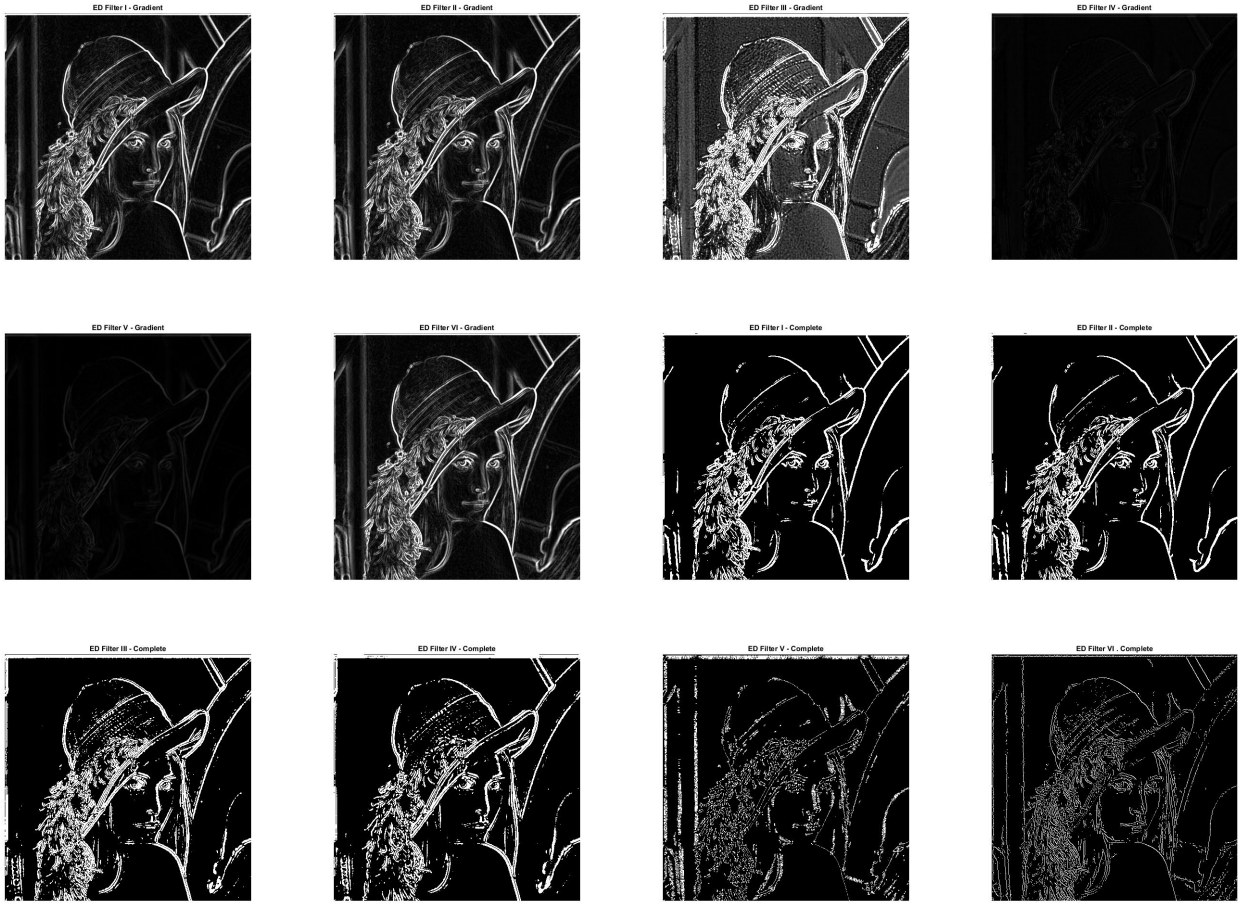

Fig. 13 Resulting images after filtering image "Lena" with chosen filters in Matlab

tion algorithms are tested on an 8-bit gray scale image Lena of size 512×512. Fig. 13 shows some results from convolution stage and final results from edge detection gathered in Matlab for image Lena.

The cores are designed in Xilinx ISE, whereas the functional and timing verification are performed on Kintex 7 board. However, due to the fact that only one core is chosen to perform the edge detection, these cores can be multiplexed in time in a reconfigurable manner within the same FPGA partition.

The implementations of the filters are adapted for communication on FPGA chip, i.e. the input and output signals are 32-bit wide and it is assumed to receive one pixel (32 bit) per clock cycle. The surrounding microprocessor system dictates 100 MHz clock frequency. The tools used for partial reconfiguration of adaptive video filtering system implementation are composed of Xilinx Design Tools - EDK 14.7, ISE 14.7, SDK 14.7, Impact and PlanAhead. After generating netlists of the RC modules in ISE, EDK tool is used to build a processor system with interfaces and memories. The processing system AXI Lite bus links the Microblaze processor to the interfaces and the reconfiguration partition. PlanAhead is used to constraint the floorplan, to place the reconfigurable module within the FPGA and to generate the full and partial bitstreams. Finally, Impact tool programs the flash memory with the full bitstream for boot up and partial bitstreams declared as data at defined memory ranges. Software development of the drivers and embedded applications for Microblaze processor is done within SDK tool. A detailed overview of the design floorplans with DPR and without DPR are shown in Fig. 14. The reconfigurable region is placed in the upper left quadrant. The pink rectangle represents the constrained region of the reconfigurable core. The size of the reconfigurable region is chosen considering the resource utilization of each reconfigurable partition and is set so that it fits the largest possible realization. Matlab program adapted for serial communication is used to send original data and fetch the output data from the SoC. The initial configuration of the recon-

Table 5 Hardware Resources for various Image/Video Filters [512 × 512]

Submodules	Kernel Size	Sigma	Slices	LUTS	Flip-Flops	BRAM Memory [kbits]	Operating Frequency [MHz]
Mean Filter	3×3	-	43	109	92	8	496
Gaussian Filter	5×5	1	110	350	249	16	360
Gaussian Filter	9×9	$\sqrt{2}$	215	696	470	32	267
Gaussian Filter	9×9	2.3	289	686	457	32	306
Sobel Filter	3×3	-	82	190	92	8	496
Sobel Filter	5×5	-	181	383	402	16	375
Laplace	3×3		67	131	92	8	430
LoG	3×3	$\sqrt{2}$	53	102	80	8	430
LoG	5×5	$\sqrt{2}$	94	191	260	16	387
Gradient of Gaussian	9×9	$\sqrt{2}$	601	1488	925	32	290
Binarization	1×1	-	5	9	-	-	
NMS + Double Thresholding +	3×3	-	48	86	110	9	496
Edge Tracking +	3×3	-	37	58	78	8	496
Complete Edge Detector Implementation [512 × 512]							
ED 1	[3×3, 3×3]		187	470	415	16	256
ED 2	5×5 [3×3, 3×3]		117	260	182	16	250
ED 3	9×9		383	1026	883	32	310
ED 4	9×9		553	1377	1087	64	267
ED 5	9×9		694	1573	1251	49	290
ED 6	5×5		274	648	607	33	218
Total	5×5		2208	5354	4425	210	218
Edge Detection Reconfiguration Module							
RM	$N \times N$ (up to $N = 9$)		480	1920	3840	288	100 (sys. clock)

figuration partition on RC is Filter V - Canny edge detector and the resulting image after filtering received on the external PC side is given in Fig. 15.

After the partial reconfiguration is performed for Filters I and IV, the resulting images received on the external PC are shown in Fig. 16. The resulting images are compared with the results of the detection process performed in Matlab and no mismatch has been found.

The theoretical reconfiguration throughput of the ICAP suggested by Xilinx, cannot be achieved since the system is not able to provide data in each clock cycle. In this particular case, the partial bitstream of size 164 kB is sent to the programmable logic within 2.6 s. This limits the proposed SoC to the non-time critical applications and directs the issue of the speed-up of partial reconfiguration process towards development of fast internal controller interface for real-time multimedia applications. Apart from the timing limitations, algorithms placed within the same reconfiguration area extend the resources and cut the functionality limitation in the small-scale systems.

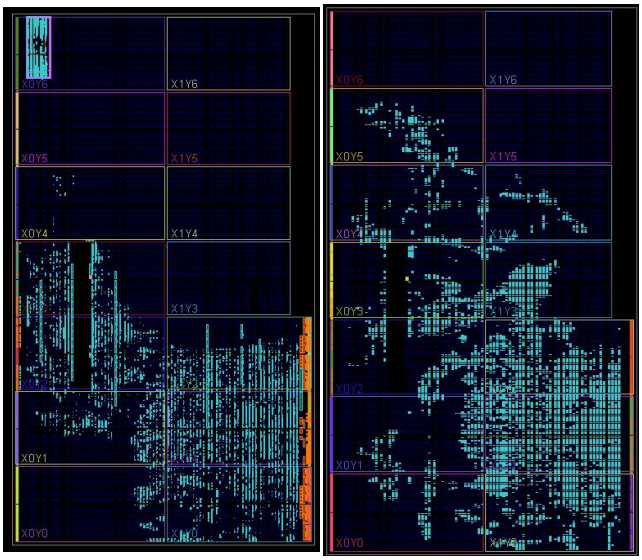

Fig. 14 Chip layout with and without DPR

The proposed SoC implementation gives clear idea about capabilities of the new generations of FPGAs and

Fig. 15 Output image on serial communication port - initial configuration ED Filter V

Fig. 16 Output images after the use of partial reconfiguration - ED Filter I, ED Filter VI

its feature of resource re-usage in time. There are large possibilites for storing large convolution matrix modules, since the complete SoC with Microblaze processor, interfaces and edge detection reconfiguration module utilizes only 15% and 12% of Slices and RAM on Kintex FPGA, respectively. Further development of the proposed implementation of Canny edge detection algorithm with increased IO pixel-parallelism is a good candidate for cutting edge reimplementation. Furthermore, the image processing pipeline can be greatly improved by building stream system discarding microprocessor as intermediate stage in the communication between reconfiguration modules, memory and external sources. Thus, the future work can concern establishment of the direct communication between external source, memory and the reconfiguration region.

8 Conclusion

We have proposed a parallel and pipelined architecture that enables fast and robust edge detection in an embedded system using an FPGA device. A SoC design including a number of edge detection algorithms is proposed. The system offers a variety of functions such as smoothing, first and second derivative edge detectors and segmentation algorithms which can be combined for improving detection process. The chosen combination of available modules varying also in kernel size for each image/video processing phase in edge detection algorithm can be performed within the limited hardware area by use of DPR. By detailed analysis of data dependencies and computation complexity of various phases, in particular convolution phase, a design methodology for efficient paralellization and implementation of algorithms has been followed. The complete system has been validated, simulated, placed and tested on FPGA hardware in communication with external image/video source. There is room for improvements, in particular, concerning increased input parallelism in the current system. Furthermore, the filter pipeline can be greatly improved by building stream system which discards microprocessor for communication purposes. As a future work, the intention is to exclude the processor from the video/image pipeline, to establish the direct communication between DDR memory and the reconfigurable region, and, in this way, to release the processor for system control purposes only. The proposed DPR system includes video algorithms/filters as reconfigurable modules varying in complexity and resource utilization.

References

1. Liu, J and Wang, S and Li, Y and Han, J and Zeng, XY (2010) Configurable pipelined Gabor filter implementation for fingerprint image enhancement. Solid-State and Integrated Circuit Technology (ICSICT), 2010 10th IEEE International Conference on, IEEE, 584–586
2. Wang, XX and Shi, B E (2010) GPU implemention of fast Gabor filters. Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on 373–376
3. Iandola, FN and Sheffield, D and Anderson, MJ and Phothilimthana, PM and Keutzer, K (2013) Communication-minimizing 2D convolution in GPU registers. Image Processing (ICIP), 2013 20th IEEE International Conference on, 2116–2120
4. Lourenzo LHA, Weingaertner D and Todt E (2012) Efficient Implementation of Canny Edge Detection Filter for ITK Using CUDA. Computer Systems (WSCAD-SSC), 2012 13th Symposium on, 33-40
5. Luo, Y and Duraiswami, R (2008) Canny edge detection on NVIDIA CUDA. Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on, 1–8
6. Rao, L and Zhang, B and Zhao, J (2015) Hardware Implementation of Reconfigurable 1D Convolution. Journal of Signal Processing Systems, Springer, 1–16
7. He, W and Yuan, K (2008) An improved Canny edge detector and its realization on FPGA. 7th

World Congress on Intelligent Control and Automation, IEEE, 6561–6564
8. Gentsos, C and Sotiropoulou, CL and Nikolaidis, S and Vassiliadis, N (2010) Real-time canny edge detection parallel implementation for FPGAs. 17th IEEE International Conference on Electronics, Circuits, and Systems (ICECS), IEEE, 499–502
9. Xu, Q and Varadarajan, S and Chakrabarti, C and Karam, LJ (2014) A distributed canny edge detector: Algorithm and FPGA implementation. IEEE Transactions on Image Processing 23(7), 2944–2960
10. Sangeetha, D and Deepa, P (2016) FPGA implementation of cost-effective robust Canny edge detection algorithm. Journal of Real-Time Image Processing, 1–14
11. Huang, J, Parris, M, Lee, J and Demara, RF (2009) Scalable FPGA-based architecture for DCT computation using dynamic partial reconfiguration. ACM Transactions on Embedded Computing Systems 9(1)
12. Ahmad, A and Amira, A and Nicholl, P and Krill, B (2013) FPGA-based IP cores implementation for face recognition using dynamic partial reconfiguration. Journal of Real-Time Image Processing 8(3), Springer, 327–340
13. Lee, J and Ryu, C and Kim, S (2012) Self-reconfigurable approach for computation-intensive motion estimation algorithm in H.264/AVC. Optical Engineering 51(4)
14. Hentati, M and Aoudni, Y and Nezan, JF and Abid, M and Déforges, O (2011) FPGA dynamic reconfiguration using the RVC technology: inverse quantization case study. Proceedings of Conference on Design and Architectures for Signal and Image Processing, IEEE, 1–7
15. Fons, F and Fons, M and Cantó, E and López, M (2013) Real-time embedded systems powered by FPGA dynamic partial self-reconfiguration: a case study oriented to biometric recognition applications. Journal of Real-Time Image Processing 8(3), Springer, 229–251
16. Bhandari, S and Subbaraman, S and Pujari, S and Cancare, F and Bruschi, F and Santambrogio, M D and Grassi, P R (2012) High speed dynamic partial reconfiguration for real time multimedia signal processing. Proceedings of 15th Euromicro Conference on Digital System Design, IEEE, 319–326
17. Kanopoulos, N and Vasanthavada, N and Baker, R L(1988) Design of an image edge detection filter using the Sobel operator. IEEE Journal of solid-state circuits 23(2), 358–367
18. Canny, John (1986) A computational approach to edge detection. IEEE Transactions on pattern analysis and machine intelligence 6, 679–698
19. Xilinx (2012) Partial Reconfiguration of a Processor Peripheral tutorial. 1–41
20. Beckhoff, C and Koch, D and Torresen, J (2012) Go AHEAD: A partial reconfiguration framework. Proceedings IEEE 20th Annual International Symposium on Field-Programmable Custom Computing Machines, 37–44
21. Xilinx (2006) LogiCORE IP OPB HWICAP v1.00.
22. Xilinx(2010) LogiCORE IP XPS HWICAP v5.00.
23. Xilinx (2012) LogiCORE IP AXI HWICAP v2.03.
24. Vipin, K and Fahmy, S A(2012) A high speed open source controller for FPGA Partial Reconfiguration.
25. Fons, F and Fons, M and Cantó, E (2011) Runtime self-reconfigurable 2D convolver for adaptive image processing. Microelectronics Journal 42(1), 204–217