FSL-based Hardware Implementation for Parallel Computation of cDNA Microarray Image Segmentation

Bogdan Bot
Student within Technical University of Cluj-Napoca,
Faculty of Automation and Computer Science
Cluj-Napoca, Romania

Sorin Martoiu
National Institute of Nuclear Physics and Engineering
"Horia Hulubei" – IFIN-HH, Bucuresti, Romania

Simina Emerich
Department of Communication, Technical University of
Cluj-Napoca, Cluj-Napoca, Romania

Bogdan Belean
Department of Mass Spectrometry, Chromatography and
Applied Physics, INCDTIM
Department of Communication, Technical University of
Cluj-Napoca, Romania

***Abstract*—The present paper proposes a FPGA based hardware implementations for microarray image processing algorithms in order eliminate the shortcomings of the existing software platforms: user intervention, increased computation time and cost. The proposed image processing algorithms exclude user intervention from processing. An application-specific architecture is designed aiming microarray image processing algorithms parallelization in order to speed up computation. Hardware architectures for logarithm based image enhancement, profile computation and image segmentation are described. The methodology to integrate the hardware architecture within a microprocessor system is detailed. The Fast Simplex Link (FSL) bus is used to connect the hardware architecture as speed up co-processor of the microarray image processing system. Timing considerations were presented considering the levels of parallelism that can be achieved by using our proposed hardware architectures. The FPGA technology was chosen for implementation, due to its parallel computation capabilities and ease of reconfiguration.**

Keywords—microarray; FPGA; image processing; hardware algorithms

I. cDNA Microarray Technology

Measurement of gene expression can provide clues about regulatory mechanism, biochemical pathways and broader cellular function. By gene expression we understand the transformation of gene's information into proteins. The informational pathway in gene expression is as follows: DNA → mRNA → protein. The protein coding information is transmitted by an intermediate molecule called messenger ribonucleic acid mRNA. This molecule passes from nucleus to cytoplasm carrying the information to build up proteins [1]. This mRNA acid is a single stranded molecule from the original DNA and is subject to degradation, so it is transformed into stable complementary DNA for further examination. Microarray technology is based on creating DNA microarrays which represents gene specific probes arrayed on a matrix such as a glass slide or microchip. The most common use for DNA microarrays is to measure, simultaneously, the level of gene expression for every gene in a genome [2]. In this way the microarray compares genes from normal cells with abnormal or treated cells, determining and understanding the genes involved in different diseases.

DNA microarrays represent gene specific probes arrayed on a matrix such as a glass slide or microchip. Usually samples from two sources are labeled with two different fluorescent markers and hybridized on the same array (glass slide). The hybridization process represents the tendency of 2 single stranded DNA molecules to bind together. After hybridization, the array is scanned using two light sources with different lengths (red and green) to determine the amount of labeled sample bound to each spot through hybridization process. The light sources induce fluorescence in the spots which is captured by a scanner and a composite image is produced [3].

Classical genomic microarray experiment involves complex steps including slide production and scanning. A brief description of a microarray experiment can be summarized as follows: a) generation of array ready cDNA, b) cDNA selection and microarray slide printing, c) selection of specific cell material and fluorescent labeling, d) hybridization of the target material on the microarray slide, e) microarray image scanning, f) microarray image processing for gene expression evaluation, g) high order processing (clustering and interpretation, gene regulatory network estimation).

The present paper provides a detailed description of microarray image processing algorithms. The classical flow of processing a microarray image is generally separated in the following tasks: addressing, segmentation, intensity extraction and pre-processing to improve image quality and enhance weakly expressed spots. The first step associates an address to each spot of the image. In the second one, pixels are classified either as fore-ground, representing the DNA spots, or as background. The last step calculates the intensities of each spot and also estimates background intensity values.

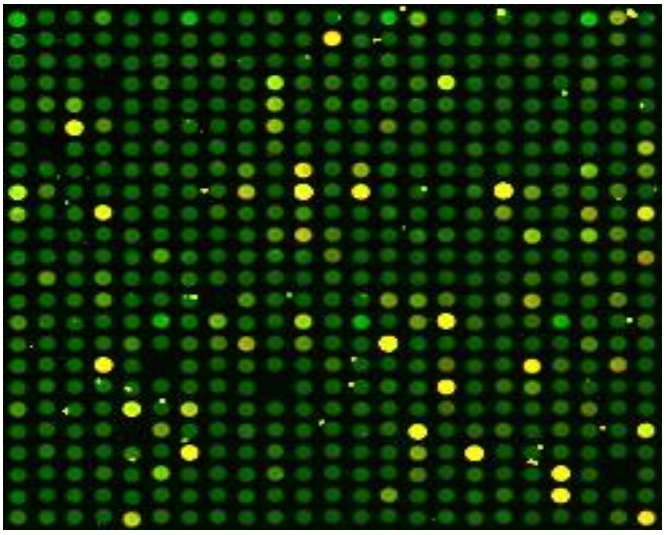

Fig. 1. Agilent pre-processed microarray image [4]

The major tasks of microarray image processing are to identify the microarray image characteristics including the array layout, spot locations, size and shape, and to estimate spot and background intensity values. In order to estimate gene expression levels using microarray analysis, spatial and distributionl methods for spot segmentation are proposed, [4-8]

Examples of microarray image processing software platforms are Agilent Feature Extraction Software (FE) [4], GenePix Pro [9], ScanAlyze [5]. In order to determine what kind of results these software platforms deliver and to validate the results, Feature Extraction software was used to process a microarray image obtained after scanning a microarray glass with DNA information from east european house mouse "mus-musculus". The image resolution is 6100x2160 pixels and covers approximately 20000 microarray spots.

TABLE I. RAW-DATA PARAMETERS FOR "MUS-MUSCULUS" EXPERIMENT DELIVERED BY AGILENT FEATURE EXTRACTION SOFTWARE

Row	***Col***	***GeneName***	***PositionX***	***PositionY***	***PValLog Ratio***
1	*1*	*BrightCorner*	*395,618*	*100.5*	*7.68E+08*
1	*2*	*NegativeCtrl*	*415,962*	*995,462*	*9.03E+08*
1	*3*	*Psma5*	*437,833*	*100,891*	*8.90E+08*
1	*4*	*Mmp14*	*459,123*	*998,774*	*9.25E+08*
1	*5*	*Cdh11*	*479,825*	*100,143*	*6.86E+08*
1	*6*	*C0152H05-3*	*501,548*	*99,529*	*7.61E+08*
1	*7*	*Pro25G*	*522,726*	*99,879*	*8.11E+08*
1	*8*	*L0951F09-3*	*543,748*	*996,792*	*8.28E+08*
…	*…*	*…*	*…*	*…*	*…*

The specified software platform provides raw-data with microarray image characteristics organized in an .xls form (Table I), which are further on used in high order analyses like clustering and gene regulatory network estimation. As the Table 1 shows, each microarray spot represents a specific gene, and it has a precise location.

A regular microarray image has up to hundreds of MB, and it can be divided in independent sub-images, which consists in a compact group of spots. Sophisticated computational tools mentioned in the previous paragraph are available for microarray image processing. Their main disadvantages are the long runtime and the user intervention needed in processing. Considering the regular distribution of microarray spots and also their regular shape, unsupervised segmentation approach can lead to application specific hardware architecture for automatic microarray image processing. Consequently, we implemented an edge detection based segmentation approach for microarray spots. Further on, the paper includes the description of image processing techniques for automatic edge-based segmentation in Section II. Section III describes the hardware implementation of the proposed segmentation methods using a parallel computing approach. A comparison between the processing time needed by a personal computer for microarray image processing and the processing time obtained using the proposed hardware architecture is performed in section IV, taking into account the levels of parallelization of the proposed algorithms. The paper ends with section V, conclusions, underlining the future directions to be considered.

II. ALGORITHMS FOR AUTOMATED MICROARRAY IMAGE PROCESSING

The variety of medical analysis to be performed and the large number of patients, lead to a novel approach in medical applications. Application specific devices are used for unsupervised analysis of medical data and medical diagnosis [12, 13]. The devices to be used in such purposes, efficiently and with a short time to market are FPGAs [14] and graphis processing unis (GPUs) [15].

Regarding microarray analysis, user intervention in microarray image processing brings up the need of a work station with a costly processing platform which will slow down the process of microarray analyses in case of large number of subjects is involved. In order to overcome the previous mentioned disadvantages, the following approaches are taken into account: image processing algorithms will be robust and independent of operator last time adjustments; microarray images are processed using FPGA technology in order to speed up computation.

A. Microarray image enhancement

Image pre-processing techniques are used in order to improve image quality and to enhance weakly expressed spots. The most common techniques used for microarray image enhancement is the spatial logarithm transformation or an arctangent hyperbolic transformation.

$$I_L(x,y) = \frac{\ln(I(x,y)+1)}{n\ln 2} \cdot 2^n \qquad (1)$$

In (1) a spatial logarithm transformation noted I_L is described for a microarray image *I(x,y)* with *(x,y)* the current pixel and *n* the number of bits for pixel representation. In (2) an arctangent hyperbolic transformation noted I_A is described for the same microarray image. In the second transformation $k = \overline{1..2^n - 1}$ determines the threshold from which the pixel intensity will be enhanced.

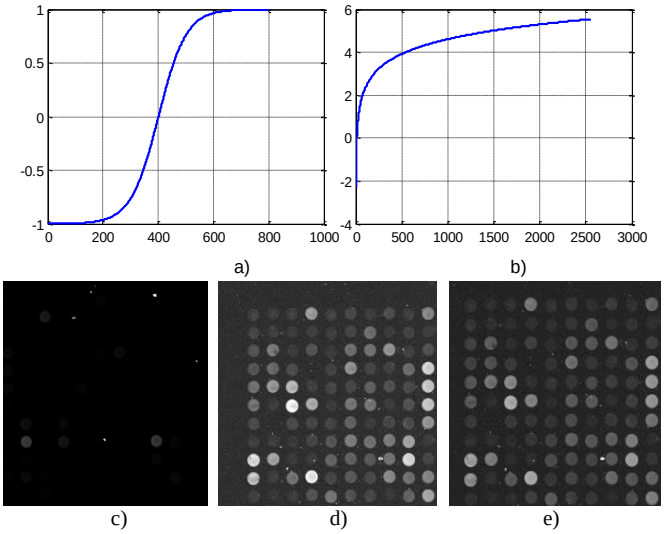

Fig. 2. a) Logarithmic transformed image, b) arctangent hyperbolic transformation, c) Original image, d) log transformation and, e) Arctangent hyperbolic transformed image

$$I_A(x,y)=\begin{cases}\dfrac{2^n\, atgh\left(\dfrac{I(x,y)-k}{k+1}\right)}{atgh(-k/(k+1))}, I(x,y)<=k;\\ \dfrac{2^n\, atgh((I(x,y)-k)/2^n)}{atgh((2^n-1)/2^n)}, I(x,y)>k;\end{cases} \quad (2)$$

In figure 2, an original image and results for both image transformations are presented. Indeed, unlike arctangent hyperbolic, the logarithm transformation does not involve another extra parameter. As a consequence, for the hardware implementation described in section 3, the logarithm transformation was chosen.

B. Microarray image addressing

For microarray image addressing an automatic estimation of spot distance is presented. After the pre-processing of the microarray images, the first step for spot localization is the computation of image projections as described in (3). It can be assumed that the profiles resulting from these projections contain a periodic signal which has been affected by noise.

$$HP(y)=\frac{1}{X}\sum_{x=0}^{X-1} I(x,y) \quad (3)$$

To be able to find the periodicity, the signal is cross-correlated with itself, procedure called autocorrelation (4).

$$pv(i)=\sum_{j=0}^{X-1} HP(i)\cdot HP((i+j)\bmod X) \quad (4)$$

$$VP(x)=\frac{1}{Y}\sum_{y=0}^{Y-1} I(x,y) \quad (5)$$

with *I(x, y)* being the microarray image, *X* and *Y* image dimensions, *i = 0, 1,...,X-1*. The first derivative of the resulted array *pv(i)* crosses the *X* axis in points corresponding to the

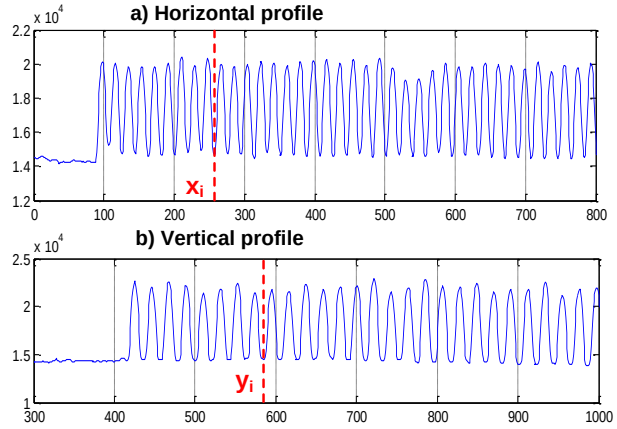

Fig. 3. a) horizontal image profile , b) vertical image profile; x_i and y_i toghether with x_{i+1} and y_{i+1} mark the borderlines which confine the microarray spot *i*

peaks and values of the spots. Taking the distance between zeros the average dimension of the spots is estimated. Microarray spot localization using image profiles can be seen in figure 3, where (x_i,y_i) represents the location of spot *i* from the microarray image.

C. Microarray image segmentation

In microarray image processing, edge detection is a fundamental tool used for intensity extraction and spot segmentation. Edges occur at images location with strong intensity contrast. For edge detection a high-pass filter in Fourier domain can be applied, or convolution with an appropriate kernel (Sobel, Prewitt etc.) in the spatial domain is useful [16]. Convolution in the spatial domain has been chosen for implementation because it is computationally less expansive and offers better results.

The algorithm used for the hardware implementation is Canny filter [17], which is considered to be optimal, based on the following: it finds the most edges, marks the edge as close as possible to the actual edges, and provides sharp and thin edges. The filter that meets all the criteria mentioned above can be efficiently approximated using the first derivative of a Gaussian function. So the first two steps in applying Canny filter would be smoothing the image and differentiating the image in two orthogonal directions. Smoothing operation is done using convolution mask. After smoothing the image, gradient calculation (magnitude and phase) is performed in order to find the edge strength of the spot. To do so, the image is differentiated on two orthogonal directions as in (6) an (7), using image convolution.

$$\frac{\partial I}{\partial x}\approx\frac{I(x+1,y)-I(x-1,y)}{2} \quad (6)$$

$$\frac{\partial I}{\partial y}\approx\frac{I(x,y+1)-I(x,y-1)}{2} \quad (7)$$

The sign and value of the orthogonal components of the gradient determined before are used in estimating the magnitude and the direction of the gradient.

Once the direction of the gradient is known, pixels values around the pixel being analysed are interpolated. The pixel that does not represent a local maximum is eliminated, by comparing it with its neighbours along the direction of the gradient (non-maximum suppression).

Up to this point, image processing algorithms were presented in order to realize a robust detection of microarray image features. A solution for implementing the previous processing chain is presented next.

III. Hardware Implementations for Microarray Image Processing Algorithms

FPGA technology uses pre-built logic blocks and programmable routing resources for configuration and for implementing custom hardware functionality. Their main benefits are the low cost, the short time to market and the ease of reconfiguration. Microarray images are analysed and processed using FPGA technology in order to speed up computation. The hardware implementations of microarray image processing techniques make use of the FPGA features, which allow accessing at the same time hundreds of memory addresses. Indeed, FPGA technology offers the possibility to exploit spatial and temporal parallelism for microarray image processing in order to create a fast automated process which delivers raw-data information about microarray image characteristics. As a consequence, FPGA are well-adapted for processing microarray images as show in [18].

Further on an FPGA based application specific architecture for microarray image processing is described. Xilinx board Virtex5 ML505 was used for the application development. The architecture includes 3 processing units PU_i: PU_1 realizes the microarray image enhancement, PU_2 computes image vertical and horizontal profiles and the last processing unit PU_3 uses spatial parallelism for image segmentation. The processing units together with a DMA controller for RAM memory access are connected to the processor trough the plb_v46 data bus. Autocorrelation and shock filters for microarray image addressing are implemented using C code. Future work aims creating processing units in order to speed up their computation. A detailed description of our application-specific architecture is presented in the figure 4. The same approach which uses hardware coprocessors for high-throughput processing was proposed in [19].

The image processing PU_i units are connected as co-processor to the Microblaze system through FSL bus in order to speed up computation. The FSL interfaces are used to transfer data to and from the register file on the processor to the hardware running on the FPGA.

The FSL represents a uni-directional point to point FIFO based communication. The methodology to interconnect the image processing hardware units to the FSL bus is detailed in section III.D.

A. Microarray image enhancement implementation

Spatial logarithm transformation is used for microarray image enhancement. The logic bloc LOG from figure 5 calculates the logarithm of image intensity for each pixel. The logarithm transformation is implemented on the luminance information Y of the image, obtained using R, G, B channels like in (8).

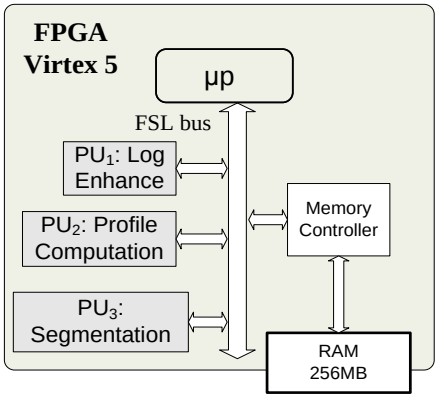

Fig. 4. Application specific architecture for microarray image processing

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \qquad (8)$$

The hardware implementation of the logarithm trans-formation is based on linear approximation of the logarithm function. The logarithm function is calculated in a number of An(x,y) points stored in a memory named ROM_LOG.

Also the slope m for each line described by two adjacent points is calculated and stored in a memory called ROM_SLOPE. In order to calculate the logarithm of the luminance, we are using (9) which represent the equation of a line which has the slope m and passes through the point $A_i(x_i, y_i)$ from the initial A_n points.

$$y_{\log} = m(y - x_i) + y_i \qquad (9)$$

For the implementation described in Fig. 6 there is a number of 3 clock cycles necessary for processing. In order to evaluate the log function estimation, mean square error was calculated for y values between 1 and Y_{MAX} = 256 and the result is shown in (10). A pipelined architecture will reduce the computational time for the logarithm unit to 1 pixel/clock cycle.

$$MSE = \frac{1}{Y_{MAX}} \sum_{y} [\ln(y) - \ln\_est(y)]^2 = 1.807 \cdot 10^{-5} \qquad (10)$$

The same type of implementation was successfully used in [20] for high-throughput decoding of LDPC codes.

B. Microarray image profile computation

Computing the horizontal and vertical image profiles for spot localization involves logarithm computation of pixel intensity. Figure 5 describes the hardware architecture for evaluating image profiles. The luminous component (Y component) of the microarray image *I(x,y)*, is extracted from the RGB colour space. The spatial logarithm transformation is

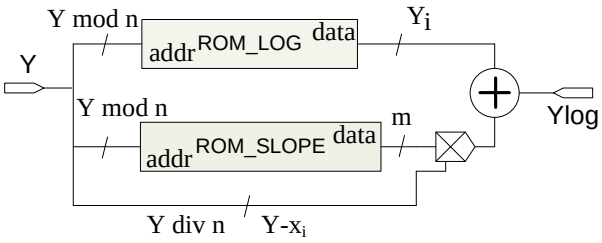

Fig. 5. Hardware implementation for logarithm function applied on the luminous image component for enhancement

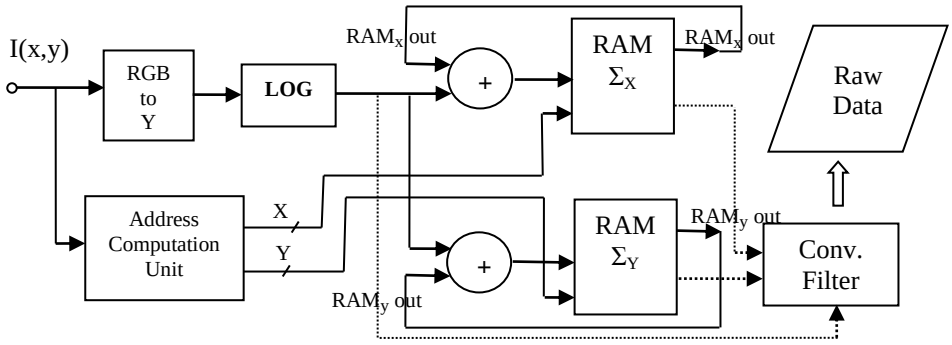

Fig. 6. Hardware architecture for image profile n

The Σ_X and Σ_Y RAM memories and the two adders are used as accumulators for horizontal and vertical profiles while the whole image is scanned. In table IV the hardware resource usage for the implementation is described. The maximum frequency to be used with the implementation is 286.2 MHz.

Once the profiles are calculated, spot location are determined as shown in Fig. 3 using discrete autocorrelation. The spot locations are delivered as partial results for further processing. The next processing step is microarray image segmentation based on spatial convolution, which aims to extract specific microarray parameters, delivered as raw data for further processing.

TABLE II. HARDWARE RESOURCE USAGE FOR MICROARRAY PROFILES COMPUTATION ON XILINX ML505 BOARD

	Used	*Available*	*Utilization*
Number of Slice Registers	108	69,120	1%
Number of Slice LUTs	6,864	69,120	9%
Number of occupied Slices	1,995	17,280	11%
Number of BlockRAM/FIFO	2	148	1%
No. of BUFG/BUFGCTRLs	1	32	3%
Number of DSP48Es	5	64	7%

C. Microarray image segmentation

This section presents a hardware implementation of an adaptive edge detection filter using FPGA, which provides the necessary performance for fast microarray image processing. For edge detection, Canny filter was used. The first two steps in applying Canny filter are smoothing the image and differentiating the image in two orthogonal directions. The next step, non-maximum suppression, computes the gradient direction and magnitude in order to eliminate the pixels that represent false edges. The previously described algorithm is applied on a microarray spot. The description of the edge detection algorithm implementation using convolution is described in detail in [21]. Other approaches for image buffering for neighborhood operation and parallel image processing are proposed in [22] and [23] respectively.

Summing up the computational time needed for each step of the border detection implementation we obtained a total processing time of 60 ns for a microarray spot. Future work aims developing a customizable processing unit for a microarray spot in order to deliver fast segmentation results. Due to the independent processing for each spot, the processing unit can be cloned for computing more than one spot at a time.

D. FSL Integration of the proposed hardware architecture

The aforementioned architectures for logarithm transformation, profile computation and spot segmentation are interconnected so, each *clk* cycle, a pixel intensity from the image is delivered to the processing unit, which, after a delay delivers sequentially the pixels intensities from the resulted image. The resulted image represents the microarray spots with detected edge. The "**Canny**" logic bloc process sequentially pixels intensities from the input image (denoted by *Y*) and delivers sequentially pixels intensities from the output image, which represents the detected edge. The "Canny" logic block has also a *clk* and *reset* pins and also a *start* pin which specifies a pixel intensity is available for processing. The canny *output* delivers sequentially the edge processed pixel intensities, validated through a "1" logic value on the *canny_valid* output. *Send_ready* output ports signals a valid output of the pixel intensity. Thus, the description of the Canny logic bloc from Fig. 7.a is presented, whereas its simulation is detailed in figure 7.c. The simulation includes the reset of all logic blocks at the beginning. Further on, pixel intensity values are sent as inputs to our Canny filter block. The first computed edge is available after an initial delay, due to the procedure which stores the

pixel intensity values within the buffers of the canny logic blocks.

The proposed logic block has to be connected to the FSL data bus. The FSL protocol is used to delivered pixel intensities values to the processing unit. Thus, the processing unit represents the slave device. The master device is the processor which reads data from RAM and delivers data to the slave device and also receives the results of the canny edge detector filter, which, as previously mentioned, acts as a slave device. The write and read operation on the FSL bus are performed using the *getfsl* and *putfsl* c functions. A finite state machine is also designed to control the Canny logic unit through the FSL bus. The FSL bus is described as follows: two *clk* inputs for master and slave, *FSL_S_Data* input port for writing the pixel intensities to be processed into the FSL FIFO, *FSL_M_Data* output port to read the resulted pixel

intensity delivered by the Canny logic unit to the FSL FIFO, *FSL_M_Write* and *FSL_S_Read* represent the control signal for read and write operation in and out of the FSL FIFO. *FSL_S_Exists* is a control signal which specifies if the FSL FIFO is empty or not. Taking into account the FSL protocol, finite state machine (FSM) is designed for the control of the proposed processing unit for Canny edge detector (see Fig. 5b). The FSM has 4 states, *st_reset*, *st_wait*, *st_work* and *End_work*, and drives the canny edge detector hardware implementation using the FSL data bus (see Fig. 5.c for the FSM). The following example is considered for testing the architecture for edge detection: a *20x20* pixels size microarray spot is written in the FSL FIFO buffer. The initial state *st_reset* initializes a counter of the number of pixels to be written in the FIFO to '0'. While FIFO is not empty (*FIFO_empty = '0'*) the pixel intensities are delivered to the Y port of the processing block through the *FSL_S_Data*, and the counter is incremented to count the processed pixel intensities. The maximum value for the counter is 400. In *St_work* state, the processing block starts the processing, and through the output port "*canny_valid*" delivers the control signal *FSL_S_read* to read the next pixel intensity from the FIFO to be processed. The read pixel intensities are processed, and when a result is available (*canny_valid = '1'*) the end_treatment signalize the end of processing and the next state becomes *st_wait*, wherefrom the processing continues if *FIFO_empty = '0'* or the FSM waits for new values to be written in the FSL FIFO.

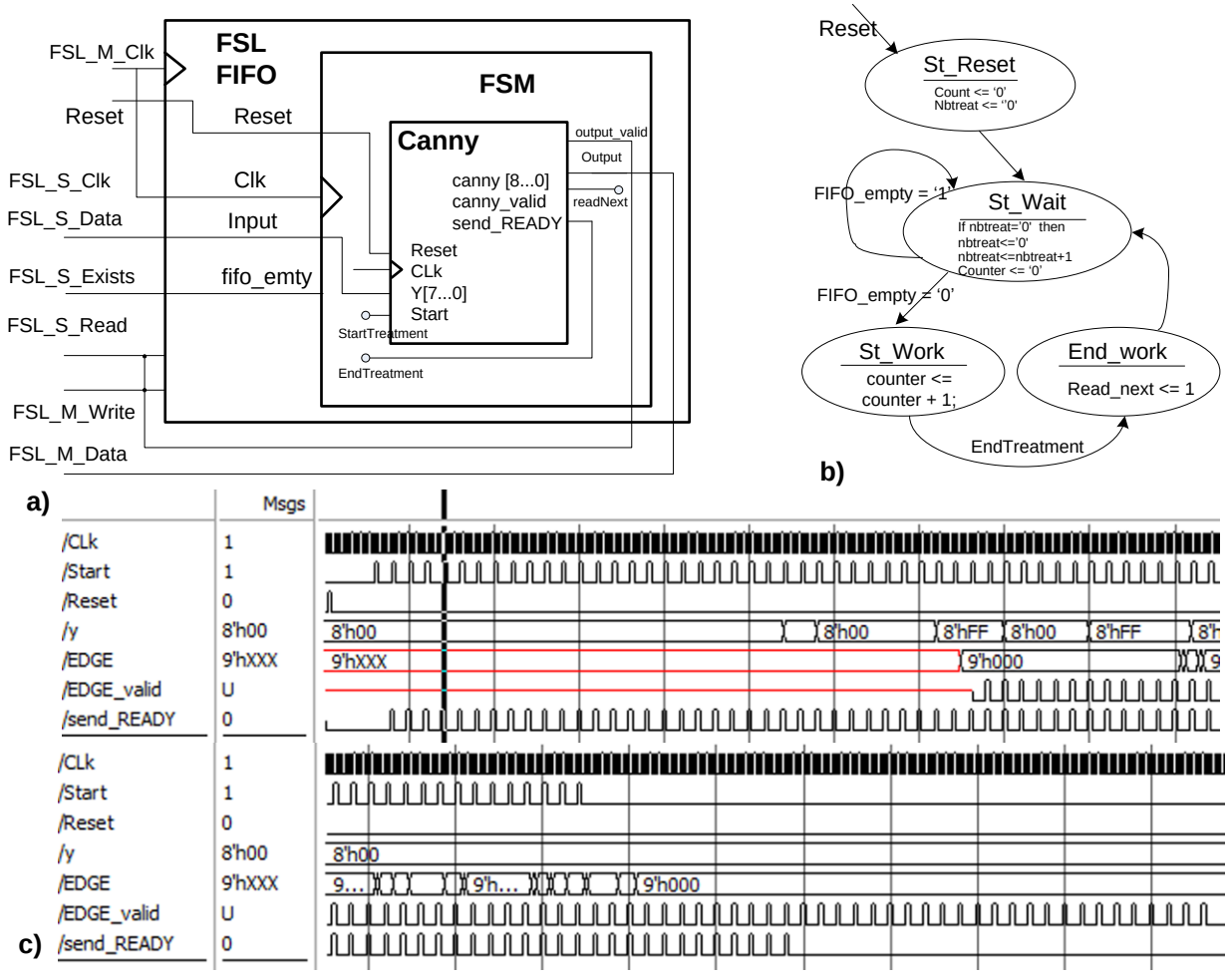

Fig. 7. Canny filter integration to a microprocessor system through FSL bus

IV. Parallel Computation for Microarray Image Processing Algorithms

Further on, the flow of microarray image processing techniques is presented, together with the parallel computation strategies which can be applied on. After image enhancement using logarithm transformation, vertical and horizontal projections are computed in order to estimate spot location and dimension. Once the spot location is established, segmentation is applied and, using border detection spot intensity extraction is performed and the level of expression for each gene is estimated. Thus, the differentially expressed genes are found by comparing the log odd ratios of the intensities from the two channel of the microarray image. If the log odd ratios are higher than 2 the corresponding genes are consider over expressed [24]. This being the interpretation of spot intensities, we proceed to the parallelization of the algorithms, considering the increased number of spots available on one microarray chip, up to 4x44k.

The levels of parallelization for the previously described image processing algorithms are discussed next. In case of image enhancement, we consider *M*, *N* the image dimensions and *p* the number of logarithm computation units. Due to the independent computation of logarithm for each pixel, the maximum level of parallelization for image enhancement is *(MxN)/p*. For spot position estimation, the level of parallelization is *M+N*. Autocorrelation and shock filters are applied on image profiles for estimating spot positions. Due to the recursive description of the algorithms they cannot be easily parallelized. Nevertheless, they are not applied over the full image. As a consequence, the parallelization is not mandatory. Thus they are not considered for describing the timing considerations presented further on.

Once the spot locations are estimated, where *k* is the number of spots, filters like Sobel or Canny for image segmentation can be parallelized, and the maximum parallelization level is *k*. In other words, for each spot, hardware architecture of the canny edge detector can be inferred. Nevertheless, the FPGA (V5 ML505) resources are limited, and *k* cannot be as high as the total number of spots.

In order to estimate the computational time, the highest level of parallelization according to the XC5VlX110T FPGA chip was taken into account. We consider the number of logarithm units *p = 100* for an *M x N = 6100x2160* pixels Agilent image. The number of hardware architectures for edge detection in case of microarray spots, denoted by *k*, is *10*. In Table III parallelization levels are listed together with the computation time for the microarray image processing algorithms.

Total computational time for logarithm transformation, profile computation and microarray image segmentation is around 23,154 ms, encouraging for future implementations.

In the next plot, on X axis, are represented different microarray images with different sizes (size defined by the number of microarray spots included) and on Y axis computational time using a personal computer and the proposed application specific architectures implemented on Virtex5 FPGA.

TABLE III. Parallelization Llelization Levels and Timing

Image processing algorithms	*Level of parallel.*	*Input data*	*Processing time*
1. Log. transformation	$MxNxp^{-1}$	≈100 MB	3480 us
2. Image profiles	M + N	≈100 MB	82,6 us
3. Autocorrelation	2	M+N	-
4. Shock filters	2	M+N	-
5. Canny filter	k	≈100 MB	16312 us

It is to be mentioned that the results presented in figure 6 correspond to the presented image processing techniques and hardware implementation with and without the levels of parallelization included. The red curves represent the processing time without the levels of parallelization applied and the green curve corresponds to the processing time with the levels of parallelization included. Compared with the work presented in [21], the levels of parallelization are included, which lead to an improvement regarding the computational efficiency, as described in figure 8.

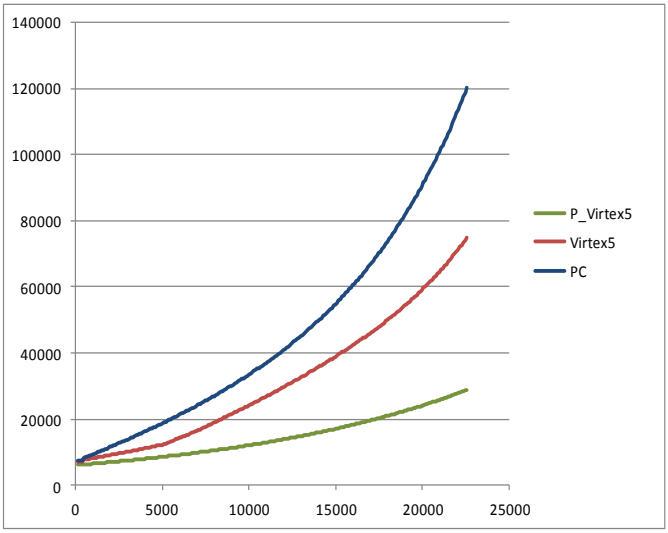

Fig. 8. Computational time on PC (Dual Core, 1800 MHz, 2GB RAM) and Virtex5 (125Mhz, 256 MB RAM)

Moreover, the hardware architectures for Gaussian filtering, gradient computation and non-maximum suppression within the image segmentation detailed in sections III.C function in a pipeline manner. Thus, the output of the Canny logic block from figure 7 is delivered each clock cycle.

V. Conclusions

The present paper proposes hardware implementations for microarray image processing algorithms, which take advantage of the FPGA technology features in order to implement an automated system for fast microarray image processing. Consequently, the proposed architectures are connected as co-processors to an FPGA based system, proving the efficiency of the proposed implementation, with respect to the computational time. The main benefit of the proposed work is the possibility to replace the workstation together with the software platform for microarray image processing with a

system on a chip. The proposed FPGA-based system can be easily integrated within the microarray canner level. Due to the reduced computational time and cost, a large number of microarray analyses can be performed, compared with the existing computational tools.

The levels of parallelism for microarray image processing algorithms are described. Considering the computation efficiency of the proposed microarray image processing task, the experimental results based on algorithm parallelization show significant improvements compared both with a general purpose processor (PC) and with a FPGA based system without levels of parallelization included. Thus, FPGA technology is proved to be an efficient solution for an application-specific architecture for microarray image processing.

Future work aims to develop application-specific hardware architecture for more complex methods for automatic microarray image processing such us, partial differential equations (PDE)-based gridding or clustering-based spot segmentation.

ACKNOWLEDGEMENTS

This paper is supported by the Human Resources Development Programme POSDRU/159/1.5/S/137516 financed by the European Social Fund and by the Romanian Government.

REFERENCES

[1] Mark Schena, Micropuce Biochip Technology: Oxford University Press, 1999.

[2] A. M. Campbell, W. T. Hatfield, L. Heyer, "Make microarray data with known ratios," CBE – Life Sciences Education, vol. 6, 196-197, 2007.

[3] Peter Bajcsy, "An Overview of DNA Microarray Image Requirements for Automated Processing," IEEE Transac-tions on Image Processing, VOL 13, NO 1, pp. 15-25, Janu-ary 2004.

[4] Yang Y, Staord P and Kim YJ. Segmentation and intensity estimation for microarray images with saturated pixels. BMC Bioinformatics; 2011. 12:462.

[5] Wanga Z et al. Hybrid clustering for microarray image analysis combining intensity and shape features. Neurocomputing; 2014. 142:408{418.

[6] Using fuzzy logic and particle swarm optimization to design a decision-based filter for cDNA microarray image restoration, Chang, Bae-Muu; Tsai, Hung-Hsu; Shih, Ji-Shiang, Engineering Applications Of Artificial Intelligence, 36 Pages: 12-26, 2014.

[7] Giannakeas Net al. Spot addressing for microarray images structured in hexagonal grids. Computer Methods and Programs in Biomedicine; 2012. 106:1.

[8] Giannakeas N et al. Segmentation of microarray images using pixel classication - Comparison with clustering-based methods. Computers in Biology and Medicine; 2013. 43:705{716.

[9] Agilent Feature Extraction Software v10.5, User guide, 2008.

[10] Handran S, Zhai YZ (2003) Biological relevance of GenePix results. Molecular Devices - Application Notes, pp 1–9

[11] M.B. Eisen, "ScanAlyze User Manual," Stanford University, 1999.

[12] Florea L, Florea C, Vertan C, Sultana A (2011) Automatic tools for diagnosis support of total hip replacement follow-up. Adv Electr Comput Eng 11(4):55–62.

[13] Tonti, Simone et al.n An automated approach to the segmentation of HEp-2 cells for the indirect immunofluorescence ANA test Computerized Medical Imaging and Graphics , Volume 40 , 62 – 69, 2015

[14]]Kamal A. ElDahshan et al., Hardware Segmentation on Digital Microscope Images for Acute Lymphoblastic Leukemia Diagnosis Using Xilinx System Generator , International Journal of Advanced Computer Science and Applications, 5(9), pp. 33-37, 2014

[15] Moulay Ali Nassiri, Jean-François Carrier, Philippe Després, Fast GPU-based computation of spatial multigrid multiframe LMEM for PET, Medical & Biological Engineering & Computing, April 2015.

[16] Tanvir Abassi, Usaid Abassim, "A Proposed FPGA Based Architecture for Sobel Edge Detection Operator," Journal of Active and Passive Electronic Devices, pp. 271–277, 2007.

[17] J. Canny, "A computational approach to edge detection," IEEE Trans Pattern Analysis and Machine Intelligence, vol. 8, no. 6, pp. 679-698, Nov. 1986.

[18] B. Belean, M. Borda, A. Fazakas, "Adaptive Microarray Image Acquisition System and Microarray Image Processing Using FPGA Technology," Lecture Notes in Computer Science 5179, pp. 327-334, 2008.

[19] Călin BÎRĂ, Lucian PETRICĂ, Radu HOBINCU, OPINCAA: A Light-Weight and Flexible Programming Environment For Parallel SIMD Accelerators, Romanian Journal of Information Science and Technology, 16(4), pp. 336–350, 2013.

[20] Belean B et al., Low Complexity Approach for High Throughput Belief-Propagation based Decoding of LDPC Codes, Advances in Electrical and Computer Engineering, 13(4):2013, pp 69-72.

[21] Belean B, Borda M, Le Gal B, Terebes R (2012) FPGA based system for automatic cDNA microarray image processing. Comput Med Imaging Graph 36(5):419–429.

[22] Kazmi M. et al., FPGA Based Compact and Efficient Full Image Buffering for Neighborhood Operations, Advances in Electrical and Computer Engineering, 15(1):2015, pp. 95-104.

[23] Maciej Wielgosz, Mauritz Panggabean and Leif Arne Rønningen, FPGA Architecture for Kriging Image Interpolation, (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 4, No. 12, 2013

[24] Marczyk M. et al., Adaptive filtering of microarray gene expression data based on Gaussian mixture decomposition, BMC Bioinformatics 2013, 14:101