

Pareto Optimal Design of an FPGA-based Real-Time Watershed Image Segmentation

M. Neuenhahn, H. Blume, T. G. Noll
Chair for Electrical Engineering and Computer Systems
RWTH Aachen University,
Schinkelstraße 2, 52062 Aachen, Germany
Phone: +49 (0)214 80 94367, Fax: +49 (0)241 80 92282
Email: [neuenh, blume, tgn}@eecs.rwth-aachen.de](mailto:{neuenh, blume, tgn}@eecs.rwth-aachen.de)

Abstract—Motion vector based field rate upconversion is a key element of image quality improvement techniques in commercial TV applications. A new trend in order to further improve the upconversion quality is to include object information. This information is computed by image segmentation. One example for a segmentation algorithm that achieves a good segmentation quality is the watershed segmentation. In this paper the implementation of a watershed segmentation algorithm on an FPGA featuring real-time processing of video-sequences in PAL-format is presented. As the watershed segmentation possesses several parameters, which strongly influence the computational effort and the segmentation quality, the possibility to trade computational effort for segmentation quality was investigated. Furthermore, Pareto optimal design points, i.e. parameter settings, were identified.

Keywords— image segmentation, FPGA design, video processing, Pareto optimization

I. INTRODUCTION

Various techniques in the field of video signal processing are aiming for picture quality improvement. Examples are motion vector based field rate upconversion, noise reduction, sharpness and contrast improvement etc. [9]. As there is a high commercial interest in high quality video reproduction much effort is spent in order to further improve the quality of these algorithms. One trend in video processing is to include more and more so-called meta information which is generated out of the original video sequence. Examples for meta information are feature-points (characteristic image elements), texture information, homogeneity information and especially object information [5], [6]. This information can be applied later on in order to improve a succeeding video processing like e.g. a motion estimation. One of the most interesting type of meta information is object information. Object information has to be computed by image segmentation techniques. But

image segmentation itself is a challenging task. Due to the increasing computational power of today's hardware platforms also highly complex image segmentation techniques are going to be transferred to digital video signal processing. These were up to now only applied for still image processing.

One characteristic of complex image segmentation techniques is that they feature a variety of parameters (e.g. local degree of connectivity, drowning levels, segment merging thresholds etc. [10]) which strongly influence the resulting segmentation quality but also the required computational complexity. Therefore, a quantitative optimization of these parameters is required keeping in mind that practical applications need real-time performance.

Given for example a specification of a target application in terms of image resolution and frame rate a Pareto optimization [7] can be performed in order to find optimal design points in the design space. This design space concerns segmentation quality and required computational complexity respectively required hardware effort.

In this paper a real-time implementation and quantitative optimization of such a complex segmentation technique, namely a rainfalling watershed based image segmentation [10] on a Field Programmable Gate Array (FPGA) is presented. Pareto fronts will be presented showing the lowest required number of computation cycles respectively the lowest number of required memory bits for a given segmentation quality evaluated in terms of quantitative segmentation metrics.

Finally, the achieved throughput rates of the watershed-based technique are compared to an implementation on a digital video signal processor.

II. SEGMENTATION

Several techniques are available today for image seg-

mentation. Examples are dynamic threshold segmentation, watershed segmentation or mean-shift segmentation (for an overview of different techniques see for example [3]). These techniques differ in their computational complexity as well as their segmentation quality. The segmentation alternatives are roughly compared in Table 1.

segmentation algorithm	segmentation quality	computational complexity
mean-shift segmentation [8]	very high	very high
rainfalling watershed [10]	high	high
dynamic threshold segmentation [17]	moderate	low

Table 1: Comparison of different image segmentation techniques

In this work the watershed segmentation technique was chosen as it provides a good trade-off between achieved segmentation quality and computational complexity.

A. Rainfalling watershed segmentation

Various watershed segmentation variants have been proposed ([4], [10]). Here, a rainfalling watershed segmentation including pre- and post-processing was implemented. The basic principle of this algorithm is that segmentation is performed by labeling connected areas of an image. For this, the luminance values of an image are regarded as a topographic surface. In this topographic surface a descending raindrop will move due to the gravity downwards to the deepest neighboring location (corresponding to the lowest neighboring luminance), i. e. the direction of the inverse gradient. This descent is stopped when the raindrop reaches a local minimum. All pixels along the path of the descending raindrop are marked with the label of the local minimum. Each label corresponds to one segment.

B. Pre-Processing

For watershed segmentation typically a pre-processing of the image is applied. For example, the number of segments is reduced by clipping the luminance values of the image to a parameter called *drowning-level* if the actual value is smaller than the drowning-level as illustrated in Figure 1.

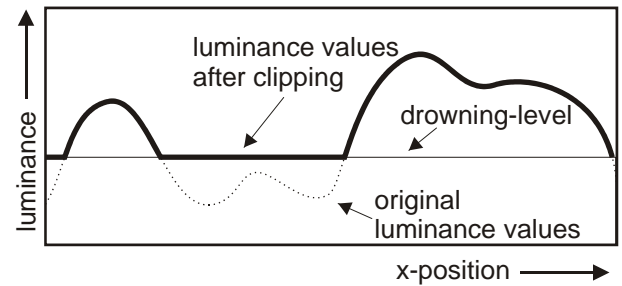


Figure 1: Visualization of the drowning-operation for a single line of an image

A further parameter influencing the algorithm is a sensitivity-threshold for the computation of image gradients (*gradient-tolerance*).

C. Post-Processing

After pre-processing and segmentation the resulting segments are typically very small and do not correspond to real objects. This problem is usually referred to as "over-segmentation". To reduce the number of segments and to increase the segmentation quality, very small segments can be merged applying different rules ([4]). Here, two rules are used to merge neighboring segments:

The first rule denoted as *pruning* is, that segments smaller than a certain size (*min. Regions*, measured in number of pixels) are combined with that neighboring segment whose average luminance is most similar to the average luminance of the segment currently being too small.

The second rule denoted as *merging* is, that segments with similar average luminance values are merged independent from their size. Two segments have to be merged, if the absolute difference between their average luminance values becomes smaller than a threshold (*Lum-Range*).

III. HARDWARE PLATFORM

The real-time watershed segmentation has been implemented on an experimental hardware platform for video processing. A block diagram of the platform is depicted in Figure 2.

The analog video signals in PAL-format are digitized in the video-decoder. The digital signals features an eight bit YUV 4:1:1 format (CCIR656-format, [11]). The DSP, a TriMedia TM1300 [15], receives the video-data from the video-decoder and is connected to a 32 MB SDRAM memory via a 32bit wide bus. The FPGA, an EP20K400BC656-3V [1], can process the video data, which comes directly from the video-decoder or the DSP. Furthermore, it can store and access data in a local memory and finally send video data to the video-encoder, which converts the digital video signal back into an analog video

signal. The memory, which is connected to the FPGA via a 32 bit wide bus, consists of two SRAM-chips, each providing a size of 16·1024 bits. The access time of the memory is 55 ns [16].

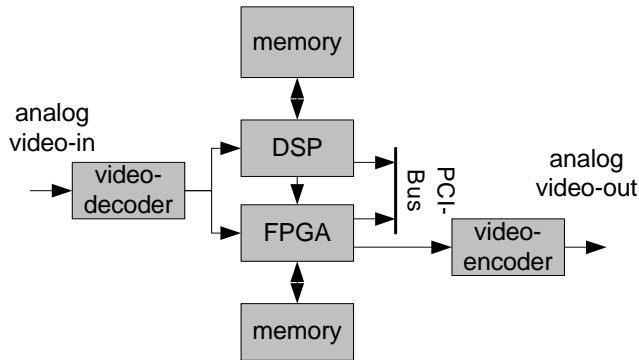


Figure 2: Block diagram of the experimental hardware platform

IV. IMPLEMENTATION

The realization of the rainfaling watershed-segmentation is separated in three tasks. At first, for each pixel in an image the direction of the steepest descent is computed. This can be done for a connectivity of four or eight pixel (see Figure 3 a and b). If there is no falling edge, the pixel is marked as a local minimum.

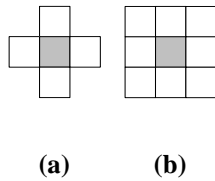


Figure 3: Visualization of a four- (a) and eight-pixel connectivity (b)

The next task is to perform the rainfaling simulation as described before, by labeling all pixels along the path to the local minimum with a unique segment number.

The third step is performed, when a local minimum has been found. Then, a so-called region-growing algorithm is applied.

An exemplary application of the complete algorithm is illustrated in Figure 4. Figure 4a shows luminance values from an image. For simplicity reasons the range of values is limited here to 0-6. The arrows in Figure 4b show the direction of the steepest descent for a four-pixel connectivity and the circles mark a local minimum. The line in the figure represents the way of one raindrop to the corresponding local minimum. The resulting segments are marked in different shades of gray.

The real-time image segmentation including the pre-processing has been mapped onto the experimental plat-

form described before. For the post-processing a cycle accurate simulation model has been generated.

2	4	6	5	4	3
0	0	5	3	1	2
2	4	3	2	3	4
6	3	1	0	1	6
6	5	3	2	3	4

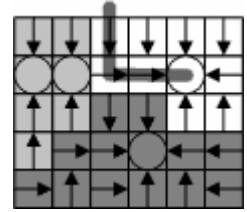


Figure 4: Luminance values from an exemplary image (a) and corresponding segmentation results with the direction of the steepest descent for each pixel (b)

V. RESULTS

Each part of the algorithm depends on parameters, which were described before and can be altered at runtime. These parameters have a strong influence on the segmentation quality, the computational effort and the size of the memory used for the computation. To quantify the influence of the different parameters and different images on the computational effort and the segmentation quality different images have been segmented. Therefore, diverse parameter combinations have been applied. The number of cycles, the memory used and the segmentation quality have been determined.

A. Implementation Results

The watershed segmentation with pre-processing was implemented on an EP20K400BC652-3V FPGA [1], which is used on the experimental hardware platform. Furthermore, it was simulated for an EP2S15F484C3 FPGA [2], which is one of the most actual FPGA devices. The required FPGA resources are given in device specific numbers of logic elements (#LE, #ALUT), memory bits and so-called DSP blocks in Table 2 and Table 3. Additionally, the maximum clock frequencies for the different blocks are shown.

In order to achieve real-time performance, the blocks *CCIR-decoder* and *pre-processing* have to operate at the clock-frequency given by the input data (for a CCIR-656-signal 27 MHz). The block *watershed* can operate at a different frequency, as the data, on which it operates, is stored in a memory, which can be accessed asynchronously.

	#LE	#Memory Bits	f_{\max}
CCIR-Decoder	171	0	38 MHz
Pre-Processing (Drowning- and Gradient-Operation)	695	16384	
Watershed	1657	9880	31 MHz
Total	2523	26264	—————

Table 2: FPGA resources needed and f_{\max} achievable for a rainfalling watershed implementation with preprocessing on an EP20K400BC652-3V FPGA

	#ALUT	#Memory Bits	DSP block	f_{\max}
CCIR-Decoder	85	0	0	93 MHz
Pre-Processing (Drowning- and Gradient-Operation)	363	16384	2	
Watershed	966	9880	2	153 MHz
Total	1416	26264	4	—————

Table 3: FPGA resources needed and f_{\max} achievable for a rainfalling watershed implementation with preprocessing on an EP2S15F484C3 FPGA

Different images featuring different resolutions taken from typical video test sequences were examined and the cycles needed for the watershed computation were counted. These examinations have been performed for several parameter combinations, which affect the computational effort. The maximum and minimum number of cycles needed for the segmentation for typical parameter settings are given in Table 4 for different images. Due to the parameter variations the required number of cycles typically varies by a factor of two.

Name of video-sequence	Resolution		Cycles	
	Width	Height	Min	Max
<i>Lego</i>	720	288	$2.46 \cdot 10^6$	$5.45 \cdot 10^6$
<i>Flowergarden</i>	720	288	$2.32 \cdot 10^6$	$4.84 \cdot 10^6$
<i>Housepan</i>	720	576	$5.40 \cdot 10^6$	$11.27 \cdot 10^6$
<i>Sailorpan</i>	720	576	$4.94 \cdot 10^6$	$10.68 \cdot 10^6$

Table 4: Cycles needed for the watershed segmentation for different images and different parameter settings

The video sequences *lego* and *flowergarden* are given in an interlaced PAL-format. According to the PAL-format a field frequency of 50 Hz is applied. This corresponds to a processing time for one image of 20 ms. Therefore, $3 \cdot 10^6$ Cycles are available for the watershed segmentation, if the FPGA works at a clock frequency of 150 MHz. This shows, that for images, taken from both video test

sequences, a parameter combination can be found, which allows real-time performance for PAL-resolution @50 Hz (2:1) on the EP2S15F484C3 FPGA. For the non-interlaced video test sequences *housepan* and *sailorpan* (50 Hz, 1:1) real-time performance is also achievable.

The experimental hardware platform featuring the EP20K400BC652-3V FPGA allows real-time processing of CIF-resolution images with 20 Hz frame repetition frequency.

Parallel to the FPGA-implementation also an implementation of the watershed segmentation algorithm on an actual TMX320DM642 digital signal processor (DSP) [18] has been realized. Though several DSP-specific code-optimizations techniques like loop-unrolling, use of custom instructions (so-called intrinsics) etc. have been applied, the achievable performance in terms of throughput rate is much lower than in the FPGA-implementation. Applying for example the test sequence *housepan*, the processing of one image out of this sequence requires approximately $115 \cdot 10^6$ cycles. As the DSP can operate at a frequency of 600 MHz, the processing time of the image is 200 ms. Compared to the DSP-implementation of the watershed segmentation the FPGA-implementation therefore roughly provides a throughput rate which is by a factor of five higher.

B. Dedicated evaluation methods

The objective evaluation of the image segmentation quality is a complex problem. The only reasonable objective approach is to compare the results of the automatic segmentation with the results of an application specific hand-segmentation, which was performed in advance by an expert. The application for which the segmentation is needed, strongly influences the results of an optimal segmentation. In this example the target application is object-based post-processing of motion vector fields [13]. For this application only the dominant image segments are needed, smaller segments have to be neglected. The following metrics are applied for the evaluation of segmentation quality:

The so-called Over Segmentation Degree (*OSD*) describes the ratio between the number of segments, which have been found in one image ($Segments_{real}$), and the number of segments that have been found by hand-segmentation for this image ($Segments_{opt}$). Therefore, it follows

$$OSD_{global} = \frac{Segments_{real}}{Segments_{opt}}$$

Segmentation quality is enhanced with decreasing *OSD*. The so-called Segmentation Accuracy (*SA*) describes the

exactness between the contours of the segments, which have been found by the segmentation algorithm, and the segments, which have been found by hand.

This metrics is computed, by inspecting for each optimum segment all those segments, which have pixels with the optimum segment in common. Therefore, the SA for an ideal segment is computed by summing up all partial contributions of the overlapping real segments.

The contribution of each real overlapping segment is computed by the ratio of the overlapping part of the real segment ($A_{i,r}^{in}$) and the complete real segment ($A_{i,r}$). This ratio is weighted with the share of the overlapping part ($A_{i,r}^{in}$) of the complete ideal segment (A_i). Here, each variable A_i , $A_{i,r}$ etc. is expressed by a number of pixels. Therefore, the segmentation accuracy SA_i for a single optimal segment i is

$$SA_i = \sum_{r=1}^{segments_{real}} \frac{A_{i,r}^{in}}{A_i} \cdot \frac{A_{i,r}^{in}}{A_{i,r}}$$

Finally, the overall segmentation accuracy SA can be computed by summing up all accuracy results SA_i weighted with their individual share in the whole image

$$SA = \frac{\sum_{i=1}^{segments_{opt}} SA_i \cdot A_i}{\sum_{i=1}^{segments_{opt}} A_i}$$

The maximum achievable SA is 1. It is important to notice, that only a good SA and $1/OSD$ value together represent a good segmentation quality.

C. Scalability of the implemented rainfalling watershed segmentation

The computational effort, which must be spent for a segmentation of an image by application of the rainfalling watershed segmentation, depends on the image and on the parameters described before. Accordingly, it is possible to trade computational effort for segmentation quality, sometimes also referred to as complexity scalability [14]. With this information it is possible to set the algorithm parameters in such a way that the best possible segmentation quality is achieved for a given computational effort.

Here, the measured results show a dependency between the parameters of the algorithm and the achieved segmentation quality respectively the computational effort that has to be invested for its implementation. The results are depicted exemplarily for an image out of the video test sequence *lego*. The original image and a segmented image

are given in Figure 5 and Figure 6.



Figure 5: Original luminance image, taken from the video test sequence *lego*



Figure 6: Segmentation results for the image out of the video test sequence *lego*

In Figure 7 and Figure 8 the segmentation evaluation metrics are depicted for the segmentation of the image *lego*. The segmentation quality is depicted in dependency of the number of cycles and number of memory bits from the external memory used. The parameters *gradient-tolerance* and *connectivity* have been varied. The *gradient-tolerance* is varied from 2 to 35 and the connectivity has the values four- and eight-pixel. The other parameters are set to fix values. On the y-axis the quality, measured in segmentation accuracy (SA) and over segmentation degree ($1/OSD$), is shown. On the x-axis the cycles (see Figure 7) and the memory bits (see Figure 8) needed for the computation (segmentation, pre- and post- processing) are depicted.

An increase of the gradient-tolerance results in fewer segments. This leads to less computational effort and less memory needed. At the same time this lower *gradient-tolerance* leads to a lower over segmentation degree (corresponding to a higher $1/OSD$ value) and a lower segmentation accuracy. Varying the *gradient-tolerance* allows to trade segmentation accuracy for over segmentation degree and computational effort.

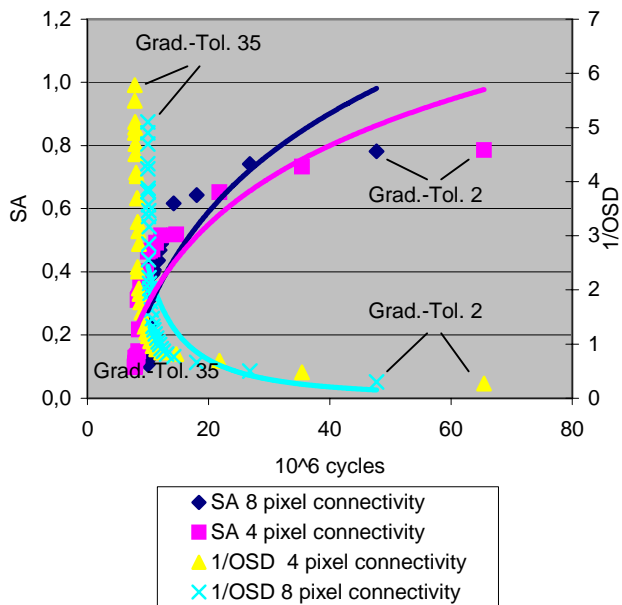


Figure 7: Segmentation quality over cycles used to process an image out of the video sequence *lego*

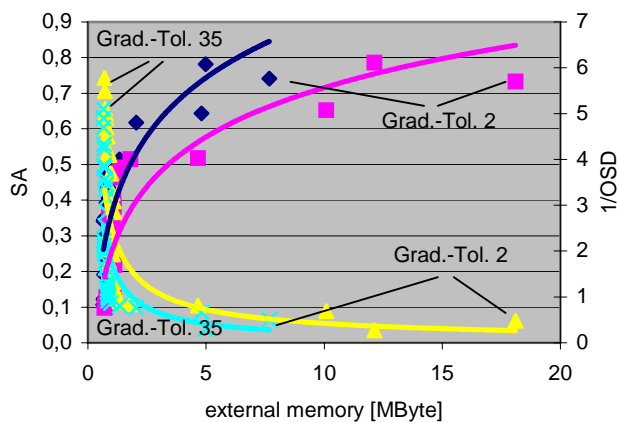


Figure 8: Segmentation quality over external memory used to process an image out of the video sequence *lego*

D. Pareto Front

To find an optimal parameter combination a variety of combinations have been tested which build up a wide design space. The results are plotted in Figure 9 and Figure 10. On the x-axis the cycles needed for the computation are shown and on the y-axis the segmentation quality is depicted (SA in Figure 9 and 1/OSD in Figure 10). With these results a Pareto front can be found. On this front the results for a parameter-combination with the best segmentation-quality for a given computational effort can be found. The Pareto front is built up by Pareto points, which represent these Pareto optimal parameter combinations. Several points with attractive parameter combinations lying either on the SA- or the 1/OSD Pareto front are marked in both diagrams and the corresponding

parameter combinations are given in Table 5.

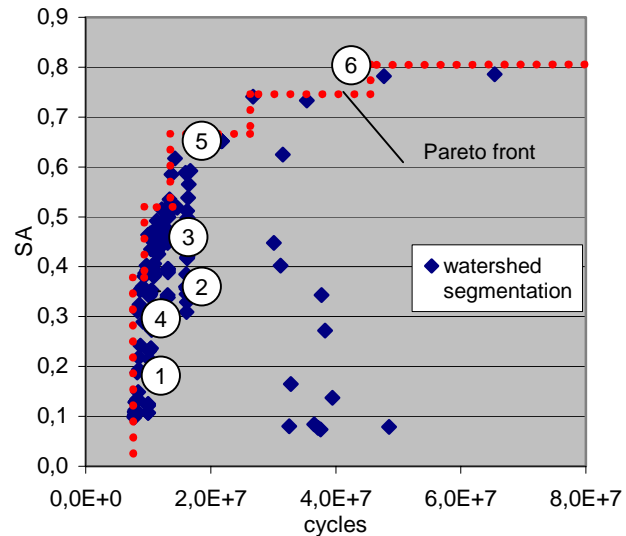


Figure 9: Pareto plot of the segmentation accuracy (SA) for an image taken from the video sequence *lego*

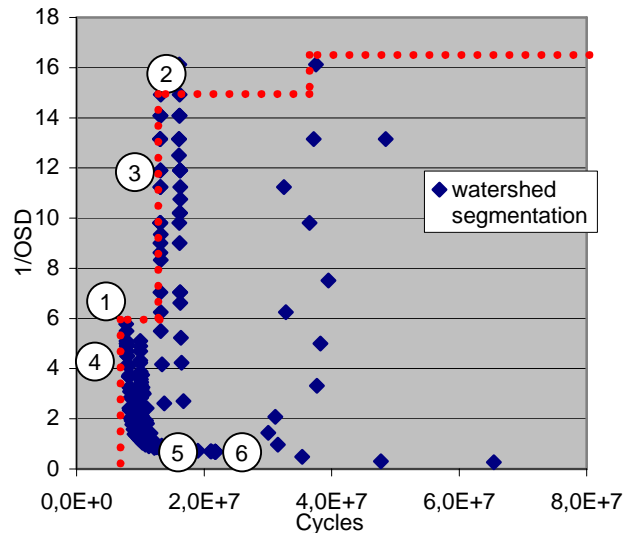


Figure 10: Pareto plot of the inverse over segmentation degree (1/OSD) for an image taken from the video sequence *lego*

Exemplary points	Watershed Segmentation		Merge
	Con.	gradient-toler.	min. regions
1	4	35	100
2	8	4	25000
3	8	4	9000
4	4	28	100
5	8	5	100
6	8	2	100

Table 5: Parameter combinations for specific points (the parameters drowning level and prune range have been fixed here to zero)

A variety of different images has been analyzed as well and different Pareto fronts have been found for the images. The parameter combinations, which define the Pareto points differ from image to image. But some parameter-combinations (e.g. parameter combination 3) show nearly optimal results for all tested video sequences.

Figure 11 and Figure 12 show the Pareto plots for an exemplary image taken from the video test sequence *flowergarden*. Furthermore, the points, representing the same parameter-combinations as the specific points depicted in Figure 9 and Figure 10 for the image *lego*, are displayed.

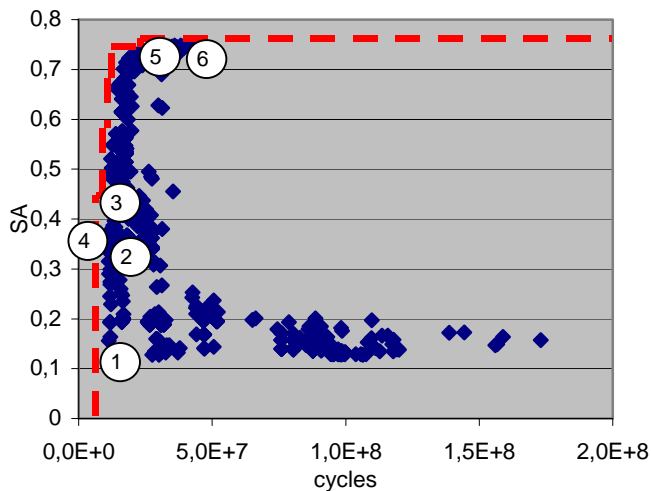


Figure 11: Pareto plot of the segmentation accuracy (SA) for an image taken from the video sequence *flowergarden*

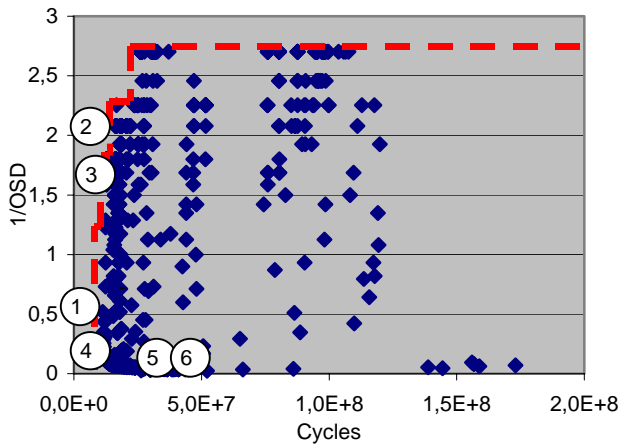


Figure 12: Pareto plot of the inverse over segmentation degree (1/OSD) for an image taken from the video sequence *flowergarden*

VI. SUMMARY

A rainfalling watershed segmentation including pre-

processing was implemented on an FPGA, which is part of an experimental hardware platform.

The implemented rainfalling watershed segmentation algorithm was applied on images taken from several video sequences. The computational costs and the segmentation quality were determined. It was shown that a real-time processing of video sequences in PAL-resolution @ 50 Hz frame rate is achievable.

Furthermore, a dependency of the computational effort and segmentation quality from images and parameters is shown. It has been worked out that the rainfalling watershed segmentation implementation, as described in this paper, is scalable. It is possible to trade computational effort for segmentation quality by varying certain parameters.

Additionally, a Pareto analysis has been performed for the implementation applying images taken from several video test sequences. The results show image dependent Pareto fronts for the images. Applying the real-time implementation presented here it was also possible to identify several attractive parameter combinations, which yield nearly Pareto optimal results for all tested video sequences.

REFERENCES

- [1] Altera Corporation: "Data Sheet APEX 20K: Programmable Logic Device Family". www.altera.com, 2002.
- [2] Altera Corporation: "Stratix II Device Handbook". www.altera.com, 2004.
- [3] Alvarado Moya, J. P.: "Segmentation of colour images for interactive 3D object retrieval", RWTH Aachen, 2004.
- [4] Beucher, S.: "The Watershed Transformation Applied to Image Segmentation", 10th Pefferkorn Conf. On Signal and Image Processing in Microscopy and Microanalysis, 16-19 Sept. 1991, Cambridge, UK, Scanning Microscopy International, suppl. 6. 1992, pp. 299-314
- [5] Blume, H.; von Livonius, J.; Noll, T. G.: "Segmentation in the Loop - An iterative, object based algorithm for motion estimation", *Proc. of the VCIP'04*, SPIE-IS&T Vol. 5308, pp. 464-473
- [6] Braspenning, R.A.; de Haan, G.: "True-motion estimation using feature correspondence", *Proc. of the VCIP'04*, SPIE-IS&T Vol. 5308, pp. 396-407
- [7] Brayton, M.; Spence, R.: "Sensitivity and Optimization", Elsevier, Amsterdam, 1980
- [8] Comaniciu, D.: "Mean Shift: A robust Approach Toward Feature Space Analysis", *IEEE Trans. On PMAI*, Vol. 24, No. 5, May 2002, pp. 609-619
- [9] de Haan, G.: "Video Processing for Multimedia Systems", University Press, Eindhoven, 2000
- [10] De Smet, P.; Pires, R.: "Implementation and analysis of an optimized rainfalling watershed algorithm", *Proc. of the SPIE, VCIP'2000*, Vol. 3974, pp. 759-766
- [11] European broadcasting union: "Tech. 3246-E: EBU parallel interface for 625-line digital video signals", Brussels 1983.

- [12] Khoral Inc.: "Khoros Pro 2001 Integrated Development Environment", www.khoral.com
- [13] Levine, M; Nazif, A.: "Dynamic Measurement of Computer Generated Image Segmentations", IEEE Trans. On Pattern Analysis and Machine Intelligence, Vol. 7, Nr. 2, March 1985.
- [14] Mietens, S.: "Complexity Scalable MPEG Encoding", Technische Universiteit Eindhoven, 2004.
- [15] Philips Semiconductors: "DATA BOOK TM-1300: Media Processor", 2000.
- [16] Toshiba: "Data Sheet TC55VBM316AFTN/ASTN40,55", www.toshiba.com, 2002.
- [17] Zamperoni, L.: "Methoden der digitalen Signalverarbeitung", Vieweg 1989 (in German)
- [18] Texas Instruments: "TMS320DM64x Digital Media Processors - Product Bulletin (Rev. B)", www.ti.com, 2004