Design of a Rapid Prototyping Environment for Stereo Vision Systems Based on Impulse C

Olaf Lüke, Bastian Nordmeyer, Frank Schmidtmeier,
Mohamed Shafik, and Bärbel Mertsching

University of Paderborn, GET Lab,
Pohlweg 47-49, 33098 Paderborn, Germany
{lueke,nordmeyer,schmidtmeier,shafik,mertsching}@
get.uni-paderborn.de
http://getwww.uni-paderborn.de/

Abstract. In this paper we present the design and development of a rapid prototyping system for algorithm developers in the field of image processing. The prototyping system is designed to give algorithm developers usually working in software the opportunity to implement and test their algorithms on FPGAs without any deeper knowledge of hardware development. Therefore, the C-to-FPGA tool Impulse C is used. As an example implementation we chose a scan-line segment based stereo vision system with a bilateral filter for image preprocessing. The actual system is targeted for the Xilinx SP601 board.

1 Introduction

Using high-level languages (HLL) such as C/C++ for the development of image processing algorithms is quite common because of several advantages. Firstly, the C/C++ is widely spread and well-known by many scientist. This leads to a variety of already existing applications and code sources for standard operations such as filtering and data import and export. Secondly, HLL support the necessary flexibility for developing innovative algorithms due to the high level of abstraction. Finally, a higher level of abstraction can reduce the lines of source code, which results in less development time, a better portability and reduced maintenance costs.

Firstly, C/C++... oder the C/C++ language...

On the other hand, especially image processing algorithms are often ideal candidates for hardware acceleration due to their inherent parallelism. Therefore, the performance can be significantly improved by implementing them on hardware such as FPGAs. One way to implement the algorithm in hardware is to manually reimplement it in a hardware description language such as VHDL. An experienced hardware developer can get a good performance out of the algorithm by doing so. But this method is time-consuming and the resulting implementation is not very flexible regarding change requests or improvements. Another way to get a hardware description of the algorithm is to use so called C-to-FPGA compilers. These tools generate for example VHDL code automatically out of a language similar to C. A number of these tools such as *Impulse C* or *CatapultC* have emerged within the last decade.

Our particular interest is in developing image processing algorithms such as stereo vision algorithms for mobile robotic applications. Because of the limited processing

power of the mobile robots – e.g. there is no GPU on our robots – we use FPGAs for the image preprocessing to unload the main processor. For developing and testing the algorithms on the FPGAs of the robots we were looking for a cost-efficient rapid prototyping environment which could easily be used by algorithm developers without a deeper knowledge of hardware development.

Therefore, we decided to chose Impulse C as input language which readily meets the easy-to-use criterion [1] and produces good results in terms of short development time, low cost, and sufficient performance.

Furthermore, the rapid prototyping system has to provide standard interfaces to Impulse C streams for the image data in- and output so that the algorithm developers can concentrate on the algorithm itself. As an input, the presented system uses an image input interface, where e.g. data from cameras can be buffered, and an output interface that provides a low level ethernet connection. Due to performance and resource considerations we developed the system presented in this paper as a pure hardware system without any usage of hardware/software co-design.

We chose a scan-line stereo algorithm with a bilateral filter for image preprocessing as an example application. The stereo vision system provides depth information of the environment and is used on mobile robots for navigation and active vision applications [2]. Compared to other publications in the field of C-to-FPGA tools such as [3] or [4] this is a relatively complex image processing system.

Die ref gibts bei mir nicht

In the following sections the theoretical background is described first, followed by the introduction of our prototyping system with an interface for Impulse C applications before a detailed description of the application implementation is given. A conclusion will close this contribution.

2 Fundamentals

2.1 Impulse C

Impulse C from Impulse Accelerated Technologies [5] is a C-based development system for programmable hardware with or without processors. The language itself consists of a subset of ANSI C and additional Impulse C libraries.

The Impulse C programming model mainly uses two constructs: *processes* and *streams*. The processes perform the real computational work while the streams connect the processes for data in and output as well as synchronization. Therefore, a process usually has a stream interface and a process body where the algorithm or a part of it is implemented in a subset of ANSI C. Processes, which are assigned to hardware, work concurrently and are independently synchronized.

Generally, the implementation starts with the partitioning of the algorithm into processes. If the algorithm is already implemented in C or even C++ numerous code segments can be reused by simply copying the code lines into the processes. After the implementation is completed the whole algorithm can be simulated in software. Then the compiler generates a functional equivalent platform specific hardware description. This hardware description can easily be integrated into the vendors synthesis flow and then be implemented on the FPGA.

Besides the normal optimization automatically done by the compiler the software engineer can apply pragmas for hardware optimization techniques such as pipelining and loop unrolling to further optimize the generated code [6].

In this way, Impulse C supports software-oriented design methods and thus is easy to use by algorithm developers with software background.

Die ref gibts bei mir nicht

2.2 Bilateral Filtering

Due to the fact that natural images are noisy, a quality enhancement of the input image is necessary. A variety of image filters specialized for different applications is available. For our purpose the bilateral filter is an appropriate filter, i.e. a smoothing filter with edge preserving properties [7]. The filter has two functions allowing it to produce smoothing and edge enhancing in the same run. For the edge enhancing, an intensity distance function f is used and smoothing is done with a geometric distance function denoted as g. Let (x, y) be a pixel in the image, $I(x, y)$ its intensity and $I'(x, y)$ the resulting intensity after filtering, then the bilateral filtering can be described as

$$I'(x,y) = \frac{\sum_{(\xi,\psi)\in\Omega} I(x,y) f(I(\xi,\psi) - I(x,y)) g((\xi,\psi),(x,y))}{\sum_{(\xi,\psi)\in\Omega} f(I(\xi,\psi) - I(x,y)) g((\xi,\psi),(x,y))} \tag{1}$$

on a filter kernel Ω.

In [8] an approach that uses histograms for the filter kernel in a median filter was introduced to enhance the runtime. In [9] the runtime was further improved by using multiple histograms and it was shown that in the bilateral filter the weighting function for the distance becomes constant for a given intensity in the case of a spatial box filter. In this case the bilateral filtering can also be done in time $O(\log(r))$, in the same manner as for the median filter.

2.3 Scan-line Based Segmentation

The goal of stereo algorithms is to establish pixel correspondences between the left and right images. In order to achieve reasonable results, two geometric constraints are used: first on the imaging systems, i. e. the input stereo images are *rectified* where the epipolar lines are aligned with corresponding scan-lines. And second on the scene, i. e. the *smoothness assumption* where the disparity map containing the depth information is smooth almost everywhere except at the border of objects assuming that scene is composed of smooth structures which in the case of autonomous mobile robots applications is not granted.

In this work, we have implemented a fast scan-line segment-based stereo vision [10] employing a modified optimization module within the scan-line framework. The inter-scan-line smoothness resulting from line segmentation leads to overcome the ambiguity problem without the use of a recursive smoothing function as in Belief Propagation approaches or facing narrow front objects problem as in Dynamic Programming algorithms.

The first step of this approach is the line segmentation of the right image as the reference image, in which the epipolar line epl_y is segmented into different labels $l_i(epl_y) \in \Gamma$ in the label space Γ based on the color differences between a seed pixel $P_s^{l_i}$ and the neighborhood pixels of the same epipolar line:

$$P_r(x+k,y) \in l_i(epl_y) \qquad \forall\, |P_s^{l_i} - P_r(x+k,y)| < \tau_c, \tag{2}$$

where τ_c is the Euclidean color distance threshold. The correspondence problem is formulated as an energy minimization function

$$E(d_\Gamma) = \operatorname{argmin}(E_{\text{data}}^{l_i}(d_\Gamma) + E_{\text{smooth}}^{l_{(i,d)\in K}}(d_\Gamma)) \tag{3}$$

between segments of the input images, where $E(d_\Gamma)$ is the disparity map of line segments and K is an adjacent segment in a particular epipolar line epl_y. The data term $E_{\text{data}}^{l_i}(d_\Gamma)$ of the energy function is the matching cost between a segment $l_i(epl_y)$ in the reference image and the opponent segments $l_{i,d}(epl_y)$ in the target image. The smoothness term $E_{\text{smooth}}^{l_{(i,d)\in K}}(d_\Gamma)$ encodes the smoothness assumption.

For performance optimization, an effective and simplified smoothness term

$$E_{\text{smooth}}^{l_{(i,d)\in K}}(d_\Gamma) = \lambda(\ell_{l_i}) \,.\, |d_{l_i} - d_{l_K}| \tag{4}$$

within the scan-line optimization framework is implemented, where $\lambda(\ell_{l_i})$ is an ascending function to the length of the current segment ℓ_{l_i} used to penalize depth discontinuities. The concept behind the function is to balance the relation between the disparity of a segment and the sum of the matching cost of the segment points. While the matching cost is affected by the length of the segment, only one disparity value is assigned to all the segment points and the best value is chosen within a winner take all (WTA) scheme.

3 Base Prototyping System for Image Processing

3.1 Overview and Basic Considerations

Our goal was to design a cost-effective and flexible prototyping system based on Impulse C which could easily be used by algorithm developers untrained in hardware design methods to implement and test their algorithms on an FPGA. Because we wanted to reserve as much space as possible and get the best performance for the actual algorithm under the given constraints we developed the system for hardware without a processor. We set image data storage, transmission of results, and of course an easy to use connection to Impulse C as basic requirements of the system so that the developers can later focus on the implementation of their algorithms. Optionally, it should support an easy transmission of internal data for a simple debugging and testing. Figure 1 shows the architecture of the system.

3.2 Image Input Interface

The *image input interface* provides two stereoscopic camera images for the image processing system. There are two ways to access the data by Impulse C application. One

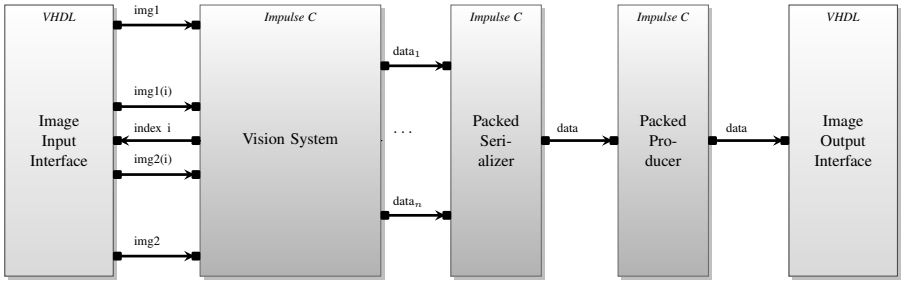


Fig. 1. System architecture with the image input interface, the vision system (user application), packet producer, packet serializer, and image output interface.

possibility is to send one pixel of the image after another as soon as they become available via a stream. These streams can be used for algorithms requiring only consecutive pixels, or to transmit real camera images for debugging purposes. But some algorithms need arbitrary pixel positions for their calculations. Because it is not possible to store the whole image data in internal memory we provide another possibility to access the image data. An Impulse C process called *index generator* sends the currently required pixels positions to the image input interface and then the requested pixel are transmitted via streams to the application. The advantage of this approach is that from the developer's view point it does not matter how the image input interface obtains the image information. It is ensured that all required information is available.

3.3 Image Output Interface

As the output image interface we chose ethernet for the transmission of processed data to a host PC. Since there is not enough space for a TCP/IP or UDP/IP stack on the FPGA, we use raw IEEE 802.3 ethernet frames to send data. Such a frame is depicted in figure 2. Besides the usual information it contains an increasing packet number for

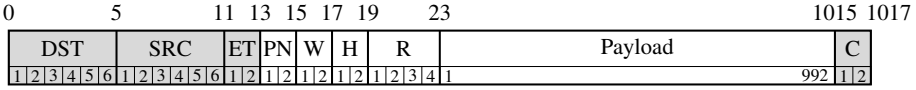


Fig. 2. Ethernet frame: The frame consists of a destination (DST), source (SRC), ethernet type (ET), packet number (PN), width (W), height (H), reserved part (R), payload, and checksum (C). The gray parts are defined by the IEEE 802.3 standard.

each image as well as width and height. This allows the receiver to construct parts of the image if there is a packet loss.

For each output stream the ethernet frames are assembled by an Impulse C process called *packet producer*. This process fragments the data into packets and attaches the

corresponding header. The *packet serializer* – another Impulse C process – buffers one packet per output stream and sends them successively to the image output interface.

The decision to implement the packet producer and the packet serializer as Impulse C processes is part of the concept of our prototyping system to support a software-oriented design method. Both processes allow a debugging in hardware similar to the *printf*-debugging well-known by software engineers by connecting streams to the packet producer in the configuration part. No new processes have to be implemented.

4 Scan-line Segment Based Stereo Vision with Bilateral Filtering

4.1 Overview of the Image Processing System

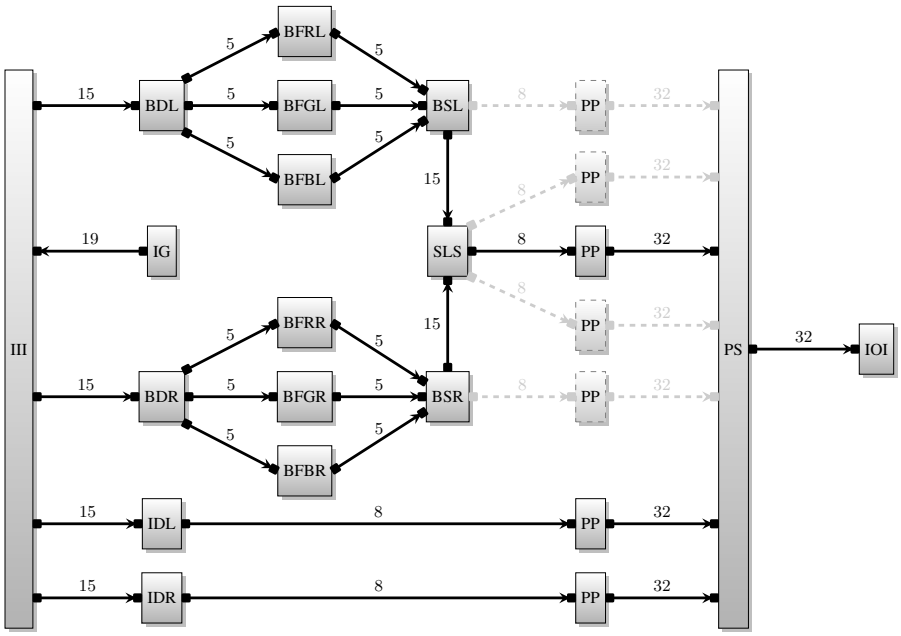


Fig. 3. Impulse C process flow chart of the image processing application. Impulse C processes consist of an index generator (IG), bilateral distributor for left and right images (BD(R|L)), bilateral filter for the left and right image as well as for the red, green, and blue color channels (BF(R|G|B)(L|R)), a bilateral serializer for left and right images (BS(L|R)), the scan-line segmentation (SLS), the packet producer (PP), and the packet serializer (PS). (III) and (IOI) are image input and output interfaces. The dotted lined streams and processes can be used for debugging.

As stated in section 1 the test application for our prototyping system is a scan-line segment based stereo vision algorithm with a bilateral filter, both of which were introduced in section 2. The application works on color images in the RGB color space.

In general, the system operates as follows: Via an index generator the current pixels of the left and right input images stored in the image input interface are accessed by the bilateral filter. The bilateral filter performs the filtering for both images and each of the three colors separately. Then the filtered data is sent to the scan-line segmentation, which generates a disparity map containing the depth information. This map is transferred via the image output interface.

As the first step of the implementation we partitioned the whole algorithm into several Impulse C processes. The bilateral filter was divided into three different processes (distributor, filter, and serializer) while the scan-line segmentation was assigned to a single process. This partitioning guarantees a flexible and clearly arranged structure of the implementation. The whole system including the index generator, the bilateral filter, the scan-line segmentation as well as the packet producer and serializer is shown as a Impulse C process flow chart in figure 3. Note that the processes in Impulse C such as the bilateral filter are only implemented once and then reused. The partitioning of the bilateral filter into three processes is owed to the fact that we are doing color image processing. Therefore, a speedup can be realized by running the filter for each color in parallel. This makes it necessary to distribute the three colors of each pixel to filters and afterwards merge them, done by the distributor and serializer process.

4.2 Preexamination of the Image Preprocessing

Before the actual implementation could be started, some adaptations and a parameterization of the bilateral filter had to be done. For the distance part of the bilateral filter equation a spatial box filter was applied, whereas $g((\xi,\psi),(x,y))$ became constant within the square window for an arbitrary intensity function [9]. But since a spatial box filter approximates a Gaussian blur when applied several times – and we are not interested in images that are too blurry – the results are acceptable with only one run. For the edge enhancing part $f(I(\xi,\psi)-I(x,y))$, however, the Gaussian function

$$G(I(\xi,\psi)-I(x,y)) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(I(\xi,\psi)-I(x,y))^2}{2\sigma^2}} \tag{5}$$

is used to calculate the weighting of the pixel difference. To find out an appropriate value of the standard deviation σ which yields sufficient results for the whole system we simulated it in software for different σ. The results are depicted in figure 4. The images show that with an increasing σ pixels with a high intensity difference to the center pixel are weighted higher, hence the image gets blurrier. The other way around, edges are better preserved with a decreasing σ. The resulting depth map has less artifacts when the input images are less detailed. The depth map for the filtered image with $\sigma = 1.00$ has no artifacts, but it lost a lot of detail of the scene in the image. E.g. the lamp holder and the red bottle disappeared completely. As a good compromise between artifacts and details we chose $\sigma = 0.15$.

4.3 Implementation of the Bilateral Filter

As described in section 2.2 our implementation of the bilateral filter uses histograms to store the image data of the filter kernel. Let H be the histogram of a filter kernel with

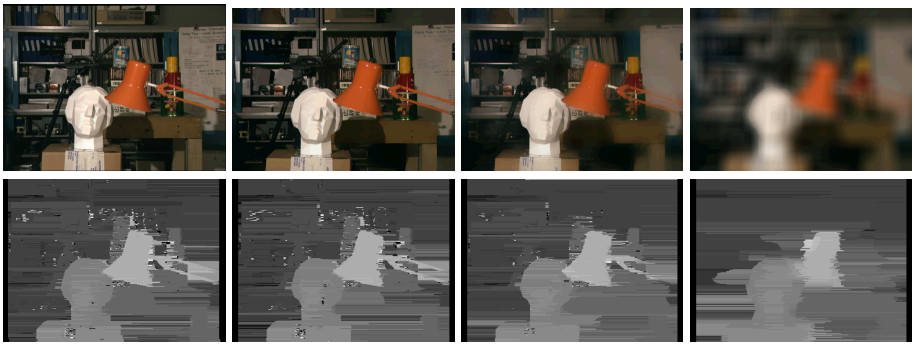


Fig. 4. Results of bilateral filtering with the Tsukuba stereo image set. From left to right: Without bilateral filtering and increasing σ. Below the filtered images, the resulting depth maps are shown. The resulting depth map has less artifacts when the input images are less detailed.

$H(i)$ representing the number of pixels in the square window with intensity i. The value for the corresponding pixel (x, y) can then be calculated with the equation

$$I'(x,y) = \frac{\sum_{i=0}^{2^d-1} i \cdot G(i - I(x,y)) \cdot H(i)}{\sum_{i=0}^{2^d-1} G(i - I(x,y)) \cdot H(i)}, \tag{6}$$

where the color depth is represented by d. A typically employed camera on our mobile robots has a color depth of five for each color which results in a relatively small histogram width of only 32. This means that for calculating one filtered value 3×32

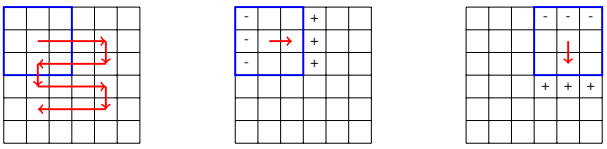


Fig. 5. (a) Processing sequence of bilateral filter, (b) horizontal histogram update, and (c) vertical histogram update.

multiplications, 2×32 additions, 32 subtractions, 32 lookups in the Gauss lookup table and one division are required. The real benefit of the histogram implementation is the independence of kernel size which allows relatively large kernel size, e.g. in our case of 25×25.

After each sliding of the filter window, the histogram has to be updated. Since histograms are distributive, this can be done by adding the pixels that are new in the filter window and subtracting pixels that fell out. The processing sequence and the updating procedure is illustrated in figure 5. Because we use fix point arithmetic on the FPGA the Gaussian function is implemented as a lookup table with an accuracy of 16 bit.

Since the storage of all necessary lines of the image – in our case 25 – takes a great amount of block RAM resources which are not available on small and medium sized FPGAs we used an index generator introduced in section 3.2. This process generates indices to address the pixels in memory needed by the bilateral filter.

4.4 Implementation of the Scan-line Segmentation

For the implementation of the scan-line segment based stereo vision algorithm in Impulse C we used an existing implementation written in C++ which shortens the development time.

The implementation of the scan-line segmentation algorithm buffers one complete line in block RAM for both images. Then, for the right image the size of the first segment is determined. Starting from the first pixel of the segment, the color distances between the left and the right image are calculated and added up. This is done with a shifted window on the left image until a maximum difference is reached. The depth for this segment is the distance between the left and the right image segment with the lowest color distance. After that, for a segment of a certain length the assigned depth values for each pixel of the segment are sent to the packet producer. This process continues with the subsequent segments until the end of the line is reached.

A problem with the scan-line segmentation algorithm occurs when a segment from the right image is matched in an area of the left image that has the same color but is not the corresponding area. This can happen in any scene with a checkerboard pattern. A possible improvement of this was found by extending the segments in the matching phase of the algorithm. The calculation of the color distance is started some pixels before the beginning of the original segment. Thereby, some color values from the former segment are also evaluated. With this setup, in a pattern of the form “blue white red white” the first white segment in the right image cannot be matched onto the second white segment in the left image. Figure 6 shows the effect of this improvement. In the

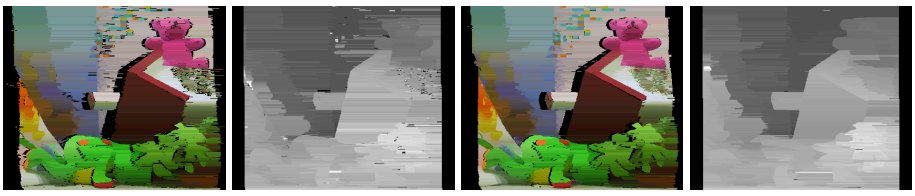


Fig. 6. The images show the matching with and without segment extension and the corresponding depth maps

first pair of images it can be seen, that the algorithm has difficulties matching the white spaces in the periodic table to the correct segments in the other image. This problem does not occur in the second pair of images, which is produced with the segment extension improvement.

Since each line in the scan-line segmentation algorithm is processed independently from all other lines, there is a parallelization potential. Similar to the procedure used by the bilateral filter, it is possible to use a distributor process that sends every second line to two different scan-line segmentation processes. The results then have to be merged again and can be streamed to the packet producer. With this approach a massive parallelization is possible. But since the scan-line segmentation process uses a big chunk of block RAM as well as slices from the FPGA, our ability to implement this idea is limited.

4.5 Results and Discussion

Impulse C allows a semi-automatical optimization of the generated hardware description by using pragmas in the source code. We use the three most important pragmas, i. e. the *pipeline*, the *nonrecursive*, and the *flatten* pragma to optimize our application.

The pipeline pragma can be applied to non-nested loops and will build a pipeline for the body. The nonrecursive pragma may be set after declaring arrays to tell the Impulse C compiler that this array is never written and read in one pipeline. The flatten pragma flattens the logic of if-else blocks and switch statements as well as hardware primitive functions. It is implicit when pipelining is used.

As a cost-efficient platform for the implementation we chose the Xilinx SP601 evaluation board with a Spartan-6 FPGA. The Spartan-6 provides different types of slices: 2278 slices, 18224 register slices and 9112 LUT slices. The only critical type for our application are the LUT slices. For this platform, we synthesized and simulated the whole application and parts of it with different optimization pragmas. All tests presented here start without any optimization. Then optimization pragmas are successively added to different parts of the algorithm.

As a first test we only examine the bilateral filter with a kernel size of 25×25 and a histogram width of 32. The results are given in table 1. The performance of the bilateral

Table 1. Results of the Bilateral Filter

Optimization	Slices	Slice Register	Slice LUT	Time per pixel
without	138	309	339	3480 ns
+ pipelining convolution	35	106	93	2870 ns
+ pipelining histogram update	51	103	132	2410 ns
+ pipelining in serializer/distributor	55	106	131	2410 ns
+ nonrecursive arrays	54	103	132	2410 ns
+ primitive function flatten	56	106	130	2410 ns

filter can be deduced from the time one pixel requires, since exactly the same operations are applied to every pixel.

Pipelining in the convolution part of the bilateral filter reduces the required time per pixel by 17.5%. The reduction in used slices is most likely due to a different optimization in the synthesis. Another speedup is achieved by pipelining the histogram update stage. Additional pragmas in the other parts have no effect on the speed. That also means the serializer and distributor are not bottlenecks in the system.

In a second test we examined the scan-line segmentation as a stand-alone application. Again we successively added pragmas. The results of this tests are shown in table 2. A speedup could be generated by using pipelining in the pixel conversion part, that

Table 2. Results of Scan-Line Segmentation

Optimization	Slices	Slice Register	Slice LUT	Time per 5 lines
without	320	340	1046	9342852 ns
+ pipelining pixel conversion	279	398	874	8665593 ns
+ pipelining distance calculation	424	825	1089	3656740 ns
+ pipelining depth calculation	403	826	1078	3657429 ns
+ nonrecursive arrays	438	826	1065	3657081 ns

reads the 15 bit input values from the stream and writes it to three arrays and by the distance calculation part, that determines the distance between a segment of the right image and the matching on the right image. Calculating the depth and using the nonrecursive pragma for the arrays did not have any mentionable effect.

The scan-line segmentation however, has different timings for different input lines. We simulated the calculation of 5 lines to determine an average value.

In the last test the whole system, as depicted in figure 3, was simulated. Table 3 shows the results. Applying pragmas can have a great impact on the performance of the

Table 3. Results of the complete system

Optimization	Slices	Slice Register	Slice LUT	Time per 5 lines
without	2094	5055	7298	12144104 ns
full	2190	6243	8211	5037055 ns

system while the demand of slices only increases slightly. Based on the results, frame rates of 5.403 fps, 5.697 fps and 4.136 fps for the optimized bilateral filter, scan-line segmentation and the complete system can be deduced for an image size of 320×240.

5 Conclusion and Future Work

We have presented an easy to use and cost-efficient rapid prototyping system based an Impulse C. The system is intended to give image processing algorithm developers usually working in software the opportunity to implement and test their algorithm on FPGAs.

As an example implementation we chose a scan-line segment based stereo vision algorithm with a bilateral filter. The algorithm was completely implemented in Impulse C and partitioned into processes by partially using existing lines of C-code. Additionally, some improvements such as the segment extension were implemented to get better results for the disparity map. A sufficient performance was reached by iterative optimizations.

As a general conclusion we can say, that Impulse C meets our expectations.

Compared to the complexity of the design the performance achieved by the system was acceptable, although it could not compete with a hand-crafted VHDL implementation. Therefore, the application was written and optimized completely in software which makes it more flexible for major modifications and shortens the development time dramatically. Especially this is a benefit for algorithm developers, who want to test different variants of an algorithm. Additionally, no deeper knowledge of hardware development is required.

For the future, we plan the implementation of further image processing algorithms in Impulse C as well as a connection of stereo cameras to the prototyping system. The system will be running on the SP601 development board that provides the Spartan-6 FPGA which was used in the simulation.

References

1. El-Araby, E., Taher, M., Abouellail, M., El-Ghazawi, T., Newby, G.B.: Comparative analysis of high level programming for reconfigurable computers: Methodology and empirical study. In: Programmable Logic, 2007. SPL '07. 2007 3rd Southern Conference on, Mar del Plata (February 2007) 99–106
2. Ali, I., Mertsching, B.: Surveillance System Using a Mobile Robot Embedded in a Wireless Sensor Network . In: International Conference on Informatics in Control, Automation and Robotics. Volume 2., Milan, Italy (April 2009) 293 – 298
3. Trexel, E., Stamler, D.: Accelerating bilateral filtering in Xilinx FPGAs. Xcell Journal (2) (2007) 53–55
4. Kayankit, W., Suntiamorntut, W.: Hardware/software co-design for line detection algorithm on FPGA. In: Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, 2009. ECTI-CON 2009. 6th International Conference on. Volume 01., Pattaya, Chonburi (May 2009) 604–606
5. Impulse Accelerated Technologies: http://www.impulseaccelerated.com
6. Impulse Accelerated Technologies: Impulse C User Guide. (2009)
7. Tomasi, C., Manduchi, R.: Bilateral filtering for gray and color images. In: ICCV '98: Proceedings of the Sixth International Conference on Computer Vision, Washington, DC, USA, IEEE Computer Society (1998) 839
8. Huang, T.S.: Two-Dimensional Digital Signal Processing II: Transforms and Median Filters. Springer-Verlag New York, Inc., Secaucus, NJ, USA (1981)
9. Weiss, B.: Fast median and bilateral filtering. ACM Trans. Graph. **25**(3) (2006) 519–526
10. Shafik, M., Mertsching, B.: Real-Time Scan-Line Segment Based Stereo Vision for the Estimation of Biologically Motivated Classifier Cells. In: KI 2009: Advances in Artificial Intelligence. Volume 5803 of LNAI. (September 2009) 89 – 96