Hardware-Software Co-Design in Practice: A Case Study in Image Processing

Ralf Joost
Institute of Applied Microelectronics and Computer Engineering
University of Rostock
Rostock, 18051, Germany
ralf.joost@uni-rostock.de

Ralf Salomon
Institute of Applied Microelectronics and Computer Engineering
University of Rostock
Rostock, 18051, Germany
ralf.salomon@uni-rostock.de

Abstract – **Recent research on field programmable gate arrays (FPGAs) has yielded devices capable to compete against both digital signal processors and general purpose processors. Due to their resources in terms of logical gates and processing speed, FPGAs have made their way into numerous new areas previously dominated by dedicated hardware architectures, i.e., ASICs, digital signal processors, and microcontrollers. This paper presents a case study in field of image processing in which the new FPGA generation allows for a system design that exhibits near-to-hardware performance at low system development costs. The system's high performance is due to application-specific hardware components that are attached to a soft-core processor; the design follows the hardware/software co-design guidelines. This standalone embedded system executes both general control tasks written in C/C++ and sophisticated image processing algorithms implemented in the hardware description language VHDL.**

I. INTRODUCTION

Today, system designers are faced with two main problems: on the one hand, the systems they are designing have to fulfil tasks with ever increasing complexities. On the other hand, the production costs are not allowed to grow significantly or are even supposed to shrink. Although the computational power was the major concern in the past, the design aspects have moved to issues, such as low energy consumption, maintenance, and robustness, in recent years.

The markets today observe that flexibility, mobility, and energy awareness are becoming more and more important to virtually all customers. This applies not only to typical end-users but also to industry, particularly cutting edge technology manufacturers. Current trends suggest that programmable hardware platforms, widely known as field programmable gate arrays (FPGAs), are a major option to fulfil the aforementioned design goals.

Currently available devices offer highly sufficient complexities at moderate costs. These devices not only allow for the implementation of various hardware components but also for the concurrent realization of a broad mixture of them. In addition, their sizes also allow for the aggregation of functional components, which used to be spatially distributed over several microchips. They thus simplify the board layout and may improve various design issues, such as timing behavior, routability, and power consumption.

It is important to note though that the available hardware platforms do not suit the complexities of all potential functionalities. It might be, for instance, that either the implementation would exceed an FPGA's capacity by far or that the design efforts are simply too high. This gap can be closed by the usage of soft-core processors. The key point of soft-core processors is that they are integrated into the very same FPGA, but that they take over those functionalities that are otherwise to be executed on additional processors or microcontrollers. In other words, by utilizing the concept of soft-core processors, an FPGA offers both enhanced software computation capabilities *and* resources for hardware implementation, and thus enables true hardware/software co-design on one single chip. This concept is visualized in Fig. 1.

This paper discusses an industrial case study in the field of image processing that demonstrates the usefulness and broad applicability of this concept. At its core, the application consists of the control of an exposing system, which is described in full detail in Section 2. The purpose of the system is to expose the printing plates with which the actual printing process is to be done. Because the printing is intended for rough surfaces, such as cardboards and wallpapers, the printing plates require a particular image processing. Those algorithms are explained in more detail in Section 3. Even so an implementation of this algorithm is straight forward, a software realization suffers from severe performance problems.

In order to illustrate the advantages of the entire hardware/software co-design process, Section 4 describes how to conceptually embed a soft-core processor into a commercially available FPGA platform. In addition, Section 4 discusses how to tailor the processor template to the designers needs.

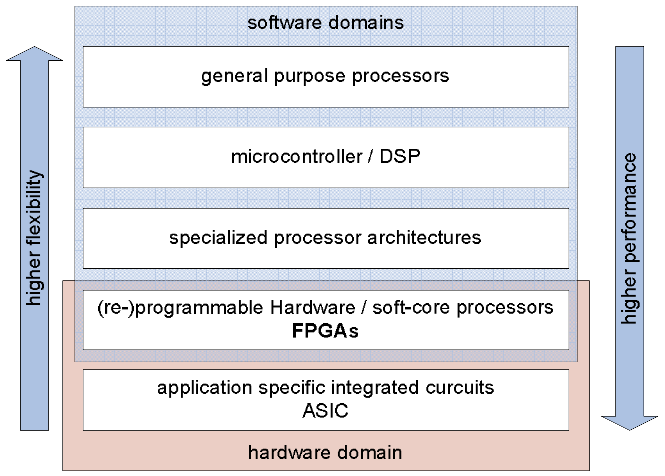

software domains
general purpose processors
microcontroller / DSP
specialized processor architectures
(re-)programmable Hardware / soft-core processors
FPGAs
application specific integrated curcuits
ASIC
hardware domain
higher flexibility
higher performance

Fig. 1: survey of hardware/software implementation techniques

Then, Section 5 shows how to realize the algorithm (as discussed in Section 3) onto the soft-core processor based approach (as already discussed in Section 4). The experimental results, as presented in Section 6, show that this approach yields a significant speedup of a factor of up to 68. Finally, Section 7 concludes with a brief discussion.

II. THE IMAGE PROCESSING APPLICATION

The case study has been done in close cooperation with Punch Graphix Prepress Germany Ltd. The goal of this research is to further improve the Computer to Conventional Plate (CtCP™) technology[1]. Computer to Plate (CtP) refers to a process with which an image, e.g. a newspaper page or advertisement flyer, is brought onto the surface of a printing plate with which later on the actual printing process is be done. Computer to *Conventional* Plate indicates that this image transfer is based on cost-effective ultra-violet (UV), rather than costly laser light. Such a printing plate exposing system manufactured by Punch Graphix is shown in Fig. 2.

For print media with smooth surfaces, such as newspapers, flyers, etc., the UV-Setter exposes offset printing plates with a regular thickness of up to 0.3mm. In recent years, offset printing has also been applied to media, such as wallpapers or cartons, that are characterized by rather *rough* surfaces. In order to obtain a homogeneous printing, thicker printing plates are required.

A. Problem Description

The main problem is that thicker printing plates impose higher requirements on the mechanical stability and that this is not guaranteed by the UV exposing process without further enhancements.

The reason is that the exposing process is a polymeric reaction caused by ultra violet light. Those areas of the printing plate that absorb a defined amount of light photons change their structural behaviour. The material hardens and becomes resistant against special liquids used to remove non-polymerized parts of the plate. Due to the effect of light diffusion, the exposing process requires additional efforts when thicker plates are utilized.

The problem with thicker printing plates is that their bottom areas do not receive as much light as necessary to sufficiently harden the polymeric material. This is especially true for single, free-standing pixels in the image content that should be transferred to the plate. In case of free-standing pixels, the mechanical stability of the polymeric base structure is furthermore interfered by the optical effect of light diffusion. Therefore, a post-processing of the entire image has to be performed to identify those free-standing pixels. Thereafter, those pixels are handled in an additional processing step[2].

[1] CTcP™ is a trademark by Punch Graphix Prepress Germany Ltd.

[2] This approach is currently under review of the German Patent Office and therefore not part of this paper.

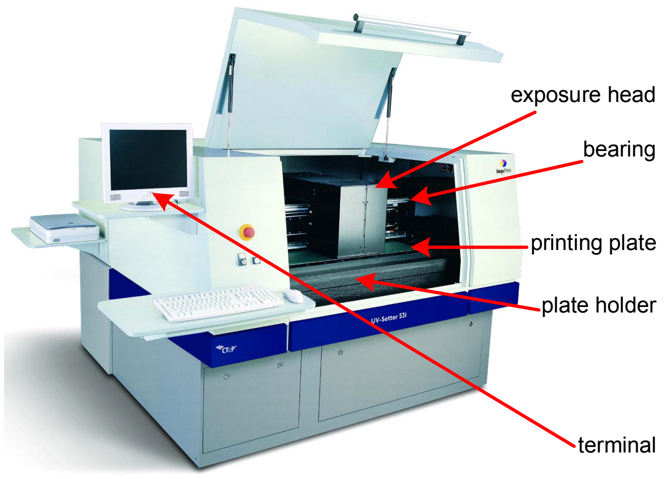


Fig. 2: A CTcP™ system, called UV-Setter

B. Design Goals

Traditionally, such data manipulating tasks are executed on specialized digital signal processors (DSPs). However, the incorporation of a DSP into the control system of the exposing system would require additional efforts to control the DSP workflow and to coordinate its activity with the memory management unit to ensure data integrity in the image memory. In combination with the usual control tasks, this approach would result in a multi device solution with the well-known problems and difficulties, such as board design, manufacturing costs, and programming issues.

In order to keep the manufacturing cost as low as possible, additional hardware components are *not* considered as a target solution. Thus, one of the central design issues of the system layout is the aim to execute the image manipulation routine inside an FPGA, which is already part of the existing control system [7]. In order to avoid the effects described above, the already existing soft-core processor could be made responsible for both the control tasks and the image pre-processing. This would be an ideal problem solution, if that soft-core processor had a performance comparable with a digital signal processor. But as the DSP is a specialized core for image processing, a pure software solution running on a NIOSII soft-core will most likely result in a poor performance.

Thus, this paper focuses on an approach that extends the processor's core with a custom designed hardware module that exactly performs the required image processing at hardware speed. This is a feasible approach, since the processor is not available in a fixed architecture but in an easy-to-modify VHDL description.

III. THE IMAGE PROCESSING ALGORHITM

Every image that is processed by the UV-Setter consists of a two-dimensional array of picture elements (pixels). Each pixel can be set to one of the following two states: "true" meaning that the pixel should appear on the printing plate, and "false" meaning that the pixel should not appear on the printing plate. Thus, each single pixel can be represented by a single bit in, for example, a C/C++ program. Due to the available memory capacity, the application at hand compresses every image. It stores eight neighbouring pixels of a row in one `char` variable. The complete data structure then consist of a two dimensional array of type `char`. Furthermore, for the purpose of a simple data storage and processing mechanisms, the image is byte aligned, where unused pixels are defined as “false”.

As mentioned before, the image processing approach has to identify all free-standing pixels of the entire image. In so doing, the Moore Neighbourhood [8] of each pixel has to be examined. This results in the necessity to determine the eight surrounding pixels of every single spot. Then, the eight pixels of the Moore neighbourhood are accessed and analyzed.

Fig. 3 shows a small segment of an image with a randomly chosen central pixel and its corresponding Moore Neighbourhood. The following simplified pseudo-code tries to illustrate this approach:

```
getCurrentPixelByte(row, column)
getCurrentPixelStatusBit(index_x, index_y)
for all neighbours
  calculateNeighbourIndex()
  getPixelByte(row, column)
  getNeighbourPixelStatusBit(index_x,index_y)
  storeStatusOfNeighbour()
end
evaluateStatusOfAllNeighbours()
decideForCurrentPixel()
storeResult()
```

The central `for`-loop is the most time-consuming element in this piece of code, since it has to perform numerous shift and mask operations to extract the desired pixel bits (the central bit and all its eight neighbours) from the byte representation. A subsequent problem is that every specific byte has to be loaded from memory at nine different points in time. The decision for a pixel whether it is free-standing or not is rather simple. It is just an OR combination between all neighbouring pixels, and if this results in zero, a central pixel that is set to “true”, is called free-standing.

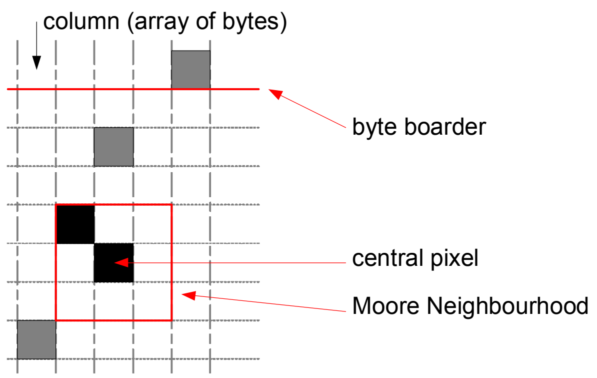


Fig. 3: segment of image content

IV. THE HARDWARE PLATFORM AND THE NIOS II SOFT-CORE PROCESSOR

Soft-core processors represent processors that are designed to fit and run inside an FPGA. In contrast to standard cores, they are not fixed hardware architectures. Instead, the core is available in a hardware description language, such as Verilog or VHDL [4]. This allows for their usage on a wide variety of FPGAs. All necessary adaptations are applied with the help of the synthesis tools of the designated FPGA platform. This also enables a wide range of possibilities for configurations and modifications of those cores according to the requirements of the system. This, for example, refers to the amount of the required data cache and/or instruction cache, the availability of hardware multipliers, or other specialized peripherals. In general, soft-core processors can be considered as equivalents to microcontrollers or “computers on a chip”. They combine a CPU, peripherals, and memory on a single chip. Through integrated standard or custom-made interfaces, they also provide access beyond the actual FPGA chip. Some available reduced instruction set computer architectures are:

1. Nios and Nios II CPUs from ALTERA [2] ,
2. LEON CPUs from Gaisler Research [6],
3. and the Microblaze CPU from XILINX[9].

Throughout this paper, the Nios II variant is of particular interest. An overview about the basic Nios II processor features can be found in [2]. The Nios II processor offers mainly three options to integrate custom-designed hardware into the FPGA-based system:

1. The very first and classical approach (Fig. 4) integrates the hardware component, aside from the processor system, as a separated instance into the FPGA. The connection between hardware and processor is established via a system bus interface, for which the Nios II processor employs the AVALON bus.

   The benefit of such a distributed design is its ability to test each component separately. Also, any modification of the extra hardware component impacts just one single instance of the entire system. This holds as long as the interface of the component remains unchanged. Due to the stand-alone design of the hardware component, a simple re-use of the design files in different architectures is supported. In such cases, only the connection interface has to be adapted to the new system. It should be mentioned though that the connection path as well as the interface impose additional latencies.
2. The second approach (Fig.5) integrates the hardware component into the processor system itself and connects it directly to the AVALON system bus. Software programs running on the processor can access the component via memory-mapped I/O.

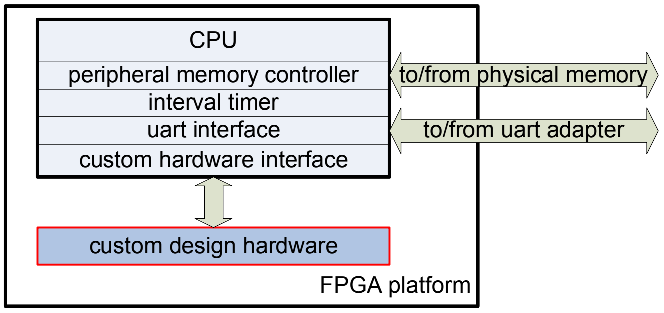


Fig 4: separated custom hardware

In comparison to the first method, this approach yields an improved performance. Furthermore, those components that feature the AVALON bus interface can be easily integrated into other AVALON-bus-based systems.

3. The third and most powerful approach implements the hardware component directly into the processor core (Fig. 6). In essence, the component presents a data path in parallel to the CPU's arithmetical logical unit (ALU). During system generation, special assembler instructions are generated for accessing the additional component. However, the application programming interface (API) offers several macros to simplify the access to the hardware component. One call to the custom hardware can transfer 64 bit of data, which is the content of two internal data registers. During the development phase, the designer has to specify how many clock cycles the additional integrated hardware requires to process its data and to examine its results. Thereafter, the results are stored in one of the processor's internal data registers. The application-specific hardware components that are integrated into the processor are called custom instructions, and are executed via the assembler's `custom` instruction. For a more detailed description of this approach, the interested reader is referred to [2]. The benefit of this approach's high performance outweighs the fact that the hardware design can only be used in combination with the Nios II processor core by far.

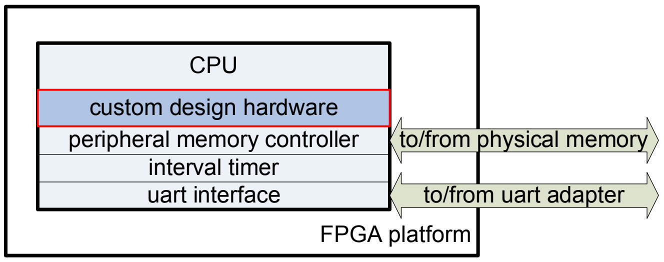


Fig 5: system integrated hardware component

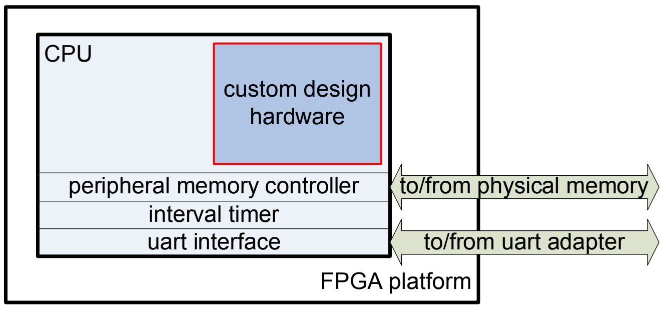


Fig 6: custom instruction approach

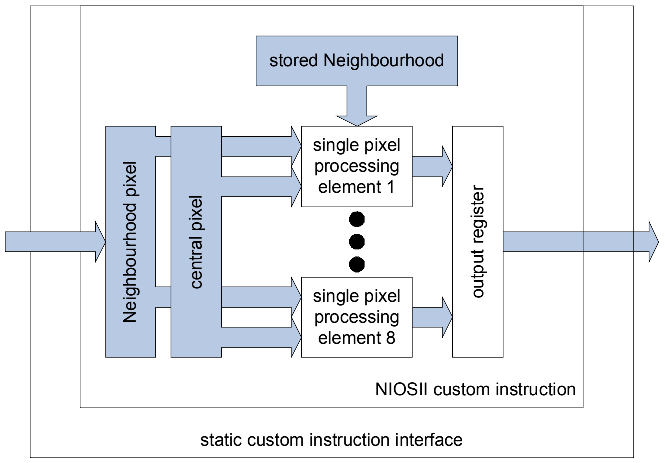
V. HARDWARE REALIZATION

As previously research has discussed [7], the UV-Setter favours an FPGA-based multiprocessor architecture for its next generation. That is, within the existing multiprocessor platform, a single soft-core processor being responsible for the image processing task is one available option. Since the existing system [7] is only capable of operating at a system clock of 50 MHz, a pure software approach runs into severe performance problems, if the images are of practically relevant sizes (see, also, Section 6). Therefore, this paper realizes the image pre-processing algorithms, as discussed in Section 3, as a hardware extension following the third system design option of Section 4.

Now, the basic question to be answered is what parts are to be realized in hardware and what parts are to be realized in software. One of the design constraints is the necessity to store both the original picture and the result of the pre-processing task in the available onboard memory. The reason for this double storage is that the UV-Setter needs *always* access to both the original unprocessed image data and the pre-processed image data. Therefore, the two images must reside in the globally accessible onboard memory. These design constraitns are another reason for choosing the third system design option.

Another reason for the chosen system design is that the image transfer between host-PC and exposing system requires a communication instance. On the host-PC side the image data transfer is accomplished via the standard Ethernet interface. On the UV-Setter's control system, the data transfer is realized by an additional onboard Ethernet connector that can be easily accessed by the soft-core processor's API. Furthermore, the memory integrity is guaranteed by memory protection mechanisms already provided by the Nios II processor.

In this system design, the determination whether a pixel is free-standing or not is done in hardware, whereas all other parts are directly executed by the Nios II processor. With respect to performance, it should be stressed that in each iteration, the software approach would be able to calculate the result just for only one single pixel. By way contrast, the hardware augmentation calculates eight pixels in parallel, as shown in Fig. 7.


stored Neighbourhood
Neighbourhood pixel
central pixel
single pixel processing element 1
single pixel processing element 8
output register
NIOSII custom instruction
static custom instruction interface
Fig. 7: schematic of image processing hardware

Furthermore, the problem-specific hardware component performs all the calculations for eight pixels in only two clock cycles, whereas a software solution would require at least eight iterations. Moreover, since the hardware processes entire bytes (containing eight pixels), any time-consuming bit-extraction operations are obsolete. Both mechanisms yield a significant speedup of the entire operation as is discussed in the subsequent section.

In summary, the software running on the Nios II processor core transmits a collection of three bytes to the hardware component at once. The already known pseudo-code section simplifies to:

```
getCurrentPixelByte(row, column)
calculateNeighbourIndex()
getNeighbourPixelBytes(row, column)
callHardwareForCurrentPixel(NeighbourBytes)
storeResult()
```

VI. PERFORMANCE RESULTS AND DISCUSSION

Both the hardware-based and software-based approaches to image processing were implemented on an ALTERA® Nios II Development Kit Cyclone Edition [3]. The development kit contains a low-cost 1C20 Cyclone FPGA that offers 20k user-programmable logic elements (LE). For further details of that device, the interested reader is referred to the pertinent literature [1]. As an out-of-the-box general-purpose processor, the Nios II soft-core comes in three different versions. In its standard version, the Nios II processor requires approximately 1400 LE to provide common functionality including a 4KB instruction cache. The *entire* system, consisting of the processor core, various peripherals, such as a memory controller and a system timer, as well as the additional hardware component for the image processing, consumes a total of 2900 LE. Without any further optimizations by the FPGA synthesis tool, the system operates at a clock rate of 50MHz.

To evaluate and compare the performances of the two approaches, the processor generates test images at random. The considered sizes were 1000x1000, 1500x1500, and 2000x2000 pixels. Then, the processor executed both approaches and validated the result.

The times necessary for each processing approach were measured according to the number of required clock cycles, and are illustrated in Fig. 8. For every case, i.e., software-based and hardware-based, the figure shows two variants, one with additional hardware multipliers and one without. The results clearly show that without additional multipliers the hardware-based calculation of the free-standing pixels in an image is up to 54 times faster than the software-based variant. This speedup is due to two important effects. First of all, the necessity to evaluate the status of the eight neighbours of each pixel requires complex index calculations in combination with expensive mask operations. Due to the fact that this operation is transferred to hardware elements, no such operations have to be calculated in the hardware-based process. Keeping in mind that eight neighbours are available, this is likely to yield an improvement of a factor of about eight. Second, it should be stressed that in opposite to the software, which performs the calculations for each pixel separately, the custom instruction performs the calculations for eight pixels (which are stored in one byte) in *parallel*.

As a simple optimization to further improve the performance, the Nios II processor was augmented with hardware multipliers; they are intended to speed up the necessary index and shift operations. With this augmentation, the size of the entire system has increased to 3600 LE. The hardware multipliers speed up the software approach by about 15%.

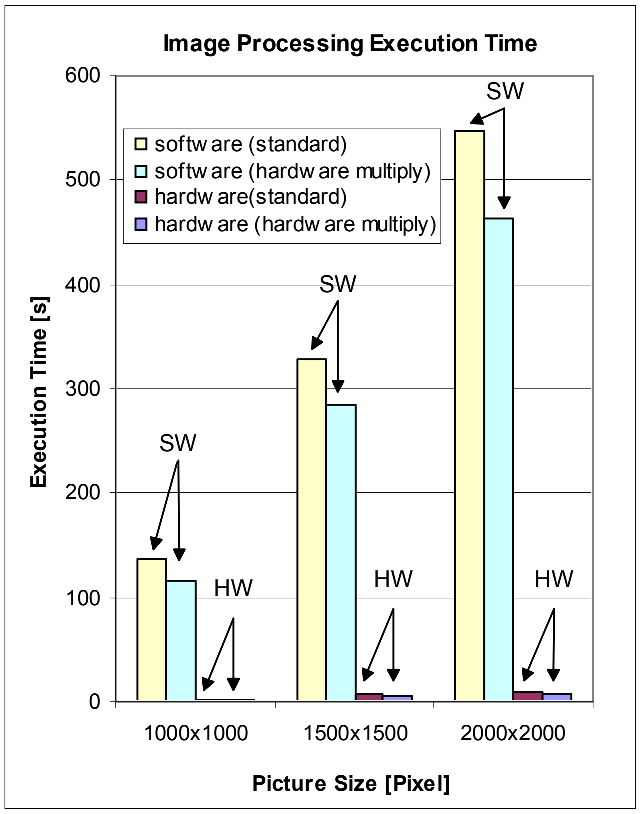


Fig. 8: image processing execution time

It might be interesting to note that the integration of hardware multipliers also significantly improve the performance of the hardware-based approach by about 35%. With hardware multipliers integrated, the custom instruction approach outperforms the software variant by a factor of up to 68.

From a system point of view, this hardware/software co-design approach also enables various other options. The pixel identification process discussed in this paper was kept rather simple. In more complex applications, such as the detection of specific pixel configurations or increase of the Moor Neighbourhood, the image pre-processing algorithms might quickly exceed practical limits, if implemented entirely in software. Hardware-based augmentations, on the other hand, are way more flexible and can easily exploit inherent parallelisms.

Another point concerns the options of how to deal with increased complexities. If a certain task would let the number of operation grow by a factor *n*, the runtime of a software-based implementation would increase by that factor *n*. That is, the software's runtime is tightly coupled to the software's complexity. By way contrast, a hardware-based realization can exploit an approach known as pipelining. Here, the processing would be done in *n* subsequent stages. After a delay of *n* clock cycles the processing chain would output the first result. But then, the processing hardware would deliver one new result in every subsequent clock cycle. In other words: rather than slowing down the (image) processing by a *factor* of *n*, the hardware realization would induce only an *additional* latency of *n* clock cycles. Overall, hardware based augmentations can decouple the processing time from the computational complexity within certain limits.

VII. CONCLUSION

This paper has presented an industrial case study on the design of the image processing parts of a printing plate exposing machine. The consequent application of hardware/software co-design principles has let to an FPGA-based solution that reduces the image processing time by a factor of about 68 as compared to pure software solutions; in order to be fair, this comparison assumes identical implementation platforms. It was furthermore discussed that without influencing the computation time too much, the presented approach is able to adapt to increasing processing complexities.

Another benefit is that the presented hardware-based approach would not require *additional* hardware components, such as further FPGAs or DSP. Rather, the utilization of user-defined custom instructions still leads to single-chip solutions with all their advantages with respect to design, validation, and maintenance efforts and costs. It should also be mentioned here that the flexibility of FPGAs enables a quick redesign of the utilized hardware. This redesign can even be done on-site, that is, when the FPGA is already installed in the target system.

Future research will be devoted to the following extensions. First of all, the approach shown in this paper just identifies free-standing pixels. The next step is to enable the hardware platform to recognize several pixel patterns, which can be used as a starting point for further image processing mechanism to improve the result of the printing plate exposure. Second, the Moore Neighbourhood might be enlarged to larger distances resulting in the nearest 24 or 48 neighbouring pixels.

ACKNOWLEDGMENTS

The authors gratefully thank Dr. Horst Steppat, managing director at Punch Graphix Prepress Germany Ltd. and John Hedde, director of research and development at Punch Graphix Prepress Germany Ltd., for providing the existing technology and the excellent cooperation. This research was supported in part by the German federal ministry of education and research, grant number 03i4919B.

REFERENCES

[1] Altera Corp., *Cyclone Device Handbook, Volume 1*, Altera Document C5V1-1.8, Altera Corp, San Jose, CA, Aug. 2005

[2] Altera Corp., *Nios II Processor Reference Handbook*, Altera Document NII5V1-1.2, Altera Corp., San Jose, CA, Jan. 2005

[3] Altera Corp., *Nios Development Board Reference Manual, Cyclone Edition*, Altera Document MNL-N2DEVLBDCYC-1.1, Altera Corp., San Jose, CA

[4] Altera *Corp., Quartus II Handbook Volume 4 - SOPC-Builder*, Altera Document QII5V4-1.0, Altera Corp., San Jose, CA, Feb. 2005

[5] basysPrint, *UV-Setters Series 7*, basysPrint Ltd., 2004

[6] Gaisler Resarch, *LEON2 Processor User's Manual – XST Edition*, Gaisler Research AB, Goteborg, Sweden, Jan. 2005

[7] Joost, R. and Salomon, R. *Advantages of FPGA-based Multiprocessor Systems in Industrial Applications*, Proceedings of 31st Annual Conference of IEEE Industrial Electronics Society (IECON), Raleigh, North Carolina, USA, 06.-10. November 2005

[8] Ozan, T. *Cellular Automata and Signals, a Review for IAM 570*, 2005

[9] XILINX Inc., *MicroBlaze Processor Reference Guide*, XILINX Document UG081 (v5.0), XILINX Inc., San Jose, CA, Jan. 2005