Elektrotehniški vestnik 70(5): 247–253, 2003
Electrotechnical Review, Ljubljana, Slovenija

HW/SW Co-design of Real-time Video Applications Using a Custom Configurable Prototyping Platform

Matjaž Finc, Andrej Trost, Baldomir Zajc, Andrej Žemva

University of Ljubljana, Faculty of Electrical Engineering, Tržaška 25, 1000 Ljubljana, Slovenia
E-mail: matjaz.finc@fe.uni-lj.si

Abstract. In this paper, a modular, configurable and versatile prototype platform for real-time video and image processing is presented. Based on the FPGA technology and a RISC softcore processor for data processing, the platform supports simultaneous HW/SW co-design and partitioning. This reduces application design cycle and shortens design iterations, especially considering the later design steps. As examples, applications of two different algorithms for motion detection are presented.

Key words: hardware (HW), software (SW), HW/SW co-design, video and image processing, FPGA, RISC softcore processor, motion detection, Markov random fields (MRF), iterated condition mode (ICM)

Sočasno načrtovanje strojne in programske izvedbe video aplikacij v realnem času z uporabo konfigurabilne prototipne platforme

Povzetek. V tem članku je predstavljena modularna, konfigurabilna in večnamenska prototipna platforma za obdelavo slik in videosignalov v realnem času. Platforma temelji na tehnologiji FPGA in mehkem procesorskem jedru RISC. Z visoko stopnjo konfigurabilnosti je omogočeno hkratno načrtovanje programske in strojne opreme pri realizaciji želenih algoritmov za obdelavo signalov. S takšnim pristopom se skrajša načrtovalski postopek in zmanjša število načrtovalskih iteracij, zlasti v kasnejših fazah načrtovanja. Kot zgled je predstavljena implementacija dveh različnih algoritmov detekcije gibanja.

Ključne besede: strojna oprema (HW), programska oprema (SW), hkratno načrtovanje strojne in programske opreme, obdelava slik in videa, FPGA, RISC mehko procesorsko jedro, detekcija gibanja, naključna polja Markova (MRF), iterativni pogojni način (ICM)

1 Introduction

The main purpose of this application was to research the theory and practice of the HW/SW co-design process from the initial specification to the final application [1, 2]. An efficient HW/SW co-design process is the key to design real-time embedded systems applications [3, 4]. To achieve the highest performance, the HW/SW partitioning should be considered in the earliest stages of the design flow. When designing HW parts, clear and distinct SW flow should be well defined and potentially critical SW parts should be recognized and predetermined. Thus, meeting requested constraints in later stages of the design flow requires less effort, reduces design cycles and cost. For prototype designs, configurable platforms should be used to speed up HW design, implementation and modification. Therefore, a modular configurable platform was set up in order to support a wide spectrum of potential video and imaging applications. The platform is based on FPGA technology and utilizes a configurable RISC softcore processor.

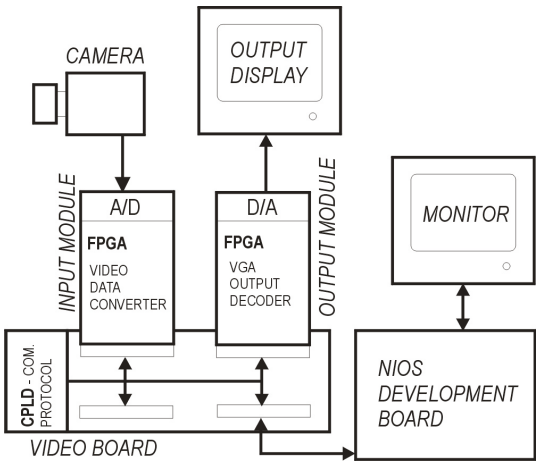

Figure 1. Video system

Received 20 March 2003
Accepted 21 May 2003

2 The Platform: Architecture

2.1 Video Board

The platform architecture can be divided in two main functional domains:

- communication (video board),
- data processing (data processing modules).

The platform, presented in Figure 1, is modular. The video board handles the data exchange between up to 4 different independent data processing modules. The data transfer is managed with the use of a master communication controller, implemented in a CPLD (Xilinx XC95144). The modules communicate via a shared bus through a special customized data stream protocol. According to the protocol, the controller handles the data transfer in terms of passing successive data streams between neighboring modules. The data transfer is unidirectional. For example, the main processing module receives processed data from the previous module (data acquisition) and sends processed data to the next module (data output).

The system communication frequency is 20 MHz, with the data transfer cycle of 5 MHz (1 clock period per each module).

2.2 The Input Module

The input module performs image/video data acquisition from an external black and white (B/W) camera through an 8 bit A/D converter. The input video data stream applies to the PAL video standard. The received video stream is decoded into sync signals and 8 bit video data (256×256 8 bit grayscale) and forwarded to the communication data bus on the video board. The PAL input frame rate is standard 50 frames per second interlaced, which means total 25 frames per second non-interlaced.

2.3 The Output Module

The output module receives and decodes the processed data from the video board and displays them onto a standard VGA monitor through an 8 bit D/A converter. The output resolution is identical to the input decoded signal (256×256 8 bit grayscale).

The layout of digital parts of input and output modules is identical. It is based on Xilinx XCS40 FPGA. Besides the dedicated functionality, input and output modules can be optionally configured to perform additional signal processing steps.

3 The Nios Development Board - The HW/SW Platform

For SW implementation of image and video algorithms, the use of a microprocessor is required. The use of additional HW for optimization contributes to the overall performance of the algorithm. For the highest degree of HW/SW integration, customization and configurability, a softcore processor was utilized.

For the main processing stage, the Altera Nios Development Board was chosen. The core of the board is an Altera APEX20K200E FPGA in which the Nios softcore processor and custom HW are implemented. The board operates at the frequency of 33 MHz.

The Nios softcore processor is a 16/32 bit softcore 5 stage pipelined RISC processor with 16 bit instruction length. It offers a high degree of HW configurability and customization: hardware accelerated multiplication, custom instructions, easy peripheral and custom HW integration, etc. The custom GNU compiler, assembler and linker are provided. With the supplied design tools, the Nios system presents a powerful HW/SW partitioning and integration stage.

The main processing core of the Nios system, illustrated in Figure 2, is the Nios CPU. It is connected to HW peripherals via a custom Altera Avalon bus. The bus is of a parametric master/slave type. The parameterized Avalon bus interfaces are automatically generated by a special Altera Nios generating tool (SOPC Builder) for every custom peripheral integrated into the design. The Nios system for our video and imaging applications comprises the following components:

- *Nios CPU (32 bit, 512 internal registers)*,
- *256 KB SRAM (external)*,
- *32 MB SDRAM (external) CAS 2 latency*,
- *1 MB FLASH memory*,
- *UART communication*,
- *timer*,
- *VGA display*,
- *video comm.*

The 32 bit version of the Nios CPU was chosen. By utilizing custom instructions, four 8 bit pixel data can be processed at once in one software step execution. For a large number of iterations performing simple mathematical operations on a single pixel data, this method can reduce the number of software execution cycles up to four times. Single 8 bit data can easily be extracted from the four-byte 32 bit word if necessary.

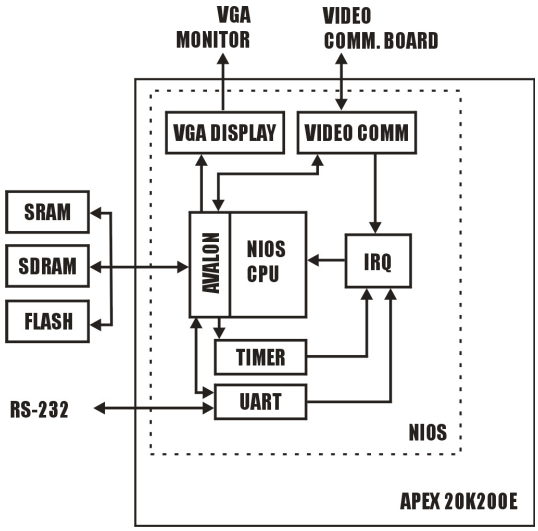

Figure 2. Nios system

3.1 Video Comm

The *video comm* component serves for video data transfer from/to the shared bus of proceeding/succeeding modules through the video communication board. The data and sync signals are fed into the Nios system byte by byte at the communication frequency of 20 MHz and the data rate of 5 MHz. As illustrated in Figure 3, the *data handling* part decodes the grant and control signals, and extracts the pixel and sync data from the bus. Afterwards, it groups four successive pixel data into 32 bit data words and stores them into a 64×32 bit *line memory* word after word. The line memory consists of two memory banks, each holding contents of a whole received horizontal line of pixel data. While the data is being forwarded to the CPU from one data bank, the other bank is being fed with new pixel data. After the completion of a line transfer, the banks switch roles. Bank switching and data exchange synchronization are performed with the use of a CPU interrupt request.

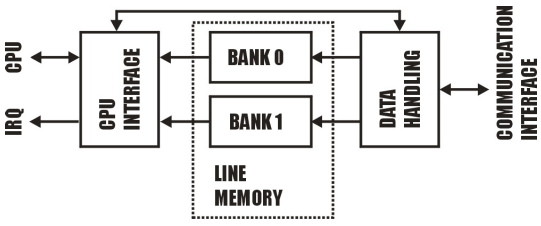

Figure 3. *Video comm* component

3.2 VGA Display

Standard 8 bit grayscale VGA display at the resolution of 640×480 is implemented [5]. Because of the system requirements, only 1 bit (black and white) 256×256 output picture is displayed for monitoring purposes. As illustrated in Figure 4, the component consists of a simple state machine for sync signals generation and a dual port RAM video memory.

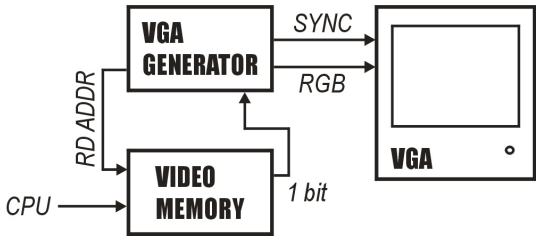

Figure 4. *VGA display* component

4 The Basic SW Flow

The basic principle to access peripherals and custom HW of the platform is through memory mapped registers. The data memory map is divided into four basic regions:

- SRAM memory (256 KB) – instruction and data memory for the fastest SW execution,
- SDRAM memory (32 MB) – picture content memory for a large number of pixel data,
- *video comm* component – line memory and control register,
- *VGA display* component – 1 bit VGA memory bank and control register.

The main idea of the SW flow can be summarized in four elementary stages:

1. receive the current image,
2. when received, disable further reception and perform algorithm processing,
3. display processed image,
4. enable further reception.

The basic SW flow is presented in Figure 5. The algorithm processing in the main loop is executed only when one complete image is received (256 new lines). The reception is done line after line by the interrupt service routine. After the interrupt signal from the *video comm* component is detected, a line of pixel data is received from the line memory bank and stored into SDRAM memory. The reception of the next successive picture is disabled (STOP) and enabled only after the completion of algorithm processing at the end of the main loop (START). This allows only the processed pictures to be received. The time frame between pictures determines the performance of the application. It depends on the speed of the algorithm implementation. The shorter is the execution time of the algorithm, the greater is the frame rate of the application.

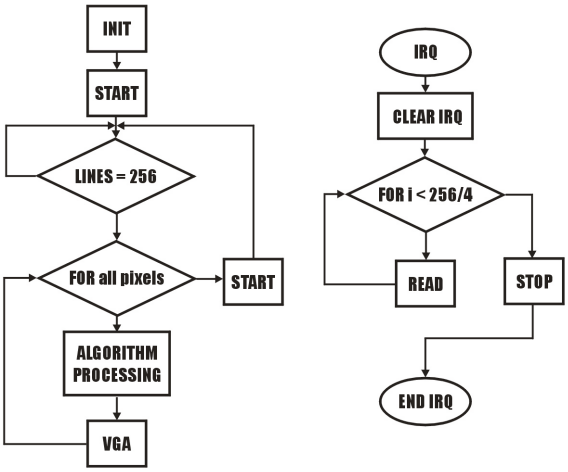

Figure 5. SW flow diagram: the main loop (left) and the interrupt routine (right)

5 The HW/SW Co-design Process

The HW/SW co-design process for the implementation into the platform can be summarized in four main steps:

1. selection of the algorithm,
2. algorithm implementation in SW,
3. detecting critical SW parts,
4. HW/SW optimization of the algorithm.

The first step in designing a video processing application is selection of the appropriate algorithm. It depends on the required performance and the functionality of the final application.

The next step is implementing algorithm in SW. The ANSI C programming language and the assembler are supported. Generally, the preferable choice is the implementation of the SW code in ANSI C. In this way, instead of rewriting the code from scratch, the use of an already existing or third party code for the algorithm shortens the design cycle. The portability of ANSI C allows also the code to be created and tested for functionality on other platforms (i.e. a PC).

After the SW code has been tested for functionality and implemented into the target platform, the performance analysis has to be applied. In order to meet the required constraints, critical SW parts have to be detected and optimized. The use of a SW profiler can be appropriate on a higher level for performance estimation. On a lower level, a CPU timer can be used for the cycle-accurate time-frame estimation of a focused part of the SW code execution.

The final step is the SW code refinement and optimization of critical SW parts in HW. The general idea is to implement parallel structures in HW for the fastest data processing. These are algorithm dependent and, regarding the interface between HW and SW, can be incorporated into the algorithm as:

1. separate HW component (memory mapped register access),
2. custom instruction.

The custom instruction solution is integrated directly into the CPU as an additional ALU. Comparing to the memory mapped register solution, it omits the read/write register operations. Therefore, it is more suitable for repeating single data-word operations.

In the HW/SW co-design process, the designer iterates through the last two design steps until the desired performance is achieved.

6 HW/SW Co-design Example: Motion Detection

In order to demonstrate the HW/SW co-design process, we decided to implement two different algorithms for motion detection. At first, the algorithms were coded in ANSI C programming language on a PC platform. The tested SW code was then rebuilt and transferred into the Nios system. The performance analysis with a CPU timer was used. Afterwards, the SW critical parts were implemented in HW with VHDL.

6.1 The Differential Algorithm

The simplest method to obtain data of the moving objects is based on evaluating the image difference. Absolute difference (O_t) between equivalent pixel data (I_t) of two successive images is calculated and compared to the preset fixed threshold (θ). It is then labeled (l) as moving or static object accordingly (1).

$$O_t = |I_t - I_{t-1}| \Rightarrow \begin{cases} l = \text{moving}, & \text{if } O_t > \theta \\ l = \text{static}, & \text{if } O_t \leq \theta \end{cases} \quad (1)$$

The absolute difference (O_t) calculation is executed inside the interrupt routine. Thresholding is executed in the main algorithm processing loop. All the data processing is executed on four pixel data in parallel, with the use of custom instructions.

6.2 The Markov Random Field Algorithm

Including the above observation difference calculation, the MRF algorithm is based on the minimization of the special energy model, which is calculated from pixel data of three successive images [6, 7, 8]:

$$U(o, l) = U_p(l) + U_o(l) \quad (2)$$

The energy function $U(o, l)$ is the sum of two terms:

1. a priori model energy *Up(l)*:

$$U_p(l) = \sum_{c \in C} V_c(l_s, l_r) \tag{3}$$

In the equation (3), *c = (s, r)* denotes any binary clique in the spatiotemporal label neighborhood shown in Figure 6. C is a set of all cliques within the central image. $V_c(l_s, l_r)$ is a stepwise potential function defined as:

$$V_c(l_s, l_r) = \begin{cases} -\beta, if\ l_s = l_r \\ +\beta, if\ l_s \neq l_r \end{cases} \tag{4}$$

Three different constant potentials are used: β_s (spatial), β_p (past) and β_f (future).

2. observation energy $U_o(l)$:

$$U_o(l) = \frac{1}{2\sigma^2} \sum_{s \in S} [o_s - \psi(l_s)]^2 \tag{5}$$

$$\psi(l_s) = \begin{cases} 0, if\ l_s = 0 \\ \alpha > 0, if\ l_s = 1 \end{cases} \tag{6}$$

It is defined as the centered Gaussian noise with variance σ^2. Constant α presents the average value of nonzero observations.

The minimization process of the energy function is performed using the iterated conditional modes algorithm (ICM). The pixel label (moving or static), producing minimal energy regarding the spatial and temporal neighboring pixels, is chosen. The process is repeated until the change in energy is negligible.

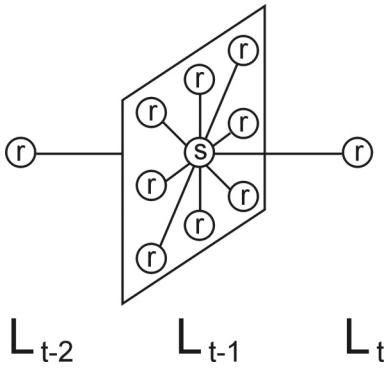

Figure 6. Binary cliques

The MRF algorithm, illustrated in Figure 7, is processing intensive and requires a large number of processing loops and iterations per picture. Besides elementary custom instructions (O_t difference calculation,

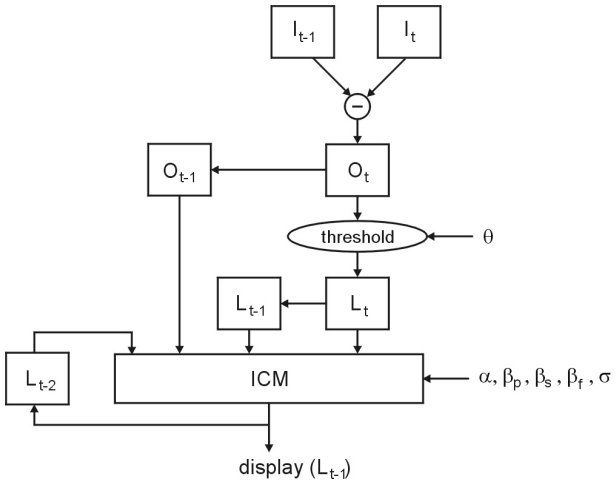

Figure 7. MRF algorithm

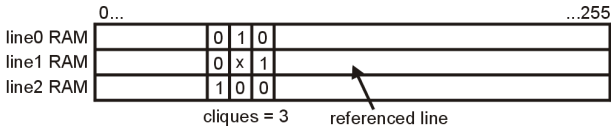

Figure 8. HW clique calculation model

quantization), another critical part of the algorithm can be assigned to custom HW: spatial cliques sum calculation. Since this is 3×3 window type calculation running through the whole image, it requires a large number of loop iterations and addition operations in SW (number of read/write operations at least 9 times the number of pixels). Our HW solution is simple and largely reduces the number of CPU operations. As illustrated in Figure 8, it consists of three RAM blocks (internal FPGA block RAM), each assigned to one horizontal VGA line of pixel label data (256×1 bit). Additional logic calculates the sum of nonzero data in the surrounding of eight pixels, according to central pixels in the referenced line. The line RAM blocks and the control register are memory mapped to the CPU. Therefore, every clique calculation requires 1 line write and 1 line read cycle per referenced line (64 write/read cycles per line).

In terms of speed and number of operations, the most critical remaining part is the last stage in the process of the ICM method, i.e. the calculation of the observation energy U_O (5). The calculation requires a large number of multiplications. The multiplication of two 8 bit words results in a 16 bit word. Therefore, the dedicated custom HW solution of 8 bit value multiplications in groups of four (32 bit word) would introduce new problems with the data word length and would demand complex HW solutions for multiplication implementation. Thus, the four word multiplication cannot be implemented in a custom instruction. As custom HW, memory mapped register type of access is preferable. To simplify and speed up the design time, the Nios builtin HW multiplication ac-

celeration was chosen (3 clock cycles per 16×16 bit multiplication), instead of implementing custom HW. Each 8 bit pixel data is extracted from an original four byte data word. Afterwards, accelerated multiplication is applied. This solution increases the number of processing cycles (4 additional 8 bit data extraction execution). Since the dedicated HW solution would require the same number of register reading cycles (memory mapped register access), the difference in the processing time is compensated. Custom HW solution would also require pipelining, which increases operation execution time. Therefore, the HW design process for custom HW multiplication would not gain any performance improvement.

In order to avoid the processing consumable division of inverse variance in the first part of equation (5), the variance was experimentally predetermined in the initialization stage of the algorithm. The inverse variance is calculated and the result is used as a constant in the main data processing part of the algorithm.

The MRF motion detection algorithm depends on four experimentally determined parameters β_s, β_p, β_f and α (β_p = 10; β_s = 20; β_f = 30; α = 10). More weight is given to the future (β_f) by taking, $\beta_p < \beta_s < \beta_f$. This simplifies dealing with motion discontinuities by taking into account any innovation in motion in a faster way, and better eliminating background areas which are uncovered during motion [6]. For a specific application, these parameters must be adjusted to optimize the quality of the results.

Owing to the large number of software loops in picture data processing, compiler supported additional loop unrolling provides reduction of compiled instruction cycles and therefore speed improvement of approximately up to 20 %.

7 Results

As illustrated in Chart 1 in Figure 10, the maximum rate of 16.7 frames/s was achieved at the input rate of 25 frames/s with the differential motion detection algorithm. With the MRF algorithm, the maximum rate of 2 frames/s was achieved (Chart 3 in Figure 10).

The comparison of frame rates and algorithm execution times, achieved with SW and optimized HW/SW implementation, is illustrated in Charts 1–4 in Figure 10 for both algorithms. Since the algorithm execution times determine the frame rates of the application, the ratios of SW and HW/SW solutions are similar in both charts for each algorithm. In Charts 5 and 6 in Figure 10, the HW resource usages of the basic Nios system configuration for the SW solution and the configuration including custom HW for the optimized HW/SW solution of the MRF algorithm implementations, are illustrated. As shown in the charts, the performance improvement is approximately proportional to HW resources (logic elements).

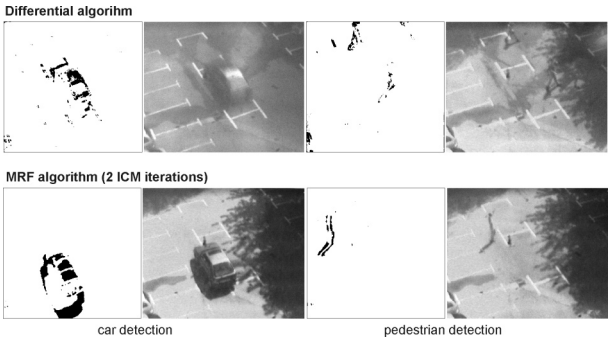

Figure 9. Visual presentation of actual motion detection

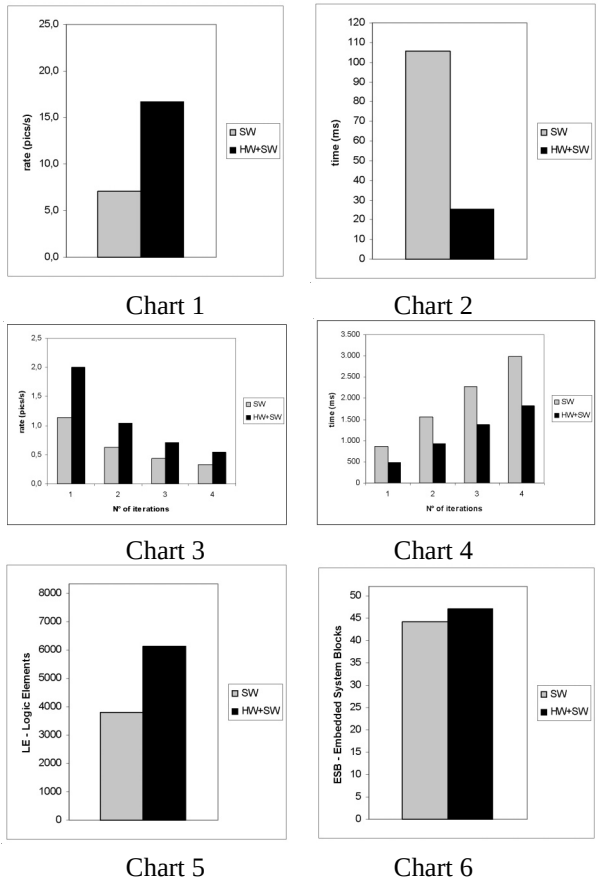

Figure 10. Charts: 1 – Frame rates (differential algorithm), 2 – Algorithm execution time (differential algorithm), 3 – Frame rates (MRF algorithm), 4 – Algorithm execution time (MRF algorithm), 5 – FPGA resource usage (LEs), 6 – FPGA resource usage (ESBs)

8 Conclusion

Based on configurable technology (CPLD, FPGA, softcore CPU), a powerful prototype platform for image processing was designed in order to support HW/SW codesign and partitioning. The high configurability and parameterization, supported with integrated design tools, allow flexible HW/SW integration into the final application. A large spectrum of potential real-time imaging and video

applications (e.g. motion tracking, segmentation) can be integrated. The platform is also highly suitable for educational purposes.

For demonstration, two motion detection algorithms were implemented. According to the achieved results, the differential algorithm is suitable for simple practical motion detection applications. The MRF algorithm realization successfully competes with other experimental systems [7] which are mostly DSP or multi CPU based or operate at much higher frequencies.

9 References

[1] I. Bolsens et al., Hardware/Software Co-Design of Digital Telecommunication Systems, *Proceedings of the IEEE*, Vol. 85, No. 3, pp. 391–418, March 1997.

[2] A. Sangiovanni-Vincentelli, M. Sgroi, L. Lavagno, Formal Models for Communication-based Design, *Proceedings of CONCUR'00*, pp. 29–47, August, 2000.

[3] F. Slomka et al., Hardware/Software Codesign and Rapid-Prototyping ofEmbedded Systems, *IEEE Design & Test of Computers*, Special issue: Design Tools for Embedded Systems, Vol. 17, No. 2, pp. 28–38, April-June 2000.

[4] M. Sgroi et al., Formal Models for Embedded System Design, *IEEE Design & Test of Computers*, Special issue: Design Tools for Embedded Systems, Vol. 17, No. 2, 14–27, June 2000.

[5] J. Konrad, *Handbook of Image and Video Processing*, Academic Press, 2000.

[6] F. Luthon et al., Real-Time DSP Implementation for MRF-Based Video Motion Detection, *IEEE Transactions on Image Processing*, Vol. 8, No. 10, pp. 1341–1347, October 1999.

[7] L. Lacassagne, Motion Detection, Labeling, Data Association and Tracking in Real-Time on RISC Computer, *Proceedings of ICIAP*, pp. 520–524, September 1999.

[8] A. Oukil, A. Serir, Markovian Random Fields Energy Minimization Algorithms, *Proceedings of the International Conference on Pattern Recognition*, Vol. 3, pp. 3522–3525, September 2000.

[9] *Nios documentation*, http://www.altera.com/literature/lit-nio.html

Matjaž Finc received his B.Sc. degree in electrical engineering from the University of Ljubljana in 2001. His current research interests include hardware-software co-design, embedded soft-core processors and real-time digital image processing.

Andrej Trost received his B.Sc., M.Sc. and Ph.D. degrees in electrical engineering from the University of Ljubljana in 1995, 1998 and 2000, respectively. His current research interests include study, design and implementation of digital systems for real-time digital image processing.

Baldomir Zajc received his B.Sc., M.Sc. and Ph.D. degrees in electrical engineering from the University of Ljubljana in 1960, 1965 and 1975, respectively. He is currently the head of the Laboratory for Integrated Circuit Design and Professor at the Faculty of Electrical Engineering.

Andrej Žemva received his B.Sc., M.Sc. and Ph.D. degrees in electrical engineering from the University of Ljubljana in 1989, 1993 and 1996, respectively. He is Associate Professor at the Faculty of Electrical Engineering. His current research interests include logic synthesis and optimization, test generation and hardware-software co-design.