Towards a Dynamically Reconfigurable System-on-Chip Platform for Video Signal Processing

Walter Stechele, Stephan Herrmann, Andreas Herkersdorf

Technische Universität München
80290 München
Germany
Walter.Stechele@ei.tum.de

Abstract: This paper reports ongoing work towards a dynamically reconfigurable System-on-Chip (SoC) platform for video signal processing. It consists of dedicated, statically and dynamically reconfigurable components, as well as an embedded RISC core and memory. Application-specific software libraries support control of dynamic reconfiguration of low level operations by high level instructions. Thus programmability is combined with high data throughput and low power consumption of hardwired circuits. Preliminary work presented here is focused on one selected application, video object segmentation. The architecture of a coprocessor for video object segmentation is presented, which exploits the basic concept of the dynamically reconfigurable SoC platform. A library of software functions for image processing was developed, too, which will be used as a starting point for the application-specific software parts of the platform.

Keywords: Dynamically reconfigurable architectures, video object segmentation

1. Introduction

Future systems for information processing will require increasing flexibility, combined with high throughput and low power consumption. Especially visual information processing will impose high requirements in this field. A substantial part of all kind of information, which a human being consumes, consists of visual information. Therefore visual information processing will play an increasingly important part in future multimedia system.

Current microprocessors are offering all the flexibility through programming, but data throughput is limited by the high number of cycles used for loading and decoding of instructions, as well as loading of data from memory and writing back results. These instructions are also contributing to power dissipation. ASICs *(Application Specific Integrated Circuits)* offer some advantages here, as all operations are implemented directly in hardware. This results in high throughput and low power dissipation, but does not provide the flexibility needed. FPGAs *(Field Programmable Gate Arrays)* can download any given ASIC or processor architecture into a configurable device. They normally contain configurable processing resources (e.g. look-up tables, logic, mux, Flip-Flops, arithmetic units) and configurable wire channels to allow any connection between processing elements. In addition they may contain distributed memory and bus systems. FPGAs are well established as ASIC-substitute for small volume and for fast

prototyping and verification during ASIC design. There is a trend towards partial, dynamical reconfiguration capabilities within FPGAs, known as „Multi context FPGA“.

FPGAs do not compete with ASICs. The overhead for programmable connections in FPGAs results in longer wire lengths, lower speed, and higher power consumption when compared to ASICs fabricated in the same technology.

Reconfigurable architectures with embedded processor cores, embedded FPGAs, and embedded memory are targeting to combine the programmability of microprocessors with the higher data throughput and lower power dissipation of FPGAs. Current FPGA devices, e.g. XILINX VIRTEX-II Pro [1] or ALTERA STRATIX [2] are offering multiple RISC cores, DSPs, high-speed I/O, and a number of special modules on-chip.

There exist a large number of scientific publications on reconfigurable architectures, as well as on software design, targeting to hide all details of hardware architecture from the software developers. An overview can be found in [3], [4], [5], [6], [7]. The hardware configuration can be static, i.e. on compile-time, or dynamic, i.e. on run-time.

Some commercial products are offering promising hardware architectures and highly specialized software development tools. The most well-known processor architectures with configurable instruction set include Tensilica/Xtensa [8] and ARC [9]. These architectures offer to implement application-specific instructions and to improve performance, but they do not reach throughput and power of dedicated ASICs and they do not offer dynamic reconfigurability. There is still some major potential to exploit with reconfigurable hardware.

In the above mentioned overview publications [3] - [7], many application areas for reconfigurable architectures are mentioned, e.g. DSP, networking, multimedia, speech, video, cryptography, and rapid prototyping. The focus of this paper is on visual information processing, where a key technology is video object segmentation. An algorithm for video object segmentation should find boundaries of connected areas, which have been generated through projection of a 3-dimensional scene into a sequence of 2-dimensional images. In our 3-dimensional world, objects consist of spatially connected elements. All information on these spatial connections is lost during projection into two dimensions, as objects from various spatial depths are projected into the same plane and might occlude each other. An algorithm for video object segmentation, which does not work on stereo or multi-camera images, can use only information, which is present in the 2-dimensional scene, as there are luminance, color, texture, shape, and motion. An overview on various algorithms and first commercial products can be found on the web pages of the SCHEMA project [10].

Algorithms for video object segmentation are using many different operations. They can be grouped into high level control operations and low level pixel manipulation. Control requires a high degree of flexibility and thus is good for processor implementation. Pixel manipulation on the other hand requires to apply the same operation on many many pixels, but also to change these operations during the segmentation process. So pixel manipulation seems good for implementation in a reconfigurable unit.

Instruction Count	100 %
Pixel Addressing	56.7 %
Pixel Processing	10.2 %
Data I/O	32.8 %
High Level	0.3 %

Tab. 1: Instruction count for segmentation
of 5 frames of test sequence *Akiyo* (4:4:4 / QCIF) [12]

Table 1 shows instructional load for a video object segmentation algorithm. Generally, the computational load of the high level algorithm is very low, less than 0.3% of the overall load. Comparing pixel addressing and pixel processing, it shows that the instruction count for pixel addressing is significantly higher than for pixel processing.

The next part of the paper describes the basic ideas of the reconfigurable platform for video signal processing. Then a architecture for video object segmentation, called AddressEngine, is described, which will be used to exploit the concept of static and dynamic reconfiguration.

2. Dynamically Reconfigurable Platform

Our target platform (figure 1) consists of a reconfigurable hardware and a software library, where low level pixel operations are executed in a processing unit with high throughput and low power dissipation. This unit is dynamically reconfigurable to adapt on the process of segmentation. The software library offers all the functions and configurations for program development. High level control operations are implemented on the embedded processor.

This platform will be used for studying reconfigurable architectures with dynamic partial reconfiguration capabilities. This will be exploited first for video object segmentation, later on for other applications as well.

For video object segmentation the software developer can have all flexibility to write his software for high level control operations in C/C++, but for low level pixel operations pre-defined function calls shall be used. The hardware architecture allows configuring and mapping the low level operations, whereas high level operations are mapped on the embedded processor. Examples for planned functionalities are preprocessing before segmentation (e.g. filtering, format conversion), object shape processing (e.g. gradient, watershed), object tracking, and detection of new/hidden segments (e.g. for motion compensation), as well as video analysis (e.g. segment-based operations).

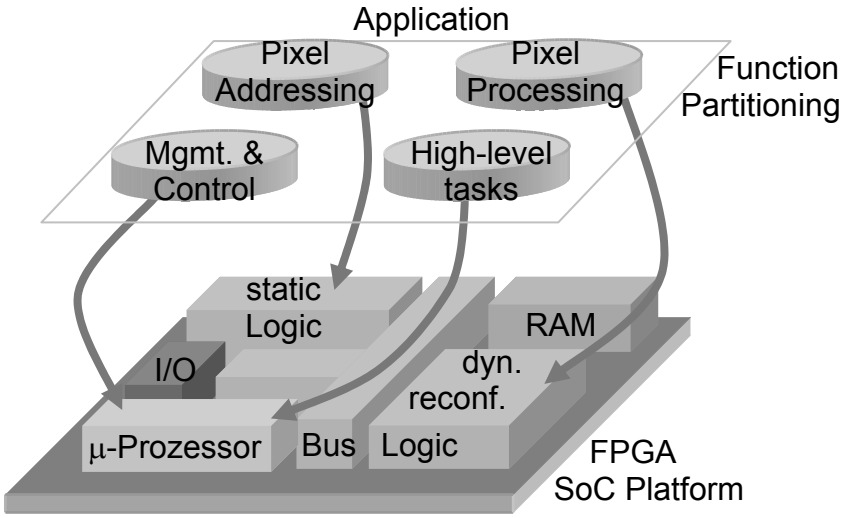

Fig. 1: Reconfigurable platform

On the hardware side of the platform, dedicated implementation will be used for those components, which are common to all applications, i.e. embedded RISC, memory, data buses, controller for data and configuration. A hardwired implementation would be best for those parts, but for a prototype a statically configurable FPGA block could be used as well, even if the performance will be lower.

Static reconfigurability will be used for application-specific components of the platform. Thus the platform can be reconfigured for the target application. For video object segmentation, pixel addressing will be implemented in a statically reconfigurable block. Dynamic reconfigurability will be used for changing operations within an application, e.g. for pixel processing in video object segmentation.

The software part of the platform will contain all functions for its operation, e.g. loading and management of configuration data, monitoring and test, RTOS if applicable. Special focus will be on control of dynamically reconfigurable blocks, i.e. buffering and run-time scheduling of configuration data downloads.

The application layer consists of hardware and software as well. For video object segmentation, low level pixel operations may be implemented in hardware, whereas high level control may be implemented in software.

For a prototype demonstrator, the platform shall be implemented completely in a single FPGA. This allows investigation of buffering, allocation, and run-time scheduling of dynamic reconfiguration data within the FPGA. Partial static reconfiguration may be used to adapt the platform to different applications. For future applications, two alternatives might be feasible: (1) The FPGA prototype could be used directly with appropriate static configuration, or (2) an application-specific instance of the generic platform could be implemented, with statically configured parts implemented in standard cell technology. Thus, higher data throughput, lower power dissipation, and lower cost for high volume production could be achieved. The drawbacks are higher development costs and the device could be used just for one application.

3. AddressEngine

Based on the generic ideas of the reconfigurable platform, an architecture called AddressEngine was developed, which exploits statically reconfigurable implementation of pixel addressing schemes, dynamically reconfigurable implementation of pixel processing, and a software library to support the addressing schemes. Addressing schemes, pixel processing, and architecture of AddressEngine are described in the following paragraphs.

3.1 Addressing Scheme

Although there are various different segmentation algorithms, many of them consist of operations, which use only four ways to access the pixel data: Inter addressing, intra addressing, segment addressing, and segment indexed addressing (figure 2). These four types of addressing were implemented in the AddressLib [11].

Inter addressing is the addressing mode where a result for each pixel position is calculated using data from two different frames. Its application may be computation of difference pictures or SAD (Sum of Absolute Differences).

Intra addressing is used in situations where a result is calculated for each pixel as a function of the pixels original value and the values of its neighbors within the same image. This is typically used for FIR filter like operations, as gradient operators and morphological operators. The tool can be applied either in a recursive or in a non-recursive way. Recursive means that the processing results of already processed pixels are used as input values; while non-recursive means that always the original values are used.

Segment addressing is used if arbitrarily shaped segments have to be processed. In this kind of processing, a segment is determined by local neighborhood criteria. First, the pixel processing is done in the same way as for intra addressing. Second, all neighbor pixels which have not been processed before, are tested if they fulfill specified neighborhood criteria. If they do, they might be processed in one of the following steps of the algorithm. By operating this way, an expansion process takes place: Beginning

with a set of start pixels, all pixels of the segment are processed in order of geodesic distance. The control of the expansion process is done using a FIFO, or with a hierarchical FIFO to allow for adaptation of the expansion speed. The start pixels have to be previously determined, e.g. by regular scanning and testing of some start criteria. Segment addressing is extensively used by the watershed algorithm.

Segment indexed addressing is an addressing method, which is used in parallel to one of the above addressing methods, when data associated to a segment is needed or generated during the pixel processing, e.g. segment identification numbers. This is done accessing an indexed table.

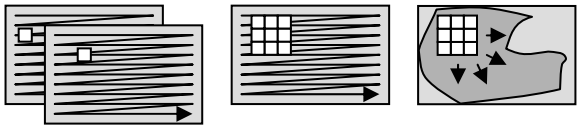

Fig. 2: Pixel addressing schemes: Inter (left), intra (middle), and segment (right) addressing. Arrows indicate direction of pixel processing.

The first two schemes are well known from frame based or block based video processing. The third addressing scheme is used for pixel addressing of arbitrarily shaped segments in a rectangular frame. The fourth scheme represents indexed table accesses and differs from the other schemes by not addressing pixel data.

3.2 Pixel Processing

Pixel-level operations, which are going to be implemented in the dynamically reconfigurable pixel processing unit, may be separated into basic sub-functions, such as add, sub, mult, grad, in order to achieve efficiency and flexibility. These sub-functions can be combined to form more complex operations, e.g. luminance/chrominance difference between neighboring pixels for homogeneity check, or morphological gradient operations, as illustrated in figure 3.

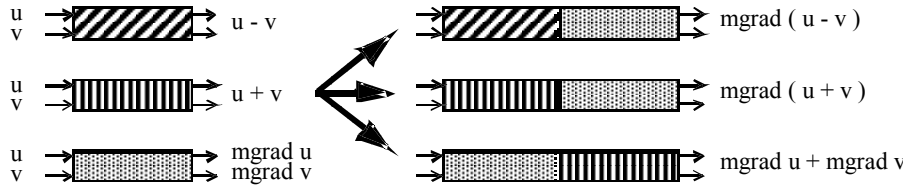

Fig. 3: Pixel processing: Combination of add/sub with morphological gradient

3.3 Hardware Architecture

A coprocessor architecture for accelerating image segmentation algorithms was developed [12]. As the previously described set of addressing schemes is common to most of the segmentation algorithms, these addressing schemes are implemented in a statically reconfigurable part of the coprocessor. The operations on pixel processing level need to be flexible, thus a configurable block was used for the processing part. A configurable processing unit offers some flexibility, but it does not need to load any instructions while processing certain operations. This behavior fits exactly the requirements of image segmentation: One arbitrary operation has to be applied to hundreds of thousands of pixels in a frame.

With respect to general aspects of performance optimization on processing level, this architecture implements pipelined data load-process-store operations. On pixel level, just one processing unit was implemented. There is a potential for further parallelization of pixel processing, using parallel processing units, sharing the same FIFO, but this would require parallel external memories or on-chip memory, in order to avoid a bottleneck in loading of pixel data.

Figure 4 shows the block level structure of the coprocessor. The statically reconfigurable address unit includes the memory interface, the scan controller, the pixel level controller, and the address generator. Beside this unit, the FIFO controller and FIFO memory are used by segment addressing. The input data is loaded from the external memory to the register matrix, which can store a 3x3 neighborhood of a pixel. The input data is then processed by the processing unit and the result is stored in the result register. The stored result is then moved to the external memory.

3.4 Preliminary Synthesis Results

The AddressEngine was modeled in VHDL and synthesized in 0.25 µm standard cell technology for area and timing estimation, as a FPGA platform was not available for these preliminary estimations. Table 2 shows area results for this preliminary synthesis. The performance for color segmentation was evaluated as well. Table 3 shows results of processing speed.

With this implementation of AddressEngine, around 49 frames/sec can be achieved for the complete color segmentation algorithm. This is more than 3x real time operation, leaving enough of time budget for implementation in FPGA.

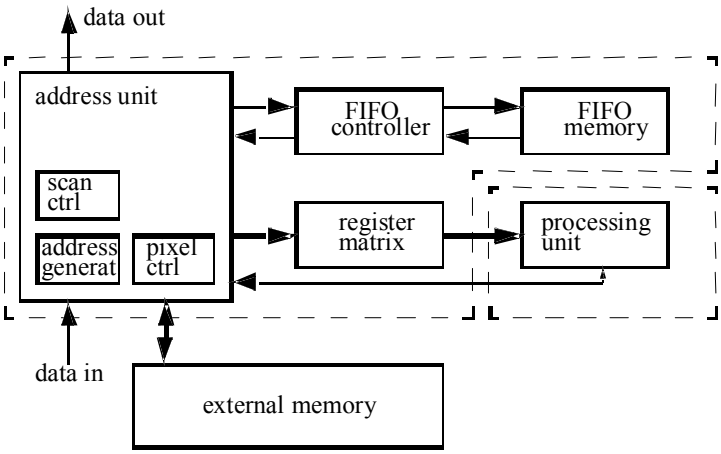

Fig. 4: Block diagram of segmentation coprocessor

Unit	[mm^2]
Processing Unit (a)	1.8
Address Unit (a)	2.5
Add'l Logic (b)	0.6
Interconnects (b)	5.1
Cache Memory (c) (122 kByte)	18.1
Coordinates Memory (164 kByte) (c)	19.6
Total	47.7

Table 2: Area estimation for AddressEngine:
(a) Synthesis Result, (b) Estimation, (c) Generated by RAM Compiler

Algorithm	
Watershed	8.56 ms (a)
Gradient	2.03 ms (a)
Erosion/Dilation	1.52 ms (a)
Relaxation	3.60 ms (a)
Iterative Color Segmentation	49.2 frames/s (b)

Table 3: Processing speed of color segmentation algorithm for AddressEngine. (a) CIF, (b) QCIF

3.5 Performance Estimation

Preliminary simulations of the segmentation algorithm presented in [13], with QCIF images (176 x 144 pixels), have shown a computational load of 2.5 Million pixel operations per second and on-chip memory requirements of 70 kBytes. The operational load consists mainly of 5 parts: Labeling, label check, fragmentation check, watershed, and relaxation. For real-time operation with 15 frames/sec there is a time budget of 66 msec per frame. An estimation for Pentium 4 with MMX and 2 GHz clock resulted in 10 frames/sec - that is below real-time, even if the processor would execute just this one single task. On the reconfigurable platform, with an estimated execution time of 20 nsec per pixel and 50 MHz system clock, which seems feasible in today's FPGAs, all operations could be executed within 50 msec, leaving 16 msec for reconfigurations of the pixel processing unit. To perform dynamic reconfiguration, the high level algorithm has to control the flow of configuration data within the architecture, using prefetch instructions and buffering concepts. The trade-off between area, execution time for pixel processing, and reconfiguration overhead still has to be investigated.

4. Outlook

The next step will be to implement the AddressEngine architecture in a FPGA. An algorithm for video object segmentation, which was developed using the AddressLib, will be taken as a test case for optimization of dataflow and reconfiguration process. Then support of a wider range of algorithms for video signal processing with more complex dynamically reconfigurable blocks will be investigated. Further goals include automatic recognition of available hardware configurations and substitution of software functions by corresponding hardware operations. In future work, the adaptation of the platform to other application areas, e.g. network protocol processing, will be evaluated.

Acknowledgment

The authors want to thank Hubert Mooshofer from Siemens for his contributions to this work.

This material is based upon work supported by the IST program of the EU in the project IST-2000-32795 SCHEMA (http://www.iti.gr/schema)

References

[1] http://www.xilinx.com/xlnx/xil_prodcat_landingpage.jsp?title=Virtex-II+Pro+FPGAs

[2] http://www.altera.com/products/devices/stratix/stx-index.jsp

[3] J. Becker: „Configurable Systems-on-Chip: Commercial and Academic Approaches“, International Conference on Electronic Circuits and Systems, ICECS 2002, Dubrovnik, September 2002

[4] R. Hartenstein: „Trends in Reconfigurable Logic and Reconfigurable Computing“, International Conference on Electronic Circuits and Systems, ICECS 2002, Dubrovnik, September 2002

[5] K. Compton, S. Hauck: „An Introduction to Reconfigurable Computing“, IEEE Computer, April 2000

[6] K. Compton, S. Hauck: „Reconfigurable Computing: A Survey of Systems and Software“, ACM Computing Surveys, Vol. 34, No. 2, pp. 171-210, June 2002

[7] A. DeHon: „Reconfigurable Architectures for General-Purpose Computing“, Massachusetts Institute of Technology Artificial Intelligence Laboratory Technical Report No. 1586, October 1996

[8] www.tensilica.com

[9] www.arc.com

[10] http://www.iti.gr/schema

[11] http://www.lis.ei.tum.de/research/bv/topics/segm/addrlib.html

[12] H. Mooshofer: Entwurfsmethodik für eine flexible Architektur zur Videoobjekt-Segmentierung. Dissertation an der Technischen Universität München, 2002 (in german)

[13] S. Herrmann, H. Mooshofer, H. Dietrich, W. Stechele: "A Video Segmentation Algorithm for Hierarchical Object Representations and Its Implementation", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 9, No. 8, Dec. 1999, pp. 1204-1215