Real-time Image Processing for Camera-based Driver Assistance Applications

Stefan Wonneberger, Thorsten Graf
Volkswagen AG / Driver Assistance and Integrated Safety
{stefan.wonneberger|thorsten.graf}@volkswagen.de
Letter Box 1777, 38436 Wolfsburg, Germany

Henning Sahlbach, Sean Whitty, Oliver Bende, Rolf Ernst
TU Braunschweig
Institute of Computer and Network Engineering
{sahlbach|whitty|ernst}@ida.ing.tu-bs.de

Abstract

Future driver assistance systems will rely on a variety of sensors, including camera-based systems, to achieve an optimal perception of a vehicle's surroundings. With increasing resolutions and the inevitable real-time requirements of these safety-relevant systems, multi-camera setups demand processing performances and data rates that cannot be satisfied by current standard PC components. To meet these requirements, this paper presents a flexible FPGA-based design approach for a central onboard image preprocessing unit, which offloads computation intensive tasks to a dedicated FPGA accelerator platform with support for real-time execution.

1 Introduction

One challenging goal of next generation driver assistance systems will be the complete digital reconstruction of a vehicle's environment in real-time. In addition to commonly used sensors like radar or ultrasonic-based systems, a visual perception of the environment will be required to obtain a maximum amount of information. The corresponding sensor systems are based on different camera configurations. Usually, front view cameras are used for applications such as lane departure warning or generic object detection. In order to obtain a complete 360 degree view of a vehicle's surroundings, additional systems composed of four cameras are used. The increasing number of cameras means a significant increase in the amount of data to be processed. For example, even for small resolutions of 512×512 pixels with 8-bit color depth, a stereo system combined with a surround view setup produces a data rate of 377 Mbit/s for an image frequency of 30 frames per second. To handle such data rates, high performance electronic control units (ECU) are required.

In contrast to the required processing performance, the power budget in cars is very limited. Therefore, these ECUs must be designed for energy efficiency. This also supports the goal of

reducing the overall carbon dioxide output, which is increasing in importance in the automotive development process.

Image processing algorithms for driver assistance systems can be divided in different classes starting with the image retrieval process and image restoration tasks. Image extraction is the next logical step and is used to reduce the amount of data. This step is then typically followed by a classification. Especially the algorithms for image restoration and image extraction can be chained as a processing graph. Figure 1 shows an example for a pre-processing algorithm graph, which is a typical part of a camera-based driver assistance system. The acquired image stream is processed by each node of the graph and streamed to each successor nodes.

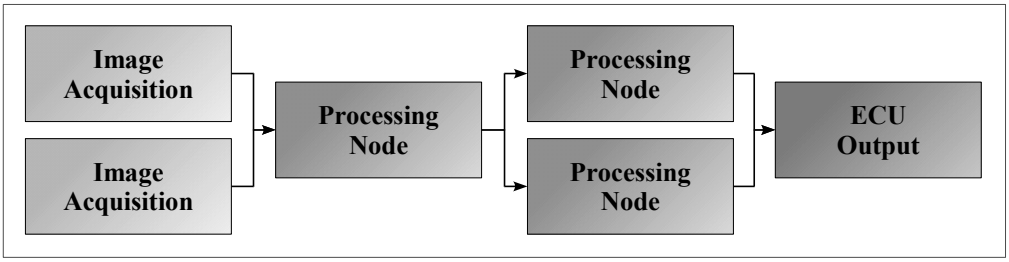

Figure 1: Algorithm graph with several processing nodes

Almost all algorithms in this class demand for high computational performance to process images in real-time. Frequently, operations must be performed on each individual pixel or on repeating parts of the input image. On the other hand, the required operations are typically quite simple, which allows the usage of simple processor architectures. Because of these characteristics these algorithms can be massively sped up if they are realized as a parallel hardware implementation. FPGAs offer a good starting point for an initial hardware development as they provide a ready-to-use hardware platform with standard I/O and hardware programming interfaces. The performance of FPGA implementations is comparable to real ASIC designs.

Therefore, this paper proposes a central FPGA-based image preprocessing platform for image data, which is acquired by all vehicle cameras. The required preprocessing algorithms for driver assistance functions can be bundled and implemented on one central ECU, which extracts and reduces the desired image information and streams it to specialized devices implementing specific assistance functions. The current trend in ECU onboard networks leads to a distinction between ECUs attached to sensors to pre-process the retrieved data, and dependent function-specific ECUs operating on the resulting data. The presented idea of a central image processing unit supports this idea of ”democratizing” the on-board ECU network.

For an assisted development of image processing algorithms the FlexWAFE architecture [1] has been selected. This architecture follows the idea of a stream oriented design. It implements a dataflow model in hardware and defines a standard communication interface for all processing node. This guarantees the reusability of processing nodes and allows a comfortable rearrange-

ment of implemented processing nodes during the design process.

To evaluate the selected architecture an existing pre-processing algorithm from the professional video domain [2] has been adapted and implemented in hardware. A motion estimation algorithm, which calculates the movement of pixels between two images, was chosen as a benchmark for the FlexWAFE architecture in automotive context. In a standard PC environment the algorithm does not achieve real-time performance for the desired resolutions, which requires an accelerated implementation on dedicated hardware. For driver assistance systems this algorithm is highly relevant because it is a basis for 3D reconstruction using mono cameras or the tracking of reconstructed 3D points using a stereo system.

For a seamless post-processing of the computed results in the driver assistance development chain the FPGA is integrated in a host PC which executes additional software algorithms. The ADTF software framework [3] is used to connect the hardware algorithm results to a software post-processing.

The rest of the paper is organized as follows. The next section gives an overview of related research. In section 3, a brief overview of the basic architecture is given, followed by an analysis of its usability for image processing in driver assistance functions. Section 4 describes the integration of the architecture in a concrete driver assistance scenario. The motion estimation algorithm and its implementation in hardware is presented in section 5 followed by benchmark results in section 6. A final conclusion is given in the last section.

2 Related work

In the past, several existing project dealt with the design and implementation of specific image processing platforms. Angermeier et. al. [4] present an approach using hardware/software codesign for vehicle detection, a specific driver assistance problem. The design implements a fixed preprocessing graph for image retrieval and feature extraction consisting of a pattern matching and segmentation unit in hardware combined with a software postprocessing. Claus et. al. [5] present a reconfigurable hardware platform for driver assistance systems. The platform consists of a dual core embedded processor design combined with reconfigurable coprocessors to accelerate certain pixel operations and different classes of algorithms. In contrast to specific, application dependent, architectures, Kyo and Okazaki [6] present a SIMD multi-core processor network for image-based driver assistance systems. Processors, described as processing elements (PE), can be clustered in certain ways to operate on image structures in parallel.

All these architectures, and classical image processing systems, are typically multi-core processors with specialized pipelines and specialized memory architectures i.e. the cell processor [7] or general purpose processors with attached coprocessors, which are required to operate on the input data in real-time. On the other hand, there exist dedicated platforms, which implement a specific driver assistance function in hardware. These approaches usually lack flexibility and

are not considered in detail.

Image processing algorithms often perform identical calculations on different image sections, which provides a very high potential for parallelization. Therefore, these algorithms are typically executed in parallel on multi processor - single instruction multiple data (SIMD) - architectures. Each processor has extended execution pipeline paths and several computation units to allow the execution of different classes of image processing algorithms. In contrast to this, most of the algorithms require only a few, simple operations provided by the processor's basic computation units. Consequently, large parts of the processor remain unused during the execution of one specific algorithm.

Moreover, the processing network is not useable for other algorithms during the execution of one specific algorithm. A complete chain of preprocessing algorithms has to be executed in sequence, which results in a decrease of available execution time for each algorithm. To target these issues, a different approach for platform development is presented.

3 System Architecture

As already mentioned, the system architecture itself is based on the FlexWAFE architecture, originally developed at the Institute of Computer and Network Engineering during the FlexFilm project [2]. The architecture has been successfully verified for different high definition video processing tasks for the digital cinema application domain, which requires high data rates and enormous processing performance. The key features of the architecture are briefly repeated for convenience, a detailed presentation can be found in [1].

The FlexWAFE architecture is based on a synchronous data flow graph model (SDF) [8]. Figure 3 shows an example instantiation with its basic components. All nodes are connected by a light-weighted interface protocol with backpressure capabilities, which provides an easy, flexible interconnect for the processing nodes. The ability to stop certain parts of the data path using backpressure allows a self-timed execution of the nodes and does not necessarily require a global fine grain schedule. Specific inbound and outbound nodes of the graph have to translate this internal protocol to the connected peripherals. The architecture consists of three different component classes, originally introduced in [2].

- Data processing units (DPU) contain hardware implementations of specific image processing algorithms or algorithm parts.
- Local memories with controllers (LMC) are used to buffer and reorder data or to provide an application interface to and from off-chip memory. This interface is necessary for certain operations such as a random read/write access to a complete image area, which cannot be realized using on-chip memories.

- The LMC and certain DPUs can be partly configured and controlled using a weakly-programmable central algorithm controller (AC).

The architecture is complemented by a high performance memory controller, which accesses the off-chip memory [9].

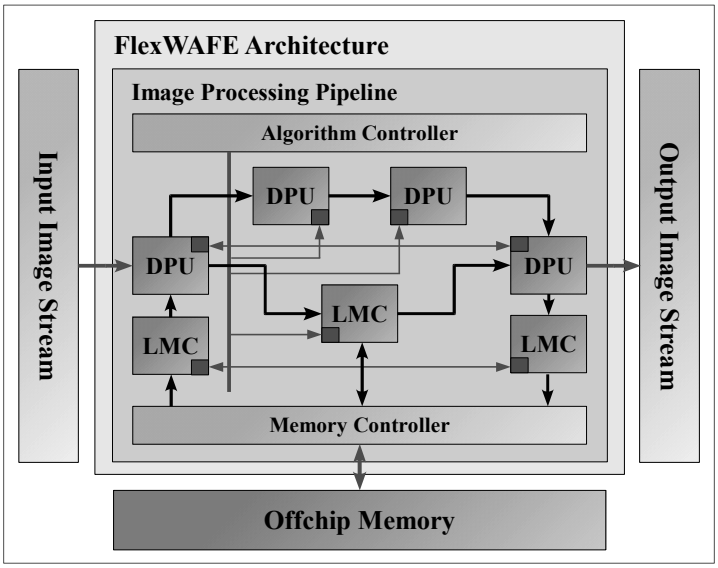

Figure 2: Example configuration of FlexWAFE architecture components

Using the FlexWAFE approach, each algorithm is implemented on a network of application specific processing elements (PE) and local memories. Depending on the selected algorithm often only a few simple mathematical operations are needed, which results in a very easy PE unit that consists of a small amount of electrical gates. To achieve a high performance execution of a specific algorithm such PEs can be instantiated in parallel. The amount of parallel PEs is defined by the algorithm designer and depends on the required performance and available chip resources or targeted hardware costs. The complete net of PEs for one algorithm can be encapsulated as a single DPU node in FlexWAFE. This allows hardware parallelism for one algorithm in an internal DPU and a parallel execution of multiple algorithms using several DPUs in a specific FlexWAFE processing system.

As the FlexWAFE library already consists of various existing image processing components the development time for new hardware accelerated algorithms decreases. Current FPGAs provide a large amount of logical elements, which allows the design of even complex image processing pipelines. Some FPGA devices are already automotive certified so they can be integrated in final ECU designs as well.

4 Implementation

Since the FlexWAFE architecture concept is based on a dataflow graph principle, no limitations to the inbound and outbound connections are necessary, except that the data streams need to be

stoppable by the succeeding nodes. Otherwise sufficient buffer space is required in the in- and outbound nodes to ensure correct communication behavior with connected peripherals. This concept offers the possibility to integrate the architecture in diverse transfer protocols or systems. During the development and evaluation process the architecture is typically embedded in a host system. Only parts of the algorithms are ported to the FlexWAFE engine. Results can be transferred to and analyzed on the host system. For the development of camera-based driver assistance functions the inbound data stream might be changed between online and offline processing. This allows a realization of live tests with real-time response for the developed algorithms. For a detailed algorithm analysis in offline mode, data streams can be transferred directly from a host system storage device. The image data can be artificially generated for simulation tests, captured during online processing or can be provided by external camera devices. Figure 3 shows a typical system configuration of the architecture for online image processing. In the example, two cameras are connected as inbound nodes of the architecture. Image processing results are transferred to the host system via an internal bus protocol (PCI Express, PCIe).

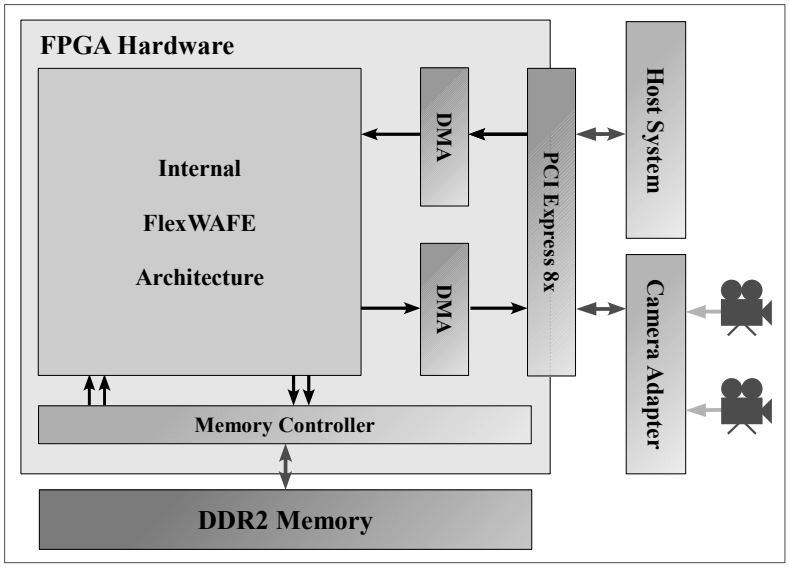

Figure 3: Integrated FlexWAFE architecture with input/output devices

Regarding application development, all single hardware components can be developed and tested as single DPUs. As soon as all required components are available a complete preprocessing system can be designed as a chain of multiple DPUs. This completed system, which is integrated as an ECU will be able to send extracted and reduced image information to subsequent ECUs via typical onboard bus systems or to local post-processing hardware such as an additional onboard processor.

A complete implementation of the FlexWAFE architecture has been realized on a Xilinx FPGA board. During the algorithm development process the hardware platform can be integrated in a common PC environment as described above. The PCIe protocol is used to transfer image data

from high level applications to the image preprocessing platform and retrieve the resulting data. An online camera mode is possible using the direct memory access (DMA) mode between an external frame grabber and the developed processing board.

On the host PC system an application programming interface for the FlexWAFE architecture has been realized. This allows the configuration of the programmable DPUs and LMCs using the algorithm controller. A set of parameters can be sent over the PCIe bus. Using the direct memory access mode (DMA), image data can be streamed from a host internal storage device or the main memory to and from the FPGA board.

The resulting output data stream can be post-processed on the host system using the automotive framework ADTF [3]. This framework implements a data flow graph in software. This allows the integration of the hardware concept of a FlexWAFE-based hardware algorithm in a full driver assistance software solution developed with ADTF. The hardware algorithm is represented as one node in the ADTF graph. Inbound and outbound data streams can also be modeled using the framework. This enables an easy switch between an on- and offline processing mode. The communication protocol between the host and the FlexWAFE board is hidden using the programming interface and can be accessed from an ADTF wrapper for a FlexWAFE-based hardware algorithm.

5 Algorithm Deployment

For the platform evaluation, a pre-processing algorithm has been deployed as a set of DPU nodes. Motion estimation has been chosen because of its high computational needs and massive parallelization potential [10]. The algorithm can be considered as a basis for 3D reconstruction, or for the tracking of reconstructed 3D points (6D Vision). In an image sequence, the movement of pixels or small pixel blocks needs to be determined. Even for a small amount of image points to be matched between two images, motion estimation is a very cost-intensive function on a general purpose processor. However, it can be massively parallelized in hardware as the pixel operations can be executed independently. In an initial step an existing block matching algorithm [10] has been adapted as a DPU. The input images are divided in pixel blocks using a predefined raster, which are then searched in the preceding image in the near neighborhood of the original block position. As the possible movement of objects in a vehicle's environment is limited by the current driving speed and frame rate, the search space is limited as well. As a first metric for the movement, a sum of absolute differences (SAD)-based algorithm using exhaustive search was chosen because of its regular integer operations. Equation 1 describes the calculation needed for each possible block position.

$$D(i,j,x_0,y_0) = \sum_{x=x_0}^{x_0+N} \sum_{y=y_0}^{y_0+N} |i_t(x,y) - i_{t-1}(x+i,y+j)| \tag{1}$$

$D(i, j, x_0, y_0)$ defines a matching rate between a reference block and a selected block in the search area defined by offsets i and j relative to the reference position. N defines the block size, x_0 and y_0 the upper left corner position of the reference block in the input image. The search space is relative to the reference position in the subsequent image. A specialized processor for the SAD calculation, first introduced in [10], has been adapted to the generic DPU interface. The structure follows the idea of systolic arrays [11]. Like many other data flow architectures, systolic arrays, can be directly implemented with FlexWAFE. Reference and search area are pumped through an array of SAD units. After an initial latency the processor computes the result for one block comparison each cycle. For example, a search area of 32×32 pixels with a block size of 16×16 pixels needs 256 single comparisons for one block match calculation.

6 Results

For a performance benchmark the motion estimation block has been instantiated as a DPU in the FlexWAFE architecture as shown in Figure 6. Two subsequent images are stored using external memory and are fed into the motion estimation DPU simultaneously, which outputs the corresponding motion vectors.

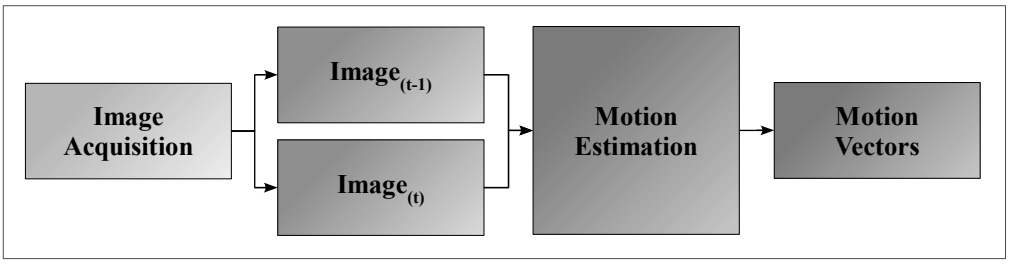

Figure 4: Instantiated FlexWAFE components for the block-matching algorithm

The application has been tested for a low resolution image of 0.5K with a block size of 16 pixels and a search area of 32 pixels. Figure 5 shows the output vectors of the motion estimation array, post-processed and displayed as an overlay on the input image and as extracted vector field. Considering the image, block and search area size the architecture requires a minimum performance of about 3.3 billion operations per second (GOPS) to achieve real-time processing (30 frames per second). If the image acquisition frame rate is assumed sufficiently high, the instantiated motion estimation DPU and the architecture is able to achieve about 400 frames per second (FPS) at a clock rate of 125 MHz. This performance enables the designer to reuse the motion estimation DPU for multiple cameras or higher resolutions.

Although the qualitative processing results can be improved by more sophisticated motion detectors, the impressive performance results demonstrate that FPGAs are an attractive hardware platform for camera-based driver assistance applications. Next generation chips announced for 2010 such as the Xilinx Spartan-6 devices show that the available logical space in automotive certified devices will increase at a very low power consumption level [12].

Figure 5: Resulting motion vectors, (a) first input image, (b) second image with resulting motion vector overlay, (c) vector field

7 Conclusion

This paper presents a design proposal for developing image preprocessing ECUs. In order to satisfy the performance requirements of tomorrow's driver assistance applications the design is based on FPGA hardware accelerators. Flexible design reuse and ECU modeling is achieved by a generic interface and interconnection model between specialized processing nodes. For evaluation purpose a concrete image preprocessing algorithm has been ported to the architecture and platform, which satisfies even multi-camera setups. In order to meet various requirements, a DPU can be instantiated multiple times by simply redesigning the processing pipeline.

In contrast to typical hardware implementations for driver assistance systems the presented approach allows flexible design reuse and scales in performance with increasing design size. This scalability is a common problem of central bus architectures, which are typical in automotive System-on-chip (SoC) design. The architecture allows the realization of real-time capable systems at a very early development phase, which dramatically decreases the gap between a first algorithm test and a final implementation. All these aspects together with next generation low power FPGA devices add up to a promising FPGA-based architecture design and test for future high-performance image processing ECUs.

References

[1] A. do Carmo Lucas, H. Sahlbach, S. Whitty, S. Heithecker, and R. Ernst, "Application Development with the FlexWAFE Realtime Stream Processing Architecture for FPGAs," *ACM Transactions on Embedded Computing Systems Special Issue on Configurable Computing: Configuring Algorithms, Processes and Architecture*, vol. 9, no. 1, 2009.

[2] A. Lucas, S. Heithecker, P. Rüffer, R. Ernst, H. Ruckert, G. Wischermann, K. Gebel, R. Fach, W. Huther, S. Eichner, *et al.*, "A reconfigurable HW/SW platform for computation intensive high-resolution real-time digital film applications," in *Proc. of the conference on Design, Automation and Test in Europe (DATE)*, Association for Computing Machinery, Inc, 2006.

[3] P. Voigtländer, "ADTF: Framework for Driver Assistance and Safety Systems," *ATZ*, vol. 2008-09, 2008.

[4] J. Angermeier, U. Batzer, M. Majer, J. Teich, C. Claus, and W. Stechele, "Reconfigurable HW/SW Architecture of a Real-Time Driver Assistance System," in *Proc. of the 4th international workshop on Reconfigurable Computing (ARC)*, pp. 149–159, 2008.

[5] C. Claus, J. Zeppenfeld, F. Müller, and W. Stechele, "Using partial-run-time reconfigurable hardware to accelerate video processing in driver assistance system," in *Proc. of the conference on Design, Automation and Test in Europe (DATE)*, pp. 498–503, 2007.

[6] S. Okazaki, S. Kyo, and F. Hidano, "IMAPCAR: A highly parallel integrated memory array processor for in-vehicle image recognition applications," in *Proc. of the World Congress on Intelligent Transport Systems (ITS)*, 2006.

[7] J. A. Kahle, M. N. Day, H. Hofstee, C. Johns, T. Maeurer, and D. Shippy, "Introduction to the Cell multiprocessor," *IBM journal of Research and Development*, vol. 49, no. 4/5, pp. 589–604, 2005.

[8] E. Lee and D. Messerschmitt, "Synchronous data flow," *Proc. of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[9] S. Whitty and R. Ernst, "A Bandwidth Optimized SDRAM Controller for the MORPHEUS Reconfigurable Architecture," in *Proc. of the International Parallel and Distributed Processing Symposium (IPDPS)*, April 2008.

[10] C. Sanz, M. Garrido, and J. Meneses, "VLSI architecture for motion estimation using the block-matching algorithm," in *Proc. of the European Design and Test Conference (ED&TC)*, pp. 310–314, 1996.

[11] H. Kung and C. Leiserson, "Algorithms for VLSI processor arrays," *Introduction to VLSI systems*, pp. 271–292, 1980.

[12] M. Klein, *White Paper: Spartan-6 and Virtex-6 Devices, Power Consumption at 40 and 45 nm*. Xilinx, Inc., WP298, V1.0 ed., April 2009.