The Design of a Smart Imaging Core for Automotive and Consumer Applications: A Case Study

Wido Kruijtzer[1], Winfried Gehrke[2], Victor Reyes[3] , Ghiath Alkadi[1], Thomas Hinz[2], Jörn Jachalsky[4], and Bruno Steux[5]

[1]Philips Research, High Tech Campus 31, 5656 AE Eindhoven, The Netherlands

[2]Philips Semiconductors Hamburg, Germany [3]University of Las Palmas GC, Spain

[4]University of Hannover, Germany [5]École des Mines de Paris, France

Wido.Kruijtzer@philips.com

ABSTRACT

This paper describes the design of a low-cost, low-power smart imaging core that can be embedded in cameras. The core integrates an ARM 9 processor, a camera interface and two specific hardware blocks for image processing: a smart imaging coprocessor and an enhanced motion estimator. Both coprocessors have been designed using high-level synthesis tools taking the C programming language as a starting point. The resulting RTL code of each coprocessor has been synthesized and verified on an FPGA board. Two automotive and two mobile smart imaging applications are mapped onto the resulting smart imaging core. This mapping process of the original C++ applications onto the smart imaging core is also presented in this paper.

Categories and Subject Descriptors

C.3 [Special-purpose and application-based systems]: Real-time and embedded systems, signal processing systems.

General Terms

Design, Performance, Algorithms, Verification

Keywords

System level design, Image processing

1. INTRODUCTION

The increasing integration of technology allows to enhance video compression cores with smart imaging functionality and to embed these cores even into low-cost camera devices. This is the starting point for new smart imaging applications that are able to analyze the content of images and video sequences enabling new consumer applications that are targeting various domains such as mobile and automotive. Cameras embedded in mobile phones are now becoming a commodity supporting applications like capturing and transmission of still images as well as video clips (Multimedia Messaging Services). With the increase of network bandwidth (e.g. 3G UMTS) real time mobile video links will become feasible, enabling new applications like mobile video telephony and video chat. It has to be noted, that the ease of use of these applications is of high importance as this is expected to be a crucial requirement for market acceptance of such new services. Thereby not only quality issues like frame and image stabilization are to be focused but also the user comfort. The automatic detection and tracking of the user's head is such an example, which helps to keep one's face in view of the camera during a mobile video telephone conference In the automotive domain, cars are equipped with more and more electronic systems that support the driver to avoid accidents. These systems are used to analyze complex driving situations and provide important and reliable information to the driver.

Some of these driving aids use radars like the automatic cruise control but driving assistance systems using cameras also appear since they have less interference with its surroundings. Furthermore, techniques like radar lack the possibility to classify detected objects. Here two examples are low speed obstacle detection [2], which deals with the detection of vehicles in a certain speed range, and pedestrian detection [3], which concerns the detection of pedestrians and an impact prediction in order to reduce the injuries of the pedestrians hit by a car.

The aim of the work described in this paper was to develop a smart imaging core that can be embedded in a camera. This core should be low-cost, low-power and suitable of supporting the above mentioned automotive and mobile communication applications. The resulting architecture integrates two coprocessors that have been designed using high-level synthesis tools taking the C-language as a starting point. Furthermore, the verification and design of the system architecture was performed using abstract transaction level SystemC models.

The remainder of this paper is organized as follows. In Section 2 we will discuss the structure of the smart imaging algorithms used by the applications. In Section 3 we disclose the smart imaging architecture while Section 4 and 5 describe the coprocessors in detail. In Section 6 we will introduce a virtual prototype that has been used to map the applications onto the smart imaging core. Finally Section 8 presents some conclusions.

2. SMART IMAGING ALGORITHMS

The applications used as a reference for developing our smart imaging core, are built up using three types of algorithms: low-level, medium-level, and high-level algorithms as depicted in Figure 1. Low-level algorithms (LLAs) process pixels typically on image segments. Examples of LLAs are linear kernel filtering, thresholding or morphological operations. The LLAs used for the smart imaging algorithms are standardized and provided through the CAMELLIA Image Processing Library (C-IPL) for which the source code can be found at Sourceforge [1]. Medium-level algorithms (MLAs) are typically used for an abstraction of the scene contents (feature extraction). Their input data are mostly pixel data, whereas output data represent abstracted image data. MLAs use several LLAs to perform their task. High-level algorithms (HLAs) deal with abstract semantic information extracted from a video scene and are typically control dominated. HLAs make the fusion between several MLAs, which individually are not sophisticated enough to yield a good result and also comprises the output stage that produces the result of the system. As an example, the automotive application low speed obstacle detection [2] (LSOD) is composed of an HLA that combines the output of several vehicle detectors (MLAs) in order to obtain an exact detection and localization of vehicles. The MLAs used in LSOD are: shadow detection, edge detection, rear lights detection, symmetry detection and motion segmentation.

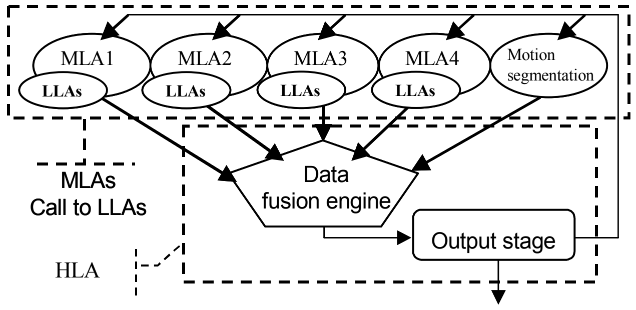

Figure 1. Smart imaging application structure

Our goal is to bring smart imaging into the consumer market, in which high performance general-purpose processors are not accepted as a cost efficient solution. Typically the architectures in this domain contain a rather modest general-purpose processor (e.g ARM9) running at a few hundred MHz. The HLAs nicely fit on such a general-purpose processor as they are control dominated and their computational load is limited. The LLAs however are associated with a high amount of inherent parallelism and relatively simple operations. Due to the high throughput rate required for the execution of LLAs, efficiency can be significantly improved by exploiting data-level parallelism. This typically is not efficient on a general-purpose processor. As an example both scaling and 3x3 linear kernel filtering need a 225 Mhz ARM9 (un-optimized C-code) to execute on video data at 25 frames per second with the resolution of 352x288 pixels per frame. With optimized C-code we might achieve each single LLA to execute for the specified frame size and rate at a 100Mhz processor. Still the smart imaging application would not fit as each MLA uses several LLAs and typically a couple of MLAs are used in each application (e.g. LSOD). Clearly some form of hardware acceleration that exploits the inherent parallelism of the LLAs is needed.

3. SMART IMAGING CORE

The architectural class that has been chosen for the realization of the system is the classical coprocessor architecture. The coprocessors execute a specific algorithm or a class of algorithms with similar computational requirements faster than a general-purpose programmable architecture resulting in a significantly higher ratio of computational performance and system cost compared to other architectural approaches. Compared to [4] and [5], this core is low-cost, low-power and targets a wide application area unlike [6] that is optimized for a single application. Furthermore our solution has built-in logic to work on image segments instead of complete images, something not present in these other solutions.

The core architecture is depicted in Figure 2. The HLA and the more control-oriented part of the MLAs are combined together in a task, which fits well to be mapped onto the embedded ARM9. All the LLAs are combined in a pixel-processing task, which is mapped onto a single smart imaging coprocessor (SI). Likewise, the pixel processing part of the motion segmentation MLA is distinguished as an independent task, which is mapped onto a motion estimation coprocessor (ME). Tasks in the CAMELLIA system communicate and synchronize with each other using a standardized TTL interface and corresponding primitives [8] thereby facilitating easy integration.

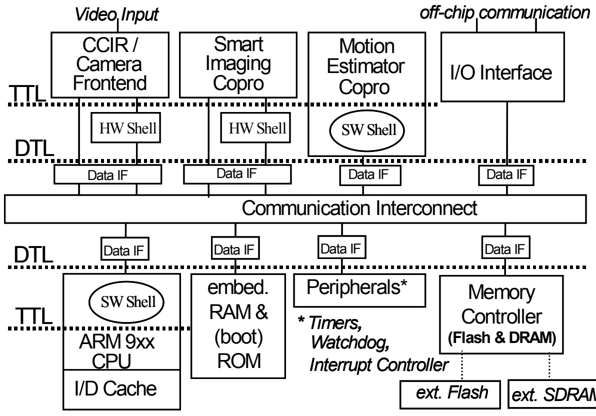

Figure 2. Architecture of the smart imaging core

One of the major challenges for the definition of such a system is the specification of the coprocessor architectures. On one hand, it is generally desired to support a wide range of algorithms that can be accelerated with the defined coprocessors. On the other hand, the efficiency of the coprocessor should be as high as possible. As typically an increase of flexibility is associated with a decrease of efficiency, these two goals can easily be highly contradictory. The impact of this basic architectural issue can be reduced if the coprocessors are aiming at the implementation of a limited set of similar algorithms. In this case, the architecture can be adapted to the requirements of the envisaged algorithmic class while supporting a minimum amount of programmability. In the next sections the design of both coprocessors is explained.

4. SMART IMAGING COPROCESSOR

As smart imaging applications have a clear need for acceleration of basic image processing tasks, the architecture of the smart imaging coprocessor (SI) has been adapted for the efficient execution of this algorithmic class. The requirements of smart imaging applications have been extracted by the analysis of sample applications from different fields of smart imaging applications. The SI accelerates most of the functions of the C-IPL and includes: Arithmetic and Morphological operations, Linear kernel filtering, Horizontal and vertical summing, Scaling, Lookup-Table, Histogram, Moments and Min-Max computation. The resulting architecture is depicted in Figure 3. It consists of an arithmetic data-path that is capable to process several image pixels concurrently. For the definition of this data path some well-known design approaches like SIMD parallelism have been incorporated to achieve a reasonable flexibility/cost trade-off. Typically the data-path processes 4 grayscale pixels in parallel or, for some binary operation, 32 pixels. This type of processing is well known from multi-media extensions used in general-purpose CPUs. An example of these extensions is the MMX instruction set [10]. In contrast to these approaches, the SI implements a deeper arithmetic pipeline. This enables the encoding and execution of complex arithmetic operations with a single microinstruction. A more detailed overview of the SI is described in [9].

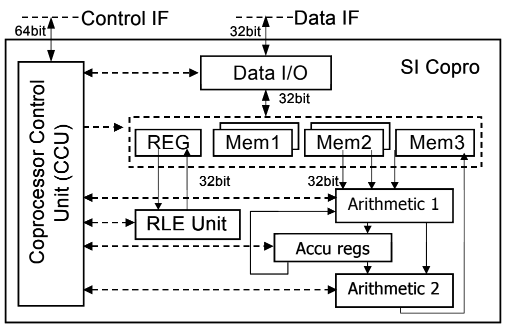

Figure 3. Smart imaging coprocessor

As image processing tasks are typically associated with a highly predictable data access, the design complexity of transparent local memories, like caches, can be avoided and the area required to provide the SI local storage can be kept relatively small. The idea of adaptation to the envisaged set of algorithms has also been applied for the control part of the SI using an hierarchical approach. At the lower level of this hierarchy the arithmetic units and the memory accesses are controlled by a VLIW approach that supports a high degree of flexibility. In order to avoid the drawbacks of the classical memory- and bandwidth-hungry VLIW architectures, a second level of control hierarchy has been introduced in the SI. This level is used to translate mighty so-called macroinstructions into a sequence of VLIW microinstructions. Several classes of macroinstructions have been defined: I/O instructions control the data traffic with system memory and allow to initiate a transfer of arbitrarily sized 2-dimensional blocks of data with a single instruction. Execution instructions typically execute a basic image-processing algorithm on an image segment previously loaded into local memory. Configuration instructions are used to set pseudo-static data, like image base addresses, segment information and data like filter coefficients. The described hierarchical control approach can be viewed as another important extensions to the principle of the vector based SIMD programming model of current general-purpose CPUs. The adaptation of the data-path's arithmetic and the chosen hierarchical control strategy allows to chose a well-suited trade-off between flexibility and area efficiency for the envisaged application domain.

4.1 SI design

The design approach chosen for the SI has applied a model-based strategy. In order to support the architectural refinement and verification on system level, a bit-true SystemC based reference implementation of the SI combined with a high-level environment comprising a functional CPU model, system memory, and a video I/O module has been created. The model is depicted in Figure 4. As a first step the basic arithmetic requirements have been derived by an analysis of LLAs mapped onto the SI. Based on this analysis the initial functionality of the data path has been selected and modeled in SystemC. Based on this model an exact performance analysis for the envisaged application range has been performed and the model has been adapted in an iterative way according to the findings of this analysis. The result of this work was a bit-true representation of the SI, which has been used as a reference for the actual implementation of the SI design.

In order to support a smooth migration from the reference representation, written in SystemC, to the actual design it is reasonable to aim at a design flow that is entirely SystemC-based. This approach enables an iterative refinement of the reference implementation towards a bit- and cycle-true design on RT-Level. Another important advantage of this approach is the concurrent HW/SW design: As the application SW is developed in C/C++, the reference as well as the RTL database can be easily linked as a library together with the application code. Thus, the SW designer is able to derive detailed performance figures of the final system before the HW has been implemented in silicon. Moreover, the HW architect can perform final optimizations of the HW implementation based on the received feedback on performance of the applications.

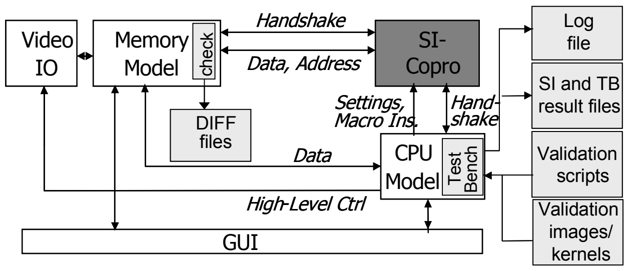

Figure 4. SI Reference model

Several vendors claim to support a set of tools achieving significant decrease of design time by applying a SystemC based behavioural synthesis flow. For a coarse classification of these tools, two basic approaches can be distinguished. Some tools are template based, i.e., the basic architectural concept is fixed and contains already a set of basic architectural components, like address generation units, basic controllers and arithmetic functions. The designer just adds the special functionality required for his application domain. Other tools give more architectural freedom to the designer and support the design on a lower level of granularity. The advantage of template-based approaches is the reduced design effort, because the tool itself provides already major infrastructure components. As the basic architectural template of these approaches is fixed, the opportunities for architectural optimization are restricted. The other tool class mentioned gives more control on the chosen architecture to the designer. As the architectural model does already sketch a clear view on the target architecture, the implementation of the SI has been based on a tool from this second class.

One promising approach was Cocentric SystemC Compiler from Synopsys Inc. This tool supports a SystemC-based design entry and allows for behavioral as well as RTL synthesis. The behavioral synthesis option is based on the automatic generation of memory structures, datapath elements as well as the required control FSM for a specific design block. Moreover, the tool supports several useful features like operator and memory sharing or automated memory instantiation. During the design phase of the SI it turned out that the tool is especially useful for dataflow-oriented designs, aiming at a dedicated implementation of core functions of the envisaged application. The approach is less useful for microinstructions-controlled designs like the SI. The main issues observed were the missing support for the automated generation of flexible arbitration circuitry of embedded memories, shared by different modules or processes and a relatively rigid approach for applying pipelining in architectures controlled by microinstructions. Mainly these issues led to the final decision to migrate from the architectural SystemC model to a SystemC-based RTL description that was synthesized with Cocentric SystemC Compiler. However, the described advantages of a SystemC-based design have been exploited during the implementation of the SI. We expect that more advanced tools will enter the CAD market place and will solve the identified flow issues. For the time being a promising alternative is to apply template-based tools for the basic architecture implementation and to combine these with dataflow-oriented tools for the implementation of core arithmetic units.

For the verification of the SI a test bench was developed that targets to perform the verification of the SI-Core with a reasonable coverage. Since the SI supports a certain range of programmability, i.e. different image segment sizes and instruction parameters, it is important to validate the functionality of all the SI coprocessor macroinstructions for various parameters in a structured way. Therefore the test bench was implemented based on scripts as an extension of the original SI SystemC reference model (see Figure 4). The test bench itself is executed on the CPU model. One major function of the test bench is the execution of the reference code of all macroinstructions. A macroinstruction is validated for a certain parameter set by first running the reference code on the CPU model producing the reference data that is written into the memory model. Afterwards the SI is programmed to perform exactly the same macroinstruction with the same parameter set. As the SI is writing its result into the memory model it is compared with the reference data by the checker module. Any deficiency is monitored and can be reported in various ways, depending on the validation settings. A *DIFF* file can be generated for instance, which indicates every pixel that differs between the SI and reference implementation including the results from both implementations.

5. MOTION ESTIMATION COPROCESSOR

Motion estimation is one of the time-critical tasks in the application algorithms. Apart from the typical sum-of-absolute-difference operations performed at pixel level, the combination of the required sub-pixel (quarter-pixel) accuracy, the size of the blocks, the number of motion vector candidates, the number of passes (scans) per frame, the frame or region-of-interest size, the frame rate, contribute to a practically intractable problem if no optimizations at all design levels (e.g. system, algorithm, architecture) are performed. Therefore, one of the decisions at system level was to map the motion estimation task onto a coprocessor. The block-based Motion Estimation coprocessor (ME) accelerates the motion segmentation MLA. The goal of motion segmentation is to identify moving objects from their motion. The motion segmentation is integrated tightly with motion estimation through a loop in which candidates for motion estimation are generated based on the result of segmentation. First a motion model for each block is calculated after which blocks are grouped that have a similar motion model and low sum-of-absolute-differences using a Breadth First Search algorithm.

Currently, two contrasting implementations are often considered for high performance video processing: ASICs and DSPs. ASICs optimally meet performance and power requirements, but lack flexibility. DSPs are highly flexible, but have significant overhead in achieving the performance requirements for a low power budget. The ME has been designed as an Application Specific Instruction Processor (ASIP). ASIPs offer performance, power and area that are comparable to ASICs but are superior in terms of performance, power and area compared to DSPs for applications in their domain. ASIPs, tuned to an application domain, can be based on any processor architecture template such as a very long instruction word (VLIW) architecture , or a vector processing architecture. It is interesting to note that the choice of the ASIP template architecture greatly depends on the characteristics of the application domain and the tool flow available. Among the available tool flows for ASIP design, namely A|RT [13], LISA [12] and CHESS [10], the A|RT-based tool flow has been used that uses a VLIW architecture template. So in contrast to the SI design the ME uses a template-based approach.

The data-path of the ME ASIP consists of standard functional units, e.g. Arithmetic-Logic Units (ALUs) and Address Calculation Units (ACUs), and Application Specific Units (ASUs), tailored for accelerating the inner kernels of motion estimation. The ME is flexible within an application domain and can be programmed for different video applications while benefiting from the instruction-set that accelerates motion estimation functionality.

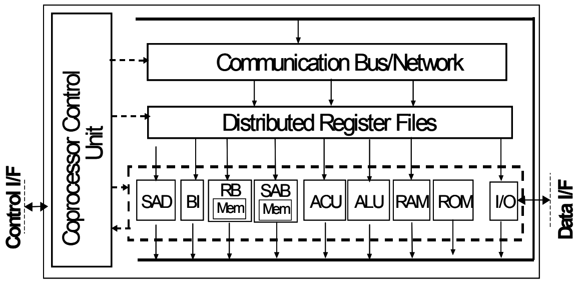

Figure 5. Motion estimation coprocessor

First a hardware/software partitioning by determining the compute- and control-intensive tasks of the application set has been performed. As a result four ASUs that can process 16 pixels in parallel have been used (see Figure 5). The ASUs are based on a previously developed general ME template [14]. The complete search area (from previous frame) is stored in the search area buffer (SA buffer) ASU and the reference block buffer (RB buffer) ASU is used to store the reference block (from the current frame). The bi-linear interpolation (BI) ASU is used for generating corresponding pixels for the SAD calculation in case sub-pixel accuracy of motion models is required. The sum-of-absolute-differences (SAD) ASU is used to calculate the SAD of every candidate motion model. It compares a block within the current frame and the corresponding block within the previous frame shifted by the motion model candidates. The ALU and ACU perform the arithmetic operations of the ME core. These ASUs are standard A|RT Designer library components. In contrast to the motion estimator described in [14] the ME calculates SAD values per 16x16 macro-block as a weighted sum of SADs from both luminance and chrominance pixel data. Each video component (Y,U,V) is calculated sequentially using the single set of ASUs described above. As mostly in popular video formats the chrominance pixel-data is sub-sampled the ASUs are also able to process 8x8 pixel blocks.

5.1 ME Design

The design of the ME starts from the C++ behavioral description of the motion estimator algorithm. First the behavioral description of the ME was partitioned into a SW task that prepares the motion model candidates and should run on the ARM CPU and a HW task that performs the main processing loop of the motion estimation algorithm. The input parameters to the HW task consist of two parts namely frame constants (e.g. frame size) and run-time parameters (e.g. motion model candidates and block coordinates). The C-code of the HW task was translated into ANSI-C, as required by A|RT Designer, and an initialization state was introduced such that frame constants are communicated only once. Furthermore several new data-types were introduced to allow communication of run-time parameters on a stripe (eight pixel blocks) basis. Next, we modified the C-code of the processing functions (data-path) in the HW task by integrating behavioral models of the ASUs. Finally bit and cycle true models of the ASUs are integrated replacing the behavioral code of the ASUs. Each step is verified with the reference C++ code of the ME by comparing the intermediate results of the motion estimator such as the generated candidate motion models and the resulting motion models calculated by the HW task of the ME.

The C-code of the HW task resulting from this last step can directly be used as an input of A|RT Designer. The result is a synthesisable RTL description of a custom VLIW processor, consisting of a data-path and a controller. A|RT Designer connects the ASUs with a set of register files and generates their interconnects. The controller contains an FSM that determines the next instruction to be executed, and a micro-code ROM, that contains the scheduled VLIW code of the HW task C algorithm. For the validation of the RTL description a separate test bench has been created in which the SW task of the ME is simulated by means of a script. The script contents is generated by the partitioned and refined C-code described above and includes read and write commands of both run-time parameters and image data.

6. Application mapping

To validate the correctness (and quality) of the applications executed in the targeted system architecture an FPGA based demonstrator or prototype was built. This validation comprises the verification of both the implemented coprocessors (functionality and performance), as well as the software optimizations required for its execution in an embedded system. Instead of directly moving into the FPGA based prototype an intermediate step using a virtual prototype was used. This approach allows verifying the HW/SW integration in an early stage. Furthermore the ME and SI can be intensively verified together with the software before pursuing their actual FPGA implementation by using the virtual prototype as a system test bench. This helps in early bug detection and eases the tuning of different architectural parameters.

6.1 Virtual prototype

The virtual prototype was built using CASSE [7]. CASSE models the architectural elements at the higher abstraction level using transaction-level modeling techniques. Transaction-level modeling (TLM) [15] has been promoted as the next abstraction level above RTL and its aim is to achieve increased simulation speeds, while keeping enough accuracy for system analysis and verification. Furthermore, CASSE eases the fast modeling of complex embedded systems by using an interface-based design approach, where the communication among architectural elements is based on predefined interfaces and protocols [8].

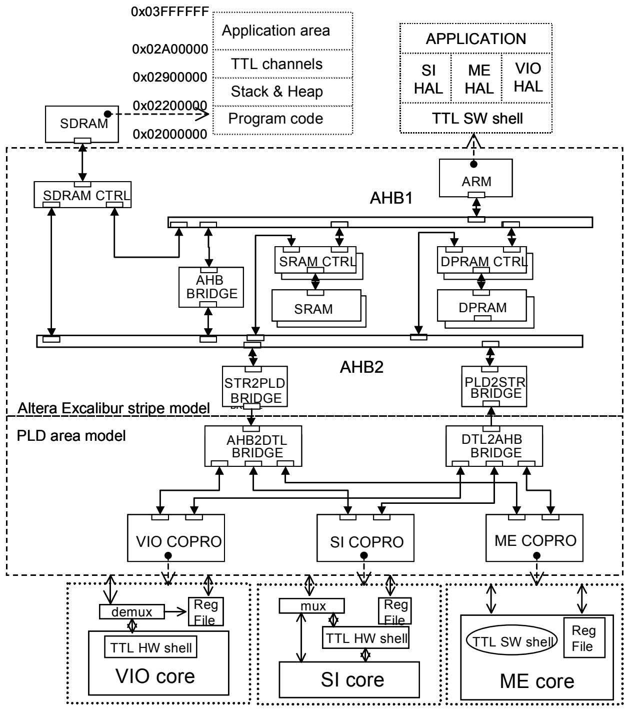

Figure 6. Virtual prototype

The virtual prototype, as shown in Figure 6, is composed of a Video-IO coprocessor, an embedded CPU, two dedicated coprocessors (ME and SI), a shared memory and a communication network that in turn is composed of several buses and bridges. This setup reflects the internal FPGA architecture. The TLM models developed for the SI and ME when compared with their equivalent RTL models result in equal functionality but hundred times faster simulation speed.

Once the architectural model is finished the work of integrating the software application is started. Although the low-level communication and synchronization between the software and the coprocessors is solved by the usage of the TTL interface, still the application functionality has to be adapted to use the coprocessors instead of the software routines and low-level libraries used before (i.e. LLAs). Therefore, different hardware abstraction layers (HAL) are created (on top of the programming model) in order to provide the software tasks with a high level API to use the SI, ME and Video-IO coprocessor. The software structure is depicted in Figure 7. These HALs hide to the designer the low-level details of the system and provides a well-structured API with function calls and parameters passing for embedded software development. This approach allows that further modifications in the system architecture would only required slight changes in the HAL and/or TTL implementation, keeping the rest of the software application unchanged. This eases significantly the porting of the reference and future applications (i.e. software reuse).

Furthermore this approach allows also the use of an abstract CPU model instead of an instruction set simulator (ISS). The CPU model is an encapsulation of the embedded SW into a SystemC model. This approach is significantly faster compared to using an ISS. Note that the encapsulated SW uses the address map of the real prototype and all relevant data structures such as image data and the TTL channel contents reside in the simulated memory model. This has been achieved by using the HALs that are build on top of the TTL interface. Each TTL port is configured with physical addresses from the embedded memory map as shown in Figure 6.

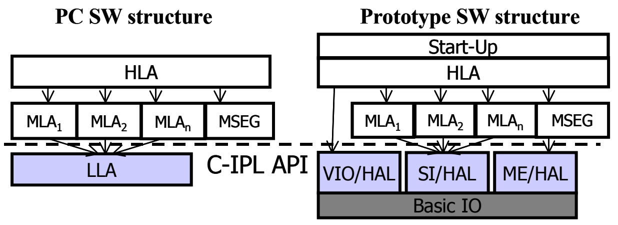

Figure 7. Software structure

6.2 FPGA prototype

The FPGA prototype is build using a PCI based prototyping board with two Altera FPGA devices: an Excalibur XA10 device with 1 million logic gates and an APEX-1500 with 1,5 million logic gates. The Excalibur also embeds an ARM9 subsystem that is used to run the embedded software parts of the applications. The FPGAs are used to implement the hardware coprocessors and the toplevel communication infrastructure. This FPGA prototype is very close to an actual chip implementation. Since the size of the SI logic after the synthesis and the place&route exceeded 1 million FPGA gates, the most likely partitioning of the smart imaging architecture on the prototyping board was to map the SI co-processor and its memory on the APEX1500 FPGA device. The ME and the infrastructure are mapped to the Excalibur device. The infrastructure comprises the multiple DTL, AHB and PCI bridges.

Next to the various embedded TTL implementation such as ARM code the TTL interface has also been implemented on a PC on top of the PCI driver shipped with the prototyping board. By using this PC version of TTL it is now possible to migrate individual components from the smart imaging core such as the SI coprocessor into the FPGA board while keeping the rest of the architecture on the PC as a SystemC model. The part running on the PC serves as system test bench for the component integrated in the FPGA. This helps significantly to manage the verification complexity by gradually moving components from the virtual prototype into the FPGA.

7. SYNTHESIS RESULTS

The synthesis results for both the FPGA and standard cell implementation are listed in Table 1. In total ten single-ported 256x32 bit RAM blocks are used as embedded memory inside the ASUs of the ME. All intermediate and motion model results are mapped into a single RAM with a size of 64 Kbits. Furthermore the controller of the ME integrates several ROM blocks with a total size of 172 Kbit. The SI integrates in total 40 Kbits RAM With a target clock frequency of 150 MHz, the arithmetic unit of the SI has a peak performance of about 3 GOPS and the ME can process 150 frames per second for a frame size of 352*288, using a single scan and 15 motion models per block.

Table 1. FPGA and Standard cell synthesis results

Technology	**Altera FPGA**		**CMOS 90nm @150 Mhz (mm^2)**			
Resources	Logic	Memory	Logic	RAM	ROM	Total
SI	1,2 Mgates	40 Kbits	0,72	0,13	-	0,85
ME	0,8 Mgates	246 Kbits	0,45	0,26	0,14	0,85

8. CONCLUSIONS

In this paper the design of a smart imaging core was presented. The core integrates two coprocessors that have successfully been designed using high-level synthesis tools. Furthermore SystemC based system level design tools have been applied both in the verification of the complete core (both hardware and software) and of the individual coprocessors. The major advantage of a SystemC-based design flow compared to traditional approaches, is the easy integration of SystemC design descriptions into a reference model that support concurrent HW and SW development. In terms of C-based synthesis flows we believe a promising approach is to apply template-based tools for the basic architecture implementation and to combine these with dataflow-oriented tools for the implementation of core arithmetic units.

9. REFERENCES

[1] Camellia Image Processing Library http://camellia.sourceforge.net

[2] B. Steux, Y. Abramson, “Robust real-time on-board vehicle tracking system using particles filter”, IFAC IAV’04, July 2004

[3] Abramson, Y., B. Steux, “Hardware-friendly pedestrian detection and impact prediction,” IEEE IVS’04, June 2004

[4] Kyo, S., et al “A 51.2GOPS Scalable Video Recognition Processor for Intelligent Cruise Control Based on a Linear Array of 128 4-Way VLIW Processing Elements”, IEEE ISSCC’03, February 2003

[5] Raab, W., Bruels, N., Hachmann, U., Harnisch, J., Ramacher, U., Sauer, C., “A 100-GOPS Programmable Processor for Vehicle Vision Systems”, IEEE Design & Test of Computers, 2003.

[6] Imagawa, K.; Iwasa, K.; Kataoka, T.; Nishi, T.; Matsuo, H., “Real-time face detection with mpeg4 codec lsi for a mobile multimedia terminal”,.ICCE’03, June 2003

[7] Reyes, V.; Bautista, T.; Marrero, G.; Carballo, P.P.; Kruijtzer, W.; “CASSE: A system-level modeling and design-space exploration tool for multiprocessor systems-on-chip”, DSD’04, August 2004

[8] Pieter van der Wolf, et al “Design and Programming of Embedded Multiprocessors: An Interface-Centric Approach”, CODES +ISSS ’04, Stockholm, September 2004,

[9] Jörn Jachalsky, et al, “A Coprocessor for Intelligent Image and Video Processing in the Automotive and Mobile Communication Domain, ISCE2004,September 2004.

[10] A. Peleg, and U. Weiser, "The MMX Technology Extension to the Intel Architecture," IEEE Micro, Vol. 16(4), Aug. 1996.

[11] D. Lanneer., et al, “CHESS: retargetable code generation for embedded DSP processors”, Code Generation for Embedded Processors, P. Marwedel, ed., Kluwer Academic Publishers, 1995.

[12] A. Hoffmann, et al “A novel methodology for the design of application-specific instruction-set processors (ASIP) using a machine description language”, IEEE TCAD, Nov. 2001.

[13] A|RT Designer and A|RT Builder tools, formerly from Adelante Technologies, now marketed by ARM Ltd. as OptimoDE, http://www.arm.com/products/CPUs/families/OptimoDE.html

[14] H. Peters et al, “Application Specific Instruction-Set Processor Template for Motion Estimation in Video Applications”, IEEE TCSVT, Vol. 15, No.4, April 2005.

[15] Lukai Cai and Daniel Gajski, “Transaction Level Modeling: An Overview”, in CODES+ISSS’03, California, USA, October 2003