Profiling-Based Hardware/Software Co-Exploration for the Design of Video Coding Architectures

Heiko Hübert and Benno Stabernack

***Abstract*—The design of embedded hardware/software systems is often subject to strict requirements concerning its various aspects, including real-time performance, power consumption, and die area. Especially for data intensive applications, the number of memory accesses is a dominant factor for these aspects. In order to meet the requirements and design a well-adapted system, the software parts need to be optimized and an adequate system and processor architecture needs to be designed. In this paper, we focus on finding an optimized memory hierarchy for bus-based architectures. Additionally, useful instruction set extensions for application-specific processor cores are explored. For complex applications, this design space exploration is difficult and requires in-depth analysis of the application and its implementation alternatives. Tools are required which aid the designer in the design, optimization, and scheduling of hardware and software. We present a profiling tool for fast and accurate performance, power, and memory access analysis of embedded systems. This paper shows how the tool can be applied for an efficient hardware/software co-exploration within the design flow of processor-centric architectures. This concept has been proven in the design of a mixed hardware/software system with multiple processing units for video decoding.**

***Index Terms*—Computational complexity, optimization, performance evaluation, video coding.**

I. Introduction

THE PROCESSING of visual data is often very demanding in terms of processing power and data transfers. In order to find an appropriate implementation of these algorithms as embedded system-on-chips (SoCs), it is very important to adapt the system to the application in order to find an efficient solution. Additionally, the different algorithmic alternatives need to be explored in their implementation efficiency on the various system setups. This exploration has to take place in an early design phase in order to keep the design space broad.

The evaluation of the influences of the different algorithmic, software coding, and hardware architecture alternatives on the system's performance and efficiency requires an in-depth analysis of the system. When changing a parameter in one of these alternatives, the influence is not always obvious. For example, the influence of caches depends on the type of memory accesses and may vary highly across the application. However, it is crucial to recognize the parts of the system that benefit or suffer from a specific change in order to adapt the system properly. In complex applications with up to 100 000 lines of code, identifying these influences is difficult and requires tools to pinpoint the locations. Besides the computational complexity, especially in video computing data transfers have a huge influence on the performance. Therefore, a memory access analysis is required in addition to clock cycle distribution analysis across the application.

Manuscript received January 30, 2009; revised May 23, 2009 and July 31, 2009. First version published September 1, 2009; current version published October 30, 2009. This paper was recommended by Associate Editor G. G. Lee.

The authors are with the Department of Image Processing, Fraunhofer Institute for Telecommunications, Heinrich-Hertz-Institut, Berlin 10587, Germany (e-mail: huebert@hhi.fraunhofer.de; stabernack@hhi.fraunhofer.de).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2009.2031522

II. State of the Art

Nowadays, SoCs for video signal processing are built around an increasing number of processors, with each processor taking over a dedicated task in the overall process of the particular signal processing flow. In contrast to hardwired architectures, which have been mainly used in the late 1990s and the beginning of the 21st century, these architectures offer a lot of advantages over the traditional architectures, e.g., flexibility, programmability, faster time to market, enhanced power reduction mechanisms, etc. The drawback of processor-based architectures is their inherently suboptimal behavior with respect to data or memory access due to the memory bound nature of data processing. On the other hand, it has been shown that the combination of programmable architectures with dedicated accelerator functions, e.g., application specific processor cores tends to solve these problems, with the processors taking over the task of control flow and coprocessors being used for data intensive parts of the signal processing algorithms. These heterogeneous architectures can be found predominantly for applications in the mobile domain, where a complete system is built around a so-called application processor. Typical state-of-the art examples of these architectures are NVIDIA Tegra, STMicroelectronic Nomadik ST8820 [1], or Broadcom BCM2820 [2]. To increase the available processing power for more demanding applications, e.g., high definition decoders in set-top boxes, the trend is moving from heterogeneous systems toward homogeneous multiprocessor architectures with a growing number of processor cores [3], so-called many core computing fabrics. Depending on the number of processor cores, these architectures are characterized by the used communication topology [4]. Up to a number of eight processor cores, shared memory architectures can be efficiently used. With a growing number of processor cores,

TABLE I
PROFILING TOOL COMPARISON

	Cycles	Memory Accesses	Power	Per Function	Per Line	Callgraph	Instrumentation	Accuracy	CPUs
Gprof	+	−	−	+	+	+	+	10 ms [4]	Many
ARM Profiler	+	+	−	+	+	+	−	μs/ns [4]&estimation	ARM
ATOMIUM	+	+	−	+	−	+	+		Generic
VTune	+	+	−	+	+	+[1]	−	[4]	Xscale
HDL Profiling[2]	+	+	+	−	−	−	−	ns	Any HDL
Valgrind	+	+	−	+	+	+	−	Cycle[5]	x86/PPC
SimpleScalar	+	+	+	−	+[3]	−	−	Cycle[5]	Synthetic
MEMTRACE	+	+	+	+	+	−	−	Cycle	Any ISS

[1] not for embedded processors, [2] very slow, [3] per assembly address, [4] sampling based, [5] simulated CPU.

network architectures are better suited as a communication structure.

The performance of such architectures is highly dominated by the accompanying application code. Therefore, the need for software-centric profiling tools arises. Besides the classical profiling methodologies of the software execution, the data transfer to memories and other processing units as well as the resulting bus load needs to be analyzed.

Numerous tools exist for this purpose as given in Table I. On a high abstraction level, tools such as SkillsInsight Tool (SIT). SIT [5] or ATOMIUM [6] can be used. These are especially well suited for analyzing the complexity and data intensity of an application. SIT and ATOMIUM use instrumentation code, and use abstract models. SIT adds a highly adjustable memory subsystem simulator. In [7] and [8], methodologies are described for dataflow profiling. These are also very useful for estimating the computational complexity, architecture exploration, and parallelization by analyzing the dependences within the computation. However, all these tools cannot provide timing analysis due to the level of abstraction, as they either use an abstract hardware architecture [5], [6] or are restricted to dataflow analysis [7], [8]. In [9], a methodology is presented which combines high level profiling with a cycle-accurate timing database. It profiles the number of executions of algorithmic processing kernels, such as inverse discrete cosine transform (IDCT), of an application and creates platform specific performance estimation by database mapping of these numbers to their execution times on the specific platform. The methodology requires instrumentation of the software code and cannot provide data access profiling results.

Performance profiling [10]–[15] solutions especially have been available for decades. Memory profiling [6], [13]–[15] for embedded systems has become a major issue within the last ten years. However, the existing tools either cover only parts of the required information [6], [10], [11], [14], [15] or the statistics are not provided in the required level of detail [10], [11], [15] or are restricted to a specific processor architecture [11]–[13]. Some of the tools provide results only for the entire application and not on a function or source code line level. This restricts the optimization potential, as the cause of a performance loss cannot be localized. Other tools suffer from a restricted level of accuracy [6], [10], [11], [15]. Results are based on generic processor architectures or taken with a low sample rate, or the source code is instrumented. Available highly accurate profiling mechanisms suffer from a long simulation time which makes a comprehensive analysis unfeasible.

In order to overcome the restrictions of existing profiling tools, a novel methodology has been developed that combines fast, accurate, and comprehensive profiling, incorporating the specification of the applied processor and memory architecture. This paper describes the technique and its implementation as the MEMTRACE profiling tool.

MEMTRACE delivers cycle-accurate profiling results on a C function level. The results include clock cycles, various memory access statistics, and optionally energy consumption estimation for reduced instruction set computer (RISC)-based processors. A focus is placed on memory access analysis, as for data-intensive applications this aspect has a high potential for increasing system efficiency.

Additionally to targeting software optimization, hardware specific exploration is also supported. Besides the exploration on the system level, the profiling has shown that the tool can be used for defining and optimizing an application specific instruction set. Using this technique, a RISC-like application specific processor has been developed. In order to cover codesign issues, the profiling technique has been expanded to analyze bus-based hardware/software systems.

This paper shows how co-exploration can be applied within the design flow for efficient hardware/software systems. This concept has been proven in the design of a system with multiple processing units for video decoding.

III. MEMTRACE: A PERFORMANCE AND MEMORY PROFILER

MEMTRACE [16] is a nonintrusive profiler, which analyzes the memory accesses and real-time performance of an application without the need of instrumentation code. The application is executed on a cycle-accurate system simulator to obtain the profiling results. The specification of this system, e.g., the processor or memory type, is provided manually by the designer. Any system can be specified for which processor simulation models are available and incorporated in the profiler, as later described in Section III-C.

Thus the results of the profiling reflect the application's complexity when implemented on a specific processor, so the results are platform specific. However, taking the influence of

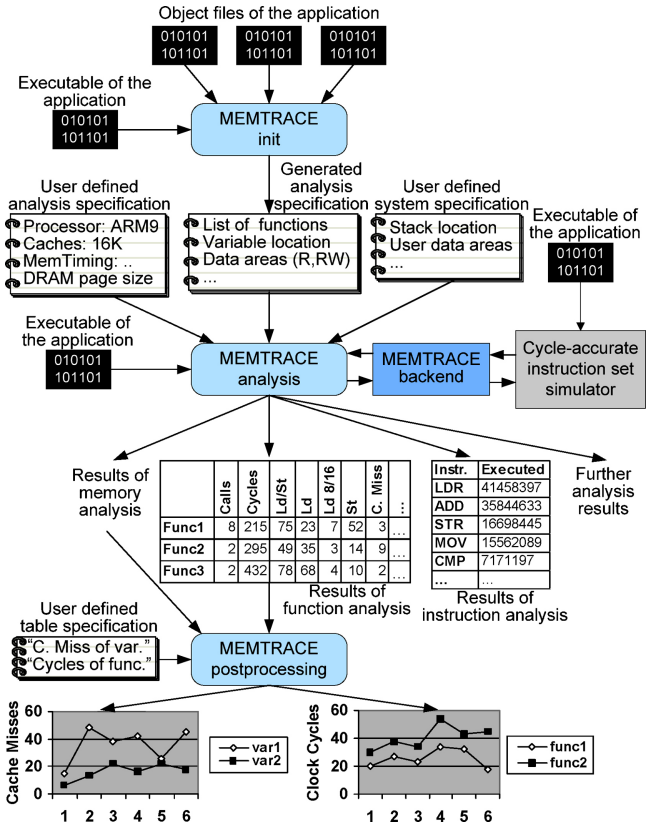

Fig. 1. MEMTRACE profiling toolflow performed in three steps.

the architecture into account is very important for a system architecture exploration, as also considered in [9].

The influence of the architecture can be based on many architectural features, for example, the usage and type of caches highly influence the performance or applying single instruction, multiple data (SIMD) instructions can accelerate video applications significantly. Profiling of such influences is essential for making a profound decision on adding these features to the system architecture.

A. Tool Structure and Usage

The performance analysis with MEMTRACE is carried out in three steps: the initialization, the performance analysis, and the postprocessing of the results, as shown in Fig. 1.

During initialization, MEMTRACE extracts the names of all functions and variables of the application and writes them to the analysis specification file.

In the second step, the performance analysis is carried out, based on the analysis specification and the system specification. The system specification includes the processor, cache, and memory type definitions (see the next section for a detailed description of this step). MEMTRACE applies an instruction set simulator (ISS) for the simulation of the user application and writes the analysis results of the functions and variables to files. Table II shows an excerpt of the profiling results. The output files serve as a database for the third step, where user-defined data is extracted from these tables.

In the third step, a postprocessing of the results can be performed. MEMTRACE allows the generation of user-defined

TABLE II
EXCERPT OF A RESULT TABLE FOR FUNCTION PROFILING

f	ca	cyl	ls	ld	st	cm	BI	BC	E	···
f1	8	215	75	22	52	5	123	92	1.45	···
f2	2	295	39	35	14	9	55	153	1.78	···
f3	2	432	78	68	10	17	143	289	3.21	···

Abbreviations: f = function name; ca = calls; cyl = bus (clock) cycles; ls = all load/store accesses from the core; ld = all loads; st = all stores; cm = cache misses; BI = bus idle cycles; BC = core bus cycles; E = energy (in μJ).

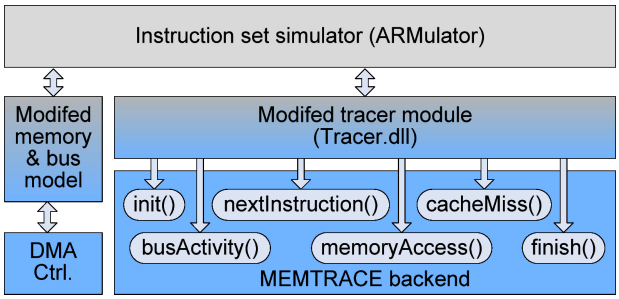

Fig. 2. Software structure of the interface between MEMTRACE backend and ISS (here ARMulator).

tables, which contain specific results of the analysis, e.g., the load memory accesses for each function.

Furthermore, the results of several functions can be accumulated in groups to compare the results of entire application modules. The user-defined tables are written to files in a tab-separated format. Thus they can be further processed, e.g., by spreadsheet programs for creating diagrams.

B. Profiling Data Acquisition

During the performance analysis (i.e., second) step the profiling data acquisition takes place. The user initializes this step for each system setup of interest by choosing a processor type, memory setup and hardware accelerators. The processor must be available in MEMTRACE as cycle-accurate ISS or generated as described in Section III-C. The hardware accelerators are supported by simple timing-annotated C-models, see Section IV-D.

MEMTRACE communicates with the ISS via its backend. Fig. 2 shows the implementation of the MEMTRACE backend for the ARMulator [17] ISS from ARM Ltd. The backend is implemented as DLL and provides six interface methods. These are introduced to the ARMulator's tracer module as callback functions. Additionally, the memory model (mapfile.dll) is extended by a mechanism for identifying the status of each bus cycle. This status can either be CORE, direct memory access (DMA) or IDLE depending on the bus usage.

At startup time the ISS calls the interface method init() to initialize the profiler. The method creates a list of all functions and global variables. For each function a data structure is created, which contains the function's start address and counters for collecting the analysis results. If the executable was created in debug mode, i.e., including source code information, a table for mapping assembly code lines to source code lines is created.

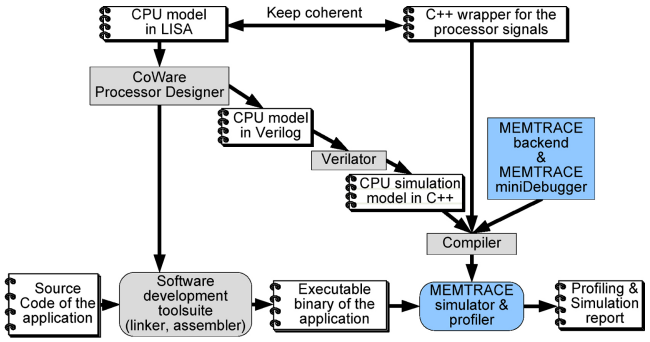

Fig. 3. Profiling toolflow incorporating CoWare Processor Designer, Verilator, and MEMTRACE.

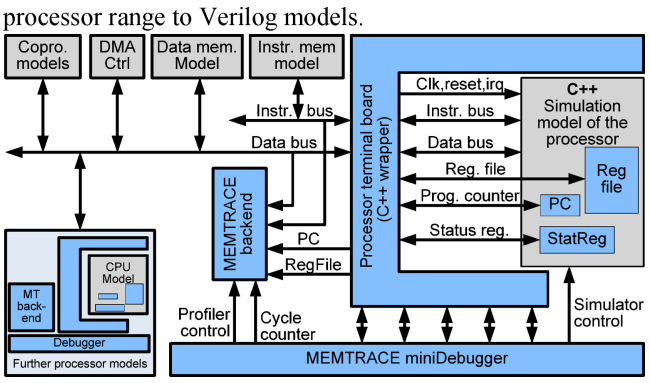

Fig. 4. Interconnection of the C++ processor model with the MEMTRACE backend and miniDebugger and with further system components.

Each time the program counter changes, the ISS calls the interface method nextInstruction(). The method checks if the program execution has changed from one function to another. If so, the cycle count of the previous function is recalculated and the call count of the new function is incremented.

For each access that occurs on the data bus (to the data cache or tightly coupled memory), the memory access counters of the current function are incremented in interface method memoryAccess(). Depending on the information provided by the ISS, it is decided if a load or store access was performed, and which bit-width (8, 16, or 32-bit) was used.

For each bus cycle (on the external memory bus) the method busActivity() identifies the bus status (idle cycle, core access or DMA access) and increments the appropriate counter of the current function.

If a data cache is available in the processor, each time a cache miss occurs the method cacheMiss() is called. It increments the number of data cache misses for the current function and also for the accessed variable.

The interface method finish() is called when the ISS terminates the simulation. It writes the profiling results to the output files.

C. MEMTRACE with a Processor Model Generator

MEMTRACE has been used successfully with the ARMulator ISS. For profiling other processors, cycle-accurate ISSes are required. The toolchain given in Fig. 3 has been developed to create such models from processors described in a high-level description language.

The design procedure starts with a processor description in the LISA language [18]. This description is processed by the CoWare Processor Designer [19] to generate a Verilog description of the processor and the required software development tools, such as an assembler and a linker. The Verilog description generated is further processed by the Verilator [20] to generate a C++ simulation model. This model is then compiled with the MEMTRACE backend and the miniDebugger libraries to form the combined simulator, debugger, and profiler. In order to ease the retargeting process, a generic interface between the simulation model and the debugger is defined. Therefore, the simulation model needs to be enclosed by a wrapper mapping the processor signals to the MEMTRACE backend interfaces functions. Fig. 4 illustrates this interconnection in more detail. The right-hand side shows the C++ simulation model generated by the Verilator. A wrapper, which acts as a terminal board, is used to provide a generic interface to other system and simulation modules. This interface provides access to the most common and important parts of the processor, including the instruction and data busses, register file, program counter and status register, as well as clock, reset and interrupt signals. The mandatory system extension is a model of the data and instruction memory connected to the system bus. Multiprocessor systems can be generated by adding further processor models, including their debuggers and profilers, as shown in Fig. 4 on the left-hand side. Even hierarchical bus systems can be created within this environment. The processor simulation is controlled by a debugger, e.g., the rudimentary MEMTRACE miniDebugger, which could also be replaced by a full-featured debugger, e.g., as a plugin to the Eclipse software development platform. The simulation environment described here allows a simple retargeting of the profiler to any processor that is available as a C source code model. The Verilator extends the supported processor range to Verilog models.

In contrast to the VLIW-SIM [21] tool, the described usage of the LisaTek/Verilator tools ensures a coherency between the hardware description and the simulation model. The VLIW-SIM environment integrates a parameterizable processor simulator with a commercial SoC simulation environment and also provides basic profiling results, such as overall cycles, memory accesses and cache misses, however not on a per function basis.

D. Incorporating Energy Estimation in the Profiler

A power model [22] has been incorporated into the profiler, for estimating the energy consumption of each function. The model is based on power measurement of an ARM processor.

E. Simulation Platform Requirements

The MEMTRACE profiler runs on PC platforms. The profiling speed depends on the ISS used, e.g., with ARMulator it is in the range of a few MIPS on a 3 GHz PC. The influence of the profiler on the simulation performance depends on the profiler features applied, e.g., instruction profiling, and leads to a reduction of the ISS speed by a factor of 1.1 to 3.

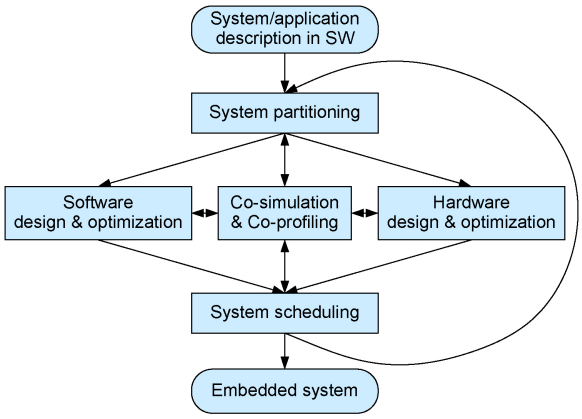

Fig. 5. Typical embedded system design flow.

IV. MEMTRACE Within the Design Flow

This section describes how the profiler can be applied during the design of embedded systems. Fig. 5 shows a typical design flow for such hardware/software systems.

Starting from a functionally verified system description in software, this software is profiled with an initial system specification, in order to measure the performance and see if the (real-time) requirements are met. If not, an iterative cycle of system partitioning, optimization, and scheduling starts. In this process, detailed profiling results are crucial for all steps in the design cycle.

A. System Partitioning and Design Space Exploration

For the definition of a starting point of a system architecture, an initial design space exploration should be performed. These steps include a variation of the following parameters:

1) processor types and quantity;
2) cache size and organization and tightly coupled memory;
3) bus and memory system and timing [dynamic random access memory (DRAM), static random access memory (SRAM)];
4) coprocessors and DMA controller.

MEMTRACE can be run in batch mode and thus different system configurations can be tested and profiled. Thus, the influence of the system architecture on the performance can be evaluated. This initial profiling also reveals the hot-spots of the software. The most time consuming functions are good candidates for either software optimization or offloading to application specific instruction set processors (ASIPs) or hardware coprocessors. Especially computationally intensive functions are well-suited for hardware acceleration in a co-processor. Thus, this information can lead to an initial system partitioning into hardware and software, which can then be profiled in order to evaluate its overall performance.

Control-intensive functions are better suited for software implementation on ASIPs, as a hardware implementation would lead to a complex state machine, which requires long design time and often does not allow parallelization. With support of a DMA controller, even the burden of data transfers can be taken from the processor. In order to get a first idea of the influence of hardware acceleration, a factor (determined by well-educated guess) can be defined for each hardware candidate function. This factor is used by MEMTRACE in order to manipulate the profiling results.

B. Software Profiling and Optimization

After the system is partitioned into several processors and hardware coprocessors, the software parts can be optimized. Numerous techniques exist that can be applied for optimizing software, such as loop unrolling, loop invariant code motion, common subexpression elimination or constant folding and propagation. For computational intensive parts, arithmetic optimizations or SIMD instructions can be applied, if such instructions are available in the processor. If the performance of the code is significantly influenced by memory accesses, as is mainly the case in video applications, the number of accesses has to be reduced or the accesses must be accelerated. The profiler gives a detailed overview of the memory accesses and allows therewith the identification of their influence.

C. Profiling-Based Hardware Optimization

The profiling information can be applied to adjust the processor and memory architecture. In the next section, a method is described for configuring and using fast on-chip cache and memory efficiently. Section II shows how the instruction set and the address generation modes of a processor can be adjusted to the needs of the application.

1) *Memory Subsystem Optimizations:* A well-defined memory subsystem should be developed to minimize the processor stalls caused by accesses to slow memory devices, for example, external DRAM.

Especially for systems with slow memory, caches are mandatory for achieving a reasonable performance. The spatial and temporal locality of memory accesses and instructions found in most applications can be used efficiently with caches. Caches are very costly in terms of die area and power consumption. Therefore, the size of the data and instruction caches should be adjusted to the requirements of the application. The detailed profiling results for cycle times, memory accesses, and cache misses which are delivered by MEMTRACE allow a comprehensive exploration of different cache sizes and their influence on the performance.

In addition to using caches, the system performance can be increased by using fast on-chip memory (SRAM). This memory can be used to store frequently used data for fast access. As SRAM is very costly in die area and power consumption, it is usually small. Therefore, in order to use it efficiently, the frequently accessed memory areas need to be identified, these being adequately valuable candidates for internal storage.

If caches are available in a system, not every load/store access is passed to the slow external memory, but instead only if a cache miss occurs. A cache miss leads to a halt of the processor and increases the execution time. Thus, the number of cache misses must be reduced to speed up the application.

The decision whether a specific data area should be stored in on-chip or off-chip memory is quantified by the number of cache misses due to accessing the data area. Therefore,

an analysis of cache misses per data area (e.g., variable) is required. The cost ratio, given in (1), expresses the ratio between cache misses that occur when accessing a data area and its size

$$\text{cost ratio} = \frac{\text{cache misses}}{\text{data size}}. \quad (1)$$

The data areas with the most cache misses (profit) and the smallest size (weight) should be stored in on-chip memory according to solving algorithms of the knapsack problem [23]. This evaluation can be performed for several SRAM sizes and thus an appropriate memory size can be found.

Numerous other techniques for optimizing the structure and accesses to the memory subsystem are described in [24].

MEMTRACE uses the memory modeling of the ARMulator for the simulation of the memory subsystem. It allows the customization of cache sizes and line length, the bus speed as a factor of the CPU speed, as well as a cycle-accurate timing of the memories. The model simplifies the memory timing by differentiating only between sequential and nonsequential read and write accesses. When using other processor architectures, generic cache simulators, such as Dinero IV [25] and Cheetah [26] need to be integrated.

2) *Instruction Set Architecture Optimizations:* As their name states, RISC processors come with a reduced instruction set. However, some of the current RISC instruction sets provide more than 100 instructions. If the instruction set of the processor is customizable, such as with ARC [27], Tensilica [28] or the CoWare Processor Designer [19], it can be helpful for the processor designer to acquire information about the actual instruction set and addressing mode usage.

An instruction set analysis can be performed by parsing the compiler-generated assembly code. However, this static analysis neglects the real instruction usage during program execution, since not every assembly code line is executed equally often. As many instructions can be replaced by a series of other instructions, it can be helpful to see how often a specific instruction is really used. This is important as the replacement with other instructions often comes with an overhead, and therefore the influence of the overhead can be estimated already by this dynamic profiling. A reduced instruction set helps to minimize the complexity of the instruction decoder. Frequently used instructions should be considered as targets for optimization during the processor architecture development.

Besides the instruction set, the supported addressing modes also influence the complexity of a processor. The addresses are either calculated in a separate address generation unit or within the regular arithmetic logic unit. Depending on the processor, a more or less wide range of addressing modes for code and data is available. These modes include absolute, PC-relative and register-indirect addressing for code access. For data access, numerous modes supporting offsets, shifts, indexes and arithmetical calculations based on immediate and register values exist.

Supporting all these addressing modes has two major impacts on the processor architecture. On one hand, the coding of the modes in the instruction set requires a portion of the instruction bit-width for encoding mode, offset register, shift and immediate value. On the other hand, the required hardware support for calculating the addresses leads to an overhead in die area and power consumption. If a processor is targeted to a specific application, the addressing mode profiling can be used to adapt the architecture to the applications needs.

D. Hardware/Software Profiling

Besides the software profiling and optimization a system simulation including the hardware accelerators needs to be carried out in order to evaluate the overall performance. Usually hardware components are developed in a hardware description language (HDL) and tested with an HDL simulator. This task requires long development and simulation times. Therefore HDL modeling is not suitable for the early design cycles, where exhaustive testing of different design alternatives is important. Furthermore, if the system performance is data dependent a huge set of input data should be tested to get reliable profiling results. Therefore, a simulation and profiling environment is required, which allows short modification and simulation time.

For this purpose, we extended the ISS with simulators for the hardware components of the system. The ARMulator ISS provides an extension interface, which allows the definition of a system bus and peripheral bus components. It provides a bus simulator, which reflects the industry standard advanced microprocessor bus architecture bus and a timing model for access times to memory mapped bus components, such as memories and peripheral modules.

1) *Coprocessors/Hardware Accelerators:* We supplemented this system with a simple template for coprocessors, including local registers and memories and a cycle-accurate timing. The functionality of the coprocessor can be defined as C code. With this methodology the C code acts as a functional simulation model of the hardware. Thus the software function can be simulated as a hardware accelerator by copying the software code to the coprocessor template without translation or redefinition using another description language such as SystemC.

The timing parameter can be used to define the delay of the coprocessor between activation and result availability. The timing value can be achieved either from reference found in literature or by an educated guess of a hardware engineer. The profiling of different implementations of a task can be accomplished by varying the timing parameter and viewing its influence on the overall performance. Thus a good trade-off between hardware cost and speed-up can be found quickly.

In a later design phase, when the hardware/software partitioning is fixed and an appropriate system architecture is found, the hardware component need to be developed in a hardware description language and tested using a HDL simulator, such as Modelsim. Finally, the entire system needs to be verified including hardware and software components. For this purpose the instruction set simulator and the HDL simulator have to be connected. The codesign environment PeaCE [29] allows such a connection with Modelsim.

2) *DMA Controller:* As data transfers can have a tremendous influence on the overall performance, their burden can be

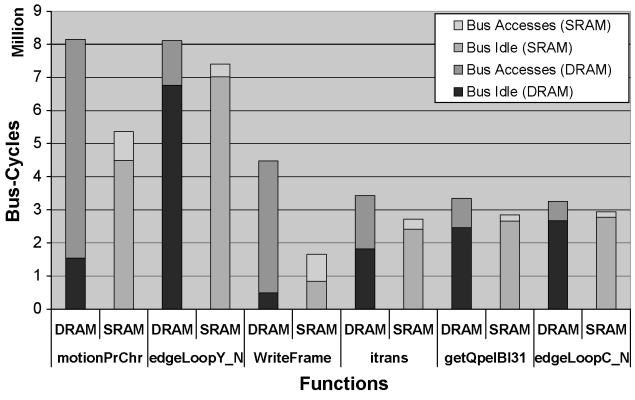

Fig. 6. Bus usage for each function, depending on the memory type.

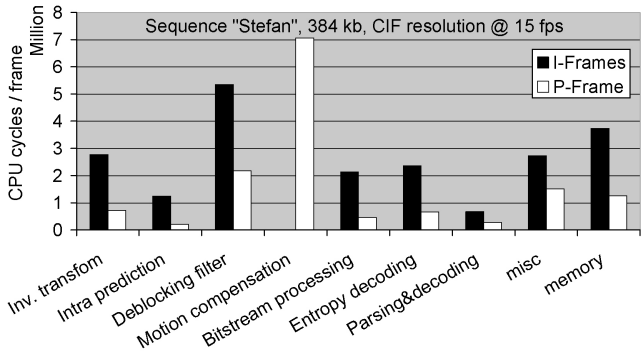

Fig. 7. Profiling results for the H.264/AVC software decoder.

taken from the CPU by means of a DMA controller. Therefore, the MEMTRACE hardware environment includes a DMA controller model. It supports 1-D and 2-D-transfers, burst transfers and multiple channels with activation first in, first outs. Thus, the designer is enabled to determine the influence of different DMA modes in order to find an appropriate trade-off between controller complexity and required CPU activity.

E. Data Transfer Optimizations for System Scheduling

After the software and hardware tasks have been defined a scheduling of these tasks is required. For increasing the overall performance a high degree of parallelization should be accomplished between the different processing units. In order to find an appropriate scheduling for parallel tasks, their dependencies, execution time and data transfer overhead need to be considered.

Concerning the overhead for data transfers to the coprocessors its dependency on the bus usage must be considered. Furthermore, side effects on other functions may occur if bus congestion occurs or when cache flushing is required in order to ensure cache coherency. In order to find these side-effects, detailed profiling of the system performance and the bus usage is necessary. MEMTRACE provides these results; for example Fig. 6 shows the bus usage for each function depending on the access time of the memory.

V. Application Example: SoC Design for Video Signal Processing

The proposed design methodology has been applied to the design of an H.264/AVC [30] video decoder as part of a mobile digital TV receiver. Starting from an executable specification of the video decoder a profiling-based partitioning of the system in processor and coprocessors has been performed. The hardware and software components of the system have been optimized and scheduled for a high degree of parallelization. In another case study, application specific processor architectures have been developed tailored to the needs of video signal processing. Appropriate instruction sets and addressing modes have been defined based on the comprehensive profiling results of the algorithms.

A. H.264/AVC Video Compression

The H.264/AVC video compression standard is similar to its predecessors, however it adds various new coding features and refinements of existing mechanisms, which lead on one hand to a two to three times increased coding efficiency compared to MPEG-2. On the other hand the computational demands and required data transfers have increased significantly.

The bitstream parsing and entropy decoding interpret the encoded symbols and are highly control flow dominated. The inter and intra prediction modes are used to predict image data from previous frames or neighboring blocks, respectively. Both methods require filtering operations, whereas the inter prediction is more computational demanding. The residuals of the prediction are received as transformed and quantized coefficients. The reconstructed image is post processed by a deblocking filter for reducing blocking artifacts. The filter includes the calculation of the filter strength, which is control flow dominated, and the actual filtering, which requires many arithmetic operations.

B. Design and Optimizations

The H.264/AVC baseline decoder has been profiled with MEMTRACE using a system specification typical for mobile embedded systems comprising an ARM946E-S processor core, a data and instruction cache (16 kB each) and an external DRAM as main memory. The execution time for each module of the decoder has been evaluated as depicted in Fig. 7. The results show that the distribution over the modules differs significantly between I and P-frames. Whereas in I-frames the deblocking has the most influence on the overall performance, in P-frames the motion compensation is the dominant part.

Based on the acquired profiling results several software and hardware architectural optimizations are applied.

1) *System Partitioning:* In order to increase the system efficiency and decrease power consumption and hardware costs compared to a single processor implementation, a system with tailored coprocessors can be developed. Following Amdahl's law [31], those parts of the system should be considered for outsourcing and optimization first, which take up most of the execution time. Fig. 7 shows that motion compensation, deblocking, inverse transformation and memory functions are those candidates.

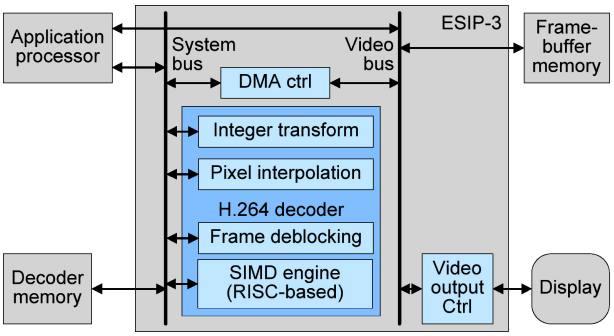

Fig. 8. System layout of the H.264/AVC decoder chip based on the profiling results with a system bus and a separate video bus.

TABLE III
COMPARISON OF THE EXECUTION TIME IN HARDWARE AND SOFTWARE

Implementation	Deblocking	Pixel Interpolation	Inverse Transform
Software	3000–7000 cycles	100–700 cycles	320 cycles
Hardware	232 cycles	16–34 cycles	30 cycles

Memory transfers are not included in cycle counts

All these components are rather demanding on an arithmetical than on a control flow level. Therefore, they are well suited for hardware implementation as coprocessors, which can be controlled by the main CPU. In order to ease the burden of providing the coprocessors with data, a DMA controller can be applied allowing memory transfers concurrently to the processing of the CPU. The coprocessors should be equipped with local memory for storing input and output data for processing at least one macroblock at a time preventing fragmented DMA transfers. As the video data is stored in the memory in a 2-D fashion, the DMA controller should feature 2-D memory transfers.

2) *System Design:* The profiling and implementation results of the previous sections lead to a mixed hardware/software implementation of the video decoder, which is given in Fig. 8.

An application processor is extended with a companion chip for acceleration of the video decoding. The companion chip contains the hardware accelerators for H.264/AVC decoding. Table III shows a comparison of the required cycle times of the accelerators with their software counterparts.

3) *Memory Subsystem Optimization:* Besides the processing power of the system components the memory architecture determines the overall performance. Caches have a huge influence on the performance and are mandatory for most applications. They are especially efficient for data areas with frequent accesses to the same memory location, e.g., the stack. The influence of the cache size needs to be considered during the design of the memory architecture, as described later. However for randomly accessed data areas, e.g., lookup tables, a fast on-chip memory (SRAM) is more appropriate. As the H.264/AVC decoder requires about 1.1 MB of data memory (at QVGA video resolution), only small parts of the used data structures (less than 3% with 32 kB of SRAM) can be stored in the of on-chip memory. In order to find a useful partitioning of data areas between on-chip and off-chip memory, it is required to profile the accesses to each data area of the decoder. Since a data cache is instantiated, accesses to the memory only happen if cache misses occur. Therefore, the cache misses have been analyzed separately for each data area in the code including global variables, heap variables and the stack. Afterward the data partitioning has been performed as described in Section IV-C.

TABLE IV
PROFILING RESULTS FOR REGISTER ALLOCATION OPTIMIZATION: MAXIMUM NUMBER OF MEMORY ACCESSES TO A SINGLE ADDRESS

Function	Calls	Max Accesses	Max Accesses Calls	Speed-Up
flushBits	32 969	7	230 783	5 %
edgeLoopY_N	2376	57	135 432	5 %
itrans	5167	16	82 672	15 %
...	...	...	...	...

4) *Software Optimizations:* The software of the system was optimized by means of standard optimization techniques as mentioned in Section IV. In order to prove their influence on the performance to ensure the negative correlation between source code modifications, a comprehensive profiling has been performed during the development process.

Besides these optimization techniques, some specialized memory-centric optimizations have been developed and applied, for example an optimized register allocation. As memory accesses are very time consuming, frequently accessed variables should be kept in registers if possible, as described in [32]. Compiler may allocate registers inefficiently if global variables, pointers or pointer chains are used. As an indicator for inefficient register allocation, the maximum number of memory accesses to a single memory address has been analyzed for each function. Multiplying this number with the number of calls of the function provides an indicator for the influence of these accesses on the overall performance, as given in Table IV.

5) *Hardware/Software Interconnection and Scheduling:* After the software optimization is performed and the coprocessors are implemented, a scheduling of the entire system is required. The scheduling is static and controlled by the software. The coprocessors are introduced step-by-step to the system. Starting from the pure software implementation, at first software functions are replaced by their coprocessor counterparts. This also requires the transfer of input data to and output data from the coprocessors. These transfers are at first executed by load-store operations of the processor and in a next step replaced by DMA transfers. This might also require flushing the cache or cache lines, which may decrease the performance of other software parts. Finally, parallelization of the coprocessor and software tasks takes place. All decision taken in these steps are based on detailed profiling results.

The following example, results are given in Fig. 9, shows how the hardware accelerator for the deblocking is inserted into the software decoder.

The coprocessor only includes the filtering process of the deblocking stage; filter strength calculation is performed in

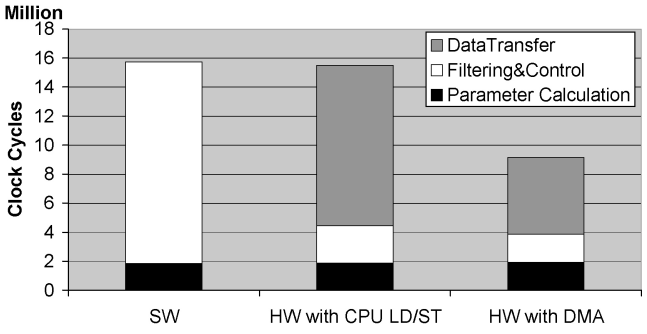

Fig. 9. Clock cycle comparison of different deblocking implementations.

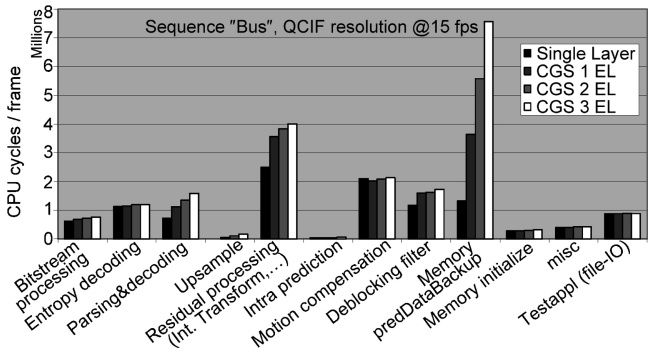

Fig. 10. Profiling results for the SVC decoder: clock cycles per frame for each software component of the SVC decoder for quality scalability.

software, because it is rather control intensive and therefore more suitable for software implementation. The filter processes the luminance and chrominance data for one macroblock at a time. It requires the pixel data and filter parameters as an input and provides filtered image data as an output. Fig. 9 shows the results for the pure software implementation, when using the filter coprocessor with data transfer managed by the processor, and when additionally using the DMA controller.

As can be seen, if data is transferred by the processor, the performance gain of the coprocessor is dissipated by the data transfers; only in conjunction with the DMA controller the coprocessor can be used efficiently.

6) *Implementation:* To fully evaluate the proposed concept, the complete SoC architecture has been implemented as an application-specific integrated circuit design [33] using UMC's L180 1P6M GII logic technology.

C. Profiling for Design Reuse (Scalable Video Coding (SVC) Decoding Example)

In a second step the profiling methodology is applied for evaluating if the before mentioned SoC design can be reused for an efficient implementation of the new video coding standard SVC [34]. SVC is the scalable extension to H.264/AVC based on a layered approach representing different modes of scalability in a single bitstream. The scalability modes provided by the codec are temporal, spatial and quality (SNR) scalability and any possible combination of these three base modes. The base layer found in the bitstream is fully compatible to H.264/AVC. A more detailed description of SVC can be found in [35].

The real-time performance of the decoder has been analyzed with MEMTRACE. Fig. 10 shows the results of a profiling run, given in numbers of bus clock cycles differentiated by the various function groups in the decoder for quality scalability. We performed a profiling of the SVC software and compared the decoding of four different bitstreams with similar bit rates. The first stream (single layer) has only a H.264/AVC compliant base layer, without any quality enhancement layers. Furthermore, three bitstreams with one to three enhancement layers (CGS 1 EL to CGS 3 EL) were applied.

As can be seen in Fig. 10, except for inter-layer prediction data backup (Memory predDataBackup), the number of clock cycles for most function groups show only a small increase for additional quality layers. The reason for this result is that the inter-layer prediction signal needs to be updated for each additional layer separately, whilst many of the other functions like deblocking, motion compensation and inverse transformation are only performed once at the reconstruction layer.

As can be seen in Fig. 10, base layer (H.264/AVC) and enhancement layer (SVC) decoding show similar hot spots and results. This is due to the fact that SVC uses the same coding tools as H.264/AVC. Thus the hardware accelerators used for H.264/AVC are perfectly suited to accelerate the SVC decoder as well. For the motion-compensated prediction and the integer transformation, no changes need to be applied. For the deblocking, a modified filter strength value computation is proposed by the SVC standard. Anyway, this is no problem for the current hardware implementation, because these filter strength values are always computed in software and transmitted as configuration values to the hardware deblocking filter. Nevertheless, we expect to have a lower overall performance due to the inter-layer dependencies which raise the amount of memory accesses. Also, for spatial scalability accelerating the upsampling process is mandatory.

The major difference between SVC and H.264/AVC is located in the control flow. Therefore, in order to reuse an H.264/AVC implementation for SVC, the control flow needs to be adapted, which can be done easily, if the control flow is implemented in software. This shows that the proposed hardware/software architecture is very well suited for the video coding domain, as modifications and extensions to codec standard are very common.

D. Application Specific Processor Architecture Exploration

Taking an optimized system architecture into account the next possible candidate for an optimization is the used processor core itself. In most cases the processor core is fixed in its architecture and instruction set, e.g., the ARM946E-S, which is only available as a predefined IP core. Using a configurable processor core, e.g., ARC or Tensilica, it is possible to build the system upon an application specific optimized processor core. In the before mentioned example the commercially available ARC 610 [36] has been chosen due to its configurability of the instruction set and the number of additional registers. The ARC 610 has a 32-bit load/store architecture with 16 to 32 32-bit registers and a

TABLE V
EXTENSION INSTRUCTIONS AND THEIR PERFORMANCE GAIN

	Area (Comp. to ARC)	No. of CE81 Cells	Gate Count	Min. Performance Gain	Max. Performance Gain
ARC with XALU	100 %	1 808 120	45 203		
Bitstream processing	25.3 %	456 560	11 414	1.8 %	4.4 %
ALIGN	0.5 %	9480	237		
INTERPOLATE	2 %	35 360	884	5.7 %	30 %
ADD CLIP	1.9 %	34 360	859	1.8 %	7 %
IDCT	94.8 %	1 713 760	42 844	30 %	35 %

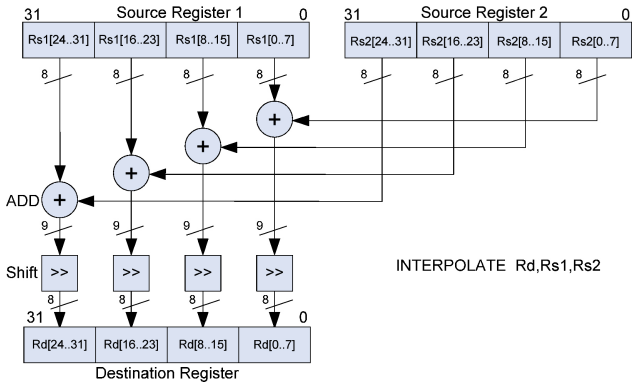

Fig. 11. SIMD extension instruction INTERPOLATE.

TABLE VI
INSTRUCTION PROFILING RESULTS FOR DECODING TWO FRAMES OF THE SEQUENCE ''STEFAN 256 KB'' (CIF RESOLUTION)

Instr.	Executed	As % of All Exe. Instr.	Skipped	As % of Decoded Instr.
LDR	4 984 793	27.85 %	70 353	1.39 %
ADD	3 717 539	20.77 %	123 891	3.23 %
MOV	1 917 986	10.72 %	192 936	9.14 %
STR	1 752 056	9.79 %	47 106	2.62 %
SUB	1 365 637	7.63 %	13 703	0.99 %
CMP	976 494	5.46 %	16 756	1.69 %
B	642 921	3.59 %	443 651	40.83 %
...	...	...	...	...
sum	17 896 454	100 %	1 158 402	6.08 %

set of basic instructions, which can be found in similar RISC processors.

Additionally it is possible to extend the given instruction set by means of instruction slots. These instruction slots can be filled with application specific instructions e.g., bitstream extraction or SIMD like instructions. Fig. 11 shows an example of a SIMD extension.

The shown SIMD instruction can be applied for the interpolation of four pixels in parallel as used by standard video compression algorithms in the motion compensation.

Table V gives an overview of a number of user-defined instructions, their performance gain in respect to the overall performance of the application and the gate count of the data path part of the instruction [37].

To evaluate the performance of an implemented extension instruction it is needed to profile a base system which runs the pure software implementation of the given algorithm. Using the profiler each instruction can be profiled for itself and the performance gain of the optimized function using a specialized instruction can be evaluated.

1) *Instruction Set Architecture Definition:* The definition of instruction set architectures suited for application specific tasks in heterogeneous multiprocessor systems often starts with a base instruction set, which will be modified, extended and/or reduced according to the needs of the specific application. As described in Section III, the results for each source code function of the application profiling, the profiler delivers information concerning the instruction set usage. In this example we have implemented a basic 32-bit RISC processor core named embedded systems group RISC suited for application specific tasks in the image processing domain especially for H.264/AVC. The design of the core is based on the definition of a generic instruction set architecture, which has been implemented using LISA for the first evaluations. Based on this LISA description a simulation model of the processor has been derived as described in Section III-C.

Table VI shows instruction profiling results as provided by the profiler for the execution of an H.264/AVC decoder. The source code of the decoder, which includes more than 20 000 lines of code, is translated to a usage of only 23 assembly instructions. Thus, an applications specific processor design with only these instructions would be sufficient to execute the code. Furthermore it can be seen that five instructions (LDR, ADD, MOV, STR, and SUB) are responsible for more than 75% of the decoded instructions. So, the processor architecture, including the instruction set and decoder, pipeline and memory interface should be designed such that these instructions require a low latency.

In order to test the suitability of this processor for other video coding applications, an H.264/AVC encoder, the SVC decoder and a system for gesture and facial characteristics recognition have also been profiled. The video encoder and decoders show a very similar instruction profile, whereas the recognition system utilizes a different instruction set. There, the instruction set is dominated by control flow and logical instructions, the five top-most instructions are MOV, ADD, B, ORR and CMP, which cover 53% of the decoded instructions.

2) *Choosing Appropriate Addressing Modes:* Table VII shows the results for addressing mode profiling. For each of the load and store operations one of the addressing modes is used, either with no offset at all, a program counter relative offset or a pre or post-indexed offset. These offsets can be either an immediate value or taken from a register value. Furthermore, the register value can be shifted by a given value and a specific shift operation.

TABLE VII
ADDRESS MODE PROFILING RESULTS FOR DECODING TWO FRAMES OF THE SEQUENCE ''STEFAN 256 KB'' (CIF RESOLUTION)

Details on Load and Store Operations	
Loads	5 055 146
Stores	1 799 162
Address Type	
Zero-offset	1 205 709
Program counter-relative	324 642
Pre-indexed	4 974 457
Post-indexed	349 500
Detail on Indexed Modes	
Immediate offset	3 340 911
Register offset	1 983 046

As can be seen, here most of the memory accesses are to pre-indexed addresses with an immediate offset value. Post-indexed and program counter-relative addressing is only used for 5% to 6% of memory accesses. It could be considered to abandon these addressing modes for data memory accesses.

VI. CONCLUSION AND FUTURE WORK

The design of an efficient system for applications with high demands on real-time performance requires the selection of an appropriate system architecture and incorporated hardware and software components. For this decision, detailed knowledge of the computational demands of the application is mandatory. Furthermore, for data intensive applications, the influence of memory accesses also has to be taken into account. We have presented a profiling tool which provides this information and have shown how it can be integrated in the design flow. The tool aids the designer in taking the right decision during each step of the design, including the system partitioning, the optimization of the components, and the system scheduling. We have applied this methodology for the development of an SoC for video decoding and for the definition and implementation of application specific processor architectures.

Future work includes the co-exploration of programming models suited for many-core systems, e.g., component-based, and their corresponding multiprocessor system-on-chip architectures. Additionally, the energy estimation metrics will be extended in order to strengthen the design space exploration process.

REFERENCES

[1] ST Microelectronics: Nomadik STn8820 Mobile Multimedia Application Processor (2008, Feb.). Data brief. [Online]. Available: www.st.com

[2] Broadcom: BCM2820 Low Power, High Performance Application Processor (2006, Sep.). Product brief. [Online]. Available: www.broadcom.com

[3] G. de Micheli and L. Benini, *Network on Chips*. San Francisco, CA: Morgan Kaufmann, 2006.

[4] S. Pashira and N. Dutt, *On-Chip Communication Architectures: System on Chip Interconnect*. San Francisco, CA: Morgan Kaufmann, 2008.

[5] M. Ravasi and M. Mattavelli, "High-abstraction level complexity analysis and memory architecture simulations of multimedia algorithms," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 5, pp. 673–684, May 2005.

[6] J. Bormans, K. Denolf, S. Wuytack, L. Nachtergaele, and I. Bolsens, "Integrating system-level low power methodologies into a real-life design flow," in *Proc. 9th Int. Workshop Power Timing Modeling Optimization Simulation*, Kos Island, Greece, 1999, pp. 19–28.

[7] J. W. Janneck, I. D. Miller, and D. B. Parlour, "Profiling dataflow programs," in *Proc. IEEE Int. Conf. Multimedia Expo*, 2008, pp. 1065–1068.

[8] C. Gremzow, "Quantitative global dataflow analysis on virtual instruction set simulators for hardware/software co-design," in *Proc. 26th IEEE Int. Conf. Comput. Design*, Lake Tahoe, CA, Oct. 2008, pp. 377–383.

[9] H.-J. Stolberg, M. Berekovic, and P. Pirsch, "A platform-independent methodology for performance estimation of streaming media applications," in *Proc. IEEE Int. Conf. Multimedia Expo (ICME)*, vol. 2. 2002, pp. 105–108.

[10] S. L. Graham, P. B. Kessler, and M. K. McKusick, "Gprof: A call graph execution profiler," in *Proc. Special Interest Group Programming Languages Symp. Compiler Construction*, Boston, MA, 1982, pp. 120–126.

[11] Intel Inc. Intel VTune performance analyzers. [Online]. Available: http://www.intel.com/software/products/vtune

[12] ARC International (2004). Integrated Profiler User's Guide.

[13] ARM Ltd. ARM RealView Profiler. [Online]. Available: http://www.arm.com/products/DevTools/RVP.html

[14] CoWare Inc. (2006). LISATek Processor Debugger Manual.

[15] P. M. Kuhn and W. Stechele, "Complexity analysis of the emerging MPEG-4 standard as a basis for VLSI implementation," in *Proc. Soc. Photo-Optical Instrum. Eng. Visual Commun. Image Process. (VCIP)*, 1998, pp. 498–509.

[16] H. Hübert, "MEMTRACE: A memory, performance and energy profiler targeting RISC-based embedded systems for data-intensive applications," Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci., Tech. Univ. Berlin, Germany, 2009. [Online]. Available: http://opus.kobv.de/tuberlin/volltexte/2009/2261

[17] ARM Ltd., Cambridge, U.K. (2004). RealView ARMulator ISS Version 1.4 User Guide (DUI 0207C).

[18] S. Pees, A. Hoffmann, V. Zivojnovic, and H. Meyr, "LISA: Machine description language for cycle-accurate models of programmable DSP architectures," in *Proc. 36th ACM/IEEE Design Automation Conf. (DAC)*, 1999, pp. 933–938.

[19] CoWare Inc. Processor designer. [Online]. Available: http://www.coware.com/products/processordesigner.php

[20] W. Snyder, P. Wasson, and D. Galbi (2008). *Introduction to Verilator* [Online]. Available: http://www.veripool.com/verilator.html

[21] I. Barbieri, M. Bariani, A. Cabitto, and M. Raggio, "A simulation and exploration technology for multimedia-application-driven architectures," *J. VLSI Signal Process.*, vol. 41, no. 2, pp. 153–168, Sep. 2005.

[22] H. Hübert and B. Stabernack, "Power modeling of an embedded RISC core for function-accurate energy profiling," in *Proc. 12th Informationstechnische Gesellschaft (ITG) Workshop Methoden Beschreibungssprachen Modellierung Verifikation Schaltungen Syst.*, Berlin, Mar. 2009, pp. 147–156.

[23] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. Hoboken, NJ: Wiley, 1990.

[24] L. Wehmeyer and P. Marwedel, *Fast, Efficient and Predictable Memory Accesses: Optimization Algorithms for Memory Architecture Aware Compilation*. Dordrecht, The Netherlands: Springer, 2006.

[25] J. Edler and M. D. Hill. *Dinero IV: Trace-Driven Uniprocessor Cache Simulator* [Online]. Available: http://pages.cs.wisc.edu/~markhill/DineroIV

[26] R. A. Sugumar and S. G. Abraham, "Efficient simulation of caches under optimal replacement with applications to miss characterization," in *Proc. ACM SIGMETRICS Conf. Measurement Modeling Comput. Syst.*, 1993, pp. 24–35.

[27] ARC International. ARC website. [Online]. Available: http://www.arc.com

[28] R. E. Gonzalez, "Xtensa: A configurable and extensible processor," *IEEE Micro*, vol. 20, no. 2, pp. 60–70, Mar.–Apr. 2000.

[29] S. Ha, C. Lee, Y. Yi, S. Kwon, and Y.-P. Joo, "Hardware-software codesign of multimedia embedded systems: The PeaCE approach," in *Proc. 12th IEEE Int. Conf. Embedded Real-Time Computing Syst. Appl.*, vol. 1. Sydney, Australia, Aug. 2006, pp. 207–214.

[30] *International Standard of Joint Video Specification (ITU-T Rec. H.264—ISO/IEC 14496-10 AVC)*, document JVT-G050.doc, Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T, VCEG, Mar. 2003.

[31] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proc. Am. Federation Inf. Process. Soc. Spring Joint Comput. Conf.*, 1967, pp. 483–485.

[32] ARM Ltd. (1998, Jan.). Writing efficient C for ARM, Ref: DAI 0034A. [Online]. Available: http://www.arm.com

[33] B. Stabernack, H. Hübert, and K.-I. Wels, "A system on a chip architecture of an H.264/AVC coprocessor for DVB-H and DMB applications," *IEEE Trans. Consumer Electron.*, vol. 53, no. 4, pp. 1529–1536, May 2007.
[34] *Advanced Video Coding for Generic Audiovisual Services*, ITU-T Rec. H.264 and ISO/IEC 14496-10, Version 8, ITU-T and ISO/IEC JTC 1, Nov. 2007.
[35] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of H.264/AVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 9, pp. 1103–1120, Sep. 2007.
[36] ARC International. ARC 610D core. [Online]. Available: http://www.arc.com/upload/download/arc_610D_core.pdf
[37] M. Berekovic, H.-J. Stolberg, M. B. Kulaczewski, P. Pirsch, H. Möller, H. Runge, J. Kneip, and B. Stabernack, "Instruction set extensions for MPEG-4 video," *J. VLSI Signal Process. Syst.*, vol. 23, no. 1, pp. 27–50, Oct. 1999.

Heiko Hübert received the Diploma and Dr. Ing. degrees in electrical engineering from the Technical University of Berlin, Berlin, Germany in 2000 and 2009, respectively.

He has held the position of a Visiting Researcher at Stanford University, Stanford, CA, at the Royal Institute of Technology, Stockholm, Sweden, and at the Rheinisch-Westfälische Technische Hochschule, Aachen, Germany. In 1998, he worked as an Intern for Ericsson, Stockholm, Sweden. He has been with the Department of Image Processing, Fraunhofer Institute for Telecommunications, Heinrich-Hertz-Institut, Berlin, Germany, since 2002. He has been working for several years on developing tools for different aspects of the design flow for hardware/software systems. His research interests include the analysis and profiling of embedded systems, as well as the design of software and hardware for multimedia applications.

Benno Stabernack received the Diploma and Dr. Ing. degrees in electrical engineering from the Technical University of Berlin, Berlin, Germany in 1996 and 2004, respectively.

In 1996, he joined the Department of Image Processing, Fraunhofer Institute for Telecommunications, Heinrich-Hertz-Institut, Berlin, Germany. Here, as the Head of the Embedded Systems Group of the Department of Image Processing, he is responsible for research projects focused on hardware and software architectures for image processing algorithms. Since summer 2005, he has lectured on the design of application-specific processors at the Technical University of Berlin, Berlin, Germany. His current research interests include VLSI architectures for video signal processing, processor architectures for embedded media signal processing, and SoC designs.