Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia), Sept. 6-7, Stockholm, Sweden.
http://www.ece.cmu.edu/~sld/ESTIMedia/

Application Design Trajectory towards Reusable Coprocessors MPEG Case Study

Martijn Rutten, Om Prakash Gangwal, Jos van Eijndhoven
Philips Research Laboratories
Prof. Holstlaan 4, 5656AA Eindhoven
The Netherlands
martijn.rutten@philips.com

Egbert Jaspers
LogicaCMG
Kennedyplein 248,
5611ZT Eindhoven
The Netherlands

Evert-Jan Pol
Philips Semiconductors
Prof. Holstlaan 4, 5656AA
Eindhoven
The Netherlands

Abstract

This paper presents a structured application design trajectory to transform media-processing applications—modeled as Kahn process network—into a set of function-specific hardware units called coprocessors. The proposed design trajectory focuses on identifying hardware-implementable computation kernels that are common for a predetermined set of applications. The design trajectory is exercised in a case study that maps MPEG video decoding and encoding applications onto a set of coprocessors in a heterogeneous multiprocessor architecture. The resulting set of coprocessors can simultaneously perform both encoding and decoding functions for multiple MPEG-2 streams in an estimated 4 mm^2 (excluding memory) in 0.18μ technology

1. Introduction

Emerging consumer-electronics products need to support a mix of media processing applications, such as the combination of MPEG encoding and decoding in a set-top box with time-shift functionality. State-of-the-art media processing systems-on-a-chip (SoC) exploit the performance density of hardwired function modules to implement critical parts of these targeted applications [10]. Currently, such function modules are application specific, e.g., an MPEG-2 decoding module only decodes MPEG-2 streams and cannot be reused for MPEG-4 decoding or MPEG-2 encoding applications. This restricts the flexibility of the system to cope with evolving application standards or variations of the required application set within a product family.

MPEG-2 and MPEG-4 applications contain computation kernels such as the discrete cosine transform (DCT) that are generic for the media-processing application domain. However, current SoC designs need to implement multiple instances of such potentially reusable computation kernels, since the kernels form an integral—and therefore inaccessible—part of each function module. To increase the flexibility of mapping applications onto SoC function modules, we advocate lowering the function granularity from the coarse application level to the medium-grain level of individual computation kernels. The resulting computation kernels are typically implemented as function-specific hardware blocks, or *coprocessors*. A flexible network of such coprocessors constitutes the new function module. Hence, such a function module comprises a second hierarchical level of inter-processor communication. Deployment of medium granularity coprocessor hardware in a function module enables three types of potential reuse:

1. Reuse of coprocessor hardware in a function module over different applications by time-shared execution. A typical example is the combined execution of MPEG-2 encoding and decoding by a single function module.
2. Reuse of coprocessor hardware in a function module for new applications that were unknown when the coprocessors were created. Defining coprocessors for this kind of reuse is typically only possible by including at least some programmability in the coprocessors.
3. Reuse of existing coprocessor hardware in the design of a new function module. An example is the deployment of MPEG coprocessors in a 3D graphics module for scaling and motion compensation.

In this paper, we exploit the first form of reuse, where the desired set of applications is known up-front. Nevertheless, the medium function granularity allows us to define coprocessors that will often be generic for the application domain and form good candidates for the second and third forms of reuse. In case of the targeted run-time reuse over applications of the same hardware, our area estimates show a significant increase in efficiency over monolithic designs. Moreover, the efficiency per application of such multi-coprocessor function modules is not significantly lower when compared to monolithic designs. Furthermore, while the finer grain of the design slightly increases latency and power consumption, it does not adversely affect throughput.

2. Related work

Clearly, choosing a fine function granularity for a co-processor (e.g. to compute a sum of absolute differences in MPEG motion estimation) facilitates reuse. However, the overhead of the infrastructure (e.g. to transport and synchronize data access) becomes relatively large. Therefore, fine-grain hardware acceleration is often embedded in RISC or VLIW cores in the form of complex function units [1]. However, the overall performance increase and reduction in power consumption with such compile-time scheduled function units is limited. A coarse-grain co-processor such as an MPEG-2 decoder [10] may incur significantly less overhead, but is difficult if not impossible to reuse over a set of similar applications.

Balancing coprocessor reuse versus overhead incurred in a generic infrastructure demands a thorough understanding of the application domain and architecture at hand. Oftentimes, application information is only available as sequential, complex and ill-documented C-code, e.g., obtained as reference implementation from a standardization body. Extracting reusable computation kernels eligible for coprocessor implementation requires extensive restructuring of the application. This software development trajectory of mapping an application onto a hardware architecture requires a considerable amount of software engineering effort and calls for a structured approach.

We adopt Kahn process networks [6][7] as the model of computation. Ptolemy [3] provides an integrated framework for specification and analysis of, among others, applications modeled as Kahn process network. Modeling languages such as SystemC [12] allow a gradual transformation from sequential code and Kahn-style models towards a hardware implementation. However, how to do this transformation process remains implicit.

This paper proposes a structured approach to gradually transform application C-code into reusable coprocessors. Hereto, the design trajectory in Section 3 focuses on extracting hardware-implementable computation kernels that are common for a set of applications. Section 4 exemplifies the transformation process in a case study for time-shared MPEG-2 decoding and encoding. Finally, Section 5 outlines the initial results.

3. Proposed design trajectory

Figure 1 gives the trajectory that is followed to gradually transform a set of applications written in a high-level programming language (such as C) into behavioral models of the coprocessor hardware. Oftentimes, particular applications are transformed more than once to different hardware architectures over time. During this software-mapping trajectory, the code is restructured significantly on behalf of optimization. Optimized code has little or no reusability over different hardware platforms. Therefore, our software development trajectory is designed to promote reuse at various levels of optimization. The following sections detail the transformation stages indicated in Figure 1.

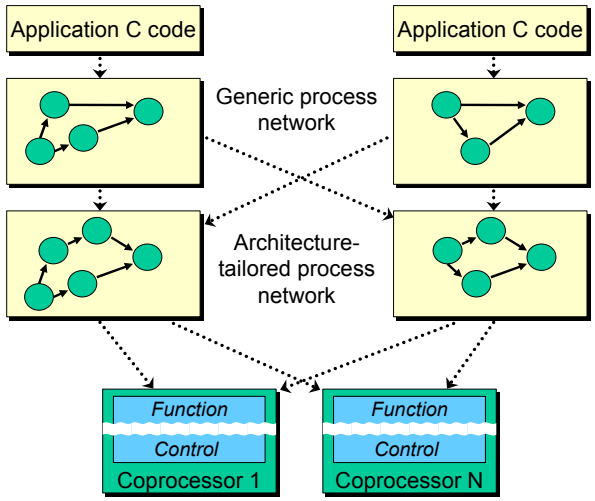

Figure 1 Proposed design trajectory.

3.1. Generic process network stage

The first step is making parallelism and communication explicit in the application. The application code is transformed into a generic process network of an application, written in YAPI [7]. A YAPI application is specified in C/C++ as a set of concurrent processes that communicate through buffered communication channels, similar to a Kahn Process Network [6]. For communication, each process calls read and write primitives on their input and output ports that connect to the communication channels. With the read primitive, the process consumes data in FIFO order from the channel or blocks when no data is available. The process produces data into the channel through the write primitive. This style of structuring applications fits well with the streaming and data-dependent nature of the targeted media-processing applications.

The target hardware architecture influences the choice of the algorithm and its partitioning into functional entities. However, a 'generic' model of an application is independent of other applications that may run on the target hardware, and has minimal dependency on the target hardware architecture. Once the generic model is available, this becomes the foundation for any further application development. This is similar to having 'reference C-code' as a basis for optimizing code towards a programmable platform. Moreover, the step of transforming an application into generic YAPI provides the insight into the structure and behavior of the application that is needed for mapping the application onto the architecture at hand. The generic YAPI model with its clean structure of paral-

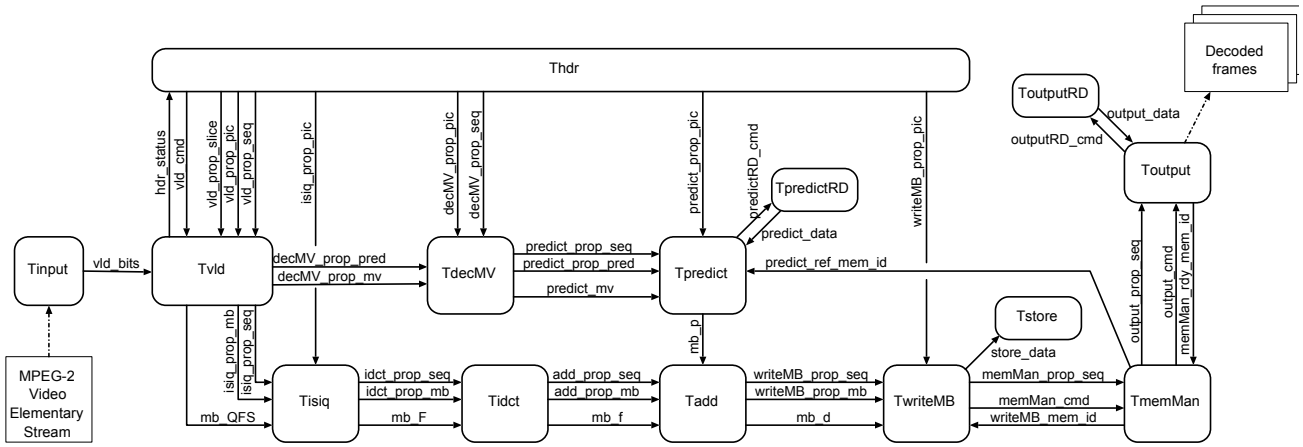

Figure 2. Generic MPEG-2 decoder model.

lel tasks and explicit communication can be reused over different architecture-definition projects. For instance, the MPEG-2 decoder model of Figure 2 has been used as starting point in very different settings: design space exploration of a dedicated MPEG-2 accelerator [13], data-parallel execution on a multi-DSP architecture [4], and the definition of the instruction set of a prototype 64-bit TriMedia processor [5].

3.2. Architecture-tailored process network stage

The second transformation step towards a hardware implementation contains the rewriting of the generic YAPI model into the architecture-tailored process network of the application. Here, the generic YAPI model is restructured such that subsequently it can be mapped onto the target architecture. Note that when an application is later mapped onto other architectures, the generic YAPI is again restructured to match the characteristics of the architecture at hand.

A key aspect of the transformation is that several applications are restructured concurrently to make the generic computation kernels explicit. These computation kernels form the candidates for coprocessor implementation. This may involve changing the actual application algorithms to extract common application kernels, as illustrated in Section 4.2.

3.3. Coprocessor design stage

The architecture-tailored model forms the starting point for hardware development. To aid the process of modeling the target hardware blocks, the YAPI code is separated into two parts: a control part with the YAPI read and write commands, and a part with bare functions. The bare functions provide the functionality of the target hardware, stripped from communication-dependent control aspects. The bare functions (e.g. the computation of a DCT) are reused as is in the hardware definition, while the control part must be rewritten to match the specific communication architecture of the target.

In our case study, communication channels between application processes are mapped onto data streams with communication buffers allocated in shared on-chip memory. The coprocessor designer needs to explicitly decide on the granularity of synchronizing access to these buffers when rewriting the YAPI read and write primitives into the communication primitives offered by the target architecture [11].

The design of coprocessor control includes the design of the data streams. For the architecture to be flexible, tasks running on the function-specific coprocessors have to support being connected into various graphs. The possibilities for reconnection increase dramatically if all streams are based on a uniform syntax that all coprocessors understand. A packet formatting of the data streams provides a common way to handle (dynamically) variable-size data chunks.

We designed the data streams as a stream of variable-length packets, consisting of a 2-byte header field next to a 1 to 255-byte payload. The header field gives the size of the payload and indicates the packet type. Using the header information, the coprocessor can decide to process the packet as basic media data, interpret it as meta-data and update the task state, or forward the packet to the subsequent coprocessor in the application pipeline. With such a packet formatting, the coprocessor designer may choose to interleave packets from different application streams in a single stream to reduce buffer-memory requirements and synchronization overhead.

4. MPEG-2 analysis and transformation

This section presents a case study for the design trajectory of Section 3. The following subsections show the transformation steps from generic YAPI models to behavioral coprocessor models for two applications: MPEG-2 video decoding and encoding.

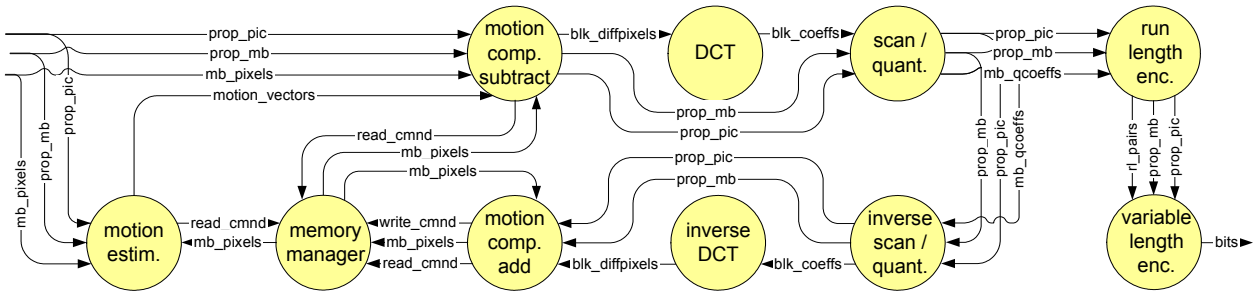

Figure 3. Generic MPEG-2 encoder model.

4.1. Generic decoder and encoder models

Figure 2 depicts the generic MPEG-2 decoder YAPI model [13], derived from C-software of the UC Berkeley MPEG decoder. The creation of this generic YAPI model involved an extensive restructuring of the functions and data structures in the sequential C-code to make parallelism and dependencies explicit.

The process Tvld parses an MPEG bitstream under control of a process Thdr. The Thdr process distributes the retrieved sequence and picture properties to other processes. The Tvld process sends motion vectors into a functional pipeline (TdecMV, Tpredict) that retrieves the prediction data for the reconstruction of macroblocks. The coefficient data for the error blocks is sent into a second functional pipeline for run-length decoding, inverse scan, inverse quantization, and inverse DCT (Tisiq, Tidct). The grain size of this coefficient data is a macroblock. A memory manager process Tmemman controls the access to the frame memories.

Figure 3 depicts the generic MPEG-2 encoder model, derived from reference C-code of Philips' EMPRESS encoder [2] The depicted encoder accepts raw picture data as input. The difference of the input data with motion compensated predication data is sent through a functional pipeline of forward DCT, quantization, run-length encoding, and finally variable-length encoding. A second functional pipeline performs inverse quantization, inverse DCT, and motion compensation to create the predication data for the next input. Both pipelines operate at a macroblock granularity. The memory manager is responsible for synchronizing access to the frame memories.

4.2. Architecture-tailored decoder and encoder models

The choice of coprocessors is influenced by the targeted performance of the MPEG-2 decoding and encoding applications. The described function module targets concurrent decoding of two high-definition (HD) MPEG-2 bitstreams. Moreover, the coprocessors should also be capable of simultaneous encoding and decoding standard definition (SD) MPEG-2 streams.

A typical hardware-software codesign problem is the trade-off between load on the control processor (CPU) and the complexity of coprocessor control as implemented in its hardware. The load on the control processor for coprocessors operating on a macroblock or sequence granularity is clearly too large for simultaneous decoding of two (worst-case) HD sequences. Therefore, the coprocessors are designed to run independently for at least an entire MPEG frame. This decision implies that macroblock and slice properties must be provided to the coprocessors through regular input streams. Picture and sequence properties may be provided through separate auxiliary input streams, thereby removing the dependency on the Thdr process of Figure 2.

For the DCT, scan, and quantization, both the forward and inverse functions can be implemented with exactly the same hardware, only requiring different constants and quantization tables. For the DCT function to be reusable in the decoder as well as in the forward and inverse path of the encoder, it needs to be implemented stand-alone. However, run-length, scan, and quantization may be combined into a single coprocessor to keep the communication between these functions local in the coprocessor.

The MPEG decoder of Figure 2 implements the sequence of run-length decoding, inverse scan, and inverse quantization. Since run-length encoding is lossless, and the silicon area for run-length encoding/decoding is very small, run-length encoding in the forward path of Figure 3 can be done before forking off the inverse path. The encoder model needs to be restructured to implement this behavior. However, this new structure significantly reduces communication buffer requirements of the encoder as the stream buffer between the forward and inverse path remains in the compressed domain.

4.3. MPEG coprocessor design

This section describes the internal architecture of the coprocessors for MPEG decoding and encoding. Based on the analysis in Section 4.2, we decided upon a 4-coprocessor model: variable-length decode (VLD), run-length, scan and quantization (RLSQ), discrete cosine transform (DCT), and motion compensation/estimation (MC/ME). The RLSQ, DCT, and MC/ME coprocessors

should be effective for both MPEG-2 decoding and encoding. Variable-length encoding (VLE) for MPEG-2 encoding is to be implemented in software.

Variable-length decoding (VLD)

The VLD parses an MPEG-encoded video bitstream. From this bitstream it extracts both the basic video data to be passed to the RLSQ unit, as well as control information contained in macroblock and picture headers, such as quantization parameters. All this information is decompressed and sent into the appropriate stream buffers. This control information will change some of the internal state of the coprocessors later in the pipeline. Sending this information as packets through the buffered streams is an important prerequisite for the concurrent (pipelined) operation of all MPEG coprocessors.

The application throughput requirement for dual stream high-definition decoding requires dedicated hardware to achieve sufficient speed in the innermost loops of the decoding process. However, to handle the less frequent (control) data around that, the parsing and decompression can be handled in software. If the hardware for the innermost loops is equipped with programmable tables, then the combination with software processing of the outer parts allows for multi-standard decoding, ranging from MPEG(-1, -2, -4) to DV. Therefore, we map the parsing process Thdr to a small, dedicated RISC core, and compression process Tvld to dedicated VLD hardware.

The interfaces between the RISC core and the VLD engine are such that the VLD engine replies on commands from the RISC core. Such commands are like: 'provide me with the initial bits currently heading the bitstream', 'advance the bitstream by n bits', 'send these provided bits into the output stream', and 'decode the block-data in the bitstream to the output and reply with a ready-code'.

Run-length, scan, and quantization (RLSQ)

The RLSQ coprocessor combines run-length decoding and encoding, zigzag scanning, and quantization functions. In case of encoding, the input to this coprocessor is a DCT coefficient block and output is run-level pairs. For decoding, the input is run-level pairs and output is a (possibly scaled/smaller) coefficient block.

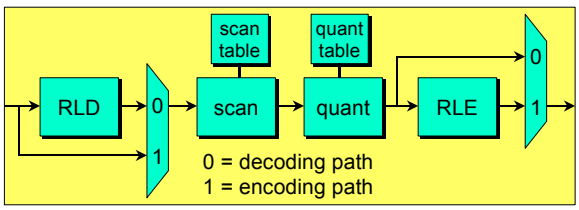

Figure 4. RLSQ coprocessor architecture.

The order of execution of scanning and quantization is different for encoding and decoding. However, the same implementation can support both encoding and decoding since the order of execution does not change the outcome of these operations. The run-length encoding (RLE) function and run-length decoding (RLD) functions are instantiated separately (Figure 4).

The RLSQ coprocessor operates on the granularity of DCT blocks and can switch tasks after producing one such DCT block. However, to enable multi-tasking, the coprocessor must take care of the internal state of each task, formed by the quantization and scanning tables. The VLD writes these tables in a stream buffer in the on-chip memory. At the start of a new task, the RLSQ coprocessor (re)loads the tables from this buffer into its local table memory. The previous tables do not need to be saved upon a task switch. The separation of data transport and synchronization in the communication infrastructure [11] allows the tables to remain in the stream buffer until the coprocessor explicitly frees the memory space by synchronizing with the VLD.

Discrete cosine transform (DCT)

The DCT coprocessor includes inverse and forward DCT functions. It operates on the DCT block level and does not require any knowledge of the picture properties or sequence properties. It supports both scaling and block compression. The internal state of this coprocessor is void after processing a DCT block. Consequently, the coprocessor can switch tasks on a block-level granularity without requiring additional hardware for state save/restore.

We chose the LLM algorithm [9] to implement both forward and inverse DCT functions. The LLM algorithm shares the same constants for multiplication in both inverse and forward DCT. Therefore, a DCT coprocessor can use the same data path for the implementation of inverse and forward DCT.

Motion compensation/estimation (MC/ME)

To efficiently compress video signals, MPEG exploits the temporal correlation between successive pictures. In a linear-wise scanning of the pictures from the left to the right and top to bottom, successive blocks of 16x16 pixels (macroblocks) are predicted from previously decoded pictures. Because these HD reference pictures are too large for on-chip storage, they are located in off-chip SDRAM memory. Consequently, the MC/ME coprocessor requires a connection to this off-chip memory to fetch the prediction data and to write the reconstructed pictures that are used as reference for the decoding of successive pictures (modeled as TpredictRD and Tstore in Figure 2).

Figure 5 depicts the internal architecture of the MC/ME coprocessor. To share the hardware resources for prefetching and interpolation, motion estimation and motion compensation are joined in one coprocessor. The prefetch and write units of Figure 5 embed sufficient buffering to hide the large access delay of the SoC infrastructure. The prefetch unit can work ahead of the MC/ME unit

by processing motion vector data provided by the VLD in a separate stream.

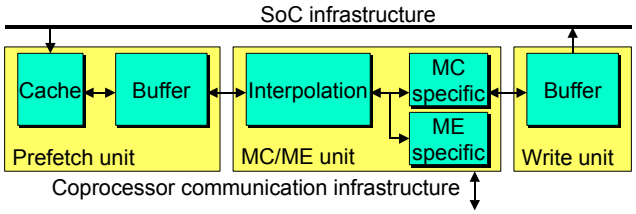

Figure 5. MC/ME coprocessor architecture.

The prefetch unit copies prediction data from the cache to its buffer. Although caching of stream-oriented data is generally not effective, the use of an SDRAM memory introduces some temporal locality due to its inherent transfer overhead. This can be exploited to reduce some of the scarce memory bandwidth. Our experiments show that a relatively small direct-mapped cache of 6 kB reduces on average 30% memory bandwidth.

5. Results

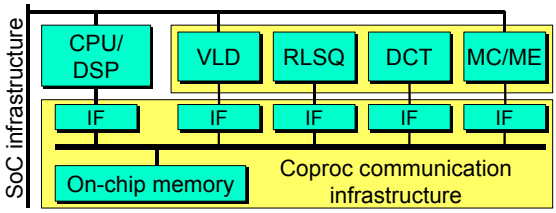

Figure 6. Resulting MPEG function module.

The generic decoder and encoder models have been reused in a number of architecture analysis and definition projects such as [4], [5], and [13]. To support the design trajectory, we developed a SystemC [12] model of the architecture that enables mixed-level simulation of YAPI tasks, behavioral coprocessor models, and cycle-accurate hardware models. This allows verification of functional correctness after each transformation step. The behavioral models of the described MPEG coprocessors reuse virtually all source code of the bare functions in the architecture tailored models. Simulation runs show that the coprocessors concurrently execute encoding and decoding tasks, reusing coprocessor hardware. The coprocessors of Figure 6 have sufficient performance to decode two high-definition MPEG streams in a time-shared fashion at a 150 MHz clock frequency with a total estimated coprocessor area of 4 mm^2 in 0.18 micron CMOS technology.

6. Conclusion

This paper presents an application design trajectory to transform sequential application C-code into a reusable multithreaded application model. The multithreaded model is subsequently mapped onto a target architecture with maximal source code reuse. The design trajectory concurrently addresses a set of applications, to identify common compute kernels that qualify for generic coprocessor implementations. Thereby, the presented trajectory results in the definition of a set of multitasking coprocessors that can be applied for various applications within a chosen application domain. This is demonstrated in a case study to define novel coprocessor hardware that concurrently executes decoding and encoding tasks, starting from sequential descriptions of an MPEG-2 encoder and decoder. It is difficult to automate the design decisions involved in the proposed design trajectory. Even so, ongoing work includes the development of tools and standardized APIs to reduce the effort involved in the code transformations.

References

[1] M. Berekovic et al., "A Multimedia RISC Core for Efficient Bitstream Parsing and VLD", *SPIE: Multimedia Hardware Architectures*, vol. 3311, pp. 131-141, Jan. 1998.

[2] W.H.A. Bruls et al., "A low-cost audio/video single-chip MPEG2 encoder for consumer video storage applications", *IEEE Int. Conf. on Consumer Electronics*, pp. 314-315, June 2000, Los Angeles, CA, USA.

[3] J.T. Buck et al., "Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems", *Int. Journal of Computer Simulation: Simulation Software Development*, vol. 4, pp. 155-182, April 1994.

[4] B.K. Dwivedi et al., "Exploring Design Space of Parallel Realizations: MPEG-2 Decoder Case Study", *9th Int. Symp. on Hardware/Software Codesign*, pp. 92-97, April 25-27, 2001, Copenhagen, Denmark.

[5] G.J. Hekstra et al., "TriMedia CPU64 Design Space Exploration", *Int. Conf. on Computer Design*, pp. 599-606, Oct. 10-13 1999, Austin, Texas, USA.

[6] G. Kahn, "The Semantics of a Simple Language for Parallel Programming", *Information Processing '74*, North-Holland Publ. Co., pp. 471-475, 1974.

[7] E.A. de Kock et al., "YAPI: Application Modeling for Signal Processing Systems", *37th Design Automation Conf.*, pp. 402-405, June 2000, Los Angeles, CA, USA.

[8] E.A. de Kock, "Multiprocessor Mapping of Process Networks: A JPEG Decoding Case Study", *15th Int. Symp. on System Synthesis*, pp. 68-73, Oct. 2-4, 2002, Kyoto, Japan.

[9] C. Loeffler, A. Ligtenberg, and G. Moschytz, "Practical Fast 1-D DCT Algorithms with 11 Multiplications", *Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 988-991, May 1989, Glasgow, England.

[10] J. Augusto de Oliveira and Hans van Antwerpen, "The Philips Nexperia Digital Video Platform", *Winning the SoC Revolution*, Kluwer Academic Publ., pp. 67-96, 2003.

[11] M.J. Rutten, J.T.J. van Eijndhoven, and E.J.D. Pol, "Design of Multi-Tasking Coprocessor Control for Eclipse", *10th Int. Symp. on Hardware/Software Codesign*, pp. 139-144, May 2002, Estes Park, CO, USA.

[12] *SystemC User's Guide*, version 2.0, 2001.

[13] P. van der Wolf et al., "An MPEG-2 Decoder Case Study as a Driver for a System Level Design Methodology", *7th Int. Workshop on Hardware/Software Codesign*, pp. 33-37, May 1999, Rome, Italy.