Mapping of MPEG-4 decoding on a flexible architecture platform

Erik B. van der Tol* and Egbert G.T. Jaspers*
Philips Research Lab., Eindhoven, The Netherlands

ABSTRACT

In the field of consumer electronics, the advent of new features such as Internet, games, video conferencing, and mobile communication has triggered the convergence of television and computers technologies. This requires a generic media-processing platform that enables simultaneous execution of very diverse tasks such as high-throughput stream-oriented data processing and highly data-dependent irregular processing with complex control flows. As a representative application, this paper presents the mapping of a *Main Visual profile* MPEG-4 for High-Definition (HD) video onto a flexible architecture platform. A stepwise approach is taken, going from the decoder application toward an implementation proposal. First, the application is decomposed into separate tasks with self-contained functionality, clear interfaces, and distinct characteristics. Next, a hardware-software partitioning is derived by analyzing the characteristics of each task such as the amount of inherent parallelism, the throughput requirements, the complexity of control processing, and the reuse potential over different applications and different systems. Finally, a feasible implementation is proposed that includes amongst others a very-long-instruction-word (VLIW) media processor, one or more RISC processors, and some dedicated processors. The mapping study of the MPEG-4 decoder proves the flexibility and extensibility of the media-processing platform. This platform enables an effective HW/SW co-design yielding a high performance density.

Keywords: media processing architecture, co-processors, heterogeneous architecture, parallel/pipelined architectures, memory hierarchy, HW/SW co-design, MPEG-4, application mapping

1. INTRODUCTION

Due to the convergence of television and computer technologies together with the continuously increasing integration density of VLSI, the total set of computational tasks in a System on Chip (SoC) becomes very diverse. High-throughput stream-oriented data processing is combined with highly data-dependent irregular processing with complex control flows and requires an integral design approach. The means for a cost-effective implementation is a heterogeneous architecture framework for high-performance computing, enabling hardware/software (HW/SW) partitioning with high flexibility. This concept has already been exploited successfully for MPEG-2 encoding and decoding. An overview of this architecture template, and its features is described in Section 3. To prove the extensibility and to evaluate the performance for the execution of very diverse tasks, this paper elaborates on the mapping of a *Main Visual profile* MPEG-4 decoder onto the architecture template. Section 2 reports on some existing LSI implementations of MPEG-4, describes some highlights in media processor architectures, and draws some parallels with the architecture template we adopted. Section 4 briefly explains the functionality of the targeted MPEG-4 decoder. To derive a suitable mapping onto the architecture, an analysis of the functionality and its processing characteristics is outlined in Section 5. Subsequently, Section 6 discusses the HW/SW partitioning and proposes a suitable implementation. Finally, Section 7 draws some conclusions and gives an outlook towards future research.

2. HIGHLIGHTS IN MEDIA PROCESSING ARCHITECTURES

An efficient implementation of a main visual profile MPEG-4 decoder for High-Definition TV (HDTV) requires a significant amount of computational power and memory resources. Therefore, it is most important to optimize the utilization of the implemented hardware, thereby reducing the system costs. Budagavi *et al.*[1] presented a *Simple Visual profile* and a *Core Visual profile* MPEG-4 decoder on the low-power TMS320C54x and TMS320C6x respectively. A straightforward comparison between the Core and Main Visual profile shows a factor of 21 for the throughput requirement, assuming an optimal utilization of the DSP. Extrapolating this number results in a required performance of

* {erik.van.der.tol, egbert.jaspers}@philips.com; Philips Research Laboratories, Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands.

33000 MIPS, without considering the increasing complexity that is required for grayscale alpha shape, interlaced video and sprite decoding. Bauer *et al.*[2], show an even higher MIPS requirement for implementation on a DSP. Although an implementation in software results in maximal flexibility, it may be concluded that the required computation power does not suffice. To solve this problem, Berekovic *et al.*[3] proposed some extensions by adding function-specific blocks into the data path of RISC processors. This paper addresses the instruction-level parallelism (ILP) and hence, only gives a partial solution to achieve an optimal performance density (i.e. the performance per unit area and per unit power). A shared-memory system architecture, containing multiple processors to address task-level parallelism, could offer a solution to further increase the computational power, but does not relax the tremendous memory requirements. An example of such approach is outlined by Stolberg *et al.*[4] They propose an MPEG-4 codec that integrates a RISC core, two DSPs, a 64-bit dual-issue VLIW macroblock engine, and an I/O processor. The following sections describe how the adopted architecture template can exploit parallelism and how memory bandwidth can be reduced by a hierarchical memory organization.

2.1. Parallelism and control

To cost-effectively increase the computational performance of programmable digital signal processors (DSP), parallelism has been exploited at different levels. Conventional Harvard architectures with a single instruction single data (SISD) structure have been extended to process multiple data entities with a single instruction (SIMD) or to simultaneously execute several instructions on multiple data entities (pipelined superscalar or very long instruction word (VLIW) architectures). To expand the amount of sustained parallelism while keeping the increasing complexity of optimal hardware utilization manageable, multi-processor and co-processor architectures[5-8] have been introduced by many vendors to exploit task-level parallelism. This technique does not replace the instruction-level parallelism (ILP), but rather adds an additional hierarchical level to the system. Thus within a task, a large amount of control (instructions) manages the processing at a small grain size, e.g. add, subtract, and multiply operations. At higher levels in the hierarchy, control is performed on task level. Thus, a single task instruction results in more processing. At an even higher level, functions or even complete applications are controlled. The various levels in this hierarchy have different properties[9], which are highly interdependent. Figure 1 shows how the properties gradually change, going from coarse-grain toward fine-grain processing via applications, functions, tasks and instructions.

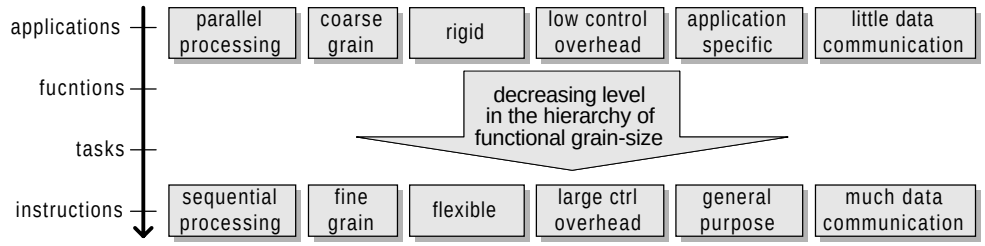

Figure 1: System properties at hierarchical levels of functional grain size.

Let us now translate this view into the consequences for parallel processing. Obviously, the mapping of tasks onto parallel processors (multi-processors) or co-processors significantly increases the amount of parallelism. However, most of these parallel processors contain special-purpose hardware to increase the performance density (i.e. the application throughput per unit area and per unit power). The disadvantage of these special-purpose processors is the lack of flexibility for reuse over different applications and different platforms, particularly if the processors contain coarse-grain functions.[5] Bove and Watlington[10] present an architecture in which task-level parallelism is provided with processors containing fine-grain functions. Although this offers a more flexible approach, the bandwidth requirement of the communication network increases significantly, thus requiring an expensive communication structure such as a crossbar. Hence, in case of a shared-memory architecture this results in a bandwidth bottleneck or a more expensive memory solution. We exploit a hierarchical communication architecture to solve the contradicting requirement for flexibility in a system with a high performance density. Section 2.2 elaborates in more detail on this approach.

2.2. Hierarchical communication

Section 2.1 shows that many system properties such as amount of control, flexibility, communication bandwidth, etc., are generally related. To establish an optimal balance between system costs and flexibility, this section focuses on the most demanding properties.

Due to the fast increase of required computational power of consumer media systems, the data communication to and from the off-chip memory has become the bottleneck in the overall system performance.[11] A major technology step to solve this problem is the embedding of on-chip SDRAM.[12] This not only reduces the bandwidth requirement to off-chip memory, but also reduces power dissipation and costs for e.g. chip package and footprint. However, because the demand for more communication bandwidth and memory capacity in such systems continuously increases, also the evolution of external-memory technology remains important.[13] Typically, computer and media architectures contain a hierarchical memory infrastructure going from small high-speed memories, locally embedded in the central processing units, to relative slow and large memories located off-chip. Patterson and Hennessy[14] describe the classical techniques for such hierarchy by means of caching and virtual memory paging. In such an approach, the content in the memory at higher levels in the hierarchy represents a superset of the content in the lower memory levels (assuming memory coherency). Consequently, all data that is communicated to and from the lowest memory level will be visible in all levels of the memory hierarchy. Bandwidth requirements and memory latency are only decreased when variables (thus memory address space) are reused within the scope that is contained in a lower memory level (locality of data). However, for streaming applications the reuse of data is limited. Instead, streaming applications have a very regular access pattern. Data is often stored for rate control, synchronization, data reordering, or simply for communication buffering between processors (FIFO-like). Examples are frame memories for picture reordering in an MPEG encoder or smaller memories to store a Discrete Cosine Transform (DCT) block and to read them in a different order (zig-zag scan). From these examples, we can conclude the following properties for stream-based applications:

- since for these types of data storage the lifetime of the data is very deterministic, the amount of memory to be allocated (worst case) can easily be derived at compile time, and
- due to the regular access patterns, the worst-case memory bandwidth requirement can easily be derived.

Considering these two observations, we exploit memory hierarchy differently. Instead of caching techniques, the streaming data is stored at one unique location in the memory hierarchy, dependent on the size of the required memory space and the required bandwidth. For example, the frame buffers for picture reordering and storage of reference pictures are mapped in external SDRAM memory, whereas the zig-zag scan memory is mapped onto a locally embedded SRAM memory. This approach offers the ability to limit the bandwidth requirements for relative slow memories (in high hierarchical levels) but to use its large capacity property to store large chunks of data. The communication of fine-grain data packets such as DCT blocks and video lines can be kept on-chip.

Note that differences in functional grain size do not necessarily lead to similar differences in the communication grain size. For example, a DCT can be performed on a complete frame before conveying it to the next processing function. However, by matching the communication grain size with functional grain size (see Section 2.1), the adopted architecture can exploit the memory hierarchy as described above. A limited amount of course-grain communication channels (for e.g. frame reordering, reference pictures and the buffer for bitrate control) are visible at the function level, whereas a larger amount of communication channels (for e.g. DCT, quantization, run-length coding, variable-length coding) are visible at the task level. At the instruction level, the amount of communication is even larger.

The Imagine media processing system exploits the memory hierarchy as explained above for media processing.[15-16] This system offers a large amount of flexibility via programmable fine-grain processors, but still requires only a limited amount of bandwidth to external memory. Eight parallel arithmetic clusters of eight functional units independently fetch VLIW instructions, thereby enabling parallel operation of all functional units within a cluster. The communication for a sequence of VLIW instructions within an arithmetic cluster, performing a task, is kept local by means of a local register file, while only results of the task are communicated to a larger shared register file. The clusters perform a set of arithmetic operations on each stream element, thereby not allowing large data-dependencies within this scope. Such a set of operations can independently read from its input stream(s), compute a series of arithmetic operations (e.g. matrix multiplication or convolution), and write to its output stream(s). This homogeneous architecture is particularly suitable for regular stream-based processing. Applications containing data-dependent processing with complex control structures and with a limited amount of ILP, do not result in an efficient utilization of the resources. For example, Variable Length Decoding (VLD) will result in a poor performance density when mapped onto the Imagine architecture.

Section 3 presents the architecture template that was used for the mapping of the MPEG-4 application. The system contains a two-level hierarchical communication structure to combine the flexibility of medium-grain task-level control with the limited bandwidth requirement for coarse-grain function-level control.

3. PROCESSOR SYSTEM OVERVIEW

For the mapping of the MPEG-4 decoding application, we use an architecture template with a two-level hierarchical communication infrastructure, each with its own set of processors. In this architecture, the communication grain size matches the grain size of the functionality in the processors. Figure 2 shows the block diagram of the two-level system architecture. Section 2 described how this architecture combines the flexibility of medium-grain functions (tasks) with the bandwidth requirements to off-chip memory of course-grain functions (e.g. MPEG decoding). To allow autonomous and asynchronous operation of the processors, communication between the processors is achieved via buffers. These buffers are mapped onto cost-efficient shared memories to maximize flexibility. Thus, for processors at the highest hierarchical level, the buffers requiring large-size data packets are located in off-chip SDRAM memory. For the processors in the low hierarchical level, the buffers can be stored in the embedded SRAM memory.

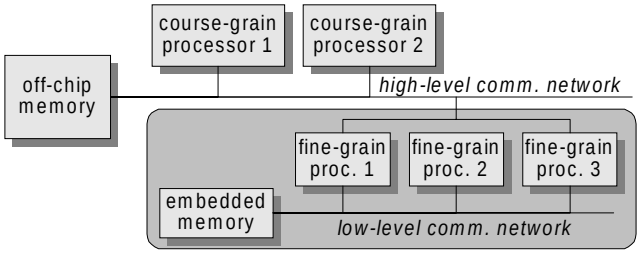

Figure 2: Block diagram of the two-level system architecture.

To describe the architecture, the properties and features can be discussed separately for the high and low hierarchical levels. Control of the high-level processors is performed in software on a host processor, because this offers maximum flexibility while the relative low synchronization rate does not require much computational effort for this task. This control software configures the high-level processors, schedules the tasks, and provides the synchronization for communication. Non-preemptive multi-tasking can be provided at the rate of communication packets, i.e. after processing of a complete picture for one media stream, a picture is processed for another media stream. To give an overview of the properties and features of the low-level processors, the grain size of computation and communication, synchronization, multitasking capabilities and task scheduling, programmable processors, and run-time reconfiguration are discussed separately below.

- *Grain size of computation and communication* - As mentioned above, the architecture provides two levels of communication grain size. At the highest level, course-grain processors such as a graphics renderer, a media processor, and an MPEG engine communicate and synchronize at the size of video pictures via off-chip memory. At the lowest level, a tasks such as DCT transform, Motion Compensation and Estimation (MC/ME), Variable Length Encoding and Decoding (VLE/VLD), etc. communicate via relative small communication buffers that are located in the shared on-chip memory.
- *Synchronization* - The architecture template that we adopted contains hardware to support data flow process networks[17-18], enabling data-driven processing. Consequently, scheduling of the tasks and synchronization of the data is performed at run-time and provides abstraction from these issues to the programmer. Moreover, the dedicated synchronization network in the architecture enables synchronization of arbitrary-sized data packets (e.g. on single pixels or a DCT block) independent of the buffer size. This is an important feature, because the grain size of communication together with the buffer sizes mainly determine the dynamic behavior of the system.
- *Multitasking* - Because the low-level processors in the architecture operate on a task-level granularity and comprise generic functions, they can be reused over different applications. This reuse is provided by the architecture by multitasking capabilities, i.e. to simultaneously reuse hardware resources for similar tasks.
- *Programmable processors* - On both hierarchical communication levels, a programmable processor can be taken up. It is even allowed to connect a programmable processor to both levels of the hierarchical communication infrastructure. This is implemented by means of a special co-processor connection that is provided by the processor, and is used to access the second-level communication network.
- *Configuration management* – For configuration of the system, a programmable host processor is required to configure the available dedicated HW processors for the application tasks, to program the signal flow graph by means of buffers allocation and assignment of the buffer pointers. Moreover, the host processor has to initiate each general-purpose processor to load the executable code of its application-specific tasks. Subsequently, the system can

synchronously run the application by executing all tasks on the autonomous processors. Apart from static configuration of the system, it is also required to do reconfiguration at run time. The architecture provides several mechanisms to establish this, and the MPEG-4 decoding system uses this feature to implement dynamic signal flow graphs. At run-time, dependent on the data in the bitstream, tasks are being inserted and/or deleted and buffers are allocated in the memories.

4. MPEG-4 DECODER FUNCTIONALITY

Section 3 showed an overview of the architecture template that was used for the MPEG-4 decoder implementation. To determine the required system components, let us now analyze the application and give some first indications of the characteristics of the functionality in terms of processing regularity, parallelism, and throughput requirements.

The MPEG-4 standard is constructed as a toolbox of functional features e.g. shape. Without the shape tool, the decoder is not capable of decoding shaped video objects. The toolbox approach allows selection of a subset of tools that is adequate for the targeted application. To guarantee compatibility between encoders and decoders, MPEG-4 specifies *profiles*, each containing a different set of tools. Hence, a decoder that is compliant with a certain profile can decode all bitstreams that are encoded using the corresponding set of tools. The toolbox concept allows the standard to be extended for future coding concepts. Moreover, it enables the standard to deploy complete new applications or to improve existing ones. Thus unlike MPEG-2 (digital television), MPEG-4 does not target a major "killer application".[19] Within a profile, also a set of *levels* have been defined which restrict the computational complexity. For example, level 2 of the Main Visual profile allows 16 visual objects at CIF resolution, whereas level 4 allows 32 objects at HD resolution. The objective of this section is to select a suitable profile-level combination to enable derivation of the system requirements.

From a technical point of view, an MPEG-4 system can be described as a layered model, which is depicted in Figure 3. Not all layers of this model are covered by the MPEG-4 standard; i.e. the Transport layer and Composition/Rendering are explicitly not described. However, interfaces between the Transport and Synchronization layers and between the Decoding and Rendering layers are specified. Hence, the delivery and presentation media are free to choose.

In order to analyze our MPEG-4 application, Sections 4.1 through 4.3 outline the particularities of the various MPEG-4 layers, while scene-graph and resource management are discussed in Sections 4.4 and 4.5. As a starting point of the mapping of an MPEG-4 decoding system onto the architecture, Section 4.6 and 4.7 elaborate on the application domains including suitable MPEG-4 profiles and levels.

4.1. TransMux, Delivery, and Synchronization Layers

The TransMux (Transport Multiplexing) layer offers transport services; only the interface on top of this layer is specified by MPEG-4. The choice in transport protocol is left to the end user or service provider, and allows MPEG-4 to be used in a wide variety of operation environments. In the Delivery layer, the MPEG-4 TransMux streams are demultiplexed into separate Elementary Streams (ESs) according to the DMIF (Delivery Multimedia Integration Framework) specification (part 6 of the standard). The subsequent layer of the model in Figure 3 consists of the Synchronization Layer. The main part of this layer concerns the timing aspects within the model. Each access unit (smallest data entity with timing information) that is received may contain a decoding time stamp (DTS) and a composition time stamp (CTS). Such time stamps convey the intended decoding time of an access unit and the intended composition time of a composition unit, respectively.

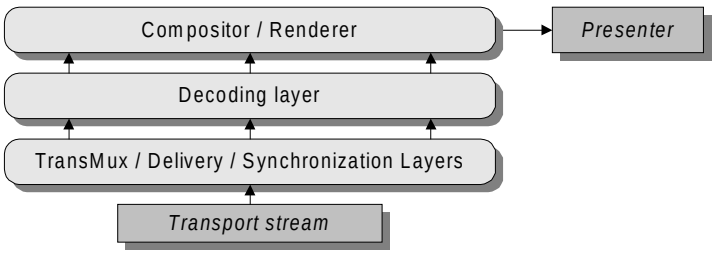

Figure 3: Simplified view of the MPEG-4 decoding system layered model.

The functionality of the TransMux, Delivery, and Synchronization Layers mainly concern streaming and management, and can be classified as event-driven processing. This type of processing is highly irregular and contains only a limited instruction-level parallelism (ILP). Moreover, the rate of control is relatively low.

4.2. Object decoding

The middle layer of the model in Figure 3 consists of the Object Decoding Layer. This layer contains the tools to decode the elementary streams from the Synchronization layer. The visual objects such as video, still textures, meshes, face and body animation, etc. may be combined or separated into one ore more elementary streams.[20] The audio objects are coded in separate elementary streams. Object Descriptors are used to relate elementary streams to media objects within a scene. Object descriptors (OD) themselves are conveyed in one or more elementary streams, since it is possible to add and discard streams (and objects) during the course of an MPEG-4 session. Similarly, the scene description (BIFS – Binary Format for Scene Description)[21] is also contained in an elementary stream, allowing to modify the spatio-temporal layout of the presentation over time (even by interaction from an end-user). Besides the scene description, the BIFS also contain a rich set of graphics operations.

In order to discuss the properties of the Decoding Layer we need to go into somewhat more detail of some decoding tools. Especially the visual decoding tools are of interest, because the most computationally expensive tools (e.g. Video and Still Texture decoding) of an MPEG-4 system can be found within this layer. For the visual as well as the audio decoding tools, two different classes of processing can be defined. Firstly, bitstream processing, which typically contains no parallelism and has a high control complexity. Examples of this are VLD and arithmetic decoding (within Shape and Still Texture decoding). Other tasks operate on macroblocks, have low control complexity, and a large degree of data parallelism. Examples of this are inverse DCT, inverse quantization, motion compensation, and context calculation (within Shape decoding). Furthermore, these kinds of tasks require a high data bandwidth. Most of these block-oriented functions are very similar to those in e.g. MPEG-2, and consequently might be reused over applications.

The elementary streams that contain the BIFS and the ODs are decoded and used in the composition/rendering layer to construct the final output to be presented. Therefore, the BIFS and OD decoders can be seen as merely parsers, which have a sequential behavior, a complex control structure (strong data dependency), and typically requiring irregular data access.

4.3. Rendering & composition and presentation

The top layer of the model consists of composition and rendering. Here the final scene, consisting of various audio-visual objects, is put together. Rendering of objects as described in the BIFS takes place in this layer. The output of the composition/rendering layer drives the presentation devices. Whether some buffering is required between composition/rendering and presentation depends on the implementation of the composition/rendering. If the architecture provides the capability to output streams with constant frame rate, synchronized with the presentation devices, additional buffering is not required. It should be noted that the composition and rendering itself are not standardized within MPEG-4, only the format of audio-visual objects and the BIFS. In this way, it is guaranteed that the MPEG-4 system is unaware of the presentation device.

Functions like composition, rendering and graphics are characterized by their need for high data bandwidth and irregular data access. However, both instruction-level and task-level parallelism can be exploited very efficiently for these functions.

4.4. Scene-graph management

In principle, there are two different BIFS nodes. The first type of node is used to create objects in the scene or to refer to elementary streams associated with media objects. The actual description of these objects is extracted from the object decoder and is conveyed via the composition buffer. This type of node is found in the *Graphics profile*. The second type of node is used to build the scene structure and to describe scene updates and user interactions. These are called “scene graph elements”, and are found in the *scene graph profiles*. The audio, visual, and graphics profiles are called media profiles as they govern the media elements in the scene.

The scene-graph manager contains the functionality that is necessary to convert the decoded scene-graph description into signals that controls the visual renderer, i.e. to draw the final scene. Hence, the scene-graph manager performs the following tasks (not necessarily in this order):

- parsing of the decoded scene graph description (BIFS nodes),
- building up a geometric representation of the 3D scene,
- updating the 3D scene with temporal behavior (e.g. animation, scene updates), and
- generating visual renderer API calls to draw the scene.

4.5. Resource management

Resource management comprises the control of the available resources in the different layers of Figure 3. The resource manager is responsible for the following tasks.

- *Configuration of the synchronization layer*: The number of packetized ESs depends on the input bitstream. Therefore, the resource manager performs run-time configuration of the synchronization layer.
- *Allocation of the required buffer resources*: The buffer-size information is conveyed via the corresponding ES descriptor within an OD and hence is extracted by the resource manager.
- *Configuration of the required decoders*: Similar to the buffer resource requirements, the type of decoder is conveyed via the ES descriptors. This information is used by the resource manager to configure the available decoders.

4.6. Application domain for the target architecture

With the architecture depicted in Figure 2, the MPEG-4 decoder is targeted for mid-range to high-end media processors for Digital TV and set-top boxes. Therefore, digital TV (DTV) broadcast is the most important application domain. Beyond DTV, MPEG-4 addresses far more applications of which many are still in a research phase. On the other hand, in the mobile (wireless multimedia communications) and Internet (combined text, graphics, video, and audio) area significant effort has already been spent in the integration of MPEG-4 technology.

The integration of Internet into consumer electronics such as TV, set-top boxes, and game consoles positively influences the introduction of MPEG-4 into the set-top box and TV world. Current stationary digital television standards[22] are based mainly on MPEG-2 compression and multiplexing techniques. Because of an increasing demand for higher interactivity and a mixture of natural and synthetically (graphics) generated video in this field, a migration toward MPEG-4 is desirable. MPEG members are working on a compatible insertion of MPEG-4 multimedia elements into a conventional MPEG-2 transport stream. Thus, an evolutionary path from MPEG-2 to MPEG-4 should be feasible.

4.7. Target profiles and levels

To feature the necessary tools for digital TV broadcast, the Main Visual profile is most likely to become the de-facto profile. Besides the coding of progressively scanned rectangular video, this profile also features interlaced video, sprite objects, grayscale alpha shape coding, and scalable textures, which enlarge the complexity of the system substantially.

To anticipate future requirements, the architecture is dimensioned to process HD pictures, which is comprised by level 4 (L4) of the *Main Visual profile*. In order to give an impression on the high complexity required for Main Visual profile, compared to profiles typically suitable for mobile or internet applications (*Simple* or *Core Visual profile*), Table 1 summarizes some characteristics of these profiles at their highest levels.[23]

Table 1: Characteristics of selected Natural Visual profiles @ levels.

	Simple profile @ L3	***Core profile @ L2***	***Main profile @ L4***
typical Visual Session size	352 x 288 (CIF)	352 x 288 (CIF)	1920 x 1088 (HD)
maximum number of objects	4	8	32
maximum number of macroblocks per VOP	396	792	16320
maximum amount of reference memory (MBs)*	396 (= CIF)	792 (= 2×CIF)	16320 (= 2×HD)
maximum number of MBs/sec*	11880 (= 30 Hz CIF)	23760 (= 30 Hz 2×CIF)	489600 (= 30 Hz 2×HD)
maximum sprite size (MBs)*	not supported	not supported	65280
maximum bitrate (kbit/s)	384	2000	38400
shape	not supported	binary	binary, gray

* MBs = macroblocks

For the graphics part of the MPEG-4 standard, the *Simple 2D profile* is sufficient for the intended application. For the Scene Description, we target for the *Simple profile*.

5. ANALYSIS OF THE FUNCTIONS

Section 4 described the functionality of the MPEG-4 system and briefly explained the characteristics of all functions. Let us now consider the signal flow graph of the application and decompose the functions into tasks that have distinct characteristics. Subsequently, we propose a HW/SW partitioning to derive the underlying architecture components.

Figure 4 shows the signal flow graph of the complete MPEG-4 decoder system. The delivery mechanism comprises the TransMux, the Delivery and the Synchronization layers as described in Section 4.1. The output of this block conveys elementary streams (ESs) that are written in separate decoding buffers. Subsequently, the appropriate decoder instantiations decode the ESs and write their output into the composition buffer. To configure the synchronization layer and the decoders and to allocate the buffers, the resource manager abstracts the resource requirements from the delivery mechanism and the ES descriptions, which are located in the ODs. To finally present the audio-visual scene, the scene-graph manager interprets the scene-graph description and the ODs and subsequently controls the rendering.

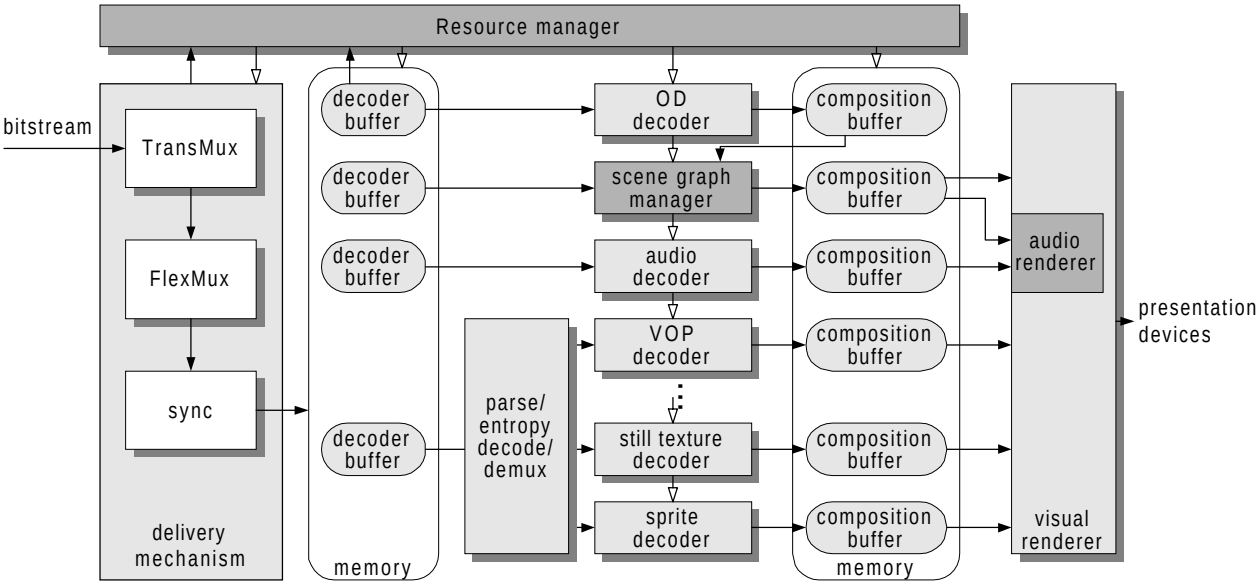

Figure 4: Signal flow graph of the complete MPEG-4 decoder functionality.

5.1. Processing characteristics

To come to a partitioning of the functionality in HW and SW, we separate the MPEG-4 decoder functions into tasks with self-contained functionality and clear interfaces. Table 2 shows an overview of these tasks, including their processing characteristics. As explained in Section 2.1, the amount of parallelism, the throughput, the flexibility and the complexity of control are related. However, this is just a rule of thumb and has to be made explicit to come to a feasible HW/SW partitioning. Moreover, the partitioning may also depend on non-technical reasons such as the design capacity, available design capabilities, and the design time to influence the time-to-market.

Section 2.1 showed that a programmable general-purpose processor offers most flexibility. However, to improve the computational power, parallelism which is inherently application dependent, has to be exploited at the expense of flexibility. This observation not only holds for parallel functions in the signal flow graph, but is also applicable for functions that sequentially process data in a signal flow graph. Partly, this is caused by the Von Neumann bottleneck; i.e. the processing of each data word in a general-purpose processor requires instruction fetching. However, all modern processors feature a Harvard architecture (separate instruction and data bus). More important, sequential processes can use pipelining to increase the throughput and hence feature parallel processing. Therefore, the fourth column in Table 2 indicates the throughput requirements of the tasks. However, this is not sufficient to determine the partitioning, since pipelining in order to increase the throughput can only be applied for stream-based processing with a low control complexity (indicated in the fifth column). Note that decision branches disrupt the pipelining, particularly if they are data dependent. Although solutions like branch prediction and speculative execution do exist, they are not straightforward for hardware implementation. For similar reasons as pipelining, parallel execution of instructions as

provided in VLIW or superscalar architectures can be used to improve the throughput. Although not explicitly shown in the table, tasks that contain ILP and may exploit pipelined or parallel execution of instructions are suitable for implementation on a VLIW architecture.

Table 2: Properties of the MPEG-4 decoding tasks that determine the mapping onto HW or SW.

function	tasks	parallelism*	throughput	complexity of control	reuse over applications	architecture
Delivery mechanism	transport protocol transport stream demultiplexing synchronization processing	sequential	low	low	low	RISC
OD decoder		sequential	low	medium	low	RISC
Scene-graph manager	BIFS parsing, 3D geometric scene reconstruction, rendering command generation	sequential	low	high	low	RISC
Resource manager	(re)configuration of the hardware, memory management	sequential	low	low	low	RISC
Audio decoder		ILP & TLP	medium	low	medium	DSP / Media proc.
Parse/entropy decode/demux		sequential	medium	high	low	RISC
Still Texture decoder	DC prediction, ZeroTree decoding, inverse quantization, inverse Discrete Wavelet Transform	ILP & TLP	low	medium	low	Media processor
	arithmetic decoding	sequential	low	high	low	RISC
VOP decoder & sprite decoding	Variable Length decoding, Context-based Arithmetic decoding	sequential	high	high	low	RISC
	up/down sampling, inverse scan, inverse quantization, inverse DCT, padding, VOP reconstruction	ILP & TLP	high	low	high	dedicated
Renderer & sprite warping	BIFS browser, geometry, lighting	sequential	medium	medium	high	DSP / Media proc.
	rasterizer set-up, rasterization	ILP & TLP	high	low	high	dedicated

* ILP = Instruction-Level Parallelism, TLP = Task-Level Parallelism

The reusability of tasks (sixth column) has two aspects. First, a task can be reused over several applications within the same system. For example, a DCT transform can be used for (de)compression schemes like MPEG-1/2/4 and H.26x. Secondly, tasks can be reused over more than one architecture design. A hardware DCT implementation may be preferred since this is usually most efficient. However, the hardware should offer sufficient flexibility to enable its use in all applications and architectures. Moreover, besides the design of the micro-architecture, a hardware design also includes a software implementation for functional specification and simulation. Hence, the design of hardware requires more effort and can be a valid reason for implementing functionality in software.

5.2. Hardware/software partitioning

Let us now briefly explain the reasoning of the chosen HW/SW partitioning as shown in Figure 5. To minimize the design effort, the default approach is to implement a task in SW, unless this significantly decreases the performance density and thus the system costs. For the delivery mechanism, the OD decoder, the scene-graph manager, and the resource manager, the choice is straightforward. For these tasks, the computation requirement as well as the amount of parallelism is limited, thereby making these tasks very suitable for a general-purpose processor.

Audio decoding has a medium throughput. For six-channel audio, the decoded stream has a bandwidth of 0.5 MByte/s. Compared to video, the audio signal processing has a higher precision and is usually implemented with a large amount of floating point operations. Moreover, standards such as MPEG and DVD contain an increasing amount of different audio coding schemes, which are normally not active concurrently. Therefore, a software implementation on a DSP or a media processor is most suitable.

For Still Texture decoding a similar reasoning holds. In contrast to video decoding, there are no hard real-time constraints and therefore it requires a lower bandwidth. Furthermore, Still Texture decoding is MPEG-4 specific, and therefore not reusable, i.e. the utilization of a hardware implementation would be poor. It can be concluded that also this task can be performed in software. Because this concerns a special-purpose task with a significant amount of instruction-level parallelism, we propose an implementation onto a VLIW media processor, e.g. TriMedia.[24]

For VOP and sprite decoding, the mapping is less straightforward because this functionality consists of a diverse set of tasks, shown in Figure 5 as a fine-grain block diagram. The numbers near the communication channels represent the bandwidth of the data transport for MPEG-4 Main Visual profile @ L4 (HD: 1920×1088 @ 30 Hz, see Section 4.7). Because the memory requirements are mainly determined by the large bandwidth numbers, only these numbers are shown. Part of the functionality consists of entropy decoding (VLD, CAD), which is mainly sequential processing with complex control. These properties make an implementation that matches a RISC-like architecture very attractive. However, because the bitstream is non-word aligned with a high throughput requirement, some hardware acceleration for e.g. bit manipulations and the keyword lookup is desired. An example implementation is presented by Berekovic, *et al*.[25] The remainder of the functionality contains a significant amount of parallelism at both instruction level and task level. Moreover, the computation requirements are high, particularly for targeted HD resolutions. Note that the VOP decoding requires memory access to an off-chip memory at high bandwidth.

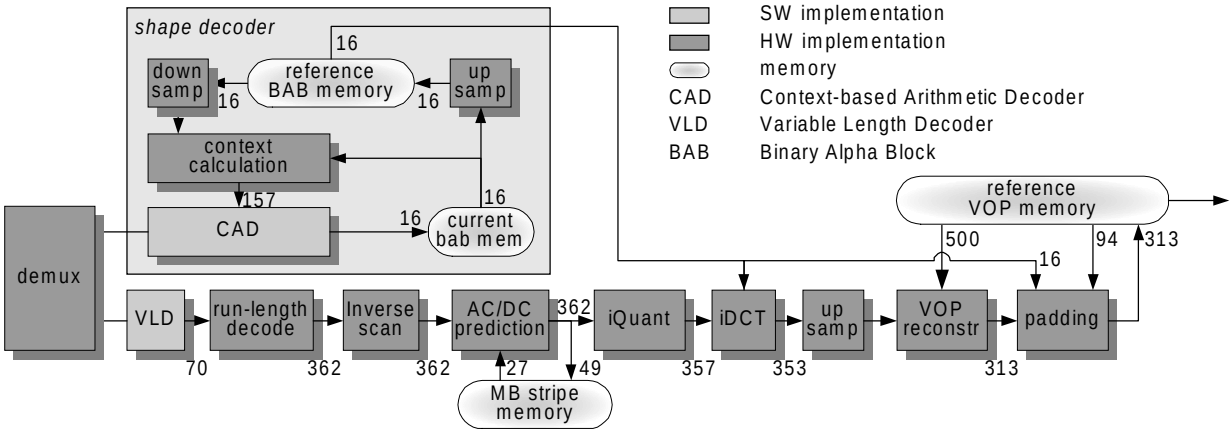

Figure 5: Block diagram of Video Object Plane decoder, including data transport bandwidths (MByte/s).

The bottom row in Table 2 represents the rendering and composition of all objects into the final scene. The first part of this functionality comprises a BIFS browser, analog to a VRML browser. The output of the browser could operate at the abstraction level of the OpenGL application programmer interface (API). However, this standardized API does not cover all functionality that is required for rendering of video objects from an MPEG-4 decoder. For that purpose, we propose that OpenML[26] is used as a more suitable API specification. This API also considers the YUV color space, interlaced video, audio, synchronization of audio/visual object, etc. The OpenML function calls are used by the geometry & lighting functionality in the renderer to create a 3-D geometric representation of the visual scene. Both the BIFS browser and geometry & lighting tasks are mapped onto a media processor, which contains ILP and includes floating point operations. After conveying the 3-D geometric model by means of vertices, the setup of the rasterizer converts the model into a large set of polygons for rasterization. This setup task within the renderer requires a significant amount of computational power, and increases the communication bandwidth when going from a vertices-based input to a polygon-based output. The final rasterization stage of the renderer accesses all pixel-based input data from the composition buffers and reconstructs the final visual scene. To provide sufficient computational resources, both the rasterizer setup and the rasterization itself are best performed with a dedicated hardware pipeline.

6. MAPPING PROPOSALS

Section 5.1 and 5.2 explained how the functionality of the complete MPEG-4 decoding system is partitioned into hardware and software and what type of processing is used for the separate functions. Now let us consider how these functions are mapped onto the overall system-architecture template as presented in Section 2, including the architecture of the separate processors in the system.

Figure 6 shows an implementation proposal. The numbers near the connections to the communication network represent the bandwidth of the data transport for MPEG-4 Main Visual profile @ L4 (HD: 1920×1088 @ 30 Hz, see Section 4.7). Because the memory requirements are mainly determined by the large bandwidth numbers, only these numbers are

shown. Although the bandwidth indications are based on the worst-case MPEG-4 bitstreams, the bandwidth numbers for rendering are typical, since the worst-case numbers for this function can be almost unlimited.

The two-level hierarchical architecture can be clearly recognized in Figure 6. All functions that require a large memory capacity or that communicate large data packets are connected to the off-chip memory. The functions that only use a relative small memory size and communicate via small buffers, are connected to the second level communication structure, which is embedded in the system. Note that it is also possible to connect functions to both networks. As shown in the figure, separate I/O modules are included to write the input MPEG-4 bitstream into the memory and to output the decoded audio and video signals. Besides streaming of video data from the memory to the display, the Video Out module also converts a YUV 4:2:0 signal to a 32-bit RGB signal.

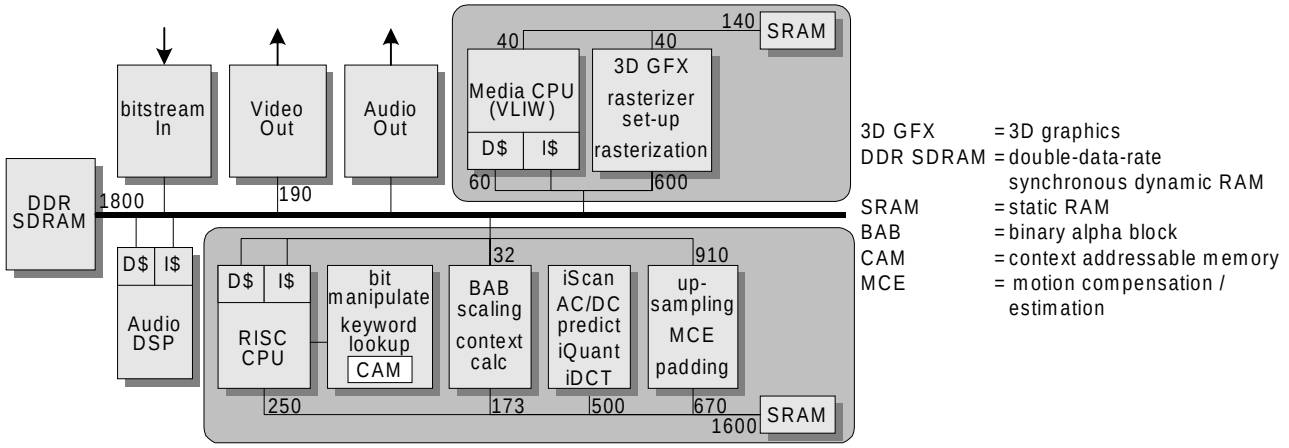

Figure 6: Architecture implementation of the MPEG-4 decoding system, including data transport bandwidths (Mbyte/s).

The delivery mechanism in Figure 4 is performed by the RISC core. It parses the MPEG-4 bitstream from the off-chip memory, processes it, and writes the output elementary streams back into the decoder buffers (off-chip memory). Besides the delivery mechanism, the RISC also provides the OD decoding, the resource management, and the scene graph management. Finally, the RISC performs the tasks for parsing, demultiplexing, and entropy decoding of the visual objects. The connection of the RISC core to the local embedded memory is provided by means of a dedicated co-processor connection, provided by the RISC. Note that the RISC does not have a cache memory for this connection. The absence of this cache can be justified by giving first priority access of the RISC to the local memory to guarantee low latency. A more detailed study is necessary to prove the feasibility to execute all above-mentioned tasks within the cycle budget of a single RISC core. Although the outcome of this study may show the requirement for more than one RISC, it should be noted that the silicon area of a single powerful MIPS (Philips PR3940) RISC core is only 5.5 mm^2 consuming less than 200 mW.

For VOP decoding and sprite decoding dedicated hardware modules are provided. The context-based arithmetic decoder (CAD) which is performed by the RISC core, parses the input bitstream, retrieves the context information from the shape decoder hardware module (via a communication buffer in local memory), and writes the output Binary Alpha Block (BAB) into a buffer in local memory. The BAB is subsequently read by the shape decoder module, and is used to calculate the context value. To do the calculation, the hardware module also reads reference BABs from off-chip memory and writes the newly calculated BAB back to the same memory. For the VOP decoding, the RISC core provides the variable length decoding and conveys the output stream to a hardware module that features a pipeline with inverse scan, AC/DC prediction, and inverse DCT. Some space in the local memory is used for AC/DC prediction (see Figure 5), which results in an additional input and output port of this hardware module to the local memory. The output of the hardware module is streamed to another hardware module that provides the motion compensated VOP reconstruction and the padding process. To store the reconstructed output and to retrieve the motion compensated prediction data, considerable off-chip memory bandwidth is consumed. Another important architectural aspect is the requirement for all VOP processors to provide multi-tasking capabilities, because a Main Visual profile bitstream may contain up to 32 independent video objects. The multi-tasking capabilities described in Section 3 are provided in the architecture template.

To provide Still Texture decoding, a very-long-instruction-word (VLIW) processor is embedded. This function only requires a connection to the off-chip memory, because both the input bitstream and the output image are stored there. However, because also the rendering process is partially mapped onto the VLIW core, this processor is connected to a second-level communication network too. Similar to the RISC connection, the VLIW uses a special co-processor connection without cache. Besides Still Texture decoding, the media processor also establishes the software tasks of the rendering functionality, i.e. BIFS browsing, geometry, and lighting. The total data bandwidth for visual rendering to off-chip memory is very large and highly dependent of the implementation. For example, Berekovic, *et al.*[27] presented an image rendering co-processor for MPEG-4 systems that contains an external local memory for autonomous rendering of two video objects and a background picture. Although they consider standard definition (SD) video, the bandwidth requirements are higher than for the visual renderer presented in this paper. This is mainly caused by the additional transfer of all visual objects from the shared memory to the local memory (reading and writing of three SD pictures). Moreover, their bandwidth calculations assume a 4:2:2:4 sampling format, whereas we assume a 4:2:0:4 sampling format. For our bandwidth estimations, we assumed a bandwidth-friendly tile-based rendering algorithm. Experimental results already proved the feasibility of such an implementation. Because the rendering functionality and the MPEG-4 decoding functionality do not share the same computational resources and only have inter-communication via the off-chip memory, the communication networks can be separated to reduce the complexity of the two-level network. However, for flexibility it is also attractive to have a single second-level communication network with one shared embedded memory. This enables implementation of additional tasks for the VOP decoder in software, thereby anticipating for future extensions.

7. CONCLUSIONS

MPEG-4, compared to other existing coding standards like MPEG-2 and H.26x, targets a much broader set of applications, with an accompanying increased number of more complex algorithms as well as coding modes. This extended functionality requires a flexible and extensible platform. However, the computational complexity of MPEG-4 applications exceeds that of state-of-the-art media processors, especially for profiles suitable for digital TV broadcast. Therefore, hardware support becomes an increasing necessity. Furthermore, the increasing demand to combine audio-visual material with (advanced) graphics within the application domain of TV and set-top boxes enhances the complexity of these systems severely.

For the adopted architecture template, an effective HW/SW partitioning was performed based on the MPEG-4 application analysis in terms like amount of parallelism, throughput requirements, the control complexity and the reuse potential. This analysis resulted in a feasible mapping of the MPEG-4 decoding application onto the architecture template. This template features a two-level hierarchical communication structure in order to increase the flexibility without alleviating the required bandwidth to the off-chip memory. It can be concluded that the stream-oriented decoding stage of an MPEG-4 decoding system can be carried out satisfactorily by the special-purpose processors, whereas amongst others resource management, demultiplexing and entropy decoding (complex control) can best be achieved in software.

To further refine the implementation proposal, a more detailed study is necessary. Especially the execution of the various tasks mapped on the RISC core might prove to be a too heavy burden within the cycle budget of a single RISC core. To circumvent this problem, it could be considered to extend the proposed implementation with an additional RISC core to distribute the computational workload. Such an extended implementation still is cost-effective, due to the small die size of a RISC core in current technology.

Since the functionality of the MPEG-4 decoder that is mapped onto dedicated hardware can be considered as a superset of the MPEG-2 functionality, the presented MPEG-4 decoder can be designed such that it is able to decode MPEG-2 bitstreams as well. In fact, the hierarchical architecture template has already been exploited to perform MPEG-2 encoding and decoding with a restricted set of hardware modules suitable for video coding only. The mapping study of the MPEG-4 decoder proves the flexibility and extensibility of the media-processing platform. This platform enables an effective HW/SW co-design yielding a high performance density.

ACKNOWLEDGMENTS

The authors would like to thank Jos van Eijndhoven, Martijn Rutten, and Koen Meinds from Philips Research for their contribution and suggestions to this work.

REFERENCES

1. M. Budagavi, *et al*., "MPEG-4 Video and Image Coding on Digital Signal Processors", *Journal of VLSI Signal-Processing Systems for Signal, Image, and Video Technology*, **23**, pp. 51-66, 1999.
2. S. Bauer, *et al*., "The MPEG-4 Multimedia Coding Standard: Algorithms, Architectures and Applications", *Journal of VLSI Signal-Processing Systems for Signal, Image, and Video Technology*, **23**, pp. 7-26, 1999.
3. M. Berekovic, *et al*., "Instruction Set Extensions for MPEG-4 video", *Journal of VLSI Signal-Processing Systems for Signal, Image, and Video Technology*, **23**, pp. 27-49, 1999.
4. H.J. Stolberg, *et al*., "The M-PIRE MPEG-4 DSP and its Macroblock Engine", *Proc. of the IEEE Int. Symp. on Circuits and Systems*, **2**, pp. 192-195, May 2000.
5. S. Dutta, *et al*., "Architecture and Implementation of a Single-chip Programmable Digital Television and Media Processor", *1999 IEEE Workshop on Signal Processing Systems. SiPS 99. Design and Implementation*, Taipei, Taiwan, pp. 321-330, 20-22 Oct. 1999.
6. D.C. Wyland, "Media processors using a new microsystem architecture designed for the Internet era", *Media Processors 2000, Proceedings of the SPIE*, Vol. 3970, San Jose, CA, USA, pp. 2-15, 26-28 Jan. 2000.
7. S. Sudharsanan, "MAJC-5200: A High Performance Microprocessor for Multimedia Computing", *Proceedings Memory Wall Workshop*, Vancouver BC, Canada, June 2000.
8. P.H.N. de With, *et al*., "A Video Display Processing Platform for Future TV Concepts", *IEEE Trans. on consumer electronics*, **45**, pp. 1230-1241, 1999.
9. E.G.T. Jaspers and P.H.N. de With, "Architecture of Embedded Video Processing in a Multimedia Chip-set", *Proceedings of IEEE Int. Conf on Image Proc., ICIP 99*, Kobe, Japan, **2**, pp. 787-791, 1999.
10. V.M. Bove and J.A. Watlington, "Cheops: A reconfigurable Data-flow System for Video Processing", *IEEE trans. on Circuits and Systems for Video Technology*, **5**, pp. 140-149, 1995.
11. E.G.T. Jaspers and P.H.N. de With, "Bandwidth Reduction for Video Processing in Consumer Systems", *IEEE Trans. on Consumer Electronics*, submitted for publication.
12. M. Schu, *et al.*, "System-on-silicon Solution for High Quality Consumer Video Processing – The Next Generation", *Digest of Technical Papers of Int. Conf. on Consumer Electronics*, Los Angeles, CA, USA, pp. 94-95, 19-21 June 2001.
13. B. Davis, *et al*., "DDR2 and low-latency variants", *Proceedings Memory Wall Workshop*, Vancouver, BC, Canada, June 2000.
14. D.A. Patterson and J.L. Hennessy, *Computer Organization & Design, The Hardware/Software Interface*, Morgan Kaufmann Publishers Inc., 2nd Ed., San Francisco, CA, USA.
15. B. Khailany, *et al*., "Imagine: Media Processing with Streams", *IEEE Micro*, pp. 35-46, 2001.
16. S. Rixner, *et al*., "Media Processors Using Streams", *Proc. of the SPIE*, **3655**, pp. 122-134, 1998.
17. G. Kahn, "The Semantics of a Simple Language for Parallel Programming", *Proc. of Information Processing '74*, Amsterdam, The Netherlands, 1974.
18. E.A. Lee and T.M. Parks, "Dataflow Process Networks", *Proc. of the IEEE*, **83**, no. 5, 1995.
19. F. Pereira, "MPEG-4: Why, What, How and When?", *http://leonardo.cselt.it/icjfiles/mpeg-4_si/2-overview_paper/2-overview_paper.htm.*
20. C. Herpel and A. Eleftheriadis, "MPEG-4 Systems: Elementary Stream Management", *Signal Processing Image Communication (Netherlands)*, **15**, pp. 299-320, 2000.
21. J. Signès, Y. Fisher and A. Eleftheriadis, "MPEG-4's Binary Format for Scene Description", *Signal Processing Image Communication (Netherlands)*, **15**, pp. 321-45, 2000.
22. ETS 300 744, Digital Video Broadcasting (DVB), *Framing Structure, Channel Coding and Modulation for Digital Terrestrial Television, European Telecommunications Standards Institute*, Sophia Antipolis, France, 1997.
23. R. Koenen, "Profiles and levels in MPEG-4: Approach and overview", *http://leonardo.cselt.it/icjfiles/mpeg-4_si/11-Profiles_paper/11-Profiles_paper.htm*.
24. S. Rathnam, G. Slavenburg, "Processing the New World of Interactive Media", *IEEE Signal Processing Magazine*, **15**, pp. 108-117, 1998.
25. M. Berekovic, *et al.*, "A Multimedia RISC Core for efficient Bitstream Parsing and VLD". *Proc. Of IS&T/SPIE Conference: Multimedia Hardware Architectures*, Feb. 1998.
26. S. Howell, "OpenML V1.0 Specification", July 2001, *http://www.khronos.org*.
27. M. Berekovic, *et al.*,"Architecture of an Image Rendering Co-processor for MPEG-4 Systems", *Proc. on Int. Conf. on Application-Specific Systems, Architectures, and Processors*, Boston, MA, USA, pp. 15-24,July 2000.