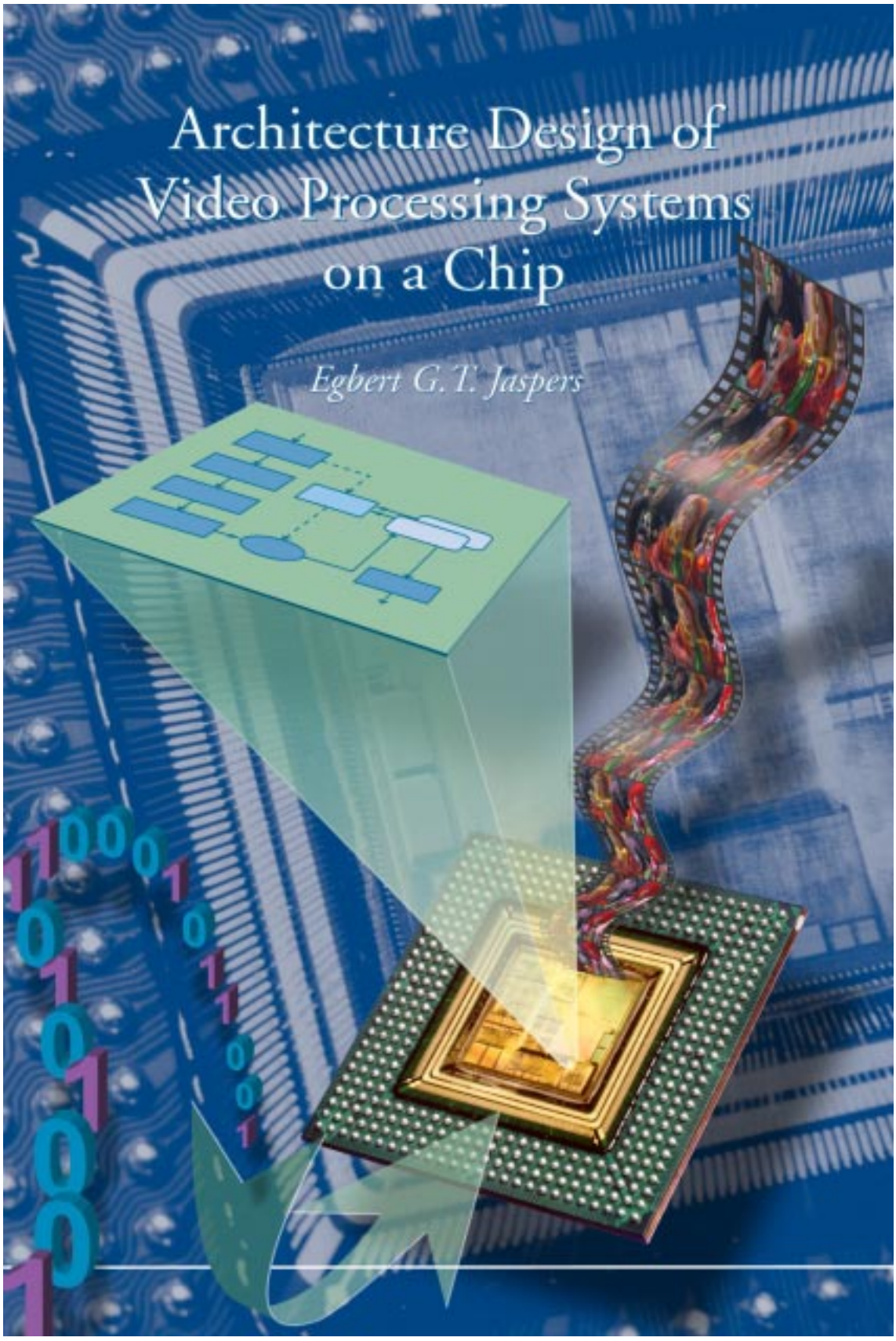
Architecture design of video processing systems on a chip


Architecture Design of
Video Processing Systems
on a Chip
Egbert G. T. Jaspers

Architecture Design of Video Processing Systems on a Chip

E.G.T. Jaspers

The work described in this thesis has been carried out at the Philips Research Laboratories Eindhoven, the Netherlands, as part of the Philips Research Programme.

Printed by: Eindhoven University Press
Cover design: Henny Herps (Philips Research AV-service) and Egbert Jaspers
The chip on the cover is a video processing system, called the CPA that is described extensively in Chapter 4.

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Jaspers, E.G.T.
Architecture Design of Video Processing Systems on a Chip
Proefschrift Eindhoven Universiteit of Technology –
Met literatuur opgave – Met samenvatting in het Nederlands
ISBN 90-74445-57-8
Trefw.: video processing, flexibility, heterogeneity, memory, programming, hierarchical communication, algorithm-architecture codesign

---

Architecture Design of Video Processing Systems on a Chip

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Eindhoven Universiteit of Technology, op gezag van de
Rector Magnificus, prof.dr. R.A. van Santen, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
donderdag 24 april 2003, te 16.00 uur.

door

Egbert Gerarda Theodorus Jaspers

geboren te Nijmegen

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr.ir. P.H.N. de With
en
prof.dr.ir. J.L. van Meerbergen.

Copromotor:
dr.ir. J.T.J. van Eijndhoven

ISBN 90-74445-57-8

Simplex sigillum veri –
Eenvoud is de zegel van het ware
(Boerhaave, 1668-1738)

Voorwoord

HET is zover: ik mag een voorwoord schrijven. Ondanks de uitzichtloze momenten, de zware tijd en de enorme inspanning die het me gekost heeft om dit proefschrift te schrijven, voel ik me nu alleen maar trots. Na een langdurige opleiding van MAVO, MTS, HTS en TU heb ik met dit promotieonderzoek wéér een grens verlegd.

Veel dank ben ik verschuldigd aan prof.dr.ir. P.H.N. de With ofwel Peter, die als begeleider, collega en promotor altijd in me heeft geloofd. Hij heeft mij al sinds 1991 in vele stadia van mijn prille carrière bijgestaan. Tijdens mijn afstuderen voor de HTS was hij, als afstudeerbegeleider bij Philips Research, een van degenen die mij gestimuleerd hebben om mijn studie te vervolgen aan de Technische Universiteit. Nadat ik daar was afgestudeerd, heeft hij me bij Philips Research binnengeloodst, als jonge onderzoeker in de groep Televisiesystemen. Als één van de vaders van het TVP-systeem (hoofdstuk 4) heeft hij me bekendgemaakt met veel facetten van een systeemontwerp en heeft hij mijn interesse gewekt voor de architectuuraspecten van videobewerkingsalgoritmen. Ook tijdens mijn promotieonderzoek heeft hij me steeds ondersteund met wijze raad, oppeppers, adviezen en velen uren samenwerken achter het toetsenbord. Bekende uitspraken van Peter als “hier belanden we in de mud”, “tante Betje stijl”en “het dooft uit als een nachtkaars”hebben uiteindelijk geleid tot een proefschrift waar ik met recht trots op ben. Ik heb nog nooit iemand ontmoet die zoveel vertrouwen heeft in de mensen om hem heen, zoveel trots uitstraalt voor het vele werk dat we in al die jaren hebben verzet, en zo toegewijd is aan zijn vak. Peter, jouw vertrouwen, trots en toewijding zijn een enorme kracht geweest in het vormen van wat ik nu ben.

Het onderzoek dat de basis van dit proefschrift vormt en is uitgevoerd bij Philips Research Eindhoven, zou nooit mogelijk zijn geweest zonder de inspanningen van velen. In het TVP-project heb ik in een team gewerkt aan het ontwikkelen van videobewerkingsalgoritmen, en heb ik collega’s

terzijde mogen staan bij het implementeren in VHDL en zelfs bij het testen en demonstreren van een werkend systeem. In het Eclipse-project (hoofdstuk 7) ben ik betrokken geweest bij de architectuurdefinitie in een team van ervaren architecten. De ervaringen met alle mensen in deze projecten zijn onderdeel van mijn vorming en hebben in grote mate bijgedragen aan de resultaten, zoals die in een 25-tal publicaties zijn beschreven.

Ik wil prof.dr.ir. E.F.A. Deprettere van de Technische Universiteit Delft en prof.dr.ir. J.W.M. Bergmans van de Technische Universiteit Eindhoven bedanken voor hun deelname aan de kerncommissie en waardevolle suggesties ter verbetering van het proefschrift. Mijn speciale dank gaat uit naar prof.dr.ir. J.L. van Meerbergen van de Technische Universiteit Eindhoven en dr.ir. J.T.J. van Eijndhoven, die naast hun bijdrage als tweede promotor en copromoter ook als specialisten en collega's een bron van inspiratie waren. Waardering gaat ook uit naar prof.dr.ir. R.H.J.M. Otten van de Technische Universiteit Eindhoven, prof.dr.ir. T. Krol van de Technische Universiteit Twente, prof.dr. L.O. Herzberger van de Amsterdamse Universiteit en prof.dr. R. Männer van de Universität Mannheim in Duitsland voor hun rol als leden van de promotiecommissie.

Uiteraard ben ik ook dank verschuldigd aan het management van Philips Research Eindhoven, dat me de mogelijkheid heeft geboden voor dit promotieonderzoek. Ook wil ik niet voorbijgaan aan mijn naaste collega's van nu en in het verleden, die naast een beroepsmatige relatie ook hebben gezorgd voor een plezierige werksfeer.

Daarnaast wil ik mijn ouders en schoonouders een warm hart toedragen voor hun onmisbare steun en interesse. Ten slotte wil ik mijn gevoel uiten voor hen die het middelpunt zijn in mijn leven en van wie ik voor de realisatie van dit proefschrift een grote opoffering heb gevraagd. Het laatste half jaar bestond voor mij hoofdzakelijk uit werken, slapen, en mijn proefschrift afmaken. Hierdoor kwamen alle werkzaamheden en verantwoordelijkheden thuis neer op mijn vrouw Janine, die naast haar baan ook de handen vol had aan onze lieve kinderen Milan en Marin. Nee, het moeilijkste van de vele avonduren achter mijn laptop was niet de enorme inspanning, maar de tijd die ik niet kon delen met mijn familie. Janine, lieve schat, bedankt voor alle opofferingen. Zonder jou was het niet gelukt.

Contents

List of Tables

List of Figures

De nihilo nihil (Lucretius, c.99 BC – c.55 BC)
Nothing comes from nothing

Chapter 1

Introduction and motivation

THIS thesis discusses the analysis of video processing algorithms and architectures for Systems-on-Chip (SoC). The design of a system comprises an interdependent design process of both the architecture and the associated functional algorithms. The functional algorithms are constrained by the architecture, whereas the architecture design is derived from the functional requirements. This mutual balancing can be described as a multidimensional design space that can be explored by constraining the design parameters and making the proper tradeoffs. This chapter has two purposes. First, it discusses some trends and the necessity for more flexibility as a general introduction to the television application domain, from which architectural requirements are derived. Second, it summarizes the individual chapters, it motivates the thesis background, and it presents the publication history.

1.1 Problem statement

The principal theme of this thesis is the analysis of video processing algorithms and architectures for SoC. The definition of architecting according to Maier and Rechtin [1] is: *the art and science of designing and building systems.* Basically, a set of requirements describing *what* the customer wants, needs to be translated into a description of *how* these requirements are satisfied. Figure 1.1 show how in addition to the requirements, also a set of constraints and cost functions are input parameters for this system design process. This process is particularly complex due to the dependen-

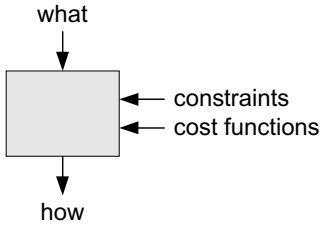


Figure 1.1: *The process of system design.*

cies of the design parameters, which are generally conflicting and require a proper balancing. For example, at first sight it seems straightforward to implement a system with sufficient flexibility. In addition, a huge amount of computation performance can be achieved by providing enough system resources. However, it becomes more difficult when implementing this at low costs. These constraints together with many more, are not orthogonal and make the design of a System-on-Chip (SoC) rather complex. This complexity becomes even more apparent due to the ever-increasing demand for more media processing and the continuously advancing VLSI technology.

The design of a System-on-Chip (SoC) involves a large design space and depends on a large set of design parameters such as, power dissipation, chip package, clock rate, memory requirements, timing constraints, communication bandwidth, network topology, silicon area, HW/SW partitioning, production volume, product differentiation, etc. Although a large part of the physics behind video processing techniques and the implementation is an exact science, the design of a system also depends on non-technical aspects such as time-to-market, business model, company capabilities, competitors, and even art[1].

The extremely large design space makes it impossible to present '*the*' architecture for a given application. A recently asked question about whether this thesis shows how a consumer-electronics company should derive solutions for TV processing is therefore poorly conditioned. To answer this

[1]The design of a system, satisfying a large set of requirements, shows many analogies with practicing art and is partially based on non-technical reasoning such as heuristic rules, intuition, company culture, business model, etc. This observation was already recognized in the early nineties by Rechtin [2] and was later even formulated explicitly by Maier and Rechtin [1] in the following quote: .... *"In contrast with science-based methodologies, the art or practice of architecting – like the practice of medicine, law, and business – is nonanalytic, inductive, difficult to certify, less understood, and, at least until recently, is seldom taught formally in either academia or industry. It is a process of insights, vision, intuitions, judgement calls, and even taste."*....

question, the full context should be known, i.e. for what time frame, for what performance, what functions, technology process, compatibility issues, or even the capabilities of the designers. Hence, there does not exist a cookbook for constructing optimal architectures and the corresponding "baking of the chips". Nevertheless, the content of this thesis gives sufficient basics for understanding the problem area and ways to approach the realization of a design.

This thesis mainly discusses the aspects of a design process for a high-level video-oriented system, starting from analyzing the required video algorithms while considering all additional constraints. Notice that this only comprises a part of the total system design. Figure 1.2 schematically shows various steps of the total design process of the system. First, functional

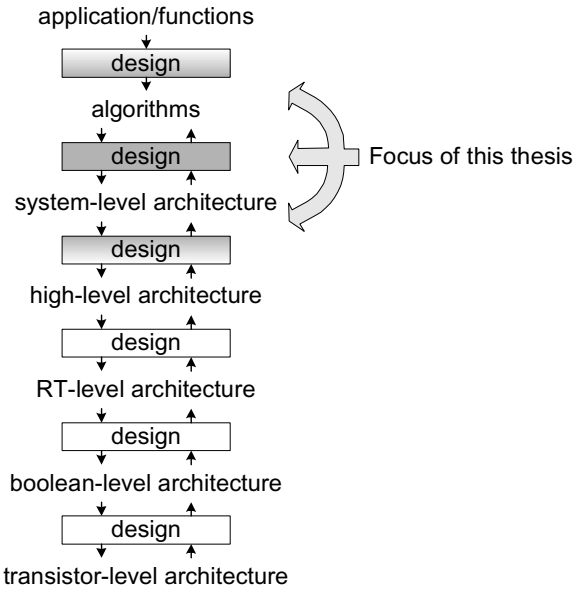


Figure 1.2: *Hierarchy in the design process of systems.*

requirements are translated into video processing algorithms. At the same time or at least highly iteratively, the system-level architecture is defined, describing the structure of the system in terms of building blocks such as processors. This codesign process is the main focus of this thesis. The next iterative design process determines the high-level architecture, which also describes the internals of the system-level building blocks. Subsequently, a design process determines the architecture at Register Transfer (RT) level. At this level, the architecture consists of clocked elements, registers, with

operations in between. As for the concept of time, we introduced a clock that is used to drive all registers. The design of the boolean- and transistor-level architecture add even more details to the system architecture.

In particular, this thesis discusses the impact of advanced video-processing applications on the architectures for high-end television, featuring various forms of flexibility or programmability. We have adopted high-end television processing as the key application field for our architectural studies. The resolution and quality of the pictures lead to significant demands on computing power and memory usage and bandwidth. Sampling rates range from 13.5 to 64 MHz, and individual pictures require memory space in the order of MBytes. As we will show later, computational requirements for such video signals go far beyond the power of single-processor system architectures as currently available in the PC domain.

The first part of this thesis focusses on TV processing for advanced features, with the aim to come to flexible and programmable system designs. Therefore, in this chapter we commence with determining the high-level functional requirements for such a system, followed by important architectural aspects. First, Section 1.2 presents the trends and developments of modern TV systems. Subsequently, Section 1.3 and 1.4 derive the functional requirements and the associated compute requirements. Finally, these functional requirements are used to determine the architectural requirements (Section 1.5), thereby also addressing non-functional issues such as ease of programming. The second part of this chapter motivates and outlines the remainder of this thesis and describes the background of the contents of the individual chapters and their publication history.

1.2 Trends and developments of TV systems

Let us now briefly discuss the architecture of traditional television systems and present some trends that require a new approach for system design. Up till now, architectures and the resulting chip sets for TV systems have been developed mostly as dedicated embodiments, having as much as possible the required memory for the functionality integrated or tightly coupled to it. The TV sets based on such a concepts have been introduced more than half a century ago [3]. The basic architecture of TV sets in those times already showed the traditional standard video path consisting of a demodulator, color decoder and the picture control block with a separate audio path. Since that time, the functionality that was offered to the customer has been increased significantly, without reconsidering the architecture of

the system. Feature extension boxes have been added to upgrade the traditional system to high-end TV sets currently available in the consumer market. Figure 1.3 shows an example of such a high-end TV architecture. In addition to the conventional TV it shows functionality for Teletext, 50-to-100 Hz conversion, PALplus decoding, an additional Picture-in-Picture (PiP), dual-screen, new sound standards and more recently, digital transmission standards based on MPEG. All these new features are an extension of the basic TV architecture half a century ago, resulting in a suboptimal structure. Because each function uses its own memory device with a dedicated communication path, the architecture becomes very rigid and the memory organization becomes cost inefficient.

Obviously, the realization of dedicated solutions is motivated by the continuous pressure on cost reduction, so that only the strictly required functionality is implemented without extensions or flexibility. This approach can be very beneficial when development costs are negligible and flexibility is required only within the product. Obviously, low costs are important, but also flexibility is and will become even more important.

Chapter 2 shows some architecture examples that feature the tradeoff between flexibility, cost, and performance for a certain application domain. Although it seems straightforward that the required applications could be implemented on a fully programmable multimedia processor, thereby enabling sufficient flexibility, the cost constraint is still not met. In Chapter 3 it becomes clear that a fully-programmable general-purpose processor(s) is not feasible for a cost-efficient solution. To determine the requirements for a TV system architecture, developments in this domain are briefly outlined.

- *Programmability* – The introduction of Internet and PC-like applications, such as information search and retrieval, multimedia playing, and games will have a clear impact on the diversity within a new TV architecture and points directly to much more programmability than was usual in the past. Another convincing motivation is that the coding standards become increasingly suitable for fully-programmable architectures, i.e. based on software implementations. Let us illustrate this with an example. The successor of the well-known MPEG-2 standard is likely to be the MPEG-4 standard. In this standard, images are not treated as rectangular frames or arrays of pixels, but an image is seen as an ensemble of *objects*. Every object is then coded and processed independently. This allows individual processing of the objects which are naturally of variable size and quality. A processing architecture for such a standard is not easily designed, but it is

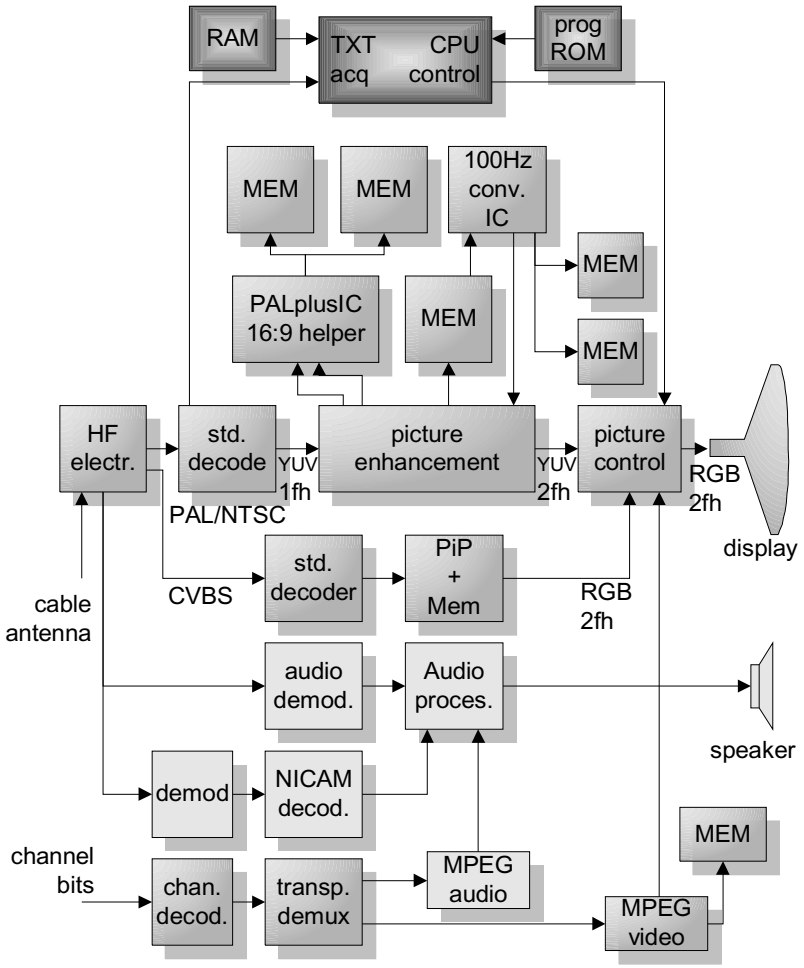


Figure 1.3: *The HW architecture of a conventional high-end TV set.*

plausible that at the top layer of the encoder and decoder, a highly programmable control system will be needed to dynamically manage the resources of the memory and functional units, to control the signal flow through the system and to compose an image at the receiver side from individually transmitted and coded objects. Furthermore, since the standard only specifies the syntax and semantics of a coded bitstream, the encoder is free to encode content in many different ways. For example, for the same video content, it can use segmentation into objects in combination with sprite coding, or it can code all objects in different Video Object Planes (VOPs), or it can render

the object texture onto an 2-D or 3-D mesh and animate the vertices of the mesh to portray the motion of the object. Since the method of encoding is done by the producer or broadcaster and is not known in advance, the decoding and composition process in the TV receiver asks for a flexible architecture, capable of decoding any format.

- *Flexibility* – The use of the above-mentioned Internet and PC-like applications implies that several video windows should be displayed simultaneously, rather than a single broadcast stream. For example, the background or a separate window could be used for TV signal watching, where in the second window, Internet (TV) could be displayed for additional information. It goes without saying that this has clear implications for a new architecture and require sufficient flexibility. Another reason for more flexibility is the increasing number of standards from international bodies. A DVD player is capable of decoding different audio standards such as AAC, AC3, DTS, Dolby Prologic, SACD, MP3, etc., but handles only one at the same time. Programmable or reconfigurable architectures provide satisfactory solutions to cost-effectively deal with this large amount of functionality. Moreover, often such standards comprise an extensible and scalable system. Parts of the functional behavior are strict and do not allow any deviation, whereas in other parts the user is allowed to configure the system accordingly to personal interest. For example, an MPEG bitstream allows the insertion of any user data and can be used for proprietary coding improvements, various forms of scalability, error resilience, or any other purpose. A flexible architecture framework is then the only way to cope with the as yet unknown additions.

- *Reuse* – Clearly, Systems-on-Chip become increasingly diverse and complex. This trend is simply due to the continuous improvement of transistor density in CMOS technology. Currently, chips for consumer electronics are designed containing up to 10M gates, having a computing power of 10 GOPS (Giga Operations Per Second) [4]. For such complex systems, it is not possible to start from scratch while having a short time-to-market. This requires a modular approach. Similarly, reuse of the architecture is also attractive to increase the range of potential products. For example, a related application area for a flexible TV system is the set-top box for digital TV reception. In this area, digital video-broadcast decoding is required with various forms of video pre- and postprocessing. Some modules, such as signal enhancement (noise, sharpness) may be applicable to both application areas, whereas others, such as MPEG decoding (compared

to PAL/NTSC) are fundamentally different. In any case, a subset of processing functions and the communication infrastructure of a new architecture together with memory access and control units can be reused. This would lead to a faster development of the secondary application field.

- *Reconsideration of TV functions* – The advent of digital television brings different picture characteristics than the conventional PAL and NTSC signals. Presently, the introduction of DVD players for movie playback is highly successful. This system is based on MPEG-2 coded video signals. Another application of digitally compressed video signals is the reception of Digital Video Broadcast (DVB) and digital cable TV. Such a new TV technology will inevitably push the quality of the conventional TV to a higher level. The new digital processing, such as decoding of MPEG-2 streams, will also lead to performance reconsideration of the conventional video enhancement algorithms, like noise reduction and sharpness improvement. Furthermore, reconsideration of the video processing functions is required for the pixel-based graphics in the afore-mentioned Internet and PC-like applications in the TV domain. Since computer graphics are artificially generated and individual pixels are addressed, the quality and performance of the conventional video processing functionality should be suitable to allow this type of signals. For example, sampling-rate conversion for up- and down scaling of the bandwidth-limited video pictures may introduce ringing and aliasing artifacts when applied to synthetic graphics.

- *Centralized off-chip memory* – As was presented earlier, the traditional TV systems contain many independent modules with features, each having its only external memory device. With respect to the total system, the distributed memory devices form a significant part of the costs. A system that would have a centralized shared memory is much more cost effective. However, such a system requires a flexible communication infrastructure to connect all video processing modules to the memory. Although this approach reduces the costs, it often remains a critical issue in system design. In many recently introduced media processors, the memory part is actually the bottleneck in system performance. Consequently, on-chip distributed memory may still be attractive. Communication and particularly the memory increasingly dominates the costs in the design. For this reason, we elaborate on optimizing the memory usage for video processing in Chapter 5 and 6.

In the following section, it is our objective to convert the aspects discussed here into a series of technical implications for the design of a new architecture.

1.3 Functional requirements

The aforementioned system aspects can be converted into a number of requirements, concerning new TV architectures.

1. It must be possible to monitor several video streams on a TV display with flexible windows, together with graphical information.

2. Conventional algorithms for TV video enhancement need to be reconsidered and redesigned for digital TV, i.e. MPEG-decoded applications.

3. If several video streams have to be displayed, appropriate video scaling is required, suitable for both high-quality video and graphics signals.

4. Video processing and pixel-based graphics will be used side by side or in mixed form. Architectures should be flexible in using various sources with different quality.

Let us now discuss these requirements in some more depth and commence with the multi-window aspect. A realistic implementation example of how an extra video signal is included is shown in Figure 1.3. A well-known feature is Picture-in-Picture (PiP), where a small video window is displayed in upper left or right corner of the display. Up till now, the required processing is mostly realized with additional dedicated hardware [5], such as a second (e.g. PAL-) decoder, and low-cost filters and down sampling for signal compression. A field memory is required for synchronizing the PiP window with the primary video signal. The extra hardware is usually minimized to limit costs, thereby accepting some quality loss. In a new architecture, it will be interesting to focus on *hardware sharing*, so that e.g. signal enhancement processing like noise reduction and sharpness improvement for a secondary signal, are executed on the same hardware that is being applied for the primary video signal.

Figure 1.3 also shows that alternative processing, such as PALplus in Europe, (or equivalently EDTV in Japan) and a 100-Hz up-conversion for large area flicker reduction, may be added.

An expensive part of this extra signal enhancement processing is the off-chip memory usage: it is distributed among the applications, and if the application is switched off, the memory cannot be reused. Similarly as with the signal processing functions, *memory sharing* can be pursued here to save system costs. This is particularly true for functions that are not used simultaneously. For example, when advanced graphics are used, it may not be required to have optimal picture quality of a PALplus signal at the same time. Thus the memory for full-quality decoding of the PALplus signal can be used for the graphics processing. Without the graphics, the memory can be used for the extra sideband video information transmitted in PALplus format.

The second point mentioned above refers to the new standards that are gradually introduced in the TV application domain. Digital Video Broadcast (DVB) is based on MPEG-2 coding, which can provide a large range of bit-rates for transmission. This means that also high bit rates will be possible and it has been proven that the MPEG-2 coding algorithm can provide excellent quality. For example, the first generation of DVD players which is also based on MPEG-2 coding, shows a quality far exceeding regularly received broadcast signals. As a consequence, an upward bias in image quality for secondary signals in the TV environment can be expected. The actual processing functions are based on the average quality of conventional broadcast signals, so that they need to be adapted to the quality level of high-quality digitally coded video signals. Moreover, the noise contained in digitally compressed signals differs clearly from the noise found in analogue broadcast signals, so that an adaptation of signal enhancement functions will be required in any case.

The display of several signals on a single display poses the question of the preferred display area usage desired by the TV viewer. This problem has a clear impact on the user interface and convenience experienced by the viewer, but this is beyond the scope of this thesis. Nevertheless, a direct technical requirement is implied by the aforementioned problem statement: the various signals need to be scaled (down) in size to offer them simultaneously to the TV viewer. Since the preferred image sizes are not known in advance, the best solution is to implement video scalers that take the scaling factor as input parameter, thereby offering flexibility in usage to the consumer. It will be shown later that the desired flexibility can be enhanced by extending this concept to upconversion (up-scaling) as well, so that digital zooming on a particular signal is also possible.

An additional aspect of scaling is that one of the extra video signals may be based on computer-generated video, i.e. graphics. In graphics signals, individual pixels can be addressed and filled with arbitrary colors. If such a signal is considered as a video signal, it is characterized by spurious high-frequency components in the signal spectrum, because extreme signal transitions can take place from one pixel to the other. Since video scaling also requires filtering, the selection of the corresponding filters and the scaling principle should be chosen carefully, since it will not be accepted that the intrinsic sharpness of graphical images is lost after a video scaling nor that ringing or aliasing artifacts appear. Hence, flexibility in video scaling for alternative types of video such as graphics is required, so that the original quality is preserved. The quality setting of the scalers should also cope with the high-quality video signals resulting from MPEG decoding.

Another point of concern in the new architecture is the optimal insertion point for digital TV signals. Depending on the signal quality, it is sometimes required to insert a signal at different stages of the signal enhancement processing chain. More generally, it is desirable that functions can operate on signals at arbitrary positions in a *task graph* (order of signal processing functions). The new architecture should enable more flexibility in these system aspects.

1.4 Computational effort

From the previous section it is clear that besides low cost, flexibility is highly required. Although flexibility may limit the costs in some sense, adding too much flexibility will undoubtedly result in the opposite effect. To get a rough indication of the system cost, we analyze how much computing power should be realized. A number of typical TV signal-processing functions were studied and computational and memory requirements were evaluated. Table 1.1 shows the intrinsic processing power of several TV functions at standard definition (SD) resolution for a realistic currently available high-end system [6]. With respect to operation counting, additions, multiplications, pixel read and writes, etc., are considered as single operations. Note that the functions in the table are not standardized unlike e.g. MPEG decoding or PALplus decoding. Thus, the table does not indicate the exact algorithm that is used, nor the quality of the result. Chapter 3 shows that different algorithms of similar functions can easily have an order of magnitude difference in the operation count. The fourth column shows the amount of memory or cache required. Here it is assumed that the video data can be retrieved or stored by single reads and writes (which

Table 1.1: *Expenses of various TV functions.*

Function	Operations per Second	Bandwidth MByte/sec.	Memory/ Cache
H zoom / compress	400 MOPS	38–64	samples
V zoom / compress	400 MOPS	38–96	lines
Filters, Comb filters	200 MOPS	64–128	samples-field
Advanced peaking	650 MOPS	32	lines
Color transient improvement	300 MOPS	48	samples
Dynamic noise reduction	500 MOPS	64–128	field
MC-100 Hz	2–4 GOPS	192-256	2–3 fields
Color space	150 MOPS	80	None
Teletext conversion	10 MOPS	48	> field
Adaptive luminance	60 MOPS	32	1 kByte

is optimistic). Moreover, it is assumed that field and frame memories are stored in the background memory thereby increasing the data communication bandwidth of the function, which is indicated in the third column. For these bandwidth numbers, it is assumed that data can be loaded or stored by read and write operations without any bandwidth overhead or any stall cycles. For a normal TV application with 50-to-100 Hz conversion, Picture-in-Picture (PiP), noise reduction and aspect-ratio conversion, the amount of operations already exceeds 6 GOPS (Giga operations per second). This is not readily implemented cost-effectively on a general-purpose processor. Such processor would give most flexibility, maximal reuse of the functional units (in this case fine-grain operations) and the memory organization is effective for such a solution, however, the sequential nature of processing limits the amount of computational power within a given IC technology and increases the power consumption, thus the system costs.

1.5 Architectural requirements

Adding flexibility in a system for future TV applications adds complexity and with this, costs such as silicon area and power dissipation will increase. To limit the costs of a flexible system, the following architectural requirements can be defined.

Parallelism – The integration of more functionality into a single SoC increases the computational requirements for the system. Simply performing

all operations in a sequential order does not satisfy the real-time constraints, since this would require unrealistic processing speed and power consumption. Hence, a large amount of parallelism is needed instead.

Communication topology – Once we have a system that features a large amount of functional processing units performing parallel computations, a communication infrastructure needs to be provided. Input data has to be conveyed to the functional units in the correct order and the final result is delivered, synchronous with the display. Furthermore, data exchange between the functional units need to be provided. The design of the communication network depends on the amount of parallelism, the granularity of data packets, the granularity of the synchronization, and the signal flow through the processing units. Generally, this part of the design is very costly because considerable buffering and wiring in the chip is involved.

Memory – Memory is an inherent part of data communication. Communication between functional processing units is provided by means of a memory that is shared by a data-producing functional unit and one of more functional units that consume the data. This memory can serve two purposes. Either the schedules between the processing units are fixed, i.e. the memory only provides a fixed delay, or the memory is used to decouple the functional processing units allowing less constrained scheduling. In both cases, the memory is only shared by two functional processing units. However, from Section 1.3 it can be concluded that from an application point of view, it is desirable to convey the video signals through the processing units in arbitrary order. To allow this type of flexible routing of data, the above-mentioned shared memory should be shared by all functional processing units. This thesis shows two example systems with different solutions for this. Chapter 4 outlines a system with small distributed FIFO memories that are connected to the inputs and outputs of the processing units. Subsequently, a switch-matrix communication network provides the sharing of the memories between any pair of functional processing units, depending on the desired task graph. Alternatively, Chapter 7 discusses an architecture with a centralized shared memory, connected to all functional processing units by means of a bus architecture.
Apart from the memory for intercommunication, memory is also required for the signal processing itself, depending on the function. For example, temporal noise reduction requires video data from both the current video picture and pictures from the past. Hence, memory is necessary to delay the video signal. Section 1.3 states that for flexibility at low cost, it is most beneficial to share these type of memory resources. Consequently, this re-

quires a communication network that allows access to a shared memory for all associated functional processing units.

Hardware-software codesign – The problem statement in Section 1.1 basically shows the trend of the ever-increasing demand for flexibility. As a result, system implementations will shift from dedicated hardware to more software-oriented systems. Because this trend is conflicting with the desire for low-power solutions, the amount of additional flexibility should be balanced carefully. Two different dimensions for hardware-software partitioning can be distinguished. The first dimension consists of functional partitioning. For example, MPEG decoding can be provided in hardware, whereas a successive histogram modification of the decoded result is applied in software. The second dimension covers the granularity of the functionality. For example, high-level software can start/stop a Picture-in-Picture feature in a TV set. However, at a low granular level, software may control multiply-accumulate operations in a video scaling function. Summarizing, applications can be decomposed into hierarchical levels of functional granularity, where each level can be implemented in hardware or software. The optimal choice for hardware-software partitioning depends on design constraints and the architectural analysis and hence, commonly results in a heterogeneous architecture (unless the constraints require homogeneous structures) with dedicated hardware, fully programmable processors, and hybrid combinations of those.

Programmability – Programmability of an architecture is directly related to the flexibility of a system and is therefore an important topic. Although this is not a central theme of this thesis, it implicitly appears in various chapters. From the thesis it will become clear that current and near future media-processing systems will be based on heterogeneous architectures for high-performance computing, having sufficient flexibility, and at a relatively low cost. To provide the application programmer with sufficient abstraction from the hardware, an adequate Application Programmers Interface (API) is required to hide the details of the architecture. However, from a programmability point of view, the programmer has to have knowledge about the details of the architecture for efficient programming of application-specific systems, whereas general-purpose systems generally do not require this. Note that this does not imply that general-purpose systems are more easy to program, because all functionality needs to be implemented in software as opposed to application-specific systems.

1.6 Outline of the thesis

This thesis is divided in three parts. The first part shows a historical evolution of general-purpose processors towards more powerful application-specific computing systems (Chapter 2). Moreover, it describes the analysis of some example video processing functions to derive the architectural requirements as discussed in Chapter 3. Finally, it extensively elaborates on the architectural analysis and implementation of an existing flexible television processor system (Chapter 4). Subsequently, the second part consisting of Chapter 5 and 6, elaborates on application-specific memory usage and the corresponding bandwidth, because this topic is a highly critical aspect in system design. The last part of the thesis proposes hierarchy in the communication infrastructure to solve the communication bottleneck and discusses the partitioning of the functionality in hardware and software. Since a growing number of fully-programmable processor cores in combination with more application-specific hardware in SoC is inevitable, the architecture should enable such mixtures of different types of hardware units.

Chapter 2 gives a short overview of the basics of computing architectures. Starting with a sequential general-purpose processor, parallelism is increased thereby making a tradeoff between costs and the amount of application-specific hardware. After the introduction of instruction-level parallelism in a Very-Long-Instruction-Word architecture and task-level parallelism in a multiprocessor system, a section is devoted to the relation between the amount of parallelism that can be exploited and data dependencies. Subsequently, the chapter discusses how the granularity of control (e.g. instruction level, task level, or function level) is related to the potential amount of parallelism and the flexibility of the system. The last section of Chapter 2 elaborates on some example processor architectures for video applications to obtain insights in system design. These examples indicate technology trends and discuss the merits and shortcomings of the systems.

Chapter 3 presents the analysis of the required computing and memory usage of several example video-processing functions. First, some algorithms for advanced sharpness enhancement and video scaling are presented in detail. The analysis of these functions gives a first insight in the type of architecture that fits naturally to the algorithms. Obviously, the choice of an architecture also depends on many other constraints as already stated in the first section. To go one step further in determining the feasibility of an architecture, the resource requirements are also analyzed. For the sharp-

ness enhancement and the scaling algorithms, but also for noise reduction and 50-to-100 Hz conversion, the computational complexity, the memory bandwidth, and the required memory capacity is determined. Considering all constraints and the results from the analysis, the architectural requirements of a system can be defined more clearly.

Chapter 4 discusses the design of the so-called *TeleVision Processor (TVP)* system, starting with a short overview of the trends, followed by a brief description of the functional requirements. Subsequently, the chapter discusses the conflicting requirements of flexibility, together with high computational performance at low cost. First, the concept of a task graph is explained. Subsequently, a set of requirements is presented such as real-time constraints, the flexibility to construct different task graphs, and the capability to processes a minimum of two input streams. Moreover, an unified off-chip memory should be provided to support memory reuse and low system cost. The resulting architecture consists of two chips: a micro-controller chip with peripherals to perform tasks that are not time critical and event-oriented control, and a coprocessor-array chip for powerful video processing. The communication network of the latter chip provides a data-flow model for asynchronous and autonomous operation of the coprocessors. The subsequent sections of Chapter 4 elaborate on the implementation of the communication network and explain how internal and external communication is provided. The last part of the chapter discusses the functionality of the system. Among other issues, this part addresses the scalability to exchange system resources for picture quality. Moreover, it describes a large set of possible applications such as multi-window TV and includes an analysis to determine the required system resources. The chapter concludes with the merits of the system but also proposes some modifications for future systems.

Chapter 5 and 6 are devoted to application-specific memory usage and the bandwidth of data communication, because the desire for a shared centralized memory and flexible inter-processor communication often induces a bottleneck in the communication infrastructure. The centralized system memory is particularly expensive and continuously grows due to the increasing complexity of video algorithms. For example, 50-to-100 Hz conversion required one field memory in the past to do median filtering, whereas current frame-rate conversion algorithms require several frame memories to do film detection, motion-compensated de-interlacing and up-conversion. Another example is the popular MPEG-2 standard [7] which requires up to three frame memories. The newly developed H.264 standard [8] requires at

least twice that memory space. Hence, the memory is an important part of the system and we can expect that it becomes an even more dominant part of the SoC. Therefore, Chapter 5 and 6 elaborate on efficient utilization of memory resources, and in particularly the off-chip memory. Chapter 5 shows that significant gains can be achieved when matching the application-specific memory accesses with the behavior of the memory device. First, it explains the physics and the operational behavior of SDRAM devices and it gives a short prospect of future memory devices. Subsequently, the chapter deals with optimization of the memory efficiency by means of an MPEG-decoder example. Chapter 6 describes some application-transparent techniques to reduce the memory bandwidth. First, a straightforward embedded compression technique is presented, followed by a section on function-specific caching. Because the MPEG-2 decoder from Chapter 5 represents one of the most critical functions for memory access, it is again adopted as an example function. The last section of the chapter reflects a collage of several discussed techniques, such as double writing (Appendix B), embedded compression (Section 6.1), and caching (Section 6.2). It shows that most techniques are complementary and can be used in combination, resulting in tremendous memory bandwidth reductions.

Chapter 7 discusses the partitioning of video-processing applications on a heterogeneous multiprocessor system called *Eclipse*. Although the above-discussed TVP system and the presented solutions for interfacing off-chip memory are satisfactory, the ever-increasing complexity of SoCs requires a new design paradigm that deals with the costly communication resources, the desire for scalability toward following generations of the same system, and reuse of the design for different products. This observation leads to a hierarchical design of the communication infrastructure and the memory architecture. Chapter 7 proposes a close match of this hierarchical design with the hierarchy that is inherently present in video processing applications. This approach forms the basis of the Eclipse system which is thoroughly discussed for implementation of an MPEG-4 decoder system. Hence, after discussing the new insights and the architecture template of the Eclipse system, an overview is presented of some state-of-the-art MPEG-4 systems that are described in the literature. Subsequently, we explain the functionality of an MPEG-4 decoder. Then, the characteristics of each task in the decoder are analyzed, leading to distinct architectural requirements for each task. Finally, a partitioning over a mixture of fully-programmable processors and application-specific processing units is presented, followed by a hypothetical proposal for implementing the MPEG-4 decoder.

Chapter 8 concludes with the most important findings and provides a future outlook. First, we recapitalize the conclusions of the individual chapter. Most important is the observation that video processing algorithms and the corresponding architecture should be designed corporately to attain a cost-effective solution. Moreover, it is concluded that a hierarchical design of the communication infrastructure and the memory architecture is inevitable to deal with the ever-increasing complexity of SoCs. To show the significance of the contribution from this thesis, the design of a state-of-the-art television SoC is first briefly outlined, followed by a proposal for a similar system with a hierarchical design. This example clearly shows that the codesign of the embedded video processing functions and the architecture is and remains important for implementing cost-effective systems on a chip.

1.7 Background and motivation of the chapters

Most parts of the chapters in this thesis have been published in conference proceedings or scientific journals. In this section, we explain the origin of the chapters and their publication history.

Chapter 2 presents a historical overview of system architectures to indicate the trends in the design of systems for multimedia computing. These trends have resulted in large shifts in design methodologies. Whereas laborious low-level IC layout design of video processors with a high transistor density was the main challenge in the past, the current emphasis is on reliability, robustness, clock skew, timing closure, power consumption, development of high-level design tools, and software architectures focussing on reuse, scalability, portability, etc. Although general-purpose processors and dedicated video processors have been separate domains in system design, Chapter 2 clearly shows a convergence of both domains. General-purpose processors are being specialized for media processing with special instructions for vector operations on video algorithms, whereas the hardwired dedicated video processing chips are becoming increasingly programmable.

Chapter 3 elaborates on the resource requirements such as operation count and memory bandwidth. The chapter was included because the architectural requirements depend a lot on characteristics of the video processing algorithms, such as the computation complexity. To provide more insight in the type of computing that is involved in video processing algorithms, Chapter 3 first presents some example video functions in detail. The design of these video functions was achieved as part of the Television Processor

project at Philips Research, which is described in more detail in Chapter 4. Results of Chapter 3 were presented in the papers "A Generic 2-D Sharpness Enhancement Algorithm for Luminance Signals"[9] and "An Advanced Sampling Rate Conversion Technique for Video and Graphics Signals"[10], which were coauthored with J.G.W.M Janssen and P.H.N. de With and presented in 1997 at the IEEE International Conference on Image Processing and its Applications. Parts of these results were later reused in system-oriented papers (see below).

In the mid-nineties, the problem statement as portrayed in Chapter 1 was already recognized, resulting in a project for designing a Television Processor (TVP). The project was a collaboration between Philips Research, Philips Semiconductors, and Philips Consumer Electronics, and was initiated by amongst others the promotors of this thesis. The TVP architecture design is described in Chapter 4 and is inspired by the work of many project members. At the end, the project resulted in two working chips. The Coprocessor Array (CPA) chip (see cover page) and the Telecommunication and Control Processor (TCP) chip were successfully implemented in a demonstration setup. This demonstrator proved all concepts and clearly indicated the usefulness of the flexibility that was offered. Furthermore, the TCP chip was successfully produced under the name SAA7430. The first overview of the system was published in the IEEE Transactions on Consumer Electronics and was titled "A Flexible Heterogeneous Video Processor System for TV Applications"[11]. The paper was coauthored by P.H.N de With and J.W.G.M. Janssen and received a Chester Sall Award for the best papers of the IEEE CE Transactions in 1999. In the same year, another overview article [12], focussing more the memory bandwidth issues, was presented at the International Conference on Image Processing in Kobe, Japan. Subsequently, a set of three papers was published, each focussing on different aspects of the system. The first one, named "Chip-set for Video Display of Multimedia Information"[13], which was coauthored by P.H.N. de With, describes the functionality of the coprocessors in more detail and discusses a range of possible applications. The second paper, entitled "A Video Display Processing Platform for Future TV Concepts"[13], elaborates more on architectural issues such as the data communication and was published in the 1999 IEEE Transactions on Consumer Electronics. The third paper was presented at the IEEE International Conference on Solid-State Circuits and particularly focusses on the VLSI aspects. The TVP system consists of relatively tightly coupled coprocessors that operate independently and asynchronously. As a consequence, the Video Input and Output modules needed to be designed under strict timing constraints

compared to similar modules in synchronous systems and systems with very loosely coupled coprocessors (e.g. decoupled with frame buffers). Because the design of these specialized modules is beyond the scope of the thesis, only a brief overview of the research is given in Subsection 4.6.4. However, a more detailed paper was presented at the SPIE conference on Visual Communications and Image Processing 2000, entitled "Synchronization of Video in Distributed Computing Systems". The paper "Synchronization of Video in Distributed Computing Systems"[14] was coauthored with P.H.N. de With and one of the VLSI circuit designers, B.S. Visser.

During the architectural analysis of the video processing in the TVP system, it was recognized that the required memory bandwidth was an important and critical issue. This is motivated by the research of Chapters 3 and 4. During the design of another video-processing system, called Eclipse, which is discussed more thoroughly in Chapter 7, again memory traffic appeared to be an important concern. The Eclipse project aimed at designing platform technology for a broad range of applications, but a first experimental instantiation was limited to multi-channel MPEG decoding and encoding. As a part of this Eclipse project, several options for memory-bandwidth reduction were explored. Chapter 5 discusses the first option, which basically outlines the optimal mapping of video data into the physical memory addresses and some implementation optimizations for further efficiency improvement. Part of this work was published in 2001 in the IEEE Transactions on Consumer Electronics with the title "Bandwidth Reduction for Video Processing in Consumer Systems"[15]. Other parts were filed as patent applications [16] [17]. Other options for bandwidth reduction are presented in Chapter 6. First, a low-cost embedded-compression technique is explored. Feasibility of the concept has been presented at the SPIE Media Processors Conference in 2002, with the title "Compression for Reduction of Off-chip Video Bandwidth"[18]. Later that year, additional experiments revealed that also the picture quality could be maintained although a lossy compression algorithm was applied. These results were published in the Proceedings of the IEEE Benelux Signal Processing Symposium of 2002. The paper, called "Embedded Compression for Memory Resource Reduction in MPEG Systems"[19], was coauthored with P.H.N. de With. In the same chapter, a section is also devoted to video-specific cache design for reduction of memory bandwidth. As a an example, it focusses on memory access for motion-compensated prediction in an MPEG decoder. Because the outcome of this work is just recently available, it has not yet been published.

Chapter 7 elaborates on the hardware-software partitioning of video applications for SoC design. Part is based on experiences from the TVP project as discussed in Chapter 3 and 4 and is published in the proceedings of the "IEE International Conference on Image Processing and its Applications 1999" in an article, entitled "A Flexible Heterogeneous Video Processing Architecture for Powerful HQ Television Applications - Finding the optimal balance between HW and SW" [20]. However, the main part of the chapter is inspired by a case study for MPEG-4 decoding that was conducted for the Eclipse project. This project aims at designing platform technology for fine-grain multiprocessors which enables a mixture of tasks being performed in both hardware and software. The first sections of the chapter describe the adopted architecture and are based on publications with the members of the involved architecture team. First, the architecture was presented at the 2002 International Workshop on Parallel and Distributed Computing in Image Processing, Video Processing and Multimedia [21] by M.J. Rutten, J.T.J van Eijndhoven, E-J. D. Pol, E.G.T. Jaspers, P. van der Wolf, O.P. Gangwal and A. Timmer. Later, these authors published an improved version under the same title [22] in the magazine "IEEE Design & Test of Computers". The actual description of the hardware-software partitioning for the MPEG-4 decoder is described in the second part of Chapter 7 and is based on a paper that was coauthored with E.B. van der Tol and presented at the SPIE Media Processors Conference 2002, called "Mapping of MPEG-4 Decoding on a Flexible Architecture Platform" [23]. Later that year, this paper resulted in an invited lecture of the author of this thesis at the 2002 International Workshop on Parallel and Distributed Computing in Image Processing, Video Processing and Multimedia (PDIVM) [24].

1.8 The main contributions of the author

The research that forms the basis of this dissertation has been performed at Philips Research Laboratories in Eindhoven and would never been accomplished without the efforts of many colleagues. The participation in two projects, containing multi-disciplinary teams of algorithm developers, architects and VLSI designers, have resulted in the design and implementation of the complete systems. The first project, called *Television Processor* (TVP) has lead to the insights of Chapter 3 and 4. For this project, the author developed the sharpness-enhancement algorithm and specified the design of the corresponding coprocessor. Moreover, he specified the design of the video input and output modules. In a later stage of the design and during the testing of the system he was responsible for the application-related support. The second project, called *Eclipse* covered the design of

the system as described in Chapter 7. For this project, the author was shortly involved in the definition of the architecture and later on focussed on the application-specific functionality of the system. Among others, he analyzed MPEG-4 decoding for modelling of the application-specific co-processors and studied the feasibility of a cache for motion-compensated prediction in an MPEG decoder. Particularly the study on caching has lead to the discussions in Chapter 5 and 6.

respice, adspice, prospice (Walter Pach, 1883 – 1980)
examine the past,
examine the present,
examine the future

Chapter 2

Developments in video computing architectures

THE problem statement and requirements of new architectures for high-end TV, in the previous chapter, have clarified that a large computational power is required for TV and multimedia systems, especially in those systems where several video functions have to be executed in parallel. In this chapter, the objective is to look into computer and DSP architecture developments and learn from the techniques that have been implemented to improve computing power, communication and memory usage, while providing sufficient programmability. This chapter also serves as a partial literature overview and broadens the problem statement of designing a new programmable architecture for digital TV systems.

2.1 Introduction

In modern TV systems, a plurality of functions is operational in parallel to offer the high performance perceived. Therefore, this chapter concentrates on the system aspects of enabling parallelism in computing. Important aspects are the type of architecture of the system, such as SIMD systems, and provisions for parallel instructions (VLIW and superscalar engines) and computational units. Afterwards, additional aspects such as control overhead and the tradeoff between general-purpose computing and application-specific hardware together with programmability are briefly addressed.

A special part of the chapter is devoted to real examples of DSP processors, where the order of the processors is put in a historical perspective. These examples illustrate how specialized processors emerged and also portray to some extent the density improvement in IC technology. The overview is by no means complete, but its purpose is more to learn from possible shortcomings from the past and to collect attractive architectural aspects which are relevant for new or upcoming processor architectures. The overview ends with a multimedia processor, called Emotion Engine, which is one of the latest examples of flexible and powerful computing.

Some time is also spent on discussing other system aspects which have to be defined in a complete architecture, such as data communication, control communication for programming or flexibility features and the usage of memory in a high-performance architecture.

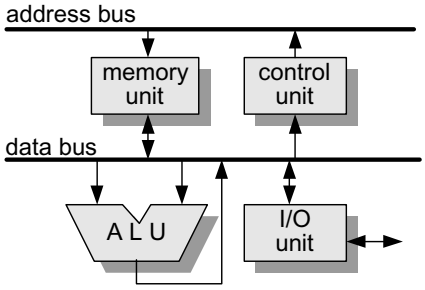


Figure 2.1: *Basic general-purpose computing architecture.*

2.2 Exploiting parallelism in computing systems

A standard general purpose-computer, e.g. an Intel 8080, has no parallelism above the level of fundamental operations such as addition, logic shift, etc. as shown in Figure 2.1. The control of the hardware in such a system is at the level of operations, and it is symbolized as the program of instructions. The most straightforward way to increase the amount of computational performance of such a system is by increasing the clock frequency of the processor. As a consequence of this performance enhancement, the power dissipation per logical gate P_g of the system increases according to:

$$P_g = C \times V^2 \times f_{max}, \tag{2.1}$$

where

$$f_{max} \propto \frac{(V - V_T)^2}{V}, \tag{2.2}$$

with V the supply voltage, V_T a constant threshold voltage, f_{max} the operation frequency, and C the capacity of the gates which is given by the feature size. If we assume the minimal supply voltage for the necessary operation frequency, the equations show that the power dissipation increases approximately with a power of three for increasing clock frequency (substitution of Equation (2.2) into Equation (2.1)). Obviously, this power dissipation is limited by the physical power durability, but can also be constrained for e.g. mobile applications, reliability, and package cost. Instead of increasing the operation speed to increase performance, it is also possible to increasing the amount of parallelism. This approach increases the total power dissipation only linearly with the added parallelism. Moreover, because parallelism increases the silicon area, the power dissipation per unit of area remains approximately equal and is therefore more easy to handle.

At a low level in a simple general-purpose processor which performs sequential processing, some forms of parallelism are already available, e.g. an adder in the ALU unit uses several gates in parallel to perform the addition of all bits. To further improve in parallel computing, the amount of functional operations in parallel can be increased.

Architectures can be classified in various forms of parallelism. Examples of enhanced parallelism are depicted in Figure 2.2, where either data and/or instruction streams are duplicated. The simplest form is called Single Instruction Single Data (SISD). More data can be processed with one single instruction, by using the instruction stream for an additional stream of data, leading to the so-called Single Instruction Multiple Data (SIMD) architecture. These architectures, control the parallel operations with a single instruction stream, i.e. a single thread of control. Often, these architectures are referred to as vector processors or array processors. An ALU of such a processor is able to process e.g. two 64-bit operands but also to process four 32-bit operands. Typically, it can use quad-words (1×64), double words (2×32), packed words (4×16) for audio and packed bytes (8×8) for graphics and video. As a trend, it can be noticed that the width of the operands increases as technology advances. Current designs, such as the PowerPC and the Pentium IV contain 128-bit operands and the first PowerPC has been introduced containing 256-bit operands.

More data streams can be processed simultaneously with multiple instructions (MIMD). This architecture is attractive if several independent functions have to be executed in parallel, but it is also the most complex system because multiple instructions basically imply multiple threads of con-

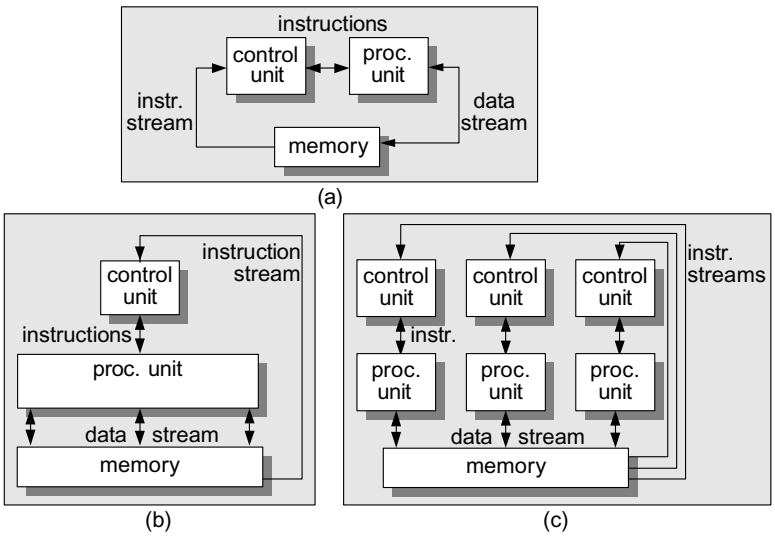


Figure 2.2: *Examples of increasing parallelism in computing from single-input to multiple output: (a) SISD, (b) SIMD, (c) MIMD.*

trol. Hence, SIMD is typically provided within a single processors whereas MIMD is typically provided by multi-processor architectures.

Besides the SISD and SIMD architectures, also other types of parallelism are introduced to increase the performance for single-threaded architectures. Examples can be found in the following architectures and concepts

- Von Neumann architecture,
- Harvard architecture,
- parallelism in time: pipelining,
- superscalar architecture,
- VLIW architecture.

In the Harvard architecture as shown in Figure 2.3, the data bus and the instruction bus are separated as well as the data memory and the instruction memory, thereby solving the *Von Neumann's bottleneck* [25] (two streams to the same memory). A double (modified) Harvard architecture contains a double data bus and data memory to retrieve two operands of an operation simultaneously using one instruction.

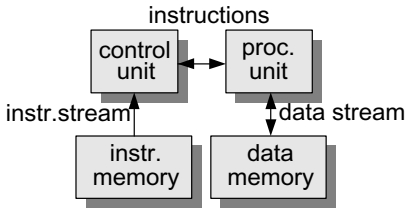


Figure 2.3: *Harvard architecture with separated data and instruction memory.*

The aforementioned architectures introduce parallel processing in *space*. Another concept of parallelism can be enabled by exploiting the dimension *time*. When an operation is split up into subsequent pipeline stages, which are separated by a predetermined amount of clock cycles, the clock rate can be increased. Although the the number of clock cycles for the operation has increased, the latency (expressed as an absolute time) remains equal. The benefit is that for every clock cycle, a new operation can be issued, resulting in a higher throughput due to the higher clock rate.

In a follow-up architecture, a single instruction stream is scheduled and partitioned over two or more independent processing-units by a single control unit at run-time. This *superscalar* architecture contains partially complex hardware to dynamically schedule the instructions over the separate instruction pipelines, and to perform some optimizations, such as branch prediction and speculative calculation. Example implementations of such an architecture are Pentium, PowerPC, and PA-RISC.

Other developments in instruction-level parallelism (ILP) have resulted in *Very Long Instruction Word* (VLIW) architectures. Like the superscalar architecture mentioned above, it is based on an single instruction stream that controls two or more independent processing units. A single full-width instruction is a concatenation or coded form of multiple instructions. Since the mapping of each small instruction onto the processing units is determined at compile time, no instruction partitioning and scheduling hardware is required. Also some optimizations like branch prediction, speculative calculation and loop unrolling, can be performed by the compiler. Example implementations of such architectures are Trimedia [26], MPact [27] and TMS320C62 [28]. Even though the superscalar and VLIW architecture are classified as single instruction architectures (single threaded), they feature instruction-level parallelism (ILP) and fit to a general template that is currently used for many implementations and is depicted in Figure 2.4.

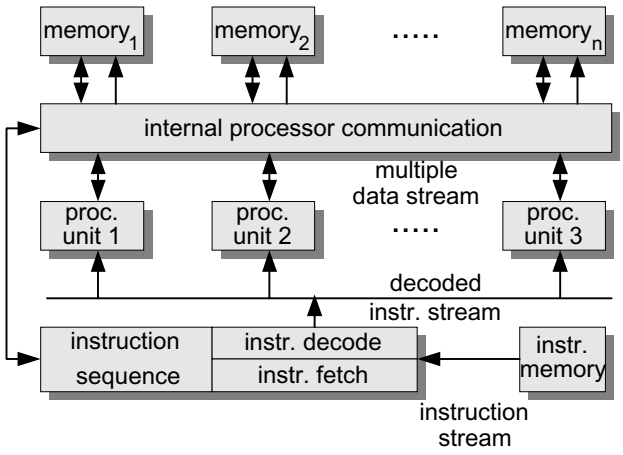


Figure 2.4: *Generalized template for architectures featuring instruction-level parallelism.*

2.3 Aspects of application-specific parallelism

As mentioned before, parallelism can be increased by simultaneously carrying out multiple operations like additions, multiplications, logic AND, etc. However, the complexity of the associated control increases rapidly. To some extent, this complexity can be hidden in a compiler, but evidently such an approach has its limits. The VLIW architecture provides an example of such a case in which a compiler performs complex scheduling of the instructions such that the resources are well utilized. Although good results can be obtained, the compilation into efficient micro code highly depends on intelligent programming of the algorithms.

The main reason for the increase of complexity with augmented parallelism results from the data dependent operations during processing, i.e. conditional instructions and branches. To explain this, let us consider for example the use of an MPEG encoder of which its processing tasks are mapped onto a imaginary processor. The processor has capabilities for unlimited parallel instructions and may carry unlimited parallel data. A multiplication in an inverse DCT (IDCT) function might be performed in parallel with an addition in the motion-compensation unit, because for some pictures the result of the inverse DCT and the motion-compensation (MC) unit is used to create a new reference picture. The compiler should be able to identify if and when the intermediate result of the addition and the multiplication are used to form a new result. Furthermore, the scheduling

of all additions in the MC and all the multiplications in the IDCT should be such that finally an inverse DCT result corresponds with the correct output of the motion-compensation unit. Even if the compiler would be able to look far ahead, it cannot determine whether it concerns a reference picture to be used in further coding iterations, or that data would be used in a different way. Such decisions are clearly data dependent and are contained in the statistics of the input bitstream. It can be concluded that instruction-level parallelism has its limits, even though this limit highly depends on the function. Commercially available VLIW processors feature five parallel issue slots [29] or more and critical loops in more application-specific processing functions can be identified that feature an ILP of 20 to 50. However, adding even more parallel functional units does not further increase the computation performance [30]. This problem is basically similar to the an observation of Amdahl [31], who states that the speedup of a multiprocessor system depends on the fraction of code that cannot be executed in parallel, according to:

$$\text{speedup} = \frac{1}{f + \frac{1-f}{p}} \ , \tag{2.3}$$

with f the fraction of code that cannot be executed in parallel, and p the number of parallel processors. Opposed to a linear speedup, Figure 2.5 shows how the speedup saturates for a larger number of parallel processors. The graphs shows that adding parallelism to the system is only useful if the parallelism can be exploited by the application.

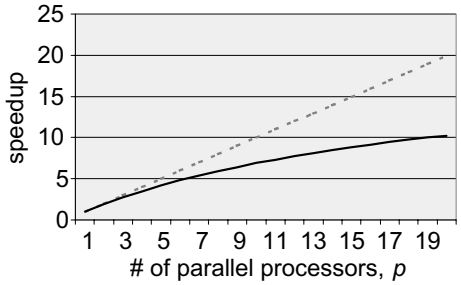


Figure 2.5: *An example graph of Amdahl's law.*

It can be concluded that data dependencies are an important hurdle for designing parallel systems. This problem can be combated by providing *task-level parallelism*. This technique does not replace the instruction-level parallelism (ILP), but rather adds an additional hierarchical level to the system. A task is defined as set of more fine-grain operations that are executed by a single thread of control. Thus, within a task parallel processing

is performed at a lower level in the hierarchy, e.g. additions, subtracts, and multiply operations. At higher levels, parallel processing is performed on task level. For example, Figure 2.6 shows how a TV application contains a hierarchy of functions, tasks, and fine-grain operations.

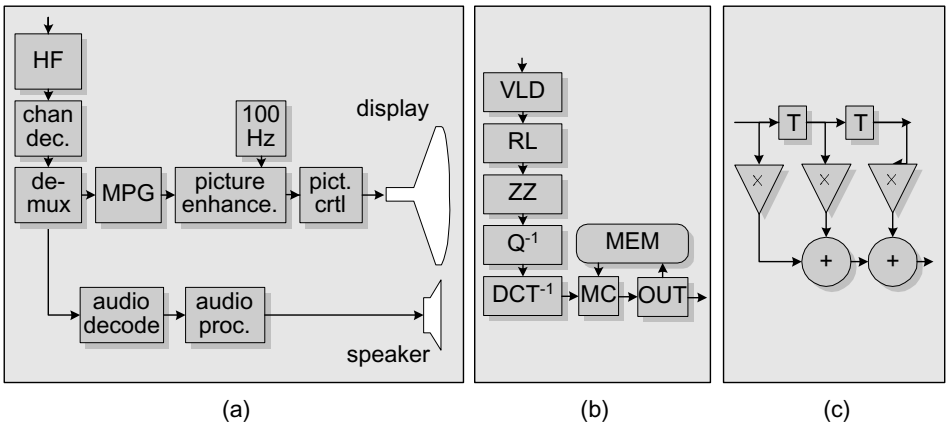


Figure 2.6: *Hierarchy of functional granularity: function level of a TV application (a); task level of an MPEG decoder function (b); fine-grain operation level of a filter task (c).*

Note that a hierarchical structure does not imply an implementation in hardware or software. A task can be implemented with CPU operations that are controlled with software, but can for example also be implemented with hardwired operations in a dedicated hardware block. However, the design of parallel systems becomes more and more complex when increasing the amount of flexibility. First, such systems imply the configuration of functionality at each level in the hierarchy. For example, a programmable task needs to be configured (loading with the correct software program) for the required functionality. Secondly, parallel systems that can be programmed with software at all hierarchical levels contain usually expensive communication networks. Such systems have a communication network to load operands, route them to the correct functional units in the correct order, and subsequently store the calculated results in the correct memory locations or registers. At the task level, software implements task switches on the processor cores, initializes the correct communication buffers in the memory that correspond to the task, and provides synchronization of executing tasks on different processor cores. At an even higher level, software configures the signal flow through the tasks (task graph), and dynamically adds or changes functions on the media processor. Chapter 7 shows that the hierarchy in the architecture of the communication networks matches

the natural hierarchy of applications. An example of such natural hierarchy in applications is a which contains functions such as channel decoding, demultiplexing, video and audio decoding, picture enhancement, etc. The functions consist of tasks such as variable-length decoding, inverse quantization, discrete cosine transform, etc. And again one level lower, the tasks comprise operations such as add and multiplications.

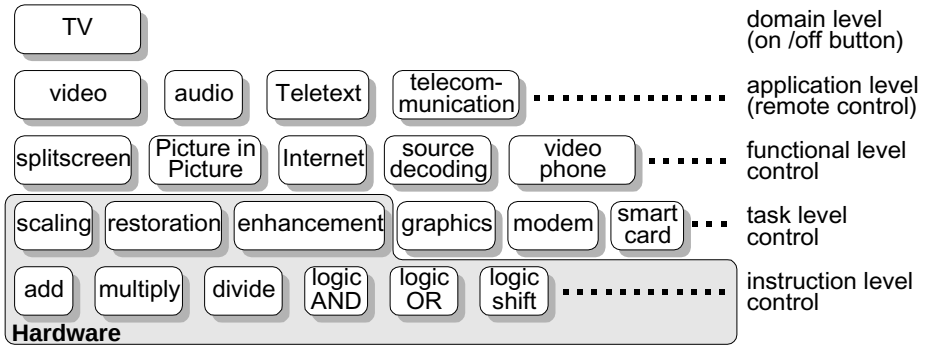


Figure 2.7: *Control hierarchy in a TV system.*

2.4 Parallelism and control

Let us first clarify the definition of hierarchy and the associated control by means of an example as depicted in Figure 2.7. The example system is used in a TV environment, where high-speed processing for video at low cost is very important. In the bottom row of the picture, fine-grain operations are depicted. The adjacent higher level represent the task level. As mentioned before, a task represents a set of operations that are executed by a single thread of control. Notice that such thread of control can be implemented by a sequence of instructions, i.e. a software program, but can also be controlled by hardwired connections between operations, i.e. dedicated hardware. Each task represents a separate thread of control. The hierarchical level on top of the task-level shows the granularity of functions. Functions represent a set of intercommunicating tasks that are executed by a single thread of control. Notice that this control results in more processing. For example a single command to start a task within a functions leads to the execution of many fine-grain operations. Hence, the amount of control per unit of computational power is different for each level. Within a task, control takes place at a small granularity such as operations, i.e. a relative large amount of control overhead. At higher levels in the hierarchy, control is performed on task level and thus shows much less control per

unit of computational power. Similar to the control of operations, tasks and functions, we can extend the hierarchy to even higher levels.

Because many functions in this domain are quite application-specific and flexibility and programmability is generally inefficient due to afore-mentioned control overhead, many tasks can be implemented with hardwired operations (not programmable), whereas tasks requiring flexibility are mapped onto a general-purpose CPU.

Apart from implementation in hardware or software, each hierarchical level also expresses a different granularity of parallelism. Within a task, many fine-grain operation can be executed in parallel, thereby providing ILP. Within a function, tasks can be executed in parallel, thereby providing TLP, etc. As a result, the amount of parallelism increases significantly.

The following list summarizes the different properties of the various levels in the control hierarchy as depicted in Figure 2.8 and is subsequently discussed briefly:

- amount of parallelism in processing;
- grain size of functionality;
- amount of control overhead;
- general-purpose vs. application-specific processing;
- amount of flexibility.

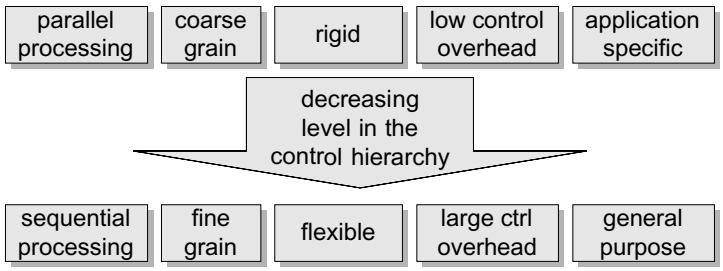


Figure 2.8: *System properties versus instruction grain size.*

Amount of parallelism
A logical gate is made from several transistors which are operating in parallel. For an adding operation, several logical gates are arranged in parallel. A convolution operation in a DSP or VLIW-processor executes several

multiplications and additions in parallel which consist of parallel gates that contain parallel transistors, etc. Since a level in the hierarchy is a subset of a higher level, the amount of parallelism increases.

Grain size of processing
Obviously, as the level of hierarchy increases, the grain size of the functionality also increases. A multiplication has less functionality than e.g. an MPEG quantizer task in which several multiplications are used to perform the complete function.

Amount of control overhead
To perform a DCT at task level, only one command or function call is required, whereas many instructions are necessary at the level of adders and multiplications. The amount of communication that is required for a sequence of instructions or commands is smaller at higher levels in the control hierarchy, as compared to the amount of processing. Thus the communication of high-level control has a low rate with respect to the data communication. This requires a different approach for implementing the high-level control communication as compared to the data communication.

General-purpose vs. application-specific processing
A task can be mapped on a dedicated processor or a general-purpose CPU, but once the hardware or CPU is configured for the functionality of the task, it remains fixed and it is controlled on task level. Therefore, at this level, the control becomes more application-specific.

Amount of flexibility
Because the functionality at higher levels in the control hierarchy is more application-specific, the flexibility decreases. A zig-zag scanning function in an MPEG decoder can perform only a limited amount of different scanning orders for a fixed block size. At a lower level, any scanning order can be programmed for any block size or even a different function can be implemented.

2.5 Examples of media processor architectures

2.5.1 Introduction to media processors

In the next chapter, we will analyze the computational complexity and communication bandwidth requirements of high-end television functions. This is an essential prerequisite for considering the problem of formulating a new architecture, as it should be known beforehand which tasks should

be carried out and with how much memory and speed. This chapter serves as an overview text that could be labelled as "lessons learned from the past". The relevant architectural aspects in a new design should be taken into account and mistakes should be avoided.

Some attempts in the past were made to deal with the requirements discussed in the previous chapter. Let us give an overview of these attempts and discuss some of the architectural aspects required for the problem statement. It is emphasized here that this chapter does not pretend to give an exhaustive overview of media processors. Such overviews can be found in alternative literature about DSPs or specialized handbooks. The purpose of this chapter is to show samples of proposed systems, both in style and of the evolution over time.

A fully programmable multimedia solution was presented by the so-called Multimedia Video Processor (MVP) [32]. This processor was based on four programmable DSPs, containing VLIW engines of 64-bit width, each of them having a substantial computing power. Furthermore, the architecture contained a powerful RISC processor for control and general-purpose computing. Communication between the processors was based on a switch matrix network, allowing arbitrary connections between processors and a bank of memory units of size 20×2kByte.

Strong points of this system are the large parallelism of the computing tasks by the set of VLIW DSPs and the corresponding large internal communication bandwidth. Furthermore, including a RISC core enables flexible control tasks to be carried out. A clear disadvantage is that the system is rather expensive for consumer products, because it is more general than the TV application domain desires. Another disadvantage is the programmability of the system. Efficient programming of the DSPs appeared to be feasible in DSP assembler language only, although this is not an inherent problem. The tool set was just lacking. A further problem was the programming of the parallelism. Tools for obtaining high parallelism were not available, so that the scheduling and synchronization of the individual processing tasks were left to the programmer. A third disadvantage of the system is the distribution of the internal memory into many small banks. For a typical TV function operating in the vertical dimension, a number of video lines in memory is already required. Hence, a group of memory blocks would be allocated for only a single function, thereby reducing the flexibility and parallelism seriously, especially with respect to memory sharing.

A highly-programmable solution, which is tailored to the specific application domain of video processing, is the so-called Video Signal Processor (VSP) [33]. This processor contains generic operations with a large amount of instruction-level parallelism, which makes the processor surprisingly powerful. Due to the large amount of parallel operations and the single thread of control that is scheduled at compile time, the VSP can be classified as a VLIW processor. The set of arithmetic logic units (ALUs) and memory elements are fully interconnected by means of a switch matrix. Although some tools exist for scheduling of all the necessary operations performing a function, a large amount of intelligent interaction by the programmer is required. Moreover, the architecture is designed for regular processing without conditional branches. Branches that are contained in the algorithms have to executed in parallel, followed by a conditional selection. Consequently, this makes the programming of video processing functions with a large amount of data dependencies extremely complicated. Due to the fully interconnecting switch matrix, the processing of the functional units is never hampered by the communication network, while the communication data can be routed from any output to any other input. The drawback of such communication network is the lack of scalability, because the number of long wires increases quadratically with the number of processing units.

Currently, the usage of Very Long Instruction Word (VLIW) becomes increasingly popular for boosting processing power [26] [34]. These architectures have a large number of functional units that can be programmed individually. High parallelism is obtained by using a very wide instruction format, so that various functional units can be addressed at each instruction issue to the processor. An advantage of a classical VLIW over the above-mentioned VSP system is that it is still reasonably effective for data-dependent control, which makes the concept very powerful and suitable for video processing and/or multimedia tasks. This explains the popularity of this processing model in recent technical proposals.

The width of the instruction word is still limited in practice, since the instruction decoding and memories would otherwise become unfeasible and the compiler should be able to exploit more parallelism in the application. This results in a parallelism factor of e.g. 4–6 for DSPs and 20–30 for more application-specific processors (ASIP).

More cost/power efficient implementations are possible by introducing task-level parallelism and by adapting the system more closely to the target

application domain. The functionality within the task can be designed rather dedicated to the application. Hence, they become only weakly programmable to change e.g. modes, resolution, pixel format, etc. In order to do so, we need to provide evidence about the computational complexity of typical TV functions. Are these kind of functions indeed significantly more expensive when implemented by means of flexible fine-grain operations, controlled by software? In general we can state that a dedicated solution is more efficient than a similar function implemented on a general-purpose processor. Figure 2.9 depicts a graph [35] that shows a difference of about two orders magnitude in computation efficiency measure in mega-operations per Watt (MOPS/W) between a dedicated hardware implementation and an implementation with a general-purpose processor.

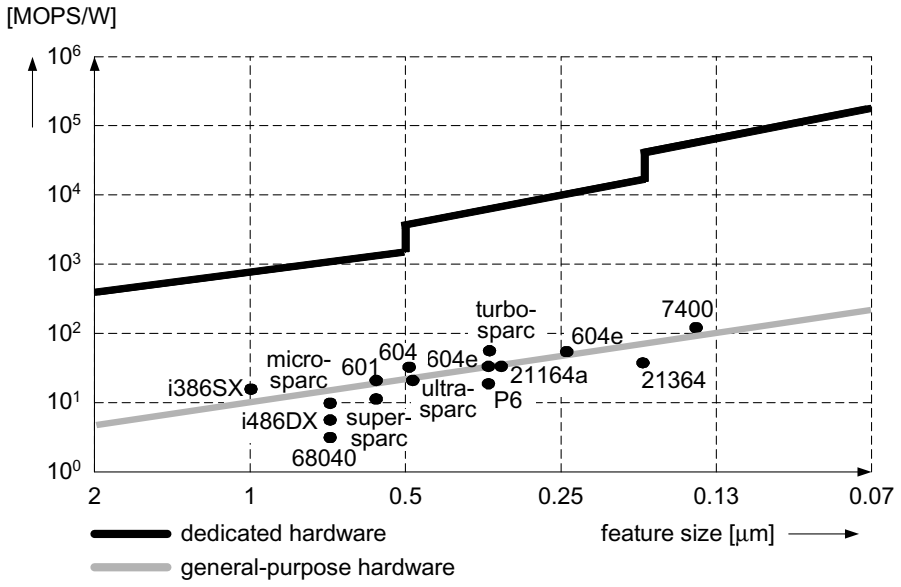


Figure 2.9: *Computational efficiency of dedicated versus general-purpose hardware.*

In the sequel, we present a few processor architectures which were summarized in this section. The purpose of this limited overview is to comprehend the architectural tradeoffs to be made and learn relevant system aspects. Afterwards, in a succeeding chapter, several TV functions will be studied and presented with complexity estimations.

Table 2.1: *Characteristics of experimentally developed VSPs.*

Parameter	VSP1	VSP2
technology	1.2 μm CMOS	0.8 μm CMOS
chip size	90 mm^2	156 mm^2
transistors	206k	1.150 M
package	QFP 160, PGA 176	QFP 208
dissipation	1 W	$<$ 5 W
clock	27 MHz	54 MHz
word length	12 bits	12 bits
ALEs	3	12
MEs, size	2, 512 $\times$ 12	4, 2k $\times$ 12
ME style	single port	dual port
BEs	–	6
inputs	5	6
outputs (OEs)	5	6

2.5.2 The Video Signal Processor

The architecture of the Video Signal Processor (VSP) is based on using a plurality of signal processors which are connected via a high-speed communication network [36]. The paradigm for this work is based on the repetition of relatively small programs, repeatedly processing all passing pixels. Branches in the program are not allowed. Instead, all conditional processing steps are performed in parallel, followed by a conditional selection. This makes it difficult to implement highly data-dependent or event-based applications. This is in contrast with the current development in general-purpose computing, i.e. the use of a powerful processor that carries out all tasks more or less sequentially at very high speed and a large uniform memory space in the background. Let us discuss some architectural aspects of the VSP system [33] [36].

Two processor ICs have been designed and implemented successfully, called the VSP1 and VSP2. The features of both ICs are given in Table 2.1. The VSP2 also served as a vehicle for research on IC design to exceed the 1-million transistor boundary in 0.8 μm CMOS technology. The VSP2 has about eight times the computing power of VSP1, because it contains four VSP1 cores, running at the double clock frequency. As the table indicates, the VSP architecture contains various elements: arithmetic logic element (ALE), memory element (ME), buffer element (BE) and output element (OE). Figure 2.10 shows the architecture in which the processing elements are connected to a switch matrix. The architecture is organized

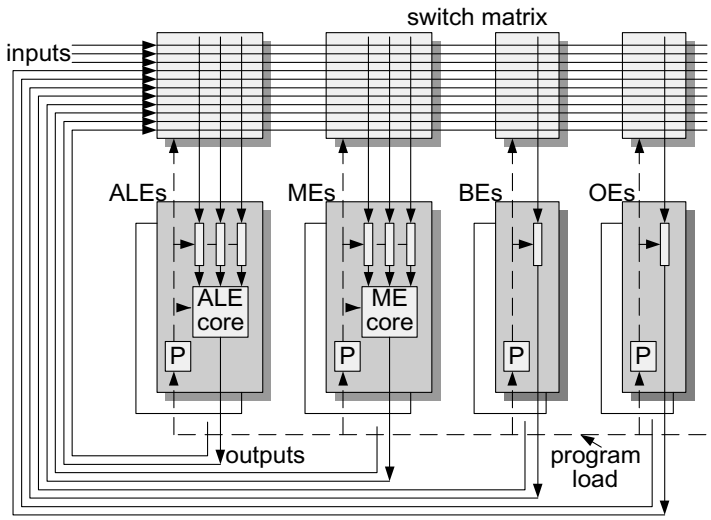


Figure 2.10: *Architecture of the Video Signal Processor.*

such that all elements are active in parallel, by using pipelined processing in all stages. For each processing element (PE), the instructions are stored in a local program memory. After initializing the system with serial downloading of programs, the PEs execute their instructions cyclically. Branches in parallel processing are implemented by mapping conditions as data-dependent selections (multiplexers). Each clock cycle, every PE can take one or several data samples from the switch matrix and produce after some delay, a new result that can be redistributed.

Figure 2.11 portrays a more detailed view on the ALE and ME building blocks. The ALE unit primarily contains input data buffers (silos) and the ALE core. Inside the ALE core, shifters, multiplexers for constant input and an ALU for conventional operations can be distinguished. A multiplication inside the ALU is carried out as a series of successive additions (Booth algorithm), to save hardware costs of a full multiplier. Typical operations of the ALU are ADDi, SUBi (subtract), LOGi (logical), BM (bit mask), UM (unsigned multiply) and SM (signed multiply). Index i refers to various forms of the same operation.

The Memory Elements (ME) contain a two-port dynamic RAM and an address calculation unit (see Figure 2.11). At the input side, so-called data silos accept data with a capacity of 32 words of 12 bits. Data is written cyclically into the RAM. The read address is calculated by subtracting the

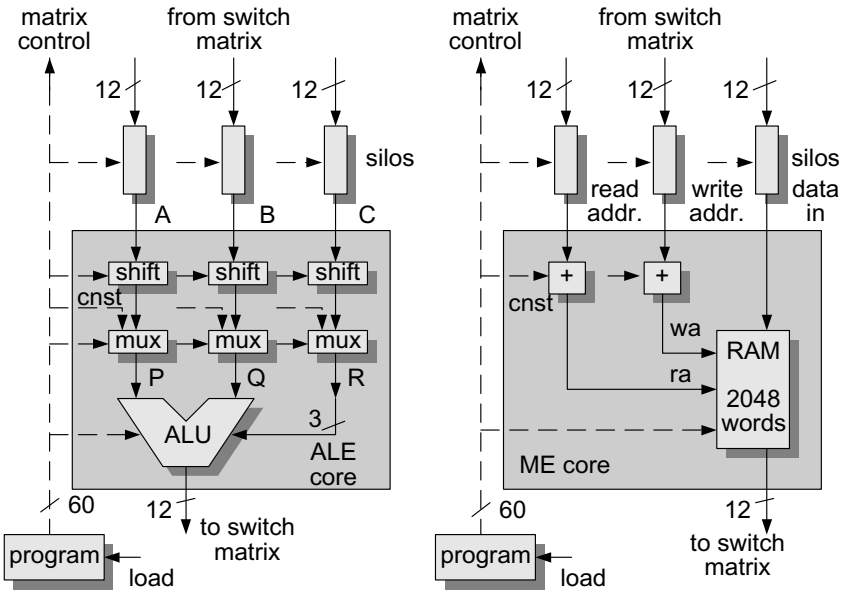


Figure 2.11: *Architecture of ALE and ME subsystems in the VSP.*

desired (to be programmed) delay from the write address. The data silos are required for time alignment of the data at the input stages of the PEs (scheduling). Moreover, the silos facilitate interleaving of data streams for (de)multiplexing operations and multi-rate processing.

Programming of the VSP is performed with signal flow-graphs, where the units are graphically connected by wires. The nature of the architecture is *stream-oriented*, meaning that small programs are repeatedly executed on a stream of video samples. The programmer has to specify the periodicity of the signal processing and the pipelining and delay of the processing units. Despite the availability of graphical tools for programming the flow-graphs, this can be a rather cumbersome task, especially if the processing task becomes so complex that it has to be partitioned over several processors. In fact, the programmer is responsible for the three major issues in the processor: *partitioning, scheduling* and *time delay management*.

Summarizing, various system and design aspects of the VSP can be listed.

1. The processor architecture clearly recognizes the high parallelism required for video processing. Important blocks such as PEs, MEs, etc., and memory silos are offered in parallel to the programmer. Because the system is merely designed for pixel processing, the number of op-

erations of the ALUs are limited. Hence, more complex operations need to be emulated thereby consuming a considerable amount of resources.

2. Communication is well covered by the implementation of a full switch matrix. This allows arbitrary connections between the modules, so that flexibility in programming of video functions is ensured. However, such a communication network does not scale properly for an increasing number of function processing elements.

3. The grain of parallelism offered to the system designer is on the level of primitive operations, that is individual instructions of a function to be carried out. For example, a video filtering operation is broken down into individual operations of multiplications, additions and delays. These operations are mapped onto the PEs described earlier.

4. All operations in the system are executed by a single thread of control which is scheduled by the compiler. This means that all parallel operations need to fit into a single complex schedule. Programming of the signal flow through the processing element in the correct order is reasonably simple, but programming of the delays (scheduling) by means of the silos and the buffer elements is cumbersome and is only partially supported by some tools. This is problematic, because the programmer has to know the processor and its subsystems in considerable detail.

2.5.3 The Multimedia Video Processor (MVP)

In 1993-1994, the Multimedia Video Processor (MVP) [28], released under the product code TMS320C80, was offered to the market. Just after the breakthrough of MPEG coding in a wide application field, it was felt that systems would be built in SW from that point on or in the nearby future. The MVP Processor was certainly advanced in various ways. The chip, designed in a 0.5 micron CMOS process, contained over four million transistors in a single package. Besides this, it was a clear multiprocessor system and even heterogeneous in its kind, since a general-purpose computer was combined with four advanced digital signal processors (ADSPs). Each ADSP can handle 32 bits of data and 64-bit instructions, thereby boosting the potential computing power to high numbers. Moreover, the chip offered wideband communication facilities. The main application field of the processor was video conferencing, general video processing and MPEG-related applications. Figure 2.12 presents the global architecture of the processor.

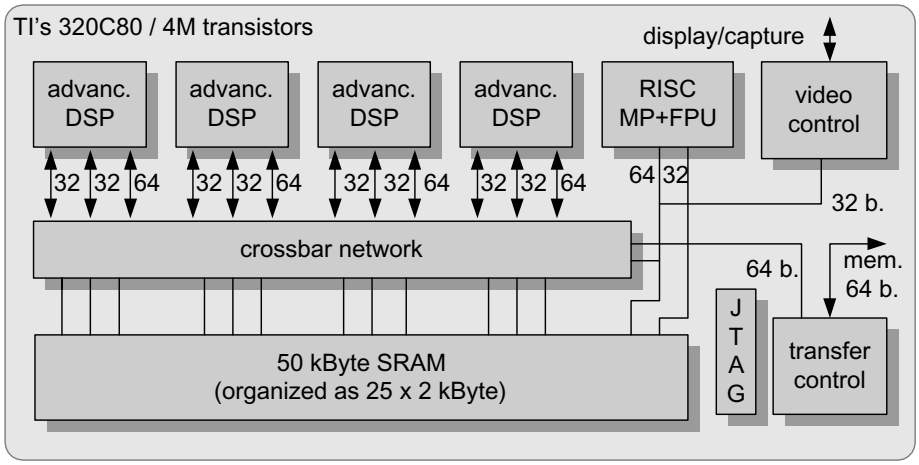


Figure 2.12: *Architecture of Multimedia Video Processor.*

Let us take a closer look at the multiprocessor architecture. The system contains four advanced DSPs and a master processor (MP) which is a customized RISC core. The MP RISC is a 32-bit computer can be programmed with efficient high-level languages and is included for simple user interfaces, control of the entire MVP, and for higher precision operations (including floating point). The ADSPs are suited for stream-based computing with high parallelism, in particular video and audio processing. The 64-bit instructions and data path enable several video samples to be processed in parallel.

Parallelism in communication is ensured by a crossbar network, which enables the connection of processors to a special part of the on-chip memory. The crossbar network supports up to 4.2 GByte/s bandwidth on-chip, consisting of 2.4 GB/s data and 1.8 GB/s instruction traffic. The system contains the considerable amount of 50-kByte memory, which is partitioned into 16 blocks of 2 kByte, two 4-kByte caches for data and instruction of the MP, and a separate scratch-path memory for the MP. The crossbar facilitates a connection between each ADSP and a group of four 2-kByte data modules. Furthermore, a second connection on each ADSP can be connected to the four data modules of its neighboring ADSP for interprocessor communication. Besides the processors, also a transfer controller is connected to the memories. This transfer controller, which is activated by the applications executing in the ADSPs, is responsible for managing the entire traffic to on and off-chip memory and controls the image data flows through the chip. The total off-chip bandwidth amounts to 400 MByte/s.

Finally, programmable video controllers enable one to capture or display video sequences in various resolutions.

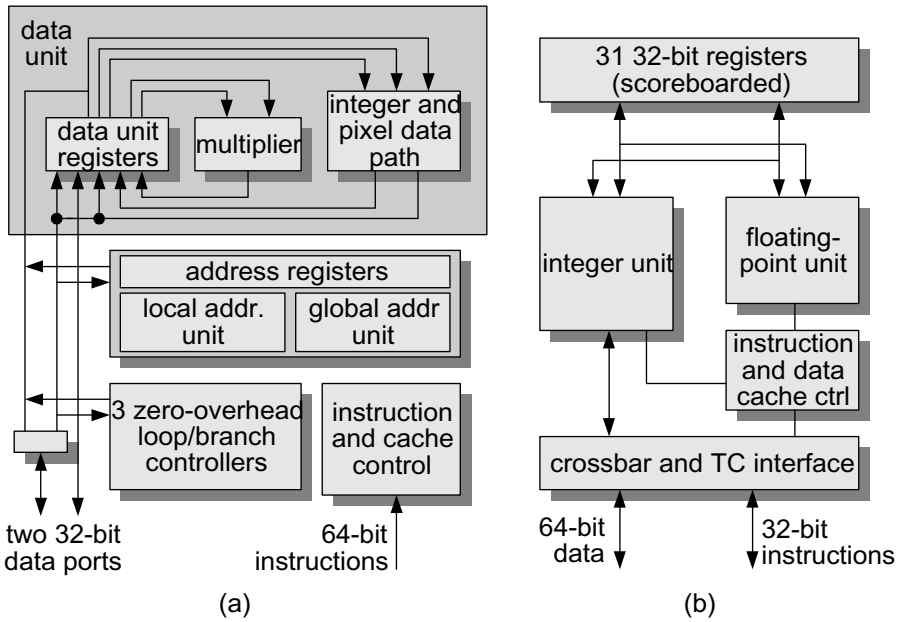


Figure 2.13: *Architecture of the ADSP (a) and MP (b) modules in the MVP.*

Figure 2.13 portrays a detailed diagram of the ADSP and the MP architecture. The key features of one ADSP unit are described below.

- Instruction word has 64-bit width and supports many parallel operations.
- Two address units are available, so that two memory operations per clock cycle can be carried out.
- A single-cycle multiplier is included, which can handle one 16-bit multiplication or produce two 8-bit results per clock cycle.
- The ALU accepts three inputs which can be split for parallel operations. In each pass when data is fetched from the registers, multiple operations can be performed and it can yield up to four 8-bit results per clock cycle.
- 44 Data registers keep data locally available for high-throughput processing.

- The ADSP operations range from simple bit-field operations to multiple arithmetic and logic functions on multiple pixels in each cycle. Moreover, the ADSP has three hardware loop controllers for fast repetitive processing which is typical for audio and video.

The Master Processor (MP) shown in Figure 2.13 is a clean RISC computer to be programmed for high-level languages as C/C++. As opposed to this, the ADSPs are programmed in assembler language in order to obtain the highest possible efficiency and to have easy access to hardware features of the DSP. The RISC core further comprises an IEEE-754 floating-point multiplier, capable of handling 100 MFLOPS, based on parallel handling of the multiplication, addition and load/store operation. Even this processor has thirty-one 32-bit registers and two 4-kByte caches for instructions and data.

Programming of the MVP is well possible, but it needs skilled programmers. Efficient programming of the ADSPs can only be done efficiently in assembler language, which requires detailed knowledge of the hardware architecture. Generally, assembler programs are not portable. Fortunately, the RISC core is programmable in a high-level language. The communication to and from the internal and external memories is provided by the transfer controller, which is activated by the application on the ADSPs or the MP. Consequently, the programmer is responsible for the management of the memory usage and the mapping of tasks onto the ADSPs. The system designer has to distribute the tasks over one or more ADSPs and then route and monitor the data flows in the architecture, in order to prevent large caching problems and/or data storage collisions. The programmer is responsible for an efficient and unique data preservation process, which is not easily done in a shared memory system.

Summarizing, the main advantages and disadvantages of the MVP are presented below.

1. The distinction between high-level control decisions and some general-purpose computing on one hand and stream-based high-throughput processing on the other hand is acknowledged in the MVP architecture.

2. The system is a multiprocessor computer that can perform the video signal processing asynchronously from the input and output video rate. This is achieved by internal buffers that decouple the chip clock speed from the video application. Moreover, the buffers also decouple and synchronize independent tasks on the processors.

3. The system exhibits large parallelism in all three major aspects: computing, crossbar communication and memory storage. Particularly the last two have been broadly dimensioned (more connections and buffers) and thereby enable to drive the chip to its maximum computing performance.

4. For advanced processing and maximum performance, the designer has to distribute computing tasks over several ADSPs. Also the split between high-level control and native signal processing is preferably performed by the designer, since the ADSPs can only achieve high throughput for stream-oriented processing containing parallelism. This still requires considerable knowledge of the processor.

5. Careful data I/O analysis is required for planning and usage of the internal 2-kByte SRAM buffers. Since the memory is shared between the four ADSPs and the MP, *data integrity* has to be kept and protected. The manufacturer delivers tools for programming the individual processors and monitoring their performance, but a clever distribution of tasks and their memory usage is handcrafted work.

2.5.4 The TriMedia processor

The TriMedia processor was one of the first experimental Very Long Instruction Word (VLIW) processors. The processor is targeted for multimedia computing applications and contains amongst others application-specific hardware for MPEG decoding. The objective for the processor usage is concentrated on embedded flexible computing for consumer electronics systems, like TVs, video phones, and set-top boxes.

The processor chip has a clear focus on MPEG-1 and MPEG-2 coding which can be seen from the block diagram in Figure 2.14 of the first embodiment, the TM1000 processor [26]. The chip contains supportive units for variable-length decoding, and an image coprocessor which is specialized to advanced handling for blocks of image samples. Such a block can be applied successfully for programmable filtering, scaling, and color-space conversion [29]. The supportive blocks are discussed below in more detail. Typical applications of the processor is are video/audio processing applications. In the sequel, we address the functional blocks individually, especially those relevant for video processing.

- *VLIW processor core.* This is a 32-bit DSP processor is suited for media applications. However, as opposed to most DSPs, the VLIW core

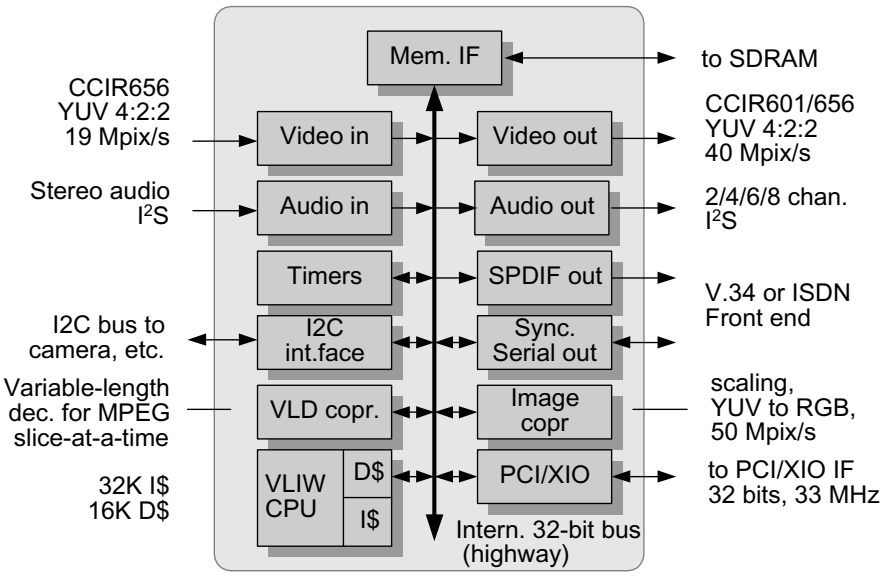


Figure 2.14: *Architecture of TriMedia processor chip.*

does not provide special hardware for loop control. The processor features a register file with 128 32-bit registers, which can be individually applied and accessed. The processor enables general-purpose operation and is based on a VLIW instruction set. Five issue slots allow the instructions to execute up to five operations simultaneously. An operation can target one of the 27 available functional units (this will be discussed later). A special feature is the use of so-called *custom-operations* or media-operations, which can dramatically accelerate the execution of a multimedia application. The processor has a 16-kByte data cache and a 32-kByte instruction cache which are both 8-way set-associative and contain cache lines of 64 Byte.

- *Video-in and Video-out unit.* The video-in unit accepts CCIR-601/656 compliant signals, based on 8-bit parallel data samples representing 4:2:2 YUV time-multiplexed data. The input unit can do limited processing during receiving a video signal, such as horizontal subsampling with a factor of two without CPU intervention. Video data is written in separated Y, U, and V planes in the SDRAM memory. The Video-out unit performs the inverse functions of the Video-in unit, with the addition of enabling graphical overlays with a blending function.

- *Image Coprocessor (ICP).* The ICP is applied for intensive but straightforward image processing tasks, such as horizontal or vertical filtering and image resizing. Internally, it has 32 FIR filters based on five surrounding samples. The filter coefficients are programmable. The 32 filters are organized in a raster to generate pixels at subpixel resolution at the output side. In a special communication mode, the ICP can also perform color-space conversion (e.g. YUV-to-RGB). In order to locate the correct pixels in memory during the above-mentioned tasks, the coprocessor contains management logic for keeping pointers to input and output images in memory and support for bit masks to manage several video windows simultaneously.

- *Variable-Length Decoder (VLD).* This unit decodes MPEG-1 and MPEG-2 Huffman-coded bitstreams into DCT-coded streams for further decoding. Since this decoding process operates on bit level, this certainly relieves the CPU from time-consuming instructions on a low system level. The decoding task runs as a memory-to-memory process, where an input bitstream is converted into data-structures with fixed-sized word lengths that have to be further decoded.

- *Internal data bus.* The data bus connects all internal subprocessing units and enables access to the background memory. General access to the bus is controlled by an arbiter, which is programmable and can prioritize the accessability to create e.g. guaranteed bandwidth or latency to specific blocks. Moreover, the bus supports access to the PCI bus. This PCI bus can be used as interface to other chips providing other interfaces such as UARTS, ethernet, USB, etc.

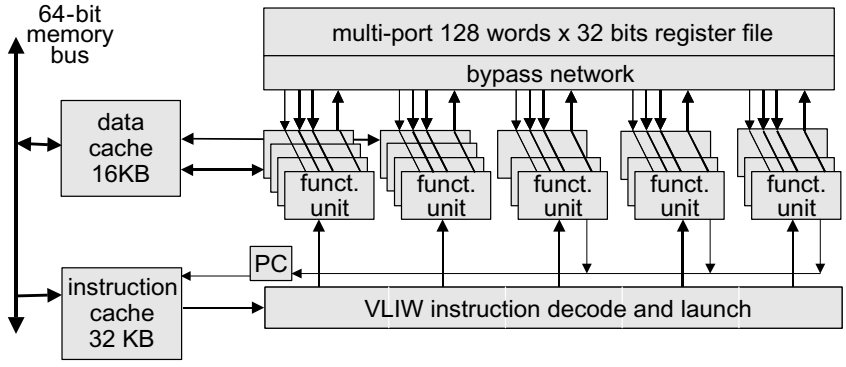


Figure 2.15: *Concept of TriMedia's VLIW instruction handling.*

Let us now briefly discuss the instruction data format in somewhat more detail. Figure 2.15 shows the concept of the VLIW processor in more detail. Instructions are fetched from memory and expanded to drive five issue slots in parallel. Each issue slot can access one of the functional units. The operands and results are read from or written in the register file.

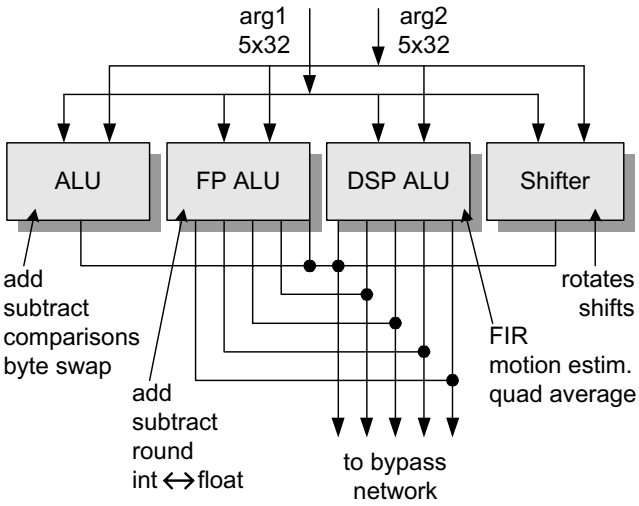


Figure 2.16: *TriMedia's different types of instructions.*

Figure 2.16 portrays conceptually how a compound unit can be utilized for addressing a particular functional unit. A *compound unit* is the partial instruction format, containing the arguments (or operands), corresponding with an issue slot. The arguments are inputs for the operation to be carried out. This operation can be of general ALU type, such as add, subtract, mask, or floating-point type, which can all use e.g. rounding and conversion. The DSP ALU enables the special custom operations mentioned earlier, such a FIR(.) for filtering, or QUADAVG(.) for averaging of groups of samples. Also a shifter can be included, featuring pattern shifting or rotation. To efficiently employ registers and data transfer units, most instructions are implemented for vector processing. This means that e.g. samples are packed together into four 8-bit or two 16-bit data items. Otherwise, it would be a waste of computing power to use a 32-bit data path if 8-bit video samples are processed.

Let us now briefly focus on the most powerful instructions, i.e. the *custom operations*. The operations are specialized to a certain application domain and would require a sequence of traditional instructions if not

Table 2.2: *Selection of custom operations of the TM1000. Regular adds and multiplies at full resolution have been omitted.*

Function	Custom Op.	Description
DSP add	dspidualadd dspuquadaddui	dual clipped add of signed 16-bit halfwords quad clipped add of unsigned/signed bytes
DSP multiply	dspidualmul	dual clipped multiply of signed 16-bit halfwords
Sum of products	ifir16 ifir8ii ifir8iu	signed sum of products of signed 16-bit halfwords signed sum of products of signed bytes signed sum of products of signed / unsigned bytes
Byte average Byte multiply	quadavg quadumulmsb	unsigned byte-wise quad average unsigned quad 8-bit multiply most signif.
Motion estimation	ume8ii ume8uu	unsigned sum of absolute values of signed 8-bit differences unsigned sum of absolute values of unsigned 8-bit differences

present. Computing power is further enhanced by using operations that are frequently required in media processing applications. The following example shows how the special custom operations can be exploited to achieve a higher computation power. The example is based on the frame reconstruction loop in the MPEG decoder, where a reference frame "back"and "forward"are used for the past image and future image, respectively. The inverse cosine transformed block is stored in "idct". A straightforward coding example in the C programming language will look as in Figure 2.17. The code addresses a byte at every clock cycle, while the infrastructure can handle 32 bits, so that 75 % of the bandwidth is wasted. Moreover, the special operations packing four samples into one word is not employed.
This packing is enabled in the converted code of Figure 2.18, where the for-loop is unrolled with a factor of four. The code enables parallelization, because the four pieces do not depend on each other. As a preparation for upcoming steps, the variable "temp"is copied into four different ones, and as an intermediate step, the copying of the result into the array "destination"can be demultiplexed and grouped together as a last step.
In Figure 2.19, a further optimization is reached by employing the special custom operations. It can be noticed easily that the instruction "quadavg"can be applied for summing up the contributions "back[i+j]"and "forward[i+j]". This result is stored in "temp"which is then used to add the

```
void reconstruct (unsigned char *back,
                  unsigned char *forward,
                  char          *idct,
                  unsigned char *destination)
{
  int i, temp;
  for (i = 0; i < 64; i += 1) {
    temp = ((back[i] + forward[i] + 1) >> 1) + idct[i];
    if (temp > 255) temp = 255;
    else if (temp < 0) temp = 0;
    destination[i+0] = temp;
  }
}
```

Figure 2.17: *Sample C code for MPEG frame reconstruction.*

contribution of the IDCT to the loop. This last step is mapped elegantly with a parallel addition "quadadd"operation. Note that the last two steps in Figure 2.19 can be combined into a single instruction, thereby eliminating the temporal storage in the variable "temp". This last step is left to the reader.

Let us now compare the complexity of the last result with the initial code. In the initial code, the requirements were 3 additions, 1 shift, 2 tests, 3 loads, and 1 data store operation per pixel. After merging the last two instructions in Figure 2.19, the code with custom operations requires 2 custom operations, 3 loads, and 1 store for 4 pixels. When comparing on a pixel basis, this represents an improvement factor of about 6. Notice that this concerns a simple example with no data dependencies, which
which can vectorized perfectly.

At the end of this section, we summarize various system and architectural aspects of the Trimedia processor.

- The aforementioned example for MPEG-loop decoding shows the powerful usage of custom operations. If repetitive signal-processing tasks occur, then parallelism of the VLIW concept can be exploited to its maximum. On the other hand, these instructions should be known well in advance, since the compiler is not capable to exploit this parallelism automatically.

- The processor is programmable in the common C language. This is a major advantage that gives great flexibility for the application programmer. The code can be developed on any platform and then

```
void reconstruct (unsigned char *back
                  unsigned char *forward
                  char          *idct
                  unsigned char *destination
{
  int i, temp0, temp1, temp2, temp3;
  for (i = 0; i < 64; i += 4){
    temp0 = ((back[i+0] + forward[i+0] + 1) >> 1) + idct[i+0];
    if (temp0 > 255) temp0 = 255;
    else if (temp0 < 0) temp0 = 0;
    destination[i+0] = temp0;

    temp1 = ((back[i+1] + forward[i+1] + 1) >> 1) + idct[i+1];
    if (temp1 > 255) temp1 = 255;
    else if (temp1 < 0) temp1 = 0;
    destination[i+1] = temp1;

     temp2 = ((back[i+2] + forward[i+2] + 1) >> 1) + idct[i+2];
    if (temp2 > 255) temp2 = 255;
    else if (temp2 < 0) temp2 = 0;
    destination[i+2] = temp2;

    temp3 = ((back[i+3] + forward[i+3] + 1) >> 1) + idct[i+3];
    if (temp3 > 255) temp3 = 255;
    else if (temp3 < 0) temp3 = 0;
    destination[i+3] = temp3;
  }
}
```

Figure 2.18: *Loop-unrolled code for MPEG frame reconstruction.*

easily mapped onto a Trimedia-based system. Subsequently, critical loops can be optimized to considerably improve the performance.

- Clearly, the final parallelization factor strongly depends on the application. If the application contains many irregular control loops or conditional tests, the factor drops significantly. The VLIW core is actually not suited for control processing, that is, although it can handle general-purpose control processing, the parallelization factor will decrease significantly.

- The input-output functions or data exchange operations between subsystems are carried out via the background memory. This gives flexibility, but it further expands the required memory bandwidth – already being scarce – to high values.

- The chip subsystems are gathered around the data bus system. For

```
void reconstruct (unsigned char *back,
                  unsigned char *forward,
                  char          *idct,
                  unsigned char *destination)
{
  int i, temp;
  int *i_back     = (int *) back;
  int *i_forward  = (int *) forward;
  int *i_idct     = (int *) idct;
  int *i_dest     = (int *) destination;
  for (i = 0; i < 16; i += 1){
    temp = QUADAVG(i_back[i], i_forward[i]);
    temp = DSPUQUADADDUI(temp, i_idct[i]);
    i_dest[i] = temp;
  }
}
```

Figure 2.19: *TM code based on two custom operations.*

highly-parallel systems containing a plurality of coprocessors, the bandwidth of such a bus structure is too limited. Moreover, the available communication resources of the bus for a coprocessor depends on the activities of the other coprocessors. This makes the system unpredictable when the system is highly utilized. Yet another drawback of the communication model of the Trimedia system is the inter-coprocessor communication via the off-chip memory. This results in a high requirement for the already scarce off-chip memory bandwidth.

- The second item mentions that the efficient mapping of software onto the Trimedia core is on of its strengths. One of the reasons for this is the centralized register file which offers a large amount of flexibility. However, because all issue slots may require simultaneous access to this single register file, multiple ports are required and therefore it becomes a critical part in the design.

2.5.5 The Emotion Engine

The Emotion Engine is a processor system which was introduced for construction of the second generation of gaming machines for connection with a TV set (known as Playstation) [37]. The system forms the core of the computational engine and is in its complete form a multiprocessor system. Playing consoles are nowadays based on advanced 2-D or 3-D graphics generation in order to provide an impressive viewing experience. The amount

of computations involved for high-quality graphics at more than TV resolution is huge and in the order of multiple GOPS. This explains why several vector processors are contained in the system.

The processor architecture is based on a two-way superscalar core [38], which is a 128-SIMD processor and two vector coprocessors [39]. This part offers together nearly 5 GOPS and 6.2 GFLOPS computing power. The chip is very large, over 240 mm^2 in a 0.25 μm CMOS process and at the same time, runs at the relatively high clock frequency of 300 MHz. The strategy that the chip designer follows is worthwhile to mention here. Instead of the usual evolutionary improvements in the chip design, the manufacturer has chosen for an extremely aggressive computing performance specification. Subsequently, the design is shrunk for every new technology step, while keeping important parameters of the design specification, such as the clock frequency, constant. To meet the specification with the initial technology process, most of the blocks are customized designs, all running at a single synchronous clock domain of 300 MHz [40]. Although difficult to achieve for such a large die size, this approach enables to quickly follow technology improvements without redesigns, so that the price goes down rapidly. The hardware costs are reducing to a fraction of the system costs, while the specification is reused for a number of generations.

The Emotion Engine portrayed by Figure 2.20 has a heterogeneous pro-

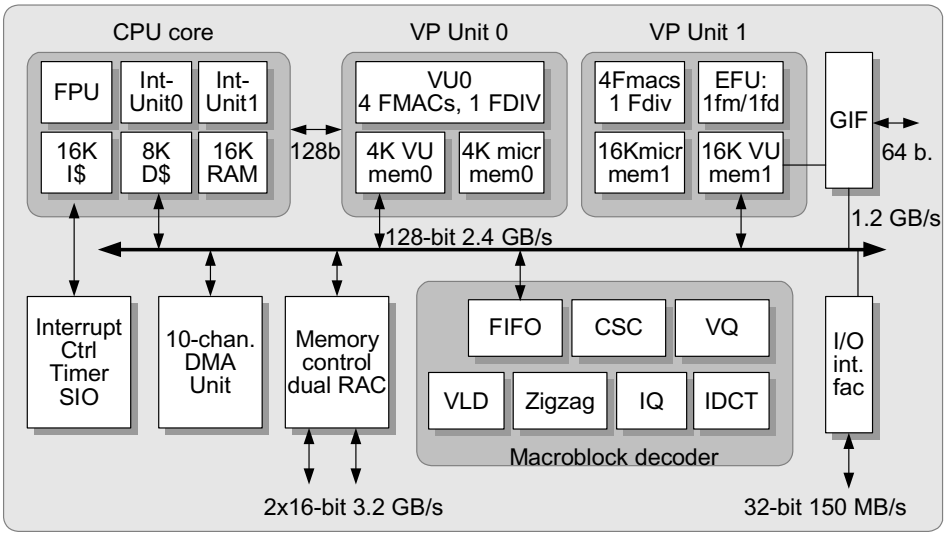


Figure 2.20: *Emotion engine processor architecture, based on a CPU core and two vector processors (VP).*

cessor architecture where three types of processors are contained in the same system: a general-purpose CPU core, two vector processors (VP) for performing many floating-point matrix operations, such as perspective transformation, and a JPEG/MPEG decoder processor [41]. The natures of these three processor types are essentially different. The CPU core is used for special actions and control. The VPs are applied for all kinds of graphical generations and need high throughput rates to obtain high-resolution graphics. For this reason, vectors of picture elements are processed at clock speed in order to acquire sufficiently high parallelism. The high clock frequency needs to be "multiplied"to generate the required resolution of the graphics pixels. Finally, the MPEG/JPEG decoder is a –to a large extent– dedicated processor for decoding the compressed pictorial data which is recovered from an optical disk inserted in the system. Let us briefly highlight the main elements of the architecture in Figure2.20 and discuss some system aspects.

- *The CPU core* is a conventional RISC core, however, with increased parallelism: the integer arithmetic has been implemented twice. This enables some processing tasks for decoding such as MPEG bit-stream conversion, while performing another task(s) in parallel. The increased parallelism is also supported from the memory side, because the core has 16 kByte RAM inside to store sufficient context.

- *The Vector Processors VP0 and VP1* are powerful engines for parallel pixel computing. The processors resemble each other but are not identical. The cache memories VU of 4K and 16K are organized for vector-based storage functions and both processors contain a scratchpad memory of similar size to contain context of graphical extension programs or local pictorial information. Differences are found in the local computation facilities. The VP1 processor contains an extended functional unit for floating-point division and multiplication. Both processors have four multiply-accumulate units suited for floating-point accuracy (FMAC) and a divider (FDIV). Note that VP0 can communicate directly with the CPU core for intermediate coprocessing activities.

- *The MPEG/JPEG decoder* contains the conventional required decoding blocks, such as inverse cosine transformation (IDCT), inverse quantization (IQ), scanning (zigzag), and variable-length decoding (VLD). Some extra functions are integrated, like vector quantization (VQ) in order to quickly modify the resolution of vectors of samples and a FIFO buffer for feeding the decoder with compressed data from

disk. The reconstruction with motion-compensated prediction, which is required for MPEG is performed in software.

It is worthwhile to mention the communication facilities here, of which the primary function are a 64-bit graphics interface (GIF) to communicate to an external graphics engine for pixel rasterization, and the double 16-bit direct RAMBUS interface which supports a bandwidth up to 3.2 GByte/s. This high bandwidth is particularly required for high-resolution graphics generation. The high external bandwidth led to a 128-bit wide bus to connect the different processors internally.

Table 2.3: *Characteristics of Emotion Engine.*

Parameter	Value
CPU core	128-bit RISC (MIPS IV subset)
Integer unit	64-bit wide
Clock frequency	300 MHz
Multimedia ext. instructions	107 at 128-bit width
Integer gen.-purp. register	32 at 128-bit width
Table-lookup	48 double entries
Instruction/Data cache	16 kB (2-way)/ 8 kB (2-way)
Scratchpad RAM	16 kB (dual-port)
Main memory	32 MB RDRAM, dual channel, 800 MHz
DMA	10 channels

Table 2.3 portrays the key parameters of the Emotion Engine. The table shows that the RISC core is already quite powerful and enables considerable parallelism with 128-bit wide instructions and 64-bit integer units. Attention has also been payed to multimedia instructions, such as indicated in the previous section. Since four different processors are contained, the memory bandwidth has been specified to an ultimate speed of 800 MHz based on RAMbus technology, offering 3.2 GB/s bandwidth.
Table 2.4 shows a few graphics performance parameters and the main technology parameters. The graphics performance is in tens of millions of polygons/s and even the MPEG decoder has a throughput rate of 150 Mpixels/s, which enables high-resolution TV pictures at real time display.

A more detailed diagram of the vector processor is shown in Figure 2.21. Instructions are entered in 64-bit format and split into two subfields of 32 bits each. One part controls the 4-way floating multiply-accumulate units, while the other part programs the lower execution unit with a.o. a divider

Table 2.4: *Performance and chip parameters of Emotion Engine.*

Parameter	Value
Perspective transformations	66 Mpolygons/s
Lighting/fogging	38 M / 36 Mpolygons/s
Curved surface (Bezier)	16 Mpolygons/s
Image Processing Unit	MPEG-2 block decoder 150 Mpix/s
Gate width / Vdd voltage	0.18 μm / 1.8 V
Metal layers	4 layers
Transistor count	10.5 Million
Die size	240 mm^2
Package	540 PBGA

and ALU. Local data for the upper unit is stored in 32×128-bit floating-point registers, enabling quick data exchange between the FMAC-units. The 16×16-bit integer registers support the lower execution unit. Note that the vector processor is basically a VLIW machine.

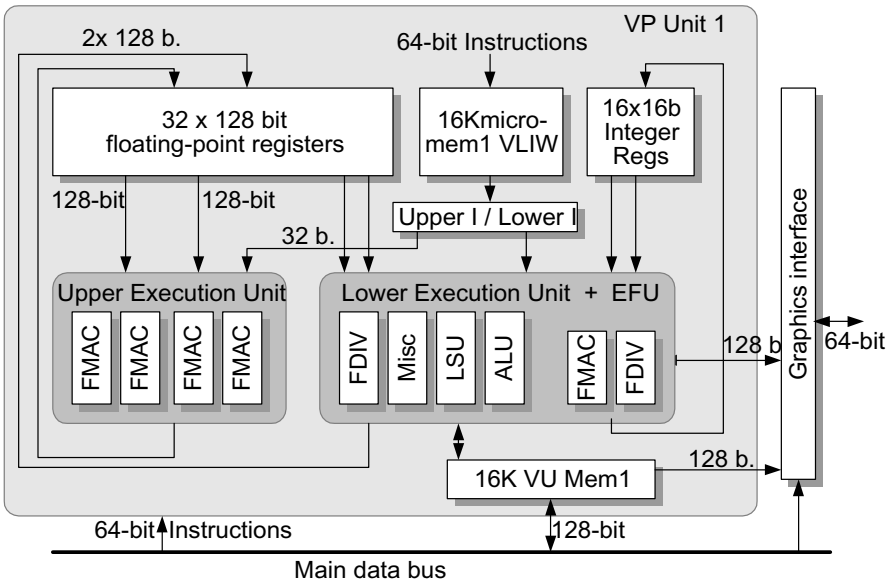


Figure 2.21: *Emotion engine vector processor architecture.*

It is emphasized here that the emotion engine chip connects to an external graphics chip which is even more powerful than the internal vector processors. This external graphics processor will not be discussed here in

detail, but some key figures will be supplied. The graphics chip also runs on 300 MHz clock frequency and has 16 pixel processing engines in parallel. The internal RAM size is 4 MByte. Memory bandwidth and internal data communication has been specified to top performance: 48 GByte/s bandwidth in total. Data bus size internally is a 2×1024-bit bus for separate reading and writing. The display color depth is 32 bits, using RGBα planes. The external graphics chip can perform all typical graphics functions, such as texture mapping, fogging, alpha blending, bi- and tri-linear filtering, anti-aliasing, multi-pass rendering, etc. Performance is in the 50–75 Mpolygons/s area, and yields a 150 Mpixel/s pixel drawing rate. The chip contains 43 million densely packed transistors on 279 mm^2 area, and it is packaged in a 384 BGA.

At the end of this section, we summarize the most relevant aspects of the Emotion Engine.

- The engine is a rather asymmetrical or heterogeneous multiprocessor system on a single chip. The RISC core is general purpose, whereas the vector processors are suited for processing groups of pixel data.

- The system architecture is heterogeneous, particularly in the hardware functions implemented. The different processors are mostly based on the VLIW principle. Besides these processors, a dedicated hardware processor for MPEG decoding is included. This leads to three different processing architectures on a single chip.

- It seems that programming of functions is separated on a functional level. Given the total complexity of the chip and the possible integrated functionality, this is a sensible choice. However, due to the heterogeneous nature, a high utilization of the hardware can only be achieved by programmers with sufficiently detailed knowledge of the architecture.

- The included MPEG decoding pipe serves as full hardware decoder for video material read from the optical disk storage system. Since the programmable processing part is rather powerful, the system can be used as a DVD video player. This improves the re-usability of the system considerably.

- The emotion engine connects to a graphics processor which is about as large as the processor system itself, thereby boosting graphics to ultimate performance. The result is a system with a very large capacity for even 3-D graphics processing.

2.6 Concluding remarks

This chapter has briefly outlined ways for improving parallelism in computing architectures and discussed also the associated control aspects. The examples of media processors have revealed that a number of aspects have to be considered carefully if an appropriate system design for television has to be adopted. The primary aspects are computing power, communication and memory, which are all related to other dimensions in the design space such as costs, parallelism, control, etc. In this section, we provide a few concluding remarks prior to presenting a new concept in the next chapter.

The VSP was designed as a generalized signal processor for pixel applications. Its main problems are the complex scheduling problem of the large amount of parallel operations and the missing capabilities for data-dependent and event-driven (irregular) processing. The MVP is a much more powerful and flexible (of course another IC technology), providing a multiprocessor architecture. Also for the MVP, the main problems are the programming, particularly for multiple tasks on low level and the passing of data from one subprocessor via memory to another one (memory management). The C-compiler does not have a high efficiency, necessitating coprogramming of C and assembler language. The TriMedia provides a fully C-programmable environment and comprises a heterogeneous architecture with more dedicated hardware. However, due to the limited instruction-level parallelism and the lack of task-level parallelism, the system does not offer sufficient computational performance for the TV domain so that more TV-oriented coprocessor engines are required. The processor seems an interesting candidate for high-end TV, if it is combined with other processors which are tailored to high-performance and/or high-throughput processing tasks.

Finally, the last example of media processors, called Emotion Engine, shows what can be achieved if the latest performance in IC technology is chosen. The system provides a heterogeneous architecture with various and different high-performance programmable embedded processors. This approach could well be adopted for high-end TV where also various processing tasks and intensive memory communication have to be covered. However, a clear discrepancy is the aspect of programmability. Since this programmable hardware is more costly than dedicated functional units, a highly programmable architecture needs to be justified by other aspects. For game applications, licensing of the associated SW could be the source for funding the extra costs. Unfortunately, in the TV world, such corresponding SW which can be sold separately, is absent. This clearly points towards

Table 2.5: *Overview of merits and shortcomings of the presented systems.*

System	Merits	Shortcomings
VSP	- Large amount of ILP - Deterministic behavior; performance known at compile time - General-purpose by fine-grain primitive operations - Computational performance does not depend on the communication network, i.e. no stall cycles	- No TLP - Switch matrix does not scale very well for more complex systems - Complex scheduling of the operations - Not programmable in a standard language such as C - Not effective for data-dependent programs
MVP	- Heterogeneous mixture of parallel DSPs and a general-purpose CPU - An adequate amount of parallel communication resources - Both ILP and TLP	- Detailed architecture knowledge is required for efficient programming in assembly - Centralized memory controller and the switch matrix limits the scalability - Mapping of tasks on the multiprocessors by hand
TriMedia	- Easily programmable in standard C - Heterogeneous mixture of a DSP and dedicated coprocessors	- No TLP by means of programmable multiprocessors - Unpredictable performance behavior due to run-time assignment of the communication resources (memory and bus)
Emotion Engine	- Heterogeneous mixture of parallel DSPs (vector processors), a general-purpose CPU, and a dedicated coprocessor - Both ILP and TLP - High computation performance	- The heterogeneous nature and the large extend of programmability requires detailed knowledge for efficient programming - Lacking on-chip shared memory for inter-processor communication hampers the programming - Dedicated design of blocks for a single product limits the reusability

Note: ILP = Instruction-Level Parallelism, TLP = Task-Level parallelism

a more low-cost solution for TV systems. A possible solution will be explored later, but to make the correct tradeoffs between programmability and application-specific processing, it is first required to analyze TV applications and associated computing and memory costs more extensively.

Table 2.5 gives an overview of the most important achievements and shortcomings of the presented systems. The table shows how certain architectural aspects result from design tradeoffs. For example, a switch-matrix network offers high-throughput communication with a predictable performance behavior. However, it results in a complex design and does not scale properly toward more complex systems. Another example is the realization of a high amount of instruction-level and task-level parallelism in a system. On the other hand, such systems are often more difficult to program efficiently. Clearly, system requirements are usually contradicting as was already stated in Chapter 1. Hence, the tradeoffs for system design with its merits and disadvantages should be made carefully.

Scire est mensurare (J. Kepler, 1571 – 1630)
To measure is to know

Chapter 3

Examples of video functions and their expenses

To determine the architectural requirements, first the functional requirements have to be understood. To this end, this chapter shows the characteristics of the video processing functions by means of some examples. Such an exercise give a first insight in the type of architectures that fit naturally to the algorithms. Obviously, the choice of an architecture also depends on many other constraints as stated already stated in Chapter 1. If these constraints are conflicting with the naturally fitting architecture, reconsideration of the algorithms might be required. Note, that determining the architecture depends on the algorithm and visa versa. Hence, this requires an integral design approach. To go one step further in determining the feasibility of an architecture, the resource requirements need to be analyzed. Therefore, the chapter also discusses resource requirements by means of example analysis of typical video processing functions. Considering the results from the analysis and all additional constraints, the architecture requirements of a system can be determined.

3.1 Tradeoffs in system design

In Chapter 1 it has already been clarified that the architecture of conventional TV systems does not satisfy the needs of future multimedia systems

in the consumer electronics domain. Future TV technologies require heterogeneous systems that are able to perform a large diversity of tasks, ranging from real-time, stream-based processing such as video filtering, to non-real-time and event driven processing for e.g. web browsing. Moreover, the increasing integration density according to Moore's law is exploited by the integration of more functionality and the growing complexity of multimedia functions. These motivations lead to a continuously growing complexity of Systems-on-Chip (SoC), where complexity is determined by the number of different component types (not number of components), the different types of interactions, and lack of structure in the interactions. To maintain a short time-to-market for such systems, reuse of design effort is absolutely essential. This requires extensible and scalable systems. Besides time-to-market, there is another important driver for requiring more flexible systems. Because Moore's law dictates more and more transistors on a silicon die and an increasing complexity of the production process, also the design costs for SoC continuously grows. At this moment in time, we are facing a new era in the TV domain in which the hardware and software design costs and the fixed production costs (e.g. cost of a mask set) significantly start to affect the total cost of the SoC, particularly in the higher market segment with its relative small production volume. Consequently, these chips need to be produced in large volumes to decrease the cost per product. To achieve this target, the market range of the SoC needs to be increased, i.e. the system should be flexible enough to be applied in a larger range of products or even product families. For example, the system should offer 50-to-100 Hz conversion for CRT-based TV, motion-compensated deblurring for LCD-based flat TV, and motion-compensated subfield generation for plasma-based flat TV. The flexibility should provide a motion-compensation unit which is required for all three different TV systems. Adding flexibility to achieve such above-discussed market increase, potentially increases system costs. Nevertheless, it can be concluded that all these considerations lead to an increasing amount of required flexibility.

This required extra flexibility can be provided by a high-speed, fully-programmable sequential processor. Such processor is most suitable for implementing any algorithm and is therefore typical for a general-purpose CPU. However, due to other constraints such as system costs, a high computational performance, and a low power consumption, a high degree of parallel computing is inevitable. To establish this high computational performance per unit of silicon area, the parallelism in the system should be optimally exploited. However, designing such a system with well-utilized hardware components, while offering sufficient flexibility to meet all requirements,

is most challenging. Hence, the amount of parallelism is an important design parameter to tradeoff flexibility, computational performance, and system costs. This chapter analyzes the expenses of several video processing functions, enabling the architect to make the correct tradeoffs. First, Section 3.2 and 3.3 elaborate on the functionality of sharpness enhancement and sampling-rate conversion as examples of video processing functions. Subsequently, Section 3.4 determines the computational complexity of video processing functions. Because the results are very function specific, Section 3.5 discusses some general observations and postulates the codesign of both the algorithms and the architecture.

3.2 Sharpness enhancement

To determine expenses in terms of computational complexity, first the algorithms of the function should be studied. Therefore, as an example this section will elaborate on a sharpness enhancement algorithm as discussed in [9]. Section 3.2.1 will first motivate the desire for sharpness enhancement, followed by detailed descriptions of each part in the algorithm.

3.2.1 Introduction to sharpness enhancement

In the consumer electronics area, the display technologies of televisions and computers are gradually merging, due to the advent of new features such as internet, games and video conferencing. Since the spatial resolution of synthetic images of the PC outperforms that of natural video scenes, the desire for improved quality in TV images will increase. The subjective attribute sharpness, which is determined by the human visual system, is one of the most important factors in the perception of image quality. The conventional techniques derived from generic image processing applications to improve the image sharpness are not suitable for broadcasted TV material, because they do not consider motion pictures and transmission impairments, such as noise, whereas the conventional algorithms from the TV world are not applicable for synthetic images. In this section, we elaborate on a generic sharpness improvement algorithm that can be applied for television images containing both natural scenes and graphically generated pictorial information.

We have confined ourselves to an approach for sharpness improvement in which an overshoot is added to the edges around objects in the image [42]. A literature study revealed that peaking, crispening, sharpness unmasking and statistical differencing are all techniques based on adding overshoot

[42] [43] [44]. It has been shown experimentally that from the above-cited possibilities, peaking gives a high subjective sharpness improvement and it yields a simple solution from the signal processing point of view. In addition we have extended this technique with an advanced adaptive control which uses the local image content, for combating various signal deteriorations such as transmission impairments and aliasing artifacts, thereby enabling the application for a much wider range of video signals. Spatially variant enhancement has already been subject of investigation for e.g. contrast enhancement [45] [46] [47].

In peaking, the added overshoot is determined by 2-D high-pass filtering of the input signal, which can be formulated mathematically as:

$$G(j,k) = F(j,k) + \sum_{n=-w}^{w} \sum_{m=-w}^{w} a_{n,m} \cdot F(j-n, k-m), \qquad (3.1)$$

where $F(j,k)$ denotes the input signal and the weight factors $a_{x,y}$ represent the coefficients of the high-pass filter with order w. In this model, the filtered signal acts as an approximation of the second derivative of the original image. With the aforementioned model, a generic sharpness improvement algorithm can be found by locally controlling the amount of overshoot added, i.e. making the addition adaptive to the local picture contents and the transmission impairments measured in the same area. We have concentrated on the following properties for adaptive control of the sharpness enhancement:

- local intensity level and related noise visibility;
- noise level contained by the signal;
- local sharpness of the input signal;
- aliasing prevention, where alias results from non-linear processing such as clipping.

Each of these control functions is very specific for a certain artifact. Consequently, dependent on the type of source signal it is desirable to enable a selection of the control functions. Moreover, future input signals may require different or additional control functions for different artifacts. For example, digitally coded signals generally contain artifacts such as blocking and ringing that are not detected by the above-mentioned control functions. As a result, these artifacts may be enhanced and become more visible. Hence, a modular architecture concept as depicted in Figure 3.1 is designed to enable

extensibility and modifications of the control functions. The blocks that denote "artifact metric", model the content-adaptive control functions to prevent artifacts. From the resulting suppression gains k_1 through k_n the overall suppression gain is derived and subsequently applied to control the output signal of the peaking filter. Let us now discuss the control functions in more detail.

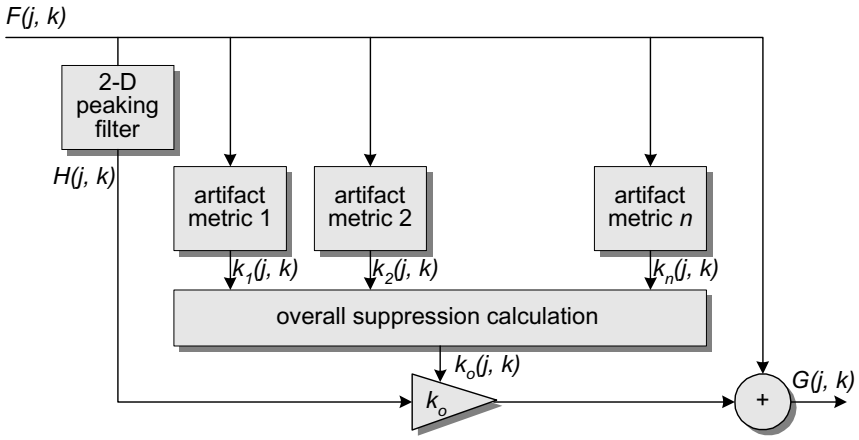


Figure 3.1: *Concept of the modular sharpness-enhancement architecture.*

3.2.2 Local intensity level and related noise visibility

The Human Visual System (HVS) is capable of perceiving a large range of intensities, but not simultaneously within an image [48]. Instead, the HVS handles this large variation by adapting itself to its overall sensitivity, a phenomenon known as brightness accommodation. The "simultaneously"perceived subjective brightness B is for a given brightness accommodation level B_b a continuous function of the intensity. B increases as a function of the intensity and is upper-bounded by peak-white and lower-bounded by pure black. The exact functional dependence of the intensity is unknown, but a mathematical approximation derived from experiments can be:

$$B = K_1 \cdot \{\mathrm{erf}(K_2(I - B_b)) + 1\}, \tag{3.2}$$

with K_1 and K_2 constants and B_b the brightness accommodation level (see Figure 3.2). Let us now consider the influence of noise on the perceived brightness. If the intensity level of an object in the image exceeds the accommodation level, noise with a negative amplitude is perceived to be more visible than noise with a positive amplitude at that position. This

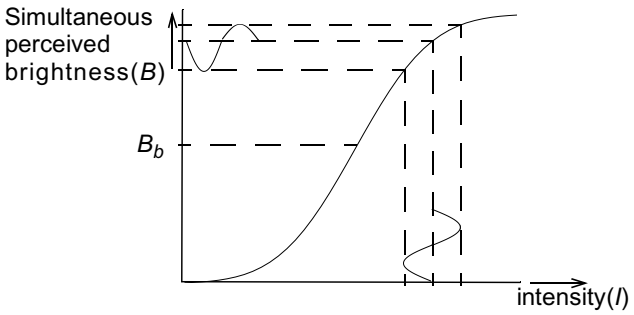


Figure 3.2: *Simultaneously perceived brightness as a function of the intensity.*

phenomenon is shown in Figure 3.2. To perceive the same amount of "positive" and "negative" noise, the noise with positive amplitude has to be larger than the noise with negative amplitude. Consequently, overshoot resulting from the 2-D high-pass peaking filter has to be larger than the undershoot from this filter for high-intensity regions in the image. For low-intensity regions in the image, an opposite statement applies. Here, high-intensity values are the intensity values higher than the accommodation level B_b and low-intensity values are the intensity values lower than B_b. However, the accommodation level also depends on intensity contributions from other sources than the display (sun light, light spots, etc.). For example, if a large amount of surrounding light is present while watching a television picture, details in dark areas cannot be distinguished. Since the surrounding light conditions are unknown in a TV set, the accommodation level B_b is assumed to be half the intensity range of the video signal ($B_b = 128$ for 8-bit intensity values). By compensating the amount of overshoot in high- and low-brightness areas, the perceived amount of overshoot is equal for all intensity levels. Experimental evaluations show that the subjective noise sensitivity decreases when the output of the peaking filter is corrected with a suppression gain $k_1(j,k)$ according to:

$$k_1(j,k) = \begin{cases} \frac{F(j,k)}{256}, & H(j,k) > 0 \\ \frac{255-F(j,k)}{256}, & H(j,k) \leq 0 \end{cases}, \tag{3.3}$$

where $F(j,k)$ is the input intensity level (0...255) and $H(j,k)$ stands for the output of the peaking filter.

3.2.3 Local sharpness of the input signal

Occasionally, the generated overshoot can be rather large, which may result in conspicuously visible sharpness or even annoying impressions of the image. For steep luminance transitions, the output of the high-pass filter has generally a high amplitude. This results in a considerable amount of – undesirable – extra overshoot at already steep edges. To prevent this, the local steepness of luminance transitions is measured. The steepness measurement is based on an approximation of the maximum derivative of the signal in all spatial directions. The approximation is based on determining the dynamic range of the signal within a small surrounding of the desired position. The steepness can be described mathematically as:

$$\begin{aligned} D(j,k) &= F_{max}(j,k) - F_{min}(j,k), D(j,k) \text{ with} \\ F_{max}(j,k) &= \max\{F(j-n,k-m) \mid n,m \in \langle -1,0,1\rangle\}, \\ F_{min}(j,k) &= \min\{F(j-n,k-m) \mid n,m \in \langle -1,0,1\rangle\}. \end{aligned} \tag{3.4}$$

Thus for 8-bit pixel values $D(j,k) \in [0..256\rangle$. This steepness indication can now be used to adaptively control the amount of sharpness enhancement. Hence, a correction gain $k_2(j,k)$ is applied to suppress the enhancement there where the steepness is too large. The relation between the correction gain $k_2(j,k)$ and the local steepness, is evidently inversely proportional to the dynamic range $D(j,k)$ according to:

$$\begin{aligned} k_2(j,k) &= \begin{cases} 0, & \hat{k}_2(j,k) < 0 \\ 1, & \hat{k}_2(j,k) \geq 1 \\ \hat{k}_2(j,k), & \text{elsewhere} \end{cases}, \\ &\text{where } \hat{k}_2(j,k) = 1.25 - 0.01 \cdot D(j,k). \end{aligned} \tag{3.5}$$

Thus for 8-bit pixel values $-1.3 \leq \hat{k}_2(j,k) \leq 1.25$. It can be seen that $k_2(j,k)$ suppresses the sharpness enhancement for steep edges, i.e. large dynamic ranges (see Figure 3.3). Relation (3.5) is an empirical relation that was found after conducting numerous experiments.

3.2.4 Noise contained by the signal (adaptive coring)

Sharpness enhancement amplifies the noise level and thus decreases the signal-to-noise ratio (SNR). Since the output of the enhancement filter is proportional to the local detail in the image, and the noise is equally distributed over the image, the SNR decreases in low-detail areas, so that noise in these image parts is mostly annoying. By suppressing the sharpness enhancement for low-detail areas, the subjective image quality can be

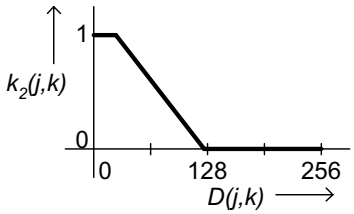


Figure 3.3: *$k_2(j,k)$ as function of the dynamic range.*

improved. If the output of the peaking filter is large, it may be assumed that an edge is detected. This is true when the signal power exceeds the noise power, thus if the SNR is larger than 0 dB. If the filter output is small, noise can be a significant part of the signal contents. Summarizing, the SNR can be preserved by suppressing the sharpness enhancement at positions in the image where the information content of the video signal is small. To explain the effect of applying a correction gain $k_3(j,k)$ at the output of the peaking filter we first define the peaking operation $P\{.\}$.

$$H(j,k) = P\{F(j,k)\} = \sum_{n=-w}^{w} \sum_{m=-w}^{w} a_{n,m} \cdot F(j-n, k-m),$$

where $H(j,k)$ denotes the output of the peaking filter and $F(j,k)$ is the input signal consisting of the original noise-free video signal $S(j,k)$ and a superimposed noise signal $N(j,k)$. Thus,

$$F(j,k) = S(j,k) + N(j,k).$$

Consequently, when applying a correction gain $k_3(j,k)$ at the output of the peaking operation, the output SNR becomes:

$$SNR_o = \frac{\mathrm{E}\left\{[S(j,k) + k_3(j,k) \cdot P\{S(j,k)\}]^2\right\}}{\mathrm{E}\left\{[N(j,k) + k_3(j,k) \cdot P\{N(j,k)\}]^2\right\}}, \tag{3.6}$$

Regularly, the noise signal is equally distributed over the image for a given noise level. When the term $P\{S(j,k)\}$ in Equation (3.6) is small, e.g. in low-detailed areas, the SNR can only be improved by decreasing $k_3(j,k)$. For other values of the filter output, $k_3(j,k)$ equals unity. The previously described behavior of $k_3(j,k)$ can be formulated by:

$$k_3(j,k) = \begin{cases} 0, & \hat{k}_3(j,k) < 0 \\ 1, & \hat{k}_3(j,k) \geq 1 \\ \hat{k}_3(j,k), & \text{elsewhere} \end{cases}, \tag{3.7}$$

$$\text{where } \hat{k}_3(j,k) = -0.25 + 0.05 \cdot |H(j,k)|,$$

with $H(j,k)$ the output of the peaking filter. This function, also known as coring, has a small transition to let $k_3(j,k)$ increase gradually from 0 to 1. Consequently, in flat low-detailed areas in the image where $S(j,k)$ is relatively large, the low filter output $H(j,k)$ will lead to a small $k_3(j,k)$. However, this solution has a disadvantage. When the noise power is small, $k_3(j,k)$ does not have to depend on $H(j,k)$ as described in Equation (3.7) to obtain a good SNR. A high SNR can also be obtained if $k_3(j,k)$ is decreased less for smaller values of $H(j,k)$. This problem is solved by giving $H(j,k)$ a penalty E depending on the noise level. Consequently, $k_3(j,k)$ becomes:

$$\hat{k}_3(j,k) = -0.25 + 0.05 \cdot (|H(j,k)| - E). \tag{3.8}$$

The noise level is determined using Equation (3.4). It was empirically found that the amount of noise is inversely proportional to M, where M is the number of times $D(j,k)$ does not exceed a certain threshold τ within a picture. This property can be formulated mathematically as:

$$M = \sum_{j=1}^{L_l} \sum_{k=1}^{L_f} U[D(j,k) - \tau], \tag{3.9}$$

where $U[.]$ denotes the unit-step function, L_l the number of pixels per video line and L_f the number of video lines per field or frame. Experimental results have shown that $\tau = 10$ is a satisfactory threshold value. The measured value of M is translated to the penalty E according to:

$$E = 50 - \left(\frac{M \cdot 2^{10}}{L_l \cdot L_f} \right). \tag{3.10}$$

Evidently, the value of M is normalized to the image size. The noise measurement as discussed above is not very accurate, but it proved to be sufficient for our application. When the image is noiseless (45 dB), the penalty

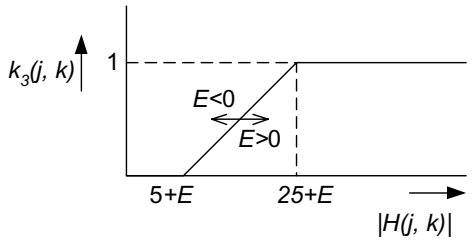


Figure 3.4: *$k_3(j,k)$ as function of the peaking filter output.*

decreases below zero and no coring will be applied. The suppression gain

$k_3(j,k)$ is depicted in Figure 3.4. It can be seen that the coring function shifts to the left when the image is noiseless and it shifts to the right if more noise is measured in Equation (3.9).

3.2.5 Prevention of aliasing from non-linear processing

Clipping of the signal to prevent the pixel values to exceed the maximum range, may cause artifacts. Clipping is a non-linear operation which creates additional cross-frequencies, which may partially fold back into the signal spectrum, causing undesired aliasing components. The only way to solve this problem is to prevent the occurrence of clipping.

The clipping problem occurs mostly locally in a number of samples on or close to a signal transient. If, in accordance with the previous subsections, a suppression factor $k_4(j,k)$ would be defined, the gain factor would show significant corrections for the clipped samples only, and little for the surrounding samples. Consequently, the parameter $k_4(j,k)$ would portray a rather discontinuous behavior at locations in the image where clipping occurs, thereby resulting in aliasing effects. Alternatively, if $k_4(j,k)$ would vary smoothly as a function of the sample position, the system would become locally linear and aliasing would not be visible. A smooth behavior of $k_4(j,k)$ is guaranteed if it is considered and controlled on an area basis (e.g. rectangular blocks) instead of a sample basis. As a result, the image is divided into a raster of blocks, each of them being represented by a single $k_4(j,k)$ value. Hence, the "subsampled"suppression gain of $k_4(j,k)$ is defined as:

$$K_4(J,K) = k_4(nJ + 0.5n, mK + 0.5m), \tag{3.11}$$

where n is the horizontal size of a block in pixels and m the vertical size of a block expressed in lines. Experimental results have shown that a block size of 32×32 is a good choice.

Let us now consider a workable definition of $K_4(J,K)$ and $k_4(j,k)$. It was found that the potential number of clippings that would occur in the actual block area is related to the smoothed version of $k_4(j,k)$ by the following reasoning. If $k_4(j,k)$ shows a strong local decrease in the correction gain in a particular area, then because of its smoothness, a significant number of samples will be affected. If the decrease is small, then the amount of samples affected is also small. Hence, the value of the correction is proportional to the number of clippings in that area. Since this relation was found to be more or less linear, $K_4(j,k)$ is defined as a value proportional to the number of potential clippings. The counted number of potential clippings,

called $N_C(J,K)$, is determined by

$$N_C(J,K) = \sum_{j=32J-16}^{32J+16} \sum_{k=32K-16}^{32K+16} (U[-H(j,k)-F(j,k)+0]+U[H(j,k)+F(j,k)-255]), \tag{3.12}$$

where $U[.]$ denotes the unit-step function and where $H(j,k)+F(j,k)$ represent the sharpness enhancement output if no correction gain would be applied. Hence, counting the number of times that $H(j,k)+F(j,k)$ is smaller than 0 and larger than 255 gives the number of potential clippings. Notice that $U[-H(j,k)-F(j,k)+0]$ and $U[H(j,k)+F(j,k)-255]$ equal 1 for $H(j,k)+F(j,k)<0$ and $H(j,k)+F(j,k)>255$, respectively. After counting these potential clippings, the counter value $N_C(J,K)$ is converted to $K_4(J,K)$ with a linear transfer function. This function equals

$$K_4(J,K) = \begin{cases} 0, & \hat{K}_4(J,K) < 0 \\ 1, & \hat{K}_4(J,K) \geq 1 \\ \hat{K}_4(J,K), & \text{elsewhere} \end{cases}, \tag{3.13}$$

$$\text{where } \hat{K}_4(J,K) = 1.3 - \frac{N_C(J,K)}{170}.$$

The division of an image into a raster of blocks is partially depicted in Figure 3.5. To reduce the amount of line memories, $K_4(J,K)$ is determined, stored and provided in the next field or frame. Prior to carrying out the actual gain correction, $K_4(J,K)$ has to be up-sampled and interpolated

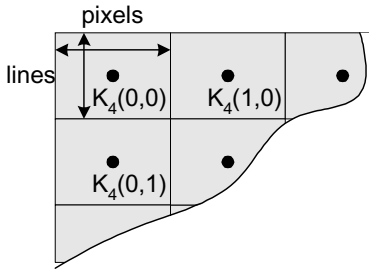


Figure 3.5: *Division of the image in a raster of blocks.*

to prevent blocking artifacts and is performed by a bilinear interpolation technique. This technique can be accomplished with low cost by using three one-dimensional linear interpolations as shown in Figure 3.6. Let us now consider the bilinear interpolation in more detail. The first vertical

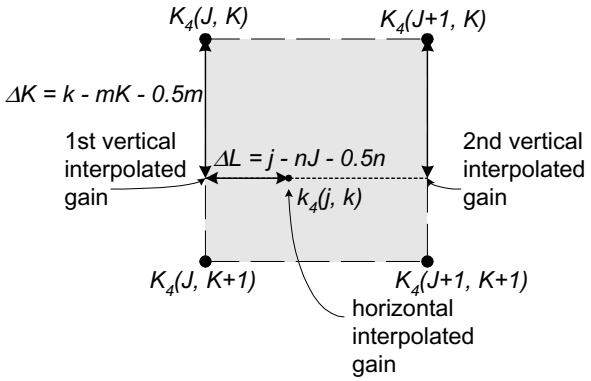


Figure 3.6: *Bilinear interpolation of $K_4(j,k)$ values.*

interpolation is done according to:

$$\begin{aligned} a_1 &= K_4(J,K), \\ a_2 &= K_4(J,K+1) - K_4(J,K), \text{ and} \\ k_4(nJ+0.5n, mK+0.5m+\Delta K) &= a_1 + a_2 \cdot \Delta K. \end{aligned} \tag{3.14}$$

Similarly, the second vertical interpolation yields

$$\begin{aligned} b_1 &= K_4(J+1,K), \\ b_2 &= K_4(J+1,K+1) - K_4(J+1,K), \text{ and} \\ k_4(nJ+1.5n, mK+0.5m+\Delta K) &= b_1 + b_2 \cdot \Delta K. \end{aligned} \tag{3.15}$$

Finally, the results of Equations (3.14) and (3.15) are used for a horizontal interpolation, leading to

$$\begin{aligned} c_1 &= k_4(nJ+0.5n, mK+0.5m+\Delta K), \\ c_2 &= k_4(nJ+1.5n, mK+0.5m+\Delta K) \\ &\quad -k_4(nJ+0.5n, mK+0.5m+\Delta K) \text{ and} \\ k_4(nJ+1.5n, mK+\Delta J, mK+0.5m+\Delta K) &= c_1 + c_2 \cdot \Delta J. \end{aligned} \tag{3.16}$$

The value of $k_4(j,k)$ is calculated sequentially by first letting j run from 1 to L_l and subsequently letting k run from 1 to L_f. Due to the sequential index augmentation, the calculation can be made recursive. Rewriting Equation (3.16) as a function of the $K_4(J,K)$ and writing $k_4(j,k)$ as a function of $k_4(j-1,k)$ leads to:

$$\begin{aligned} k_4 &= K_4(0.0) \\ k_4(j,k) &= k_4(j-1,k) + c_2 \\ &= k_4(j-1,k) + b_1 - a_1 + (b_2 - a_2) \cdot \Delta K. \end{aligned} \tag{3.17}$$

For each new block, the terms a_1, a_2, b_1 and b_2 are recomputed. It can be noticed that one multiplication and a few additions are sufficient to perform each iteration. Vertical recursion is also possible at the cost of a video line memory, in order to restore $k_4(0, k-1)$ till $k_4(L_l, k-1)$.

Experimental results showed that the problems due to clipping were still not solved completely. Aliasing was not only produced by the two spatial dimensions, but also by the temporal dimension. Apparently, $k_4(j,k)$ also contains discontinuities in the temporal domain. The aliasing produced by the time component is perceived as field flicker. This problem was solved by filtering $K_4(J,K)$ with a first-order recursive filter as a function of time. As was mentioned before, $K_4(J,K)$ is determined, stored and subsequently provided in the next field or frame. The temporal filter updates the stored $K_4(J,K)$ instead of replacing it (see Figure 3.7). The block size influences both the image quality and the implementation as follows.

- Spatial aliasing: too small blocks lead to a inconsistent $k_4(j,k)$ and too large blocks lead to loss of local adaptivity.
- Required memory resources to restore $K_4(J,K)$.
- Errors in the temporal moment of $k_4(j,k)$: due to the temporal filtering and the motion in the images, small blocks will lead to larger position errors in $k_4(j,k)$.

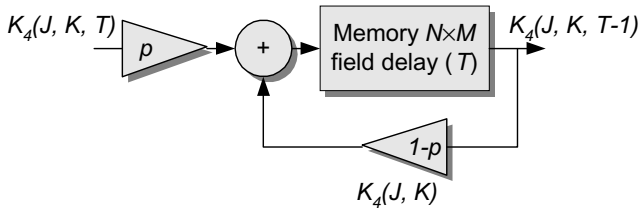


Figure 3.7: *Temporal filtering of the gain factor to suppress temporal aliasing.*

Although sharpness enhancement will locally decrease in the areas where much clipping would occur, aliasing is perceived as more annoying than the local sharpness loss.

3.2.6 The system including all controls

In the overall sharpness enhancement model, the overshoot at edges in the peaking processing is suppressed by a special control parameter $k_o(j,k)$,

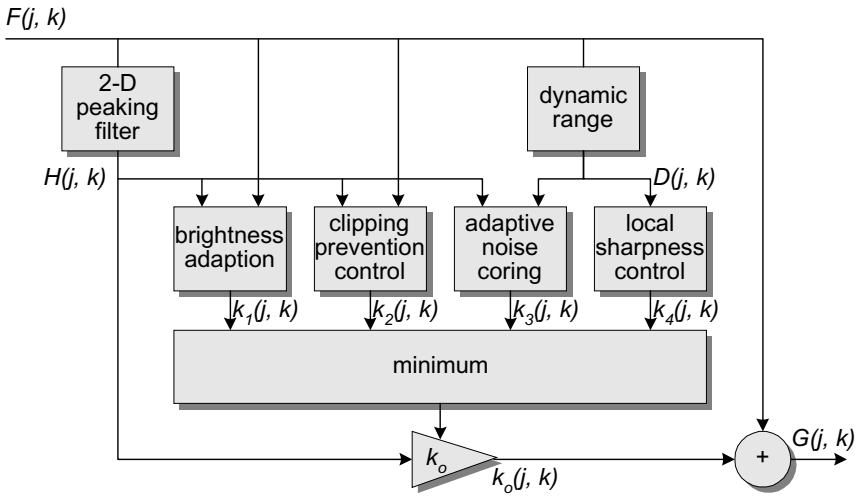


Figure 3.8: *Block diagram of the generic 2-D peaking.*

which is determined by all aforementioned artifacts and considerations. The overall block diagram of the algorithm is depicted in Figure 3.8. In the complete system, the formulation of the peaking becomes as follows:

$$G(j,k) = F(j,k) + k_o(j,k) \cdot \sum_{x=-w}^{w} \sum_{y=-w}^{w} a_{x,y} \cdot F(j+x, k+y), \qquad (3.18)$$

where $k_o(j,k)$ is bounded between $0 \leq k_o(j,k) \leq 1$. When one of the individual contributions $k_i(j,k)$ for $i = 1...4$, portrays a large occurrence of the corresponding artifact, the correction of the added overshoot will be large too, leading to a small $k_o(j,k)$. In this model, the smallest correction factor represents the most annoying artifact. When the gain of the smallest artifact measure is applied to suppress the total enhancement, the remaining artifacts will be suppressed as well. We have found that this decision criterion yields a good performance, although it is a highly non-linear operation.

3.2.7 Results and conclusions

This section presents a sharpness enhancement technique for TV applications based on adding overshoot to luminance transitions in the image. A key feature of the system is that the amount of overshoot added depends on four correction gain parameters. These parameters result from analyzing different properties of the local image signal, i.e. intensity level, noise level,

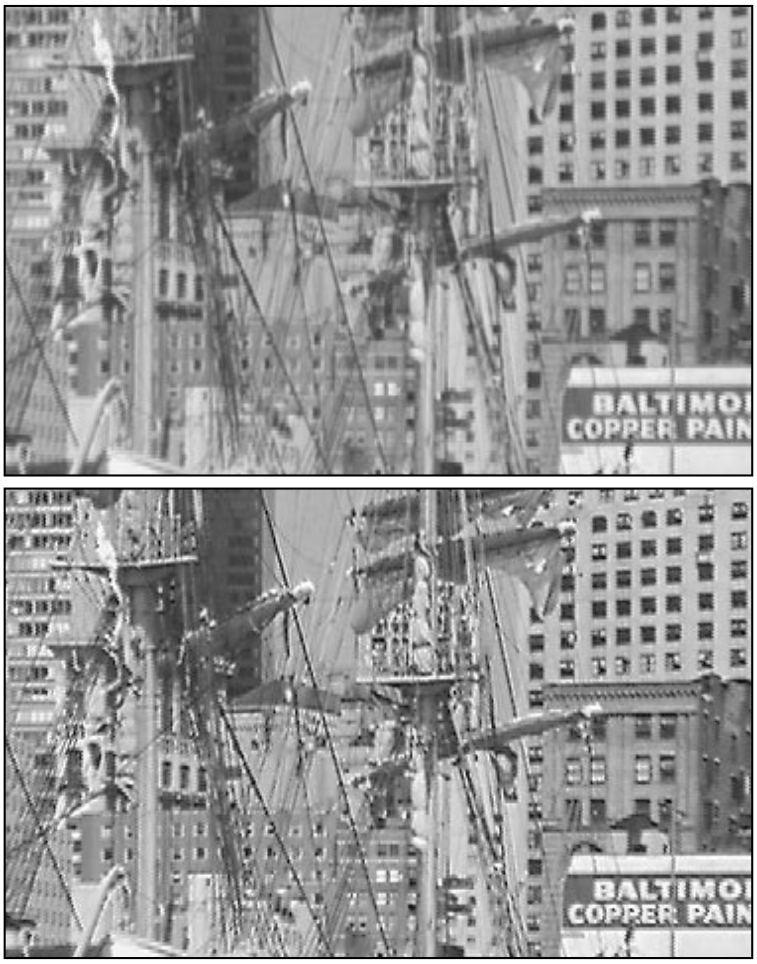


Figure 3.9: *The original 'Baltimore' image (upper picture) and the sharpness enhanced version of this image (lower picture).*

local sharpness of the input and potential clipping artifacts. The concept enables a stable sharpness improvement for a large variety of TV scenes.

The algorithm has been evaluated by computer simulations and the results of the adaptive peaking together with its special artifact control performs satisfactory. The algorithm is robust with respect to varying image statistics which is explained by the artifact controls, while the performed sharpness enhancement is good. Even for high-quality video signals, a remarkable improvement is obtained (see Figure 3.9 in which the original is

digital YUV). The appropriate suppression of overshoot in textured areas with large signal amplitudes or areas containing significant noise energy, is working quite effectively and subjectively improves the image quality.

3.3 Advanced sampling-rate conversion

To show that expenses of different functions may be related to different implementation aspects, this section discusses another example function in more detail. Where the example in the previous section is complex in terms of computations, the sampling-rate converter as described in this section [10] is particularly complex in terms of memory usage, when applied in vertical direction. The first subsection will motivate the need for such a function in the TV domain, whereas the subsequent subsections will elaborate on the algorithm.

3.3.1 Introduction to video scaling

As was mentioned in the previous section, a merge of video technology for TV and PC is foreseen. In a multimedia PC environment, it is common to display several windows containing different video signals, such as MPEG, JPEG and Graphics. The windows can be scaled in arbitrary sizes. However, the quality of this scaling for video signals is rather poor. In high-end TV sets, Picture-in-Picture (PiP) is an established feature and this is currently being upgraded to dual-window TV. In the future, with an increasing number of broadcast TV channels and the augmented application of MPEG decoding in TV (via DVD, DVB), it is expected that more windows of video signals and graphics signals will be used for display. On top of this, graphics and menus will be superimposed for user interface and control. The parallel display of several of the aforementioned signals relies heavily on sampling-rate conversion.

Until now, sampling-rate converters were more or less dedicated and optimized to perform one function (e.g. size conversion for PIP). This leads to architectures in which sampling-rate converters, tuned for different applications, are scattered, thereby leading to – sometimes – higher system costs. The objective for future TV, however, is to design a more flexible, high-quality 2-D scaler, that can be used for a broad range of TV functions and graphics applications. It is obvious that flexible advanced scalers will become an integral part of consumer electronics and multimedia products. The system requirements for defining the desired high-quality sampling-rate converter for TV applications are severe and listed below.

- **Large dynamic scaling range**. Current PIP converters yield a good visual performance for small picture sizes only. However, for Split Screen/Dual Window, the secondary picture has a comparable size to the main picture. The video quality of the secondary picture should therefore be equal. In multi-window applications, any size of a video window may be used and the quality should always be good.

- **Matching to source material**. High-quality size conversion of graphics and video should both be enabled. The visibility of aliasing for graphics signals differs substantially from that of bandwidth-limited video signals.

- **Matching to the resolution of the display**. The video signal processing with its scaling should be applicable to various displays, such as CRT, LCD and plasma displays.

- **Down-scaling and up-scaling**. The converter should not only be used for zooming (up-scaling) the video signal, but also for compression (e.g. PIP) with the same level of picture quality.

Current implementations of sampling-rate converters are usually only suited for a limited conversion range. In this section, an advanced sampling-rate conversion technique is presented which is suited for high-quality up- and down-scaling of video and/or graphics source material, using a large conversion range.

3.3.2 Basic theory of sampling-rate conversion

In order to understand sampling-rate conversion, the basics of sampling and sampling-rate conversion theory are described briefly.

Since sampling implies that the values of a signal are only evaluated at discrete points of time, sampling can be described as a product of delta sampling on equidistant time intervals and a continuous signal $x(t)$, so that the sampled signal $x_s(t)$ becomes:

$$x_s(t) = x(t) \cdot \sum_{n=-\infty}^{\infty} \delta(t - nT). \tag{3.19}$$

Note that $x_s(t)$ is still a continuous signal. The Fourier transform of the sampled function can be calculated using the product theorem:

$$\begin{aligned} X_s(\omega) &= \frac{\omega_0}{2\pi} X(\omega) * \sum_{n=-\infty}^{\infty} \delta(t - nT), \\ &= \frac{\omega_0}{2\pi} \sum_{n=-\infty}^{\infty} \delta(\omega - n\omega_0), \end{aligned} \tag{3.20}$$

where $\omega_0/(2\pi) = f_0 = 1/T$ and '$*$' represents the convolution operator. The combination of the convolution operation and the periodic delta function results in a repetition of spectrum $X(\omega)$ along the ω-axis.

It can be deduced that an error-free reconstruction (separation of the baseband spectrum) is possible if the original signal is properly limited in bandwidth with cut-off frequency ω_c, meaning that it should satisfy $\omega_0 = 2\pi/T \geq 2\omega_c$, which is known as the sampling theorem. Perfect reconstruction of the signal $x(t)$ may be performed by using a low-pass filter with the following characteristics:

$$L_p(\omega) = \begin{cases} 1 & \text{for} |\omega| < \omega_0/2 \\ 0 & \text{elsewhere} \end{cases} . \tag{3.21}$$

The impulse-response according to (3.21) is

$$l_p(t) = \frac{sin(\pi t/T)}{\pi t/T}, \tag{3.22}$$

which is the so-called *sinc* function. For simplicity, the sampled signal can be considered as a sum of weighted delta pulses. This model allows the sampled signal to be written as a sequence of numbers representing the signal samples, or alternatively, as the pulse weights of a delta sequence. Accordingly, we find that

$$\{x[n]\} = \{x(nT)\} = \sum_{n=-\infty}^{\infty} x(nT) \cdot \delta(t - nT), \tag{3.23}$$

where $\{x[n]\}$ denotes the sequence of sampled values of the analogue signal $x(t)$ at time nT. This theory can be extended to describe sampling-rate conversion in more general terms.

Let $\{x[n]\}$ denote a sequence of sampled values at time nT_i where T_i is

the input sampling period. Assuming that the bandwidth of the continuous signal $x(t)$ is less than $f_i/2$, the value of this signal at any point of time t can be calculated exactly from $\{x[n]\}$, using

$$x(t) = \sum_{n=-\infty}^{\infty} x[n] \frac{sin(\pi(t - nT)/T)}{\pi(t - nT)/T}. \tag{3.24}$$

It can be understood that (3.24) can be rewritten as a basic formula for sampling-rate conversion. If Equation (3.24) is calculated at output sampling moments mT_o, the output sampling rate is different from the input rate. Therefore, Equation (3.24) can be rewritten into an equation that changes the sampling rate [49], so that

$$y[m] = \sum_{n=-\infty}^{\infty} x[n] \frac{sin(\pi(mT_o - nT_i)/T_i)}{\pi(mT_o - nT_i)/T_i}, \tag{3.25}$$

where $y[m]$ is the exact value of the time-continuous signal at time $t = mT_o$. Equation (3.25) shows that for the calculation of one output sample, all the input samples are required. In practice, however, an output sample is computed using a set of input samples only, thereby introducing an error in the output signal. The aim of digital conversion techniques is to replace the *sinc* interpolation function by another function of finite length and which limits at the same time the error in the output signal. Note that Equation (3.25) is only valid for $T_o \leq T_i$. In the case that $T_o > T_i$, the resulting signal may suffer from aliasing. This can be prevented by filtering out a part of the high-frequency components of the input signal, depending on the ratio between T_i and T_o.

3.3.3 Considerations for SRC implementation

Digital schemes used for sampling-rate conversion (SRC) are increasingly popular, because the required hardware becomes cheaper as VLSI technology advances. In digital schemes, Equation (3.25) can be seen as a convolution of the input stream with a digital filter having an impulse response which resembles the *sinc* function. As stated already, this filter is not realizable since it is of infinite length. However, there are many techniques to design filters which have a frequency response close to the ideal low-pass filter as described in Equation (3.21).
Since the input and output sampling frequencies are different, it is difficult to visualize Equation (3.25) as a digital filtering equation. This is easier to understand if input and output of the digital filter operate on the same

sampling frequency. Equal frequencies are obtained by creating an intermediate frequency which is a multiple of both the input and the output frequency. Let us assume that the input and output sampling frequencies are related by ratio K/L. The corresponding conversion scheme is depicted in Figure 3.10. Initially, the input samples are up-sampled by an integer

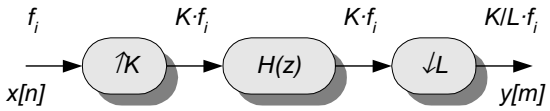


Figure 3.10: *Sampling-rate conversion by means of digital filtering.*

factor K. Up-sampling is achieved by simply adding $K-1$ zeros between every incoming sample. As a result, the intermediate sampling frequency operates at a sampling rate of $K \cdot f_i$. Subsequently, a filter $H(z)$ operating at the same intermediate frequency, filters out the undesired parts of the video spectrum. Finally, the sequence is decimated with a factor L. This implies that the output sampling rate equals K/L times the input sampling rate.
Implementations relying on the above-mentioned scheme typically have a fixed filter $H(z)$, a fixed up-sampling factor K, and a variable down-sampling factor L. Consequently, the implementation is only suited for e.g. up-scaling and limited down-scaling.

Generally, the implementation of digital Sampling-Rate Conversion (SRC) can be divided into three categories [50][51]: curve fitting, digital filtering (combination of interpolation, filtering and decimation) [52], and hybrid architectures, which are a combination of curve fitting and digital filtering. These three categories can all be modelled according to the block scheme depicted in Figure 3.10. Let us now briefly address each technique individually.

- *Digital filtering* schemes for sampling-rate conversion are becoming increasingly popular. These implementations actually perform an up-sampling, followed by filtering and decimation. An example implementation of a digital filtering scheme is a polyphase filter. A polyphase filter is a sampling-rate converter which combines the up-sampling, low-pass filtering and decimation into a single function. The low-pass filter can be fixed, selectable or programmable.

- *Curve fitting* is a more classical interpolation technique which is often used in low-cost implementations. Instead of performing up-sampling

followed by filtering and decimation, the desired pixel values are determined by calculation the coefficients of a spline (e.g. linear, quadratic, cubic or any other n-th order polynomial function), using the original input pixel values. Subsequently, the output pixel values are derived by substituting the positions corresponding to the desired sampling rate. The most simple variant of curve fitting is linear interpolation, which just takes the weighted average of two neighboring samples.

- *Variable Phase Delay* (VPD) filters applied in current TV sets, are basically hybrid implementations, using a two-phase up-sampling filter followed by a quadratic interpolation [53].

A common disadvantage of the aforementioned techniques is that they give a disappointing performance for a substantial compression of video and graphics signals.

Since up- and down-scaling are both desired key features of the converter, this section discusses a new architecture satisfying this constraint with a high performance. The key to the solution is that the filter bandwidth is coupled to the stage having the *lowest* sampling frequency (e.g. for up-scaling, this is the input stage). Considering the same solution for compression, the bandwidth of the input signal must be limited in order to prevent aliasing. The required bandwidth limitation depends on the compression factor. The three SRC categories discussed earlier usually apply an adjustable or selectable prefilter to limit the bandwidth of $x[n]$ to $f_o/2$, thereby enabling high-quality compression. Without prefiltering, high quality is only obtained for $L \leq K$.
Conventional SRC implementations aiming at up- and down-scaling lead to different architectures. In this section, the objective is to map both properties onto a single architecture. This is achieved by using the concept of transposition, which is discussed in the subsequent subsection.

3.3.4 Transposition of a sampling-rate converter

In literature [50], it is stated that if the directions of all branches in a signal flow-graph for a discrete linear time-invariant network are reversed, the function of the resulting *transposed* network is the same as that of the original network. All branch nodes in the original network become summation nodes and vice versa. Systems containing up- and/or down-sampling are in principle time-varying functions. Generalized rules for transposition of such systems were derived in [54]. The main element pairs which transpose mutually according to those generalized rules are summarized by:

- up-samplers become down-samplers and vice versa;
- an addition becomes a signal split and vice versa;
- constant multipliers and delays remain the same.

Applying the transposition rules to Figure 3.10 leads to the sampling-rate converter depicted in Figure 3.11. When considering compression, $H(z)$

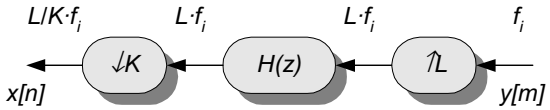


Figure 3.11: *Sampling-rate converter after transposition.*

results in a decimating filter for $L \leq K$(note that for the non-transposed case in Figure 3.10, this filter is an interpolation filter). Assume again the usual case where K is fixed and L is variable. It can be seen that in the transposed mode, $H(z)$ operates on $L \cdot f_i$. Since L is variable, $H(z)$ operates on a *varying* sampling rate. As a result, depending on L, a different part of the baseband spectrum is filtered out. In other words, an important property of a transposed SRC is that it inherently prefilters the input signal according to the compression ratio.

Let us consider an example in which the inherent prefiltering is illustrated. Assume a decimation factor $K = 8$. In Figure 3.12, the spectrum of a

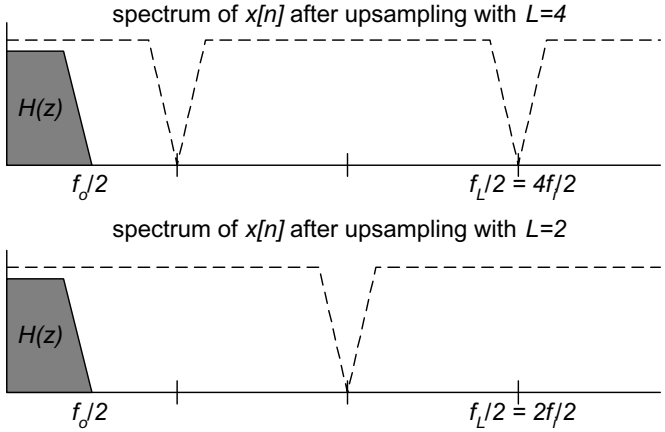


Figure 3.12: *Filter operation in transposed mode for fixed down-sampling and a variable up-sampling factor.*

signal $x[n]$ is shown after up-sampling with factor $L = 2$ and $L = 4$. The up-sampled signals are filtered with $H(z)$. After filtering, the signals are decimated with a factor 8. It can be seen that a different part of the base-band spectrum is filtered out, as a result of the choice of the up-sampling factor L. This is the strength of a transposed filter structure. From Figure 3.12 we can conclude that a filter $H(z)$ can be designed such that for all $L < K$, it is ensured that the filtered signal is bandwidth-limited to $f_o/2$ prior to decimation.

3.3.5 Requirements for transposed filters

Fundamentally, transposition can be applied to any type of SRC implementation, but this requires careful filter design. This subsection will explain this in more detail. Filters of the non-transposed type as depicted in Figure 3.10 are usually designed to comply with the following requirements.

- Sufficient passband flatness.
- Stopband attenuation is below a certain threshold (e.g. below -60 dB).
- DC input gives a DC output.

Let us consider the last requirement in more detail for both the normal and the transposed case. For the normal case, the up-sampling leads to K repetitive spectra at center points $n \cdot f_i$ for $1 \leq n \leq K$ and n being an integer. Knowing that decimation with L will follow, the spectra will fold back. Let us assume that the filter $H(z)$ operates on frequency f_F. To ensure that DC input yields DC output, it can be concluded that $H(z)$ must have spectral zeros at $n \cdot f_F/K$ for $1 \leq n \leq K/2$and n being an integer. This is easy to achieve, since K was assumed to be fixed.

For a transposed filter, the constraint that a DC input results into a DC output, the filter $H(z)$ must have spectral zeros at $n \cdot f_F/L$ for $1 \leq n \leq L/2$ and n being an integer. After decimation, these frequencies fold back to DC. Since K is a fixed down-sampling factor and L is a variable up-sampling factor, this is impossible to meet for all L. Therefore, transposed filters need additional design constraints. If these constraints are not taken into account, the output will definitively suffer from aliasing. Especially for DC (DC-ripple) and for low frequencies, the aliasing can become well visible and can be rather annoying.

Experiments have shown that when the DC ripple is below -54 dB, aliasing is no longer visible on a high-quality monitor. This implies that the requirement $\mathrm{DC}_{in} = \mathrm{DC}_{out}$ may be relaxed. Instead of having spectral zeros at $n \cdot f_F/L$, it satisfies to design a filter with sufficient stopband suppression at these frequencies.

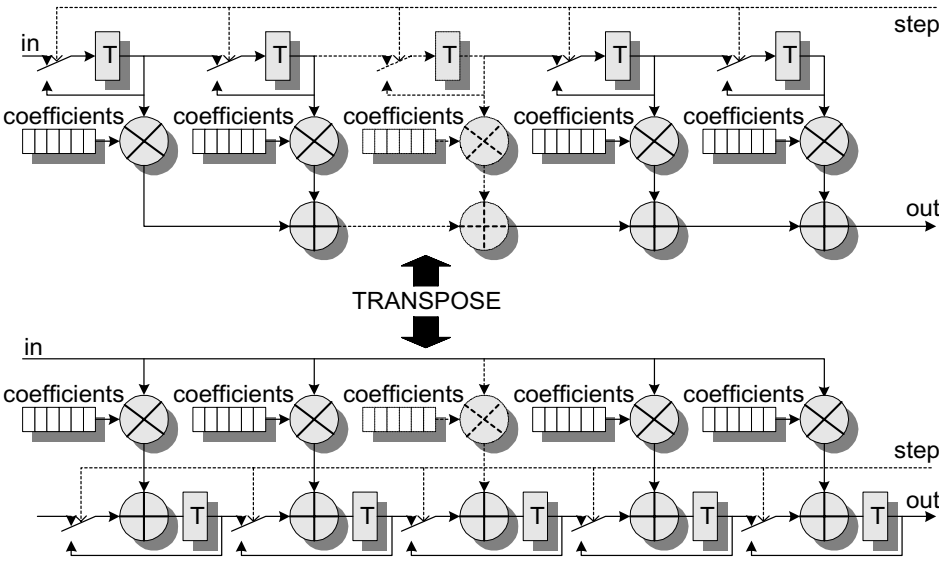


Figure 3.13: *Transposition of an N-taps polyphase filter.*

3.3.6 Experiments and results for polyphase filters

A normal and a transposed implementation of an SRC, based on a 6-tap, 64-phase polyphase filter, have been implemented in software. Figure 3.13 portrays a possible architecture of the normal and transposed polyphase filter [55]. Filter characteristics have been optimized for both the regular and the transposed mode. The results have been obtained by computer simulations. For more severe testing, the regular and transposed polyphase filter have been implemented on a real-time video system. The results of the transposed algorithm outperforms clearly the existing implementations in current TV receivers. By switching between transposed and regular polyphase operation, high-quality up- and down-scaling is possible without adding an adjustable prefilter.

Figure 3.14 shows the visual results for a up- and down-scaling factor of 2.6. The upper left picture portrays that a normal filter leads to substantial

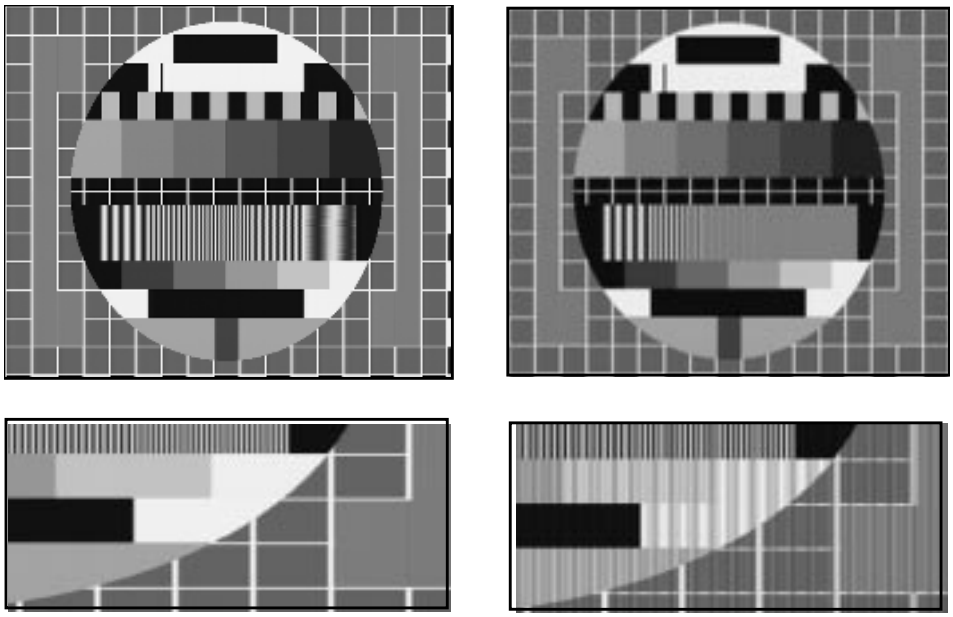


Figure 3.14: *Results of transposition of an N-taps polyphase filter.*

aliasing in the multiburst and the cross-hatch pattern. It can be seen that the performance of the transposed filter (upper right) is much better due to the proper bandwidth limitation. The opposite case is shown in the lower part for up-scaling. The left picture show up-scaling with the normal filter and in the right the picture is up-scaled using a transposed filter.

As a result of the scaling of the filter passband, the obtained picture quality remains high for a large range of up- and down-scaling factors. Extensive experiments were conducted with up- and down-scaling factors between 1 and 16. Furthermore, since the regular and transposed implementation rely on the same resources, implementations of both operation modes can be mapped on the same hardware architecture.

Finally, the suitability for sampling-rate conversion of graphics material has been examined. The spectral energy distribution of graphics source material differs significantly from that of video. Without taking extra care, conversion of graphics material may result in visible ringing around steep edges. Therefore, new filters have been designed which reduce considerably the ringing artifacts for graphical images. Also for graphics, the transposition concept has proven to be convenient to realize high-quality compression.

3.3.7 Conclusions on video scaling

The increasing amount of TV channels and alternative information sources in a TV system lead to the requirement to display several video and/or graphics signals in parallel. For the scaling of the display windows of these sources, high-quality sampling-rate converters are desired.

A generic technique has been presented to obtain high-quality sampling-rate conversion, both for up- and down-scaling. It has been found that by transposing the architecture for expansion, a sampling-rate converter can be made suitable for compression. Since the architecture of a transposed network is similar to that of the regular version, the implementations of both types can be combined into a single hardware architecture.
It has been concluded that basically any linear sampling-rate converter can be transposed. However, some extra design rules have to be applied to ensure high-quality compression. Especially the DC ripple suppression appears to be an important design constraint.

In the experiments, a 6-tap 64-phase polyphase filter was applied, which showed a very satisfactory performance. Particularly for compression, the transposed structure leading to the inherent prefiltering of the input signal, has proven to be valuable. A polyphase filter is intrinsically attractive, because the filter coefficients are freely programmable. Furthermore, the polyphase filter has shown robustness in performance with respect to varying image statistics.

3.4 Computational costs of video functions

3.4.1 Estimation model for complexity

The mapping of an algorithm onto a preselected architecture is no straightforward process. The optimal architecture depends on choices in the algorithm, e.g. for a high-performance floating point implementation of an IDCT with a 52-bit precision [56], a different computational platform is required than for a fast integer implementation with a 32 or even 16-bit precision [57][58]. On the other hand, some restrictions in the architecture may influence the choice for a particular algorithm. For example, the bandwidth to external memory may be too scarce for performing a full-search motion estimation. Instead, a three-Dimensional Recursive Search (3DRS) block-matching algorithm [59] can be used. Summarizing, it is a difficult task to compare the performance of functions without considering the platform onto which they have to be executed. For a good comparison,

the algorithms of the application have to be optimized for each platform in order to achieve optimal performance. However, changing the algorithm implies different functionality, thereby resulting in a comparison of different subjects. This is caused by the iterative nature of the design process as depicted in Figure 3.15.

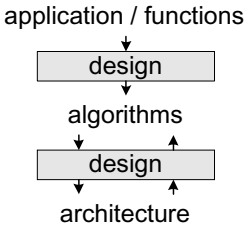


Figure 3.15: *Iterative design process.*

The peaking algorithm as described in Section 3.2 is tuned for an application-specific embedded IP block in a multiprocessor system as described in Chapter 4. Therefore, the number of Operations Per Second (OPS) does not imply the computational requirements for a software implementation on a general-purpose CPU. Depending on the architecture and therefore on the algorithm, the operation count of a function can deviate significantly. To come to clear insights by means of an example, both the above discussed sharpness enhancement algorithm and a software-optimized sharpness enhancement algorithm will be analyzed in the following subsections. Besides algorithmic differences of functions, also the operations may require a different number of cycles for execution. Nevertheless, such an evaluation gives a good impression of the computational requirements when compared with other algorithms on the same platform. In succeeding subsections, the analysis of the computational performance is elaborated in more detail. It is assumed that a very simple Harvard RISC processor is used with a very limited amount of local registers. It is assumed that no stall cycles are present because sufficient bandwidth resources are available and a sufficient caching hierarchy is provided including prefetching to hide the latency of the memory access operations. Although these assumptions result in rough estimates only, the order of magnitude gives sufficient information to formulate the high-level conclusions. Some example calculations are:

$$r1 = a + b,$$

where $r1$ is a variable in a local register and values a and b are read from the memory. Hence, this function requires 2 read (RD) operations, and 1 add (ADD) operation, thus in total 3 operations. Consider now the function

$$z = ax + by,$$

where y is stored in memory and a and b are coefficients for the data x and y. This functions requires 4 RD, 2 multiplications (MUL), 1 ADD and 1 store (STR) operation. In total 8 operations. When it is assumed that all operations can be performed within the same amount of cycles, the operations per sample (OP/sample) are proportional with the amount of cycles per sample.

3.4.2 Sharpness Enhancement complexity estimation

The functional blocks in Figure 3.8 divide the sharpness enhancement algorithm of Section 3.2 in tasks which are discussed individually.

- **2-D Peaking filter** - The filter consists of a 2-D symmetric FIR filter with seven taps in horizontal direction and three taps in vertical direction. Using the symmetry, 20 add (ADD), 8 multiply (MUL) and 29 read (RD) operations are used to do the filtering. Furthermore, 1 additional MUL, 1 logic shift right (LSR), 1 ADD and 2 RD operations are used to scale and quantize the output value. Finally, the result is clipped between -256 and 255 using 2 compare (CMP) operations. Note that an additional assign operation is required for each compare if the output value actually exceeds the clipping value. We assume that this does not hold in most cases and hence do not add an additional assign operations. In total, the 2-D peaking filter uses 65 OPs/sample including 1 operation to store (STR) the result.

- **Brightness adaptation** - This task uses 1 CMP, 2 RD, 0.5 subtract (SUB) and 1 STR operation, in total 4.5 OPs/sample. The non-integer values of the SUB operations are determined by using the probability of the binary result of the CMP operation. In this case the probability is assumed to be 50 %.

- **Clipping prevention control** - This task can be subdivided in a subtask for counting the number of potential clippings per 32×32 block, a subtask to update all pixel, line, stripe and block counters, a subtask to determine the suppression gain dependent on the amount of potential clippings, and a subtask to do the bilinear transform. For the counting of the potential clippings, 4 ADD, 4 CMP, 8 RD, 2 AND, 2 LSR and 2 STR operations are necessary. In total 22 OPs/sample. The subtask to update all counters uses on average 7 CMP, 2 OR, 2 ADD, 6 RD, 4 STR and 2 modulo (MOD) operations. In total 23 operations are used per sample. Note that these operations are only executed once every block or every stripe and therefore have only limited impact on the operations per sample. The subtask to determine

the suppression gain is only executed once per block. Consequently, the operations per block have to be divided by 32×32 (= block size). Per block 21 RD, 4 CMP, 6 STR, 8 ADD, 5 MUL, 2 LSR and 1 SUB operation is executed resulting in less than 0.05 OP/sample. Finally, the bilinear transfer is applied using 35 operations per block change (every 32 pixels) and 25 operations every sample. In total, approximately 26 operations are spent per sample. For the complete clipping prevention control, 71 OPs/sample are executed on average.

- **Dynamic range** - This function determines the maximum absolute difference within a 3×3 window around the pixel of interest. The implementation uses 12 CMP, 22 RD, 15 STR and 1 SUB operation, thus in total 50 operations per sample.

- **Local sharpness control** - This task converts the dynamic range according to a linear function, preceded by clipping, to a suppression gain. This consumes 4 RD, 2 ADD, 1 MUL, 1 LSR, 2 CMP and 1 STR operation, bringing the total to 11 OPs/sample.

- **Adaptive noise reduction** - For this task a simple noise measurement algorithm uses the result of the dynamic range measurement. This requires 40 OPs/sample. The calculation for the suppression gain occupies 1 absolute operation (ABS), 7 RD, 1 SUB, 2 ADD, 1 MUL, 1 LSR, 2 CMP, and 1 STR operation. For the complete adaptive noise reduction 56 OPs/sample are necessary.

- **Minimum** - This task determines the minimum suppression gain, applies this gain, and provides an adjustable sharpness enhancements by means of an additional multiplication. The result is then scaled and clipped. This results in 7 RD, 2 MUL, 6 CMP, 2 STR, 1 ADD, 1 SUB, 1 LSR and 1 DIV operation. In total 21 operations

- **Add** - Finally the enhancement signal is added to the original sample and clipped between the original pixel range. This requires 6 operations divided into 2 RD, 1 ADD, 2 CMP and 1 STR operation.

The summation of the total operation budget per sample for an advanced sharpness enhancement algorithm results in 285 operations. When Standard-Definition (SD) interlaced video with a 25-Hz frame rate and a resolution of 832×576 is processed, the total operation bandwidth equals 3.4 GOPS. Table 3.1 summarized the total number of operations per sample.

Another complexity metric that is important is the *local memory usage* and the *bandwidth* to the external memory. These metrics can often be

Table 3.1: *Operations per sample for sharpness enhancement.*

Task	Operations per 32 samples												Total/ sample
	RD	STR	ADD	SUB	AND	OR	CMP	LSR	MUL	DIV	ABS	MOD	
2-D filter	992	32	672				64	32	288				65
Brightness adaptation	64	32		16			32						4.5
Clipping prevention	685	330	259	97	97	64	578	98	3			64	71
Dynamic range	704	480		32			384						50
Local sharpness	128	32	64				64	32	32				11
Adaptive noise reduction	832	352	64	64			384	32	32		32		56
Minimum	224	64	32	32			192	32	64	32			21
Add	64	32	32				64						6
Total													**285**

exchanged, dependent on whether processing data is stored locally or in an external memory. For example, the sharpness enhancement algorithm uses data from three video lines simultaneously. Some embedded memory could be used to store two video lines, thereby avoiding multiple reading of data from the external memory. Obviously, it is also possible to accept the high memory bandwidth when no buffer memory is provided. Because memory bandwidth is generally the bottleneck in multimedia processing systems, it is assumed that embedded line memories are used. As a result, the amount of embedded memory is 1.8 kByte, including the temporal storage of the subsampled suppression gains $K_4(J, K)$, of the clipping prevention task.

To show the potential large variation of computational complexity of similar functions and the influence of the algorithm choice, an alternative sharpness enhancement algorithm [60] is analyzed that was designed for a different purpose and for which only small enhancements are required (see Figure 3.16). This algorithm comprises a simple fixed 2-D filter, a look-up table (LUT) for non-linear signal adaption such as coring (see Subsection 3.2.4), and a gain control to adjust the amount of enhancement.

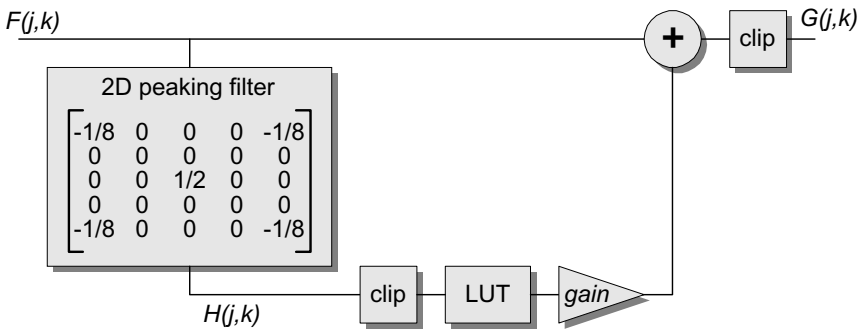


Figure 3.16: *Block diagram of an alternative sharpness enhancement algorithm.*

- *2-D Peaking filter* - The filter consists of a 2-D symmetric FIR filter with five non-zero valued coefficients. Five RD and one LSL command are used to read the pixel values and to weight the center pixel with coefficient values 4. Subsequently, three ADD and one subtraction (SUB) are necessary to compute the filter output value, followed by a LSR to normalize the DC component of the signal. Using two CMP operations, the value is clipped between -128 and 127. In total the 2-D filter uses 13 OPs/sample.

- *LUT* - The look-up function only requires one ADD and one RD operation to add the value 128 to the filter output and to read the table value of this table entry.

- *Gain* - The value from the LUT is multiplied with a gain value and subsequently normalized by truncating the six least-significant bits. Hence, one MUL, two RD and one LSR is required.

- *Add* - Finally, the result is added to the original signal and clipped, requiring one ADD, two CMP, and one write.

The summations of the individual contributions results in a total operation budget of 23 operations per sample. Comparing this with 285 operations per sample for the sharpness enhancement algorithm outlined in Subsection 3.2, a difference of more than factor 12 in computation complexity is shown. Both algorithms perform the function of sharpness enhancement, but they are implemented on a different hardware architecture and they have different picture quality performance. Although complexity and performance analysis can be very helpful, the output is only useful when considering all conditions.

3.4.3 Sampling-rate conversion complexity estimation

Similarly, the advanced sampling-rate converter from Section 3.3 is designed as a processor with dedicated hardware in a multiprocessor system. For this algorithm, the same analysis is done as for the sharpness enhancement algorithm. The complexity estimation result is depicted in Table 3.2. Unlike

Table 3.2: *Operations per sample for advanced sampling-rate conversion.*

Task	Operations per samples												Total/ sample
	RD	STR	ADD	SUB	AND	OR	CMP	LSR	MUL	DIV	ABS	MOD	
Filtering	7	1	7				2	1	6				24
Phase computation		1	1										2
Total													**26**

the sharpness enhancement algorithm, the sampling-rate conversion is applied to all color components instead of only the luminance. As a result, the amount of operations per pixel is 52, when the sampling format 4:2:2 is assumed. When SD video is considered with a 25-Hz frame rate and a resolution of 832 × 576, the total operation bandwidth equals 623 MOPS.

With respect to horizontal processing, no significant amount of memory is used, only 6 pixels to feed the filter taps. However, for vertical processing, the filter taps have to be fed with samples originating from different video lines. Therefore, 5 line memories are required for a sampling-rate converter with a 6-tap filter. For both luminance and chrominance this requires memory of approximately 8 kByte. The total bandwidth requirements for writing (1×) and reading (6×) of this local embedded memory is 224 MByte/s.
If high-quality vertical video scaling of interlaced video is required, some additional signal processing is required for de-interlacing of the input signal prior to the vertical sampling-rate conversion. For this processing, the data of two adjacent fields is required. Figure 3.17 portrays how the fields are used to create the progressive video frame. Note that only those video lines have to be de-interlaced that are actually used for the filter taps. The bandwidth at the input of the sampling-rate converter B_i is dependent on the scaling factor S and the output bandwidth B_o according to:

$$B_i = 2\frac{1}{S} \cdot B_o, \tag{3.26}$$

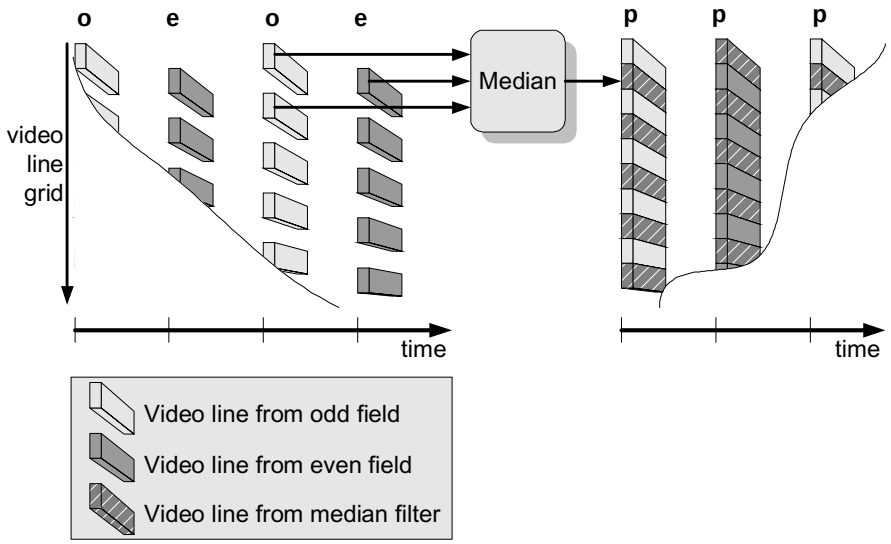


Figure 3.17: *De-interlacing by means of median filtering.*

where $S > 1$ if expansion is applied and $S < 1$ when compression is applied. The factor two represents the bandwidth cost of the de-interlacing. To create one field, two input fields are required. Consequently, when $S = 1$ and the output video has a SD video resolution, the input bandwidth $B_i = 64$ MByte/s. To be able to read two video fields, one field has to be stored in memory, requiring a memory space of 3.8 Mbit. Moreover, from Figure 3.17 it can be noticed that one additional line memory is required for applying the median filtering, bringing the total to 6 line memories (10 kByte).

3.4.4 Temporal noise reduction

Before calculating the complexity metrics for a temporal noise reduction algorithm, first the functionality is briefly explained. Figure 3.18 shows that only the low-pass frequencies are noise-reduced by means of a 2-D filter. The high frequencies, which remain after subtracting the low-pass signal from the original signal, are conveyed to the output and are subsequently added to the noise-reduced result. Temporal filtering is applied by means of a weighted average of the filtered low-pass output signal and the new input signal. The weighting factors are adapted depending on the amount of motion in the video signal. When little or no motion is present, the mixing factor k will be small, thereby yielding only little contribution from the input signal. When fast motion occurs, the input signal has the

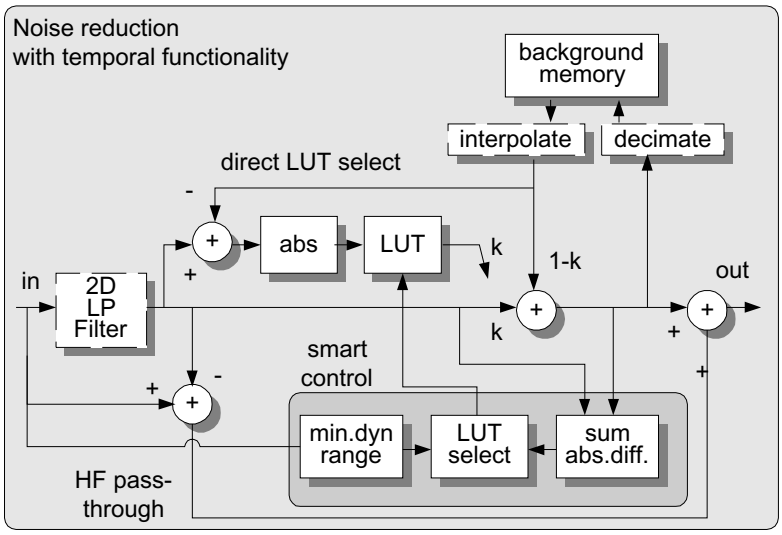


Figure 3.18: *Block diagram of a temporal noise reduction function.*

main contribution to the output signal, in order to prevent motion blur. Summarizing, the fundamental filter algorithm comprises an adaptive IIR filter. The motion is determined by the difference between the previous and the current picture. A Look-Up Table (LUT) translates the measured difference into a k-factor. Additionally, to prevent a decrease in overall picture quality, this algorithm contains functionality to adapt the amount of noise reduction to the noise level. Two different measurements are applied to determine this noise level. First, a dynamic-range measurement is used, similar to the sharpness enhancement algorithm (see Section 3.2.4). The number of times that the dynamic range does not exceed a certain threshold within one field or frame, is inversely proportional to the amount of noise. Secondly, the sum of absolute difference between the low-pass filtered signal before and after noise reduction, i.e. the input and output signal of the $(k, 1-k)-$mixer.

In the following table (Table 3.3) the number of operations per sample is shown. Note that the block diagram in Figure 3.18 indicates a decimation step before writing into the frame buffer and an interpolation step after reading from the memory. Because only the low-frequency part of the signal is stored, down-scaling of the picture can be applied without loosing information, thereby decreasing the buffer requirements. However, to indicate the complexity of the noise reduction only, this functionality is not included in the calculations. For a standard-definition video signal with a

Table 3.3: *Operations per sample for temporal noise reduction.*

Task	Operations per samples											Total/ sample
	RD	STR	ADD	SUB	AND	OR	CMP	LSR	MUL	DIV	ABS	MOD	
2-D LPF	9	1	8					1	5				24
Direct LUT select	3	1	1								1		6
HF Path	4	2	2										8
Temp. filter with k	3	1	2						2				8
Smart control	11	6	1	1			6				2		27
Total													**73**

25-Hz frame rate and a resolution of 832×576, the total operation bandwidth equals 1 GOPS. Due to the processing in both spatial dimensions, a dual-line memory is necessary and requires 3.3 kByte of memory. The bandwidth for writing into and reading from this memory should be sufficient for one video stream (2×32 MByte/s). Since a large buffer memory is also required for temporal IIR filtering, a memory capacity of 3.8 MBit is necessary, e.g. mapped onto an off-chip background memory. This buffer also requires a throughput rate of 64 MByte/s.

3.4.5 Motion-Compensated frame-rate conversion

For the sharpness enhancement algorithm, the computational requirements are most noticeable. For other algorithms, the amount of local memory is the mostly required resource. Motion-Compensated (MC) frame-rate conversion fits to a third category of algorithms for which the *bandwidth* to off-chip memory is the most demanding requirement. Figure 3.19 shows the functional block diagram of motion-compensated frame-rate conversion. This function contains several modes to handle different video formats:

- Conversion of 50-Hz to 100-Hz interlaced video,
- Conversion of 60-Hz interlaced to 60-Hz progressive video,
- Conversion of 24-Hz film material (received as a 50-Hz interlaced video applying a 2-2 pull-down scheme), to 50 or 100-Hz interlaced video. Motion judder is removed resulting in fluent motion portrayal.

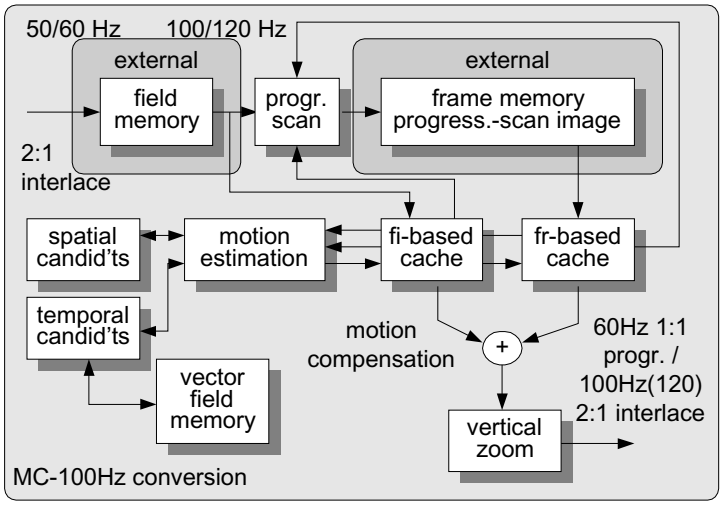


Figure 3.19: *Block diagram of the motion compensated frame-rate conversion function.*

- Conversion of 24-Hz film material (received as a 60-Hz interlaced video applying a 2-3 pull-down scheme), to 60-Hz interlaced or progressive video. Motion judder is removed resulting in fluent motion portrayal.

The algorithm uses an 8×8 recursive block matching algorithm [59] with a limited amount of candidate vectors. These candidate vectors point to predictions in the spatial and temporal surrounding of the block to be estimated.

Most operations are performed only once per block, such as retrieving candidate motion vectors, bookkeeping of the matching result in the motion estimator (ME), etc. The operations at pixel rate are limited to the Sum of Absolute Difference (SAD) calculations in the ME, median filtering for interlace-to-progressive conversion and composition of the motion compensated result. For the luminance part of the video signal, an estimate of 527 MOPS is required. Because for the chrominance part only temporal filtering is applied, 100 MOPS are required for this part of the signal. The total of 627 MOPS is significantly less than is required for the peaking algorithm.

The input field memory is necessary to double the field rate by means of field repetition. For the luminance this consumes 16 and 32 MByte/s for writing in and reading from the memory, respectively. After motion-

compensated de-interlacing by the proscan block in Figure 3.19, the frame picture is written in the memory requiring 64 MByte/s memory bandwidth. After one frame delay, this progressive signal is read again for the recursive motion estimation and de-interlacing of the subsequent frame picture. Also this read action occupies a memory bandwidth of 64 MByte/s. In total, the processing for luminance requires 176 MByte/s. For the chrominance, no motion-compensation video data is used and the frame-rate conversion is kept simple by applying picture repetition. This requires a memory bandwidth of 80 MByte/s, resulting in a total bandwidth of 256 MByte/s. Comparing this with the aforementioned functions such as sharpness enhancement, sampling-rate conversion, and noise reduction, this function consumes a considerable amount of the total memory bandwidth. Also in terms of memory capacity, this function is quite expensive, requiring five field memories.

3.5 Conclusion

The design of future multimedia systems in consumer electronics is a challenging task due to conflicting requirements such as picture quality performance, sufficient flexibility, sufficient computational performance, and low system costs. Analyzing the complexity of independent functions is necessary to make the proper design tradeoffs. For example, consumer-electronics systems require a strong focus on limited system costs, thereby trading off different parameters in the design-space. In this respect, the two algorithms for sharpness enhancement as analyzed in Section 3.4.2, target another point in the design space. The presented analysis shows a difference in computational complexity of more than a factor 12. For the computational's expensive algorithm, picture quality is most important, while the implementation only requires the programmability to adapt the processing to the amount of noise, the sharpness setting, the picture resolution, etc. Hence, it results in a dedicated hardware solution with programmable registers for the adaptations. The other sharpness-enhancement algorithm is intended for some minor sharpness recovery after video processing such as noise reduction, and is only applied in some special use cases of the system. Consequently, a simple software implementation is desirable. Clearly, this results in different algorithm requirements. It can be concluded that an algorithm can only be defined when the constraints of the design space are known. For some special cases this leads to extreme freedoms in the design. For example, if functional feasibility has to be proven within an academic environment, algorithms can be developed without constraints on the computational complexity, the flexibility, the costs, etc. However, for consumer

electronics, the algorithm development is heavily constrained, i.e. architecture dependent. On the other hand, previous chapters have shown that the design of an architecture is based on the analysis of the functionality. This chicken-and-egg problem postulates the codesign of both algorithms and architecture.

Besides computational resources, also communication resources such as bandwidth requirements for the communication infrastructure and the necessary memory space are important metrics for the system codesign. For example, for motion-compensated frame-rate conversion a relatively small amount of operations per sample is used, provided that elegant algorithms are adopted. However, the necessary amount of external memory bandwidth to perform motion estimation, de-interlacing, and motion compensated frame-rate conversion is significant.

Without further specifying the necessary amount of flexibility and the targeted price point, it can already be concluded that implementation of the complete video processing functionality in a television system cannot be achieved on a general-purpose CPU. Applying advanced sharpness enhancement, temporal noise reduction, advanced sampling-rate conversion, and motion-compensated frame-rate conversion for one main video stream requires about 6 GOPS. However, an important remark here should be made. As was mentioned in the introduction of this chapter, the design complexity starts to have a major impact on the total system costs, and hence only large production volumes can lead to an acceptable price/performance ratio. However, production volume is determined by the market share or at least by the market volume. Consequently, to increase the production volume, systems should become more generic, to increase the product portfolio for which they can be used. Another motivation that may influence the above-stated conclusion in the future is the ratio between the silicon area of computational units and embedded memory on a SoC. Because the relative silicon area of embedded memory in a system grows, the contribution of the computational units in costs decreases. As a result, more general-purpose computing may be adopted in the SoC to make the system more flexible and more transparent for the application programmer, at only limited additional costs. Notice that this increasing flexibility will lead to an undesired growth of the power consumption. Hence, the amount of additional flexibility should still be balanced carefully.

Starting from the requirement for more flexibility and at the same time sufficient performance at low cost, the following chapter discusses a tele-

vision processor design for high-end TV. It reveals proper tradeoffs and outlines both the functionality and the system architecture. Obviously, the presented system is just one solution from a large design space. Therefore, the Chapter 5, 6, and 7 elaborate on some design issues in general. However, in all parts of this thesis, tradeoffs between expenses and benefits are central.

Quod erat demonstrandum
(Greek mathematician Euclid, c.300 BC)
which was to be proved

Chapter 4

Flexible television processor system

THE VLSI complexity and the corresponding software for implementing multimedia architectures, including advanced video processing, require an increasingly large design effort. Chips for consumer-electronic products are currently being designed with 30 Million gates, having a computing power exceeding 10 GOPS (Giga Operations Per Second) [4]. Furthermore, the associated firmware after compilation already exceeds one MByte for TVs. The architecture of conventional TV sets has been introduced more than half a century ago. Since that time, the functionality that was offered to the customer has been increased significantly, without fundamentally reconsidering the architecture of the system. Feature extension boxes have been added to upgrade the conventional system to high-end TV sets currently available in the consumer market. This chapter presents a recently developed TV system called the TeleVision Processor (TVP), satisfying the need for flexible, yet powerful video computing at low cost.

4.1 Preliminary statements and requirements

The previous chapter has shown examples of signal-processing functions and the architecture of the conventional high-end TV architecture was discussed earlier. When standardization of the electronic television was carried out [3], the architecture already showed the traditional standard video

path. It consisted of the demodulator, decoder and the picture control block with a separate audio path. Since then, an additional Teletext path was introduced, 50-to-100 Hz conversion (Europe), PALplus decoding (or EDTV2 in Japan), an additional Picture-in-Picture (PiP), dual-screen, new sound standards and more recently, digital transmission standards based on MPEG, have evolved. An evaluations of the conventional solution leads to the following points (see also the previous chapters).

- From a bird's eye view, the aforementioned new features can be characterized as an extension of the conventional implementation, resulting in a suboptimal structure with respect to memory and flexibility.
- Since each function uses its own memory device and has a dedicated communication path, memories cannot be reused for alternative purposes if a memory-rich function is switched off.
- Furthermore, the solution is rigid and not flexible for adding new features, because the interfaces between the functional blocks are dedicated and specified as required for the actual part of the task graph. Optimization and fitting of a new feature inside the processing chain is difficult –if not impossible– because the order of signal-processing functions is determined by the existing chip sets.

It goes without saying that these restrictions are conflicting with modern trends in TV applications. In Chapter 1, requirements for the design of future TV systems were discussed, taking long-term trends into account. We briefly recapitulate those requirements without discussing them.

Increased diversity in products – The TV is becoming very diverse in terms of features, requiring a programmable or reconfigurable architecture. New products such as TV-telephone, TV-VCR, etc. have emerged.
More interfaces – The TV will have communication links to a network and others (e.g. mobile) devices.
Absorption of PC technology – RISC cores, peripheral devices, SW architectures, and Operating Systems are gradually adopted and accepted as standard technology.
Unified memory architecture – Since memory devices continuously become larger and cheaper, it is desirable to have one uniform shared background memory.
Flexibility for new standards – New features and standards from international bodies have to be realized in a short time frame and require an extensible and scalable system.
Low system costs – New TV systems are introduced in the highly competitive consumer electronics market.

Some attempts in the past were made to deal with the fast changing requirements. A detailed overview of several architectures has been presented in Chapter 2. We will only briefly summarize a few results from literature here. Fully-programmable multimedia solutions like the Multimedia Video Processor (MVP) [32] can be very powerful, but are rather expensive for consumer products, because they are more general than the TV application domain desires. Moreover, they dissipate more power and occupy a larger silicon area. A fully-programmable solution that is more tuned toward the application domain is the Video Signal Processor (VSP). It contains a large amount of ILP but is difficult to program if the application is more event-driven or contains many data dependencies. Furthermore, the large amount of parallelism and the single thread of control introduce a complex scheduling problem that can only be solved with tools and laborious interaction of the programmer. A recent trend is to use Very Long Instruction Word (VLIW) processors to boost processing power [26] [34]. However, this does not offer sufficient parallelism to cover all processing requirements. Simply increasing the number of functional units would increase the present scheduling problem for such solutions.

More powerful and yet cost-effective implementations are possible by adapting the system more closely to the target application domain. As a result, the processing performance and efficiency can be increased by making use of the task-level parallelism that is expressed in a functional decomposition of the application. The grain of the operations within a functional unit is increased to the level of *complete TV tasks or functions* like a video scaler, noise reduction, sharpness enhancement, field rate conversion, etc. It will become clear at the end that this approach maintains low system costs and flexibility and programmability is obtained with moderate extra costs. The task-oriented approach will be developed in the coarse of this chapter. The next section elaborates on the consequences of the required computational effort and memory bandwidth that were discussed in the previous chapter.

4.2 Consequences from the requirements

4.2.1 Computational aspects

The previous chapter concluded with an overview of computational requirements of various TV functions. Let us now analyze why the aforementioned processor systems from literature were not directly adopted in current TV systems, although most requirements would be fulfilled if the required applications could be implemented on a fully programmable multi-

media processor. One of the questions to be answered is how much computing power is involved and should be realized. The previous chapter gives a detailed examination of state-of-the-art TV image enhancement techniques and presents fairly realistic estimations of required computing power and memory expenses. In this section, we summarize the results in Table 4.1.

Table 4.1: *Computational costs and memory of various TV functions.*

Function	**Operations per Second**	**Bandwidth MByte/sec.**	**Memory/ Cache**
H up/down-scaling	400 MOPS	38–64	samples
V up/down-scaling	400 MOPS	38–96	lines
Filters, Comb filters	200 MOPS	64–128	samples-field
Advanced peaking	650 MOPS	32	lines
Color transient improvement	300 MOPS	48	samples
Dynamic noise reduction	500 MOPS	64–128	field
MC-100 Hz	2–4 GOPS	192-256	2–3 fields
Color space	150 MOPS	80	None
Teletext conversion	10 MOPS	48	> field
Adaptive luminance	60 MOPS	32	1 KByte

Table 4.1 shows the intrinsic processing power of several TV functions at standard-definition (SD) resolution. With respect to operation counting, additions, multiplications, sample read and writes, etc., are considered as single (DSP) operations (see the previous chapter). The fourth column shows the amount of memory or cache required. Here it is assumed that information can be retrieved or stored by single reads and writes (which is optimistic). For a normal TV application with 50-100 Hz conversion, Picture-in-Picture (PiP), noise reduction and aspect-ratio conversion, the amount of operations already exceeds 6 GOPS (Giga operations per second). This is not readily implemented on a general-purpose processor cost-effectively. Consequently, the use of application-specific coprocessors to increase parallelism and local computing power, in combination with general-purpose processing is unavoidable to keep system costs low.

4.2.2 Off-chip memory considerations

Since memory functions determine a significant part of the system costs, the access and the amount of external memory are important optimization parameters in the system design.

Memory aspects

Due to the trend towards more flexible and re-usable computing and the success of the PC, the distributed off-chip memory as shown in the conventional TV architecture is not acceptable for future products. Although the memory requirements of the desired system are optimized, typical memory sizes would still be 4–16 MByte with a bandwidth in the order of 350 MByte/s. For such memory devices, the synchronous DRAM (SDRAM) that are used in PCs and related products represents the mainstream market, offering the lowest price per Byte. Because the competition in the PC domain is driven by the highest computational performance, the most advanced SDRAM memories at this moment in time, such as double-data-rate SDRAM (DDR SDRAM) and Direct Rambus (Direct RDRAM) devices have been are adopted. For an efficient data transfer, all these devices should be accessed with bursts of data instead of single data words, due to the overhead in the addressing. Since multimedia processing often accesses the data with a regular pattern (e.g. line sequentially), multimedia memory caching can be used to exploit this so-called *locality of data*. Using the low-density static RAM (SRAM) for caching, a high peak bandwidth with sufficiently low latency can be provided.

Communication aspects

To avoid excessive access to off-chip memory, the processors have small local cache memories, so that the background memory is not required for small memory functions. For example, a vertical-filter coprocessor may have line memories locally, thereby requiring an input and output interface that stream the video data only once in a line-sequential scanning order. However, there are exceptions to the aforementioned rule. The caching of field memories for e.g. MC-100Hz conversion, dynamic noise reduction or comb filtering is too expensive for on-chip integration as embedded memory (1 field = 4 Mbit). Instead, these memory functions are integrated in the large unified memory in the background. As a consequence, the consumption of scarce bandwidth to this memory should be monitored carefully.

Another aspect of the communication architecture is the capability of direct inter-processor communication without necessary access to external memory. This approach limits the memory access to functional usage. As a general conclusion, we state that only (sub)systems using large data structures (e.g. video field/frames), access the background memory. In the alternative situation where subsystems communicate only large data packets via the off-chip memory, high-throughput communication between

multiple processors is required and the architecture needs extremely fast communication networks with a high bandwidth and a considerable amount of additional memory. Examples of such systems are given in [26] and [27].

4.3 Analysis of TV applications

In this section, the TV applications are further analyzed with respect to the timing requirements in order to motivate the design of a new chip architecture called the TeleVision Processor (TVP). Prior to discussing the details, the concepts of graphs and tasks are introduced, as they will be applied extensively in the system analysis.

A *task* is a basic TV function, thus a set of closely coupled calculations, which are executed on a stream of data samples (a video signal). The set of tasks is well defined for the high-end TV system. Tasks are *weakly* programmable, that is, they can be set to different video quality levels with different resource requirements, but the nature of the function itself cannot be modified. For example, a filter can be programmed to a smaller length resulting in a lower quality, thereby reducing local memory usage. This memory can be reused for other filter tasks. A *graph* consists of a set of tasks which should be executed in parallel. The graph represents an application in the TV domain, e.g. a main video signal with a Picture-in-Picture (PiP) in the upper corner.

The first step in coming to a new architecture is to classify the different tasks into three groups.

- *Event-driven* tasks (EDT). Examples are user interactions, such as channel switching and user menu generation, a modem function, etc. Control tasks are related to the interaction of the system with the environment.
- *Soft real-time* tasks (SRT). This group contains tasks having a "soft" real-time deadline, meaning that a deadline can be missed and is not critical for the system performance. An example is the decoding of a Teletext page.
- *Hard real-time* tasks (HRT). These tasks should not miss their deadline, since this would directly lead to system malfunctioning. Typical hard real-time tasks are most video processing operations, since output pictures have to be displayed at predetermined time intervals (field rate). Examples are horizontal and vertical sample-rate conversion, sharpness enhancement, noise reduction, etc.

Hard and soft real-time tasks can be represented in a so-called *task graph* as shown in Figure 4.1, which portrays a multi-window application. First, an input video stream is conveyed through a noise-reduction task. Subsequently, the result is scaled to the resolution of the display. In parallel, a second video stream is processed by a second noise-reduction task. The output of this task is scaled down to a PiP resolution before it mixed with first video stream. As will be explained later, this mixing is achieved in the background memory. Because the input video streams are not necessarily synchronous, the memory for mixing is also exploited to synchronize the input video streams. Notice that only one video stream can be synchronized with the display. Consequently, the writing and reading of the other video stream into the mixing memory is performed asynchronously. After mixing of the video streams, field-rate conversion is applied to construct a video signal with a 100 Hz field rate. Subsequently, sharpness enhancement and conversion of YUV to RGB color components is applied. All these tasks belong to the HRT classification. A third parallel video signal is generated by the Teletext decoder which generates Teletext pages in a shared memory. Because the timing of this task is not critical for the system performance it is classified as a SRT task. However, the reading of the Teletext pages from the memory and the mixing with the other two video signals is again performed under real-time constraints (HRT). As shown in the figure, the three different video windows are displayed on the TV screen. The user can control the application by, e.g. switching the PiP on and off, changing the size of the Teletext page, or the positions of the video windows. These user interactions result in modifications of the task graph or changing of the parameter settings of the tasks, and are classified as control or event-driven tasks (EDT).

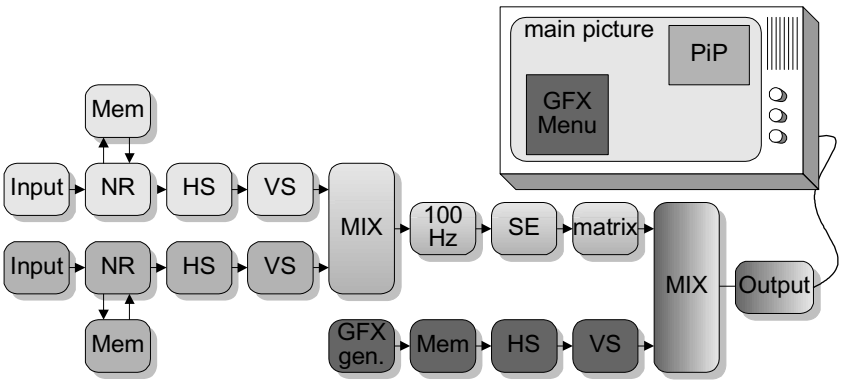


Figure 4.1: *An example of a task graph.*

The hard and soft real-time tasks, requiring a large computational effort, are mapped onto more dedicated processor hardware to increase the amount of parallelism. This approach was adopted to come to a cost-effective system design, enabling a large set of 50 to 100 applications with a single chip. In [13], it was discussed that a fully programmable architecture for a broad range of TV applications is far too expensive for consumer television. Although the aforementioned application-specific (co-)processors are weakly programmable and are able to process various tasks that are slightly different, the type of task is fixed, e.g. a sampling-rate converter can perform aspect-ratio conversion, scaling, geometric corrections or data compression, but cannot do noise reduction or sharpness enhancement. The system requires an architecture which provides multi-tasking capabilities, because a plurality of different tasks has to be processed simultaneously by the same coprocessor.

Let us now summarize the results so far of the analysis of the set of TV functions and the computational estimations from the previous section and find implications for the architectural design.

- The required computing power is in the order of 10 GOPS (see Table 4.1).

- If all the communication between tasks in a typical task graph is accumulated, several GByte/sec of required bandwidth is obtained. Each full-color video channel requires 32 MByte/s bandwidth (at 50-Hz field rate, see Table 4.1).

- The constraints for hard real-time streams must be met under all conditions. This requires worst-case design of the architecture for this part of the system.

- One of the disadvantages of current rigid TV architectures is that signal-processing functions cannot be used in different orders, i.e. modifying the order of tasks in a task graph (see Section 4.1). Such task reordering would enable an optimal TV quality setting under all conditions within the limits of available resources. The new architecture should offer this flexibility, i.e. the feature to construct different task graphs with the same set of TV tasks. Analysis has shown that many different graphs exist for common high-end TV applications, such as PiP, PiP recording, PiP replay, Split screen, with or without 50-to-100 Hz conversion, with or without noise reduction for each input stream, with or without display aspect-ratio conversion, etc.

- The resulting quality of the performed video tasks should be scalable to adapt to the amount of available hardware resources (computing power and bandwidth). The quality depends therefore on the chosen task graph. For example, if only one video stream is being processed, all video functions can be switched to the highest quality level, requiring e.g. the best filters or most advanced processing (computationally intensive for each task). If two video streams have to be processed, the same video task can appear twice in the task graph, so that each TV task is executed at medium quality. This might be acceptable, e.g. a PiP application does not require the same video quality as an application with one main video plane. More precisely, the video streams are processed at the clock rate (64 MHz), but can have four different pixel rates: 16, 32, 48 and 64 MHz. Each coprocessor can process a number of streams in parallel, constraint by the following expression:

$$\frac{1n_{16} + 2n_{32} + 3n_{48} + 4n_{64}}{4} \leq 1, \tag{4.1}$$

  where n_{16} represents the number of streams with 16-MHz pixel rate, n_{32} stands for the number of streams with 32-MHz pixel rate, and so on.

- The system should be capable of processing a minimum of *two* real-time SD input video streams, because our aim is a multi-window TV system. The clocks of the streams are not related (asynchronous), as they originate from different sources.

Up to this point, primarily signal processing and its requirements were discussed primarily. Let us now concentrate on the memory which is associated with the signal processing. As was discussed in the previous section, the objective is to use one single unified memory. The most important motivation for this is twofold. Firstly, a unified memory supports *memory reuse*. If functions are switched off, memory can be assigned to other tasks. For example, PALplus decoding memories can be made available for e.g. Teletext pages or vertical scaling. The second argument is *cost*. Clearly, off-the-shelf large-scale memory components give the lowest system costs.

An experimental system implementation is based on a memory bank of 96-MHz Synchronous DRAMs. For 32-bit words, this gives a theoretical bandwidth up to 384 MByte/s, which is lower than the maximum requirements mentioned in the previous section. However, not all data *communication* necessarily occurs via the SDRAMs and can be kept on chip. Only

the tasks that inherently require complete field memories, such as temporal noise reduction, access the external memory. Another example that requires a video frame memory is the *synchronization* of two independent video streams as already explained (the mix task in Figure 4.1). Thus, the two input video streams and the output stream may be independent, i.e. the rate in which the streams are written and read may be asynchronous.

Since memory bandwidth is a scarce parameter, external memory accesses are indicated in the task graph: they are modelled as extra inputs and outputs to the graph. Furthermore, the decoupling of tasks by a frame memory results in a decomposition of the task graphs into independent *subgraphs* as shown in Figure 4.2. The figure represents a decomposition of the tasks graph in Figure 4.1 into four subgraphs. The top subgraph generates the main picture in the background of the TV screen. The second subgraph produces the PiP. Due to the decoupling of these two asynchronous video signals by the mixing function, they can be considered as independent task graphs, i.e. subgraphs. The third subgraph performs the scaling of the Teletext pages and is decoupled by a frame buffer at the input and a mixing buffer at the output. Finally, the bottom graph outputs the resulting picture for display.

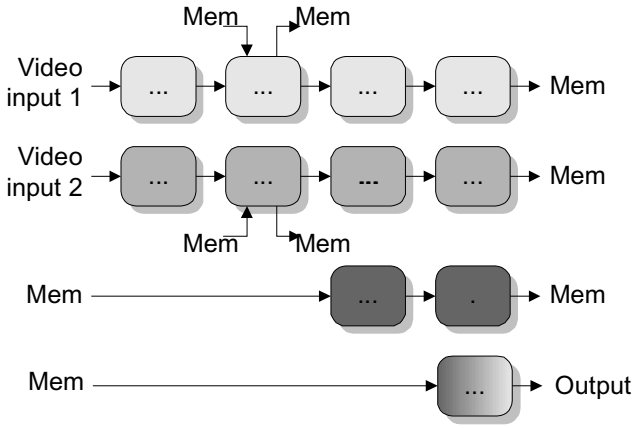


Figure 4.2: *Example subgraphs, corresponding to Figure 4.1.*

4.4 Architecture design

The application analysis of Section 4.3 and the computational requirements of Section 4.2 are starting points for designing the architecture. We have

discussed sets of tasks being combined into graphs, and the associated memory communication. Consequently, the architecture contains principal modules and their components, and as such, it has a hierarchical structure. Let us therefore start at the top of the hierarchy.

4.4.1 Top-level architecture

Since the properties of control and the signal processing are totally different, we will define three different subsystems: a control subsystem, a signal-processing subsystem and a memory subsystem. The control subsystem, comprising a micro controller with some peripherals, is responsible for the execution of all control-oriented tasks and some of the soft real-time tasks. This subsystem is also referred to as the Telecommunication and Control Processor (TVP). The signal-processing subsystem, called CoProcessor Array (CPA) executes all hard real-time tasks and some of the soft real-time tasks. The memory subsystem performs the memory functions for both the system control and signal processing. The nature of these types of processing is totally different. The control subsystem is based on handling *events*. It has to deal with unpredictable events coming from the environment. Therefore, its memory requests have a *random* characteristic. The signal-processing subsystem is characterized by the more regular periodicity of the operations. It sends requests with a more regular *periodic* to the memory subsystem. The top-level architecture is visualized in Figure 4.3.

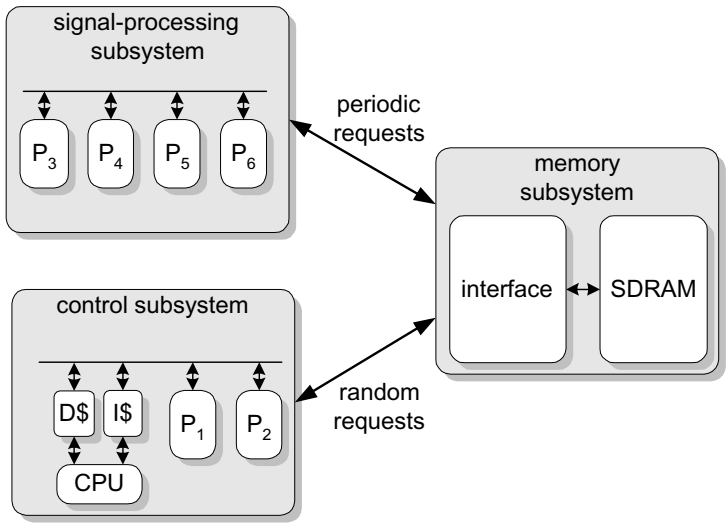


Figure 4.3: *The top-level architecture with separated subsystems.*

The principal design step at the top-level concentrates on the arbitration between the two types of requests. The difficulty is that different aspects have to be optimized. In the case of random requests by the micro controller, the *latency* should be minimized, due to the event-driven character of the tasks. In the case of periodic requests, the *throughput* of the system has to be guaranteed, because all hard real-time tasks are mapped onto the signal-processing subsystem which is periodic. Thus, in the case of periodic requests we want to design for the throughput that should be obtained. The latency is less relevant, since it can be hidden if sufficient buffering is provided. Summarizing, an arbitration scheme is needed which ensures the throughput for the periodic requests, while minimizing the latency for the random requests. For our system, we used an arbitration scheme from [61], that provides a low latency and high-priority access for the micro controller and guarantees the required bandwidth for the video signal processors.

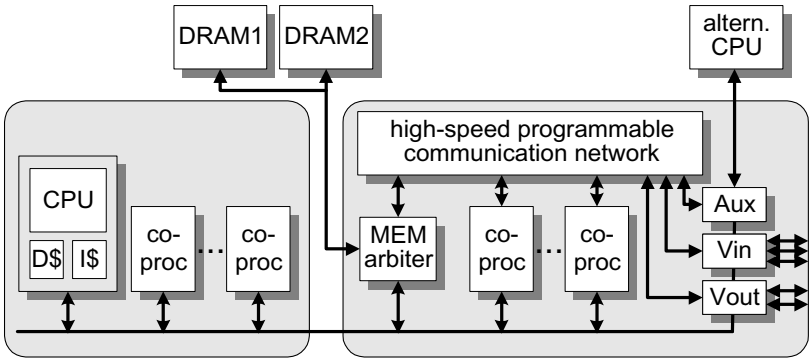


Figure 4.4: *More detailed view of architecture proposal.*

A further detailed view on the architecture is portrayed by Figure 4.4. The above-mentioned subsystems are connected via a bus and a high-speed communication network. The signal-processing subsystem consists of a number of programmable coprocessors, interconnected via the high-speed communication network. The background memory is accessible by both the coprocessors and the micro controller. A few important system aspects of the system are highlighted below.

- The coprocessors are weakly programmable and perform typical video tasks for TV systems. Since all coprocessors concentrate on a different type of task, they are much smaller than general-purpose CPU, or equivalently, they are an order of magnitude more powerful in computing power. The drawback is the restricted use of each coprocessor.

- The general-purpose CPU is used for soft real-time tasks and for event-oriented control. When the CPU is sufficiently powerful, a number of communication functions (e.g. modem) and Internet applications can be covered, while the coprocessors are executing video simultaneously.

- Since it is not known in advance what TV application is chosen, the connection network should be programmable, so that an arbitrary set and order of tasks can be chosen.

- The system should be *extensible* for future coprocessors and/or the addition of a media processor, as discussed in a previous chapter. For this reason, a generic interface is added for connecting an external processor, either weakly programmable or general-purpose. It should be possible to include the external processor as an extension of the internal set of available processors.

- Video input can be received and processed without first writing it into the unified memory. Similarly, processed video can be output directly via the Video Output unit without memory access. This approach saves the scarce memory bandwidth to the memory subsystem.

In the subsequent section, the signal-processing subsystem will be addressed and details for achieving autonomous processing without control by the micro controller, will be disclosed.

4.5 Signal-processing subsystem

4.5.1 Tasks and task graphs

The tasks to be performed for a complete task graph in a TV set can be classified globally into two groups: tasks requiring a similar type of signal processing and tasks which are different from a functionality point of view. Examples of the first group are horizontal down-scaling for PiP and aspect-ratio conversion. The required type of signal processing is sample-rate conversion in both cases. It could be stated that sample-rate conversion is the *primitive* task for various typical TV functions. Examples of the second group where tasks are clearly different, are noise reduction and sharpness enhancement which are essentially different in most algorithmic aspects. Using the aforementioned classification, a number of choices can be made for the new architecture.

- After functional decomposition, tasks that inherently contain the same function are mapped on the same processor and are mapped on a separate processor otherwise.
- Since the tasks are known in advance at design time, each processor can be optimized for a particular job.
- To allow the mapping of different graphs, a reconfigurable communication network is added. This means that signal-processing functions can be used in various orders and can be programmed.

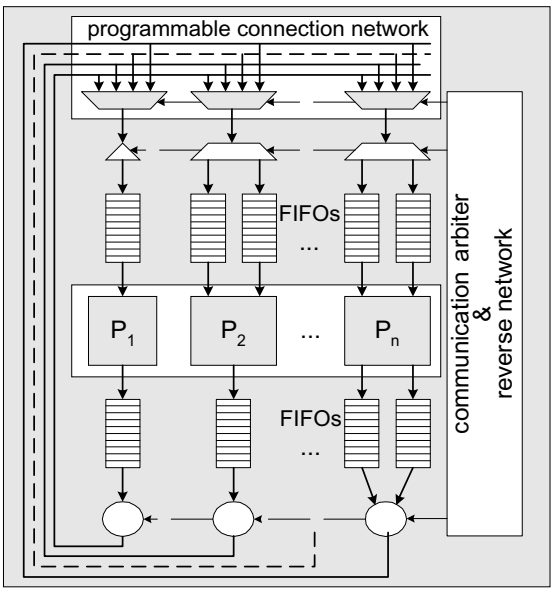


Figure 4.5: *Architecture of the signal-processing subsystem.*

The result of the architectural choices is that a heterogeneous multiprocessor architecture is obtained, which is capable of true task-level parallelism. The architecture is shown in Figure 4.5. This will now be discussed in more detail.

First, it can be noticed that all processors are surrounded by FIFO buffers at their inputs and outputs. The aim is to separate signal processing from communication, so that the activity of the processors is decoupled from the communication network. The processors can autonomously perform processing without requiring control from a host processor.

Secondly, the choice for FIFO buffers is motivated. They have been adopted because the arrows in the task graphs, called *edges*, represent streaming (video) signals, i.e. measurable physical quantities sampled at discrete points in time and binary encoded. The only identification of the different samples in the stream is given by the order of the samples. Samples are produced only once and cannot be lost on the communication channels. For streams with the aforementioned features, separation of communication and processing can be well performed with FIFOs.

Third, the architecture is based on a *data-flow model* as proposed by Kahn [62] called Kahn process network. Basically, this means that static scheduling is not required. A process or task in the task graph is activated at runtime when both input data to be processed and output space for storing the results are available. In the sequel this data-flow model is also referred to as data-driven processing. The arguments for adopting such Kahn process network model are as follows.

- When analyzing the tasks in a task graph, it can be found that some video signals cover the full display, while other signals cover a only a part of the screen, such as a PiP. When the size or the position of the PiP changes, the latencies and thus the schedules of the tasks change. Using a Kahn process network model, tasks become automatically active depending on the availability of input data and output buffer space without requiring a static schedule for each possible mode of operation.
- At some fixed moments in time, a typical video signal does not contain any pictorial data according to the CCIR 601 recommendation [63] (see Section 4.6.4). During these non-active parts of the video signal, the coprocessors are idle. Consequently, this so-called "blanking time" can be used for soft real-time (SRT) tasks. The detection whether a stream is in the blanking time or not is a runtime decision, since input streams are asynchronous with respect to each other. This reuse for SRT tasks is easily implemented with a data-flow model.
- A data-flow model is often simpler to implement in comparison with a static synchronous system, because it can perform runtime activation of processing task based on local availability of data, without a static schedule. Note that synchronous systems may potentially be less expensive since runtime computations are shifted to compile time. For example, no synchronization mechanisms is needed since synchronization is guaranteed by the schedule.

- The concept is better scalable with respect to the addition or removal of processors. It is expected that the data communication of the future functions will become increasingly dynamic, meaning that the amount of data bandwidth varies over time. A good example is a variable-length decoder, which produces and consumes a data-dependent amount of data.

Let us now discuss briefly how the signals will flow in the data-flow model. The communication behavior is such that the signal channels can be blocked locally (blocking semantic), according to the Kahn process network model. A local processor is stopped when the output FIFOs are full or the input FIFOs are empty. The decision is performed locally in the shells surrounding the processors. Furthermore, the output FIFOs of a sending task must be blocked when the input FIFOs of the receiving task are full. In order to be able to do this, the output FIFO has to know which input FIFO causes the output to be blocked. This requires backward reasoning about the communication. A special network provides control signals from each input FIFO of the receiving tasks back to the output FIFOs of the sending tasks, in order to block a output FIFO when its associated input FIFO is full. Hence, the special network provides the same connections as the data-communication network, but in the reverse direction and is therefore referred to as the *reverse* network (see Figure 4.5).

Up to this point, *resource sharing*, that is, the reuse of a processor for another task was only mentioned but not yet taken into account. For the model, a one-to-one correspondence between tasks in the graph and processors in the architecture has been assumed. In Subsection 4.5.2, resource sharing will be discussed in more detail.

4.5.2 Processor model

A typical TV function such as scaling, is based on sample-rate conversion. The same function (e.g. horizontal sampling-rate conversion, HSRC) can appear more than once in a task graph. For cost reasons, these conversions will all be executed on the same HSRC coprocessor. For the same reasons, resource sharing of the communication network can also be provided. Obviously, since the resources have limited speed, the sharing of the resources has also its limits. Let us elaborate on this by means of an example.

The processor model portrayed by Figure 4.6 has two input ports *I(p)* and two output *O(p)*ports. For example, this could be a temporal noise reduction coprocessor that needs an input video signal and the filtered result

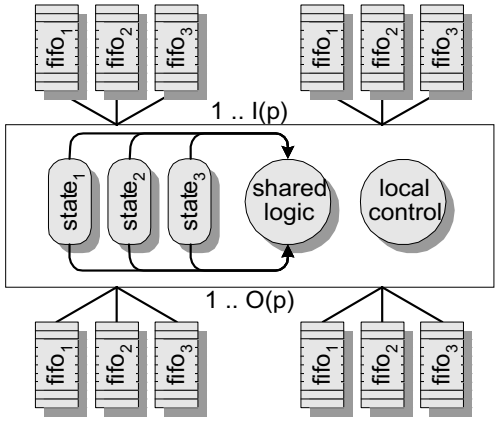


Figure 4.6: *The model of a signal processor.*

from the temporal loop as inputs, while the processor has a video output and a backward channel to memory as outputs. Each input and output port is connected to three FIFOs, labelled 1-3. When a task is started, a FIFO is assigned to this task at each input and output port. These assigned FIFOs only convey the data corresponding to this task. The maximum data bandwidth of each input and output port is limited (64 MByte/s) and is divided into four quanta of 16 MBbyte/s, which can be partitioned over the parallel tasks. For example, one task could process a 32 MByte/s video stream together with two other tasks that each process a video stream of 16 MByte/s. Thus, three tasks in parallel, communicating in total 64 MByte/s of data per input and output port.

Associated with each task, a local memory exists called *state space*, to store the state information of the task. Thus, according to the depicted model, three different state spaces can emerge, where each state space is assigned to a particular task. As a result, this model can handle maximally three tasks in parallel (the experimental chip discussed in [13] can perform one to three parallel tasks, depending on the considered coprocessor). In the figure, each task on a coprocessor occupies its own input and output FIFOs. To provide resource sharing of the communication network, these FIFOs could be shared as well. However this may cause *deadlock* situations. In general, a deadlock means that all tasks in the task graph cannot make any progress according to the data-flow model, while data is still available that needs to be processed. For the specific example where input and output FIFOs are shared and processing tasks that depend on each other are executed on the same processor, *deadlock* may occur.

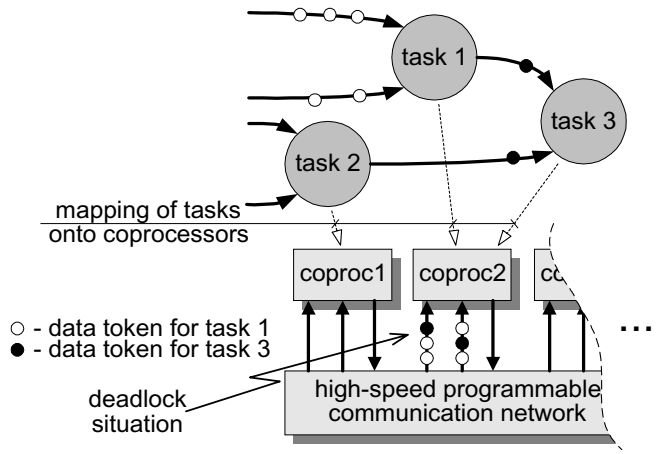


Figure 4.7: *Mapping of three tasks onto two coprocessors, causing a deadlock situation.*

This is schematically depicted in Figure 4.7. In the figure it is shown that each input channel of Coprocessor 2 receives the tokens of Task 1 and 3 in a different order. To avoid deadlock, the coprocessors should be able to reorder the tokens at the input channels. This is most straightforwardly implemented by separating the tokens of each task in separate FIFOs, thereby preventing the reordering problem.

4.5.3 Communication network

The primary task of the communication network is to provide sufficient bandwidth for the data streams between the output and the input FIFOs. A secondary task is to offer arbitrarily programmable connections between various coprocessors. Thus for every connection in the task graph, a path is created in the programmable connection network in Figure 4.5 for transport of the data. The path is created using circuit switching.

The network is a so-called *Time-Space-Time (TST) network* with space and time switches. The reason to build such a network is to ensure non-blocking connections between output FIFOs of processors and input FIFOs of succeeding processors, with a predetermined amount of bandwidth for each connection. Figure 4.8 shows a space switch at the left-hand side, making a connection between a_2 and b_3 and another connection between a_4 and b_2. At the right-hand side of Figure 4.8, the same connections are shown using a time (T-)switch. At the input, a multiplexer is added and

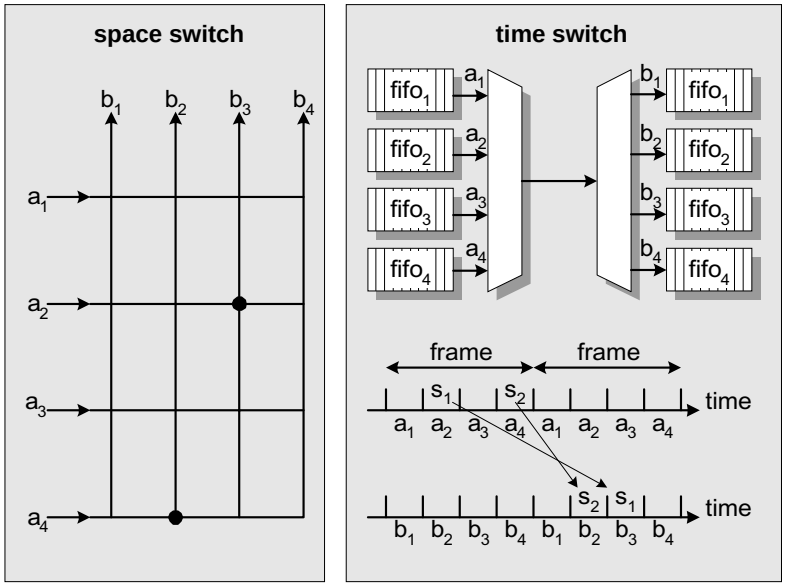


Figure 4.8: *A space switch (left) and a time switch (right).*

a demultiplexer at the output. Using four time slots, the total bandwidth equals the sum of the bandwidth contributions of the individual channels. This leads to the TST network as shown in Figure 4.9.

As an example, two paths through the network are indicated. Each path is programmed by putting the correct code for the three different parts of the network. A table with four different time slots is used for the time switches. For example, the x connection is programmed in phase one for the most left time switch via the correct code at the position labelled with an x. In this way, bandwidth is allocated corresponding to one phase. The connection labelled y has twice the bandwidth of one channel, because it is programmed for two time phases. The programming flexibility ensures that sufficient bandwidth can be provided for particular stages or signals in the task graph.

Note that at both the input and output side of the processors, FIFOs are implemented. With respect to the concept of the data-flow model, these FIFOs could be combined into a single buffer. However, to decouple the computation (pixel processing) from the communication, FIFOs are necessary at both the input and output of the coprocessors. Because the communicated data of multiple tasks on a processors is conveyed over the

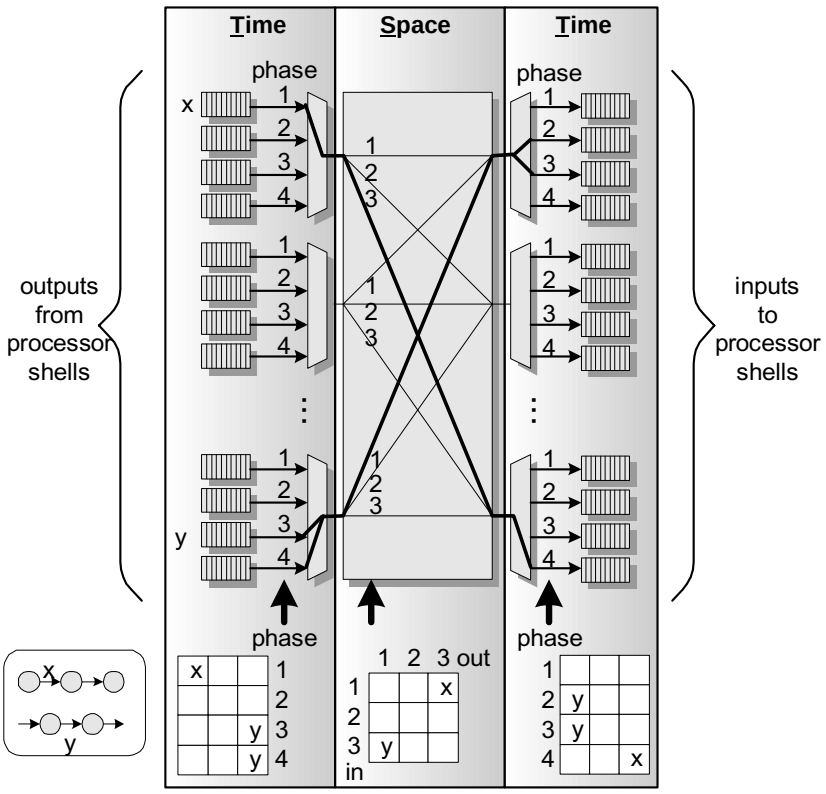


Figure 4.9: *A Time-Space-Time (TST) network, ensuring a guaranteed bandwidth for hard real-time tasks.*

switch matrix in a time multiplexed manner, the communication channels for a particular task are not available at all time phases. Hence, the FIFOs prevent that the coprocessors can only process a certain task if the corresponding network resources are available. Because this thesis only intends to discuss high-level architectural issues, it suffice to understand that the FIFOs can be rather small depending on the number of time phases. For example, the implemented system uses FIFOs with 32 entries.

Due to the static assignment of processing resources and the guaranteed communication throughput for all tasks, the behavior and performance of the system is completely predictable. Hence, it can be determined *a priori* whether a particular application can be executed with the available system resources. This predictability is a major advantage of the presented Television Processor system as opposed to systems that arbiter the resources

at run-time. For these later systems, the performance of a functional unit depends on requested shared resources, such as the off-chip memory and a communication bus, by other functional units. Moreover, these systems are less robust because the sum of the peak resource usage of the individual processing units may exceed the available resources, leading to unpredictable behavior of the system.

4.5.4 Interaction between controller and processor

The previous subsections discuss the architectural aspects of independent processors and how to provide these processors with sufficient data by means of a programmable communication network. In this subsection, we address briefly the adaptivity of the processing tasks by means of interaction with the controller.

A form of interactive communication is required when a TV function is made adaptive to the content of the video signal. For example, the noise-reduction processor measures a significant noise increase, communicates this property to the CPU, and is subsequently programmed by the CPU to perform more noise reduction. For this type of event-driven communication, an interrupt (irq) line to the CPU is used. The signal-processing subsystem contains a large range of interrupts to cover all possible events. Each interrupt is represented by one bit in a memory-mapped interrupt register. The CPU receives an interrupt if one of these bits indicates an "irq"by means of a hardware OR-function. Subsequently, the CPU may read the interrupt registers via the interrupt service routine in order to identify which event has occurred. The "irq"is then acknowledged by resetting the corresponding irq-bit in the memory. This general communication protocol is visualized in Figure 4.10.

The interrupt mechanism is also used when the processing generates erroneous behavior or when signals are not in accordance with specified video formats. An example of the former case is a coprocessor with hardware problems deteriorating the video signal and an example of the latter case is a signal with a varying number of samples per line or lines per frame caused by substantial timing jitter in the synchronization signals. Since all interrupts generated by the signal-processing subsystem operate at video field rate and can be disabled individually, the interrupt rate to the microcontroller system can be kept sufficiently low. This optimization is advantageous, because interrupts increase the communication overhead (context switching). Adaptation of the processing is sufficient at field frequency, since it provides sufficient response-time of the system. This also gives am-

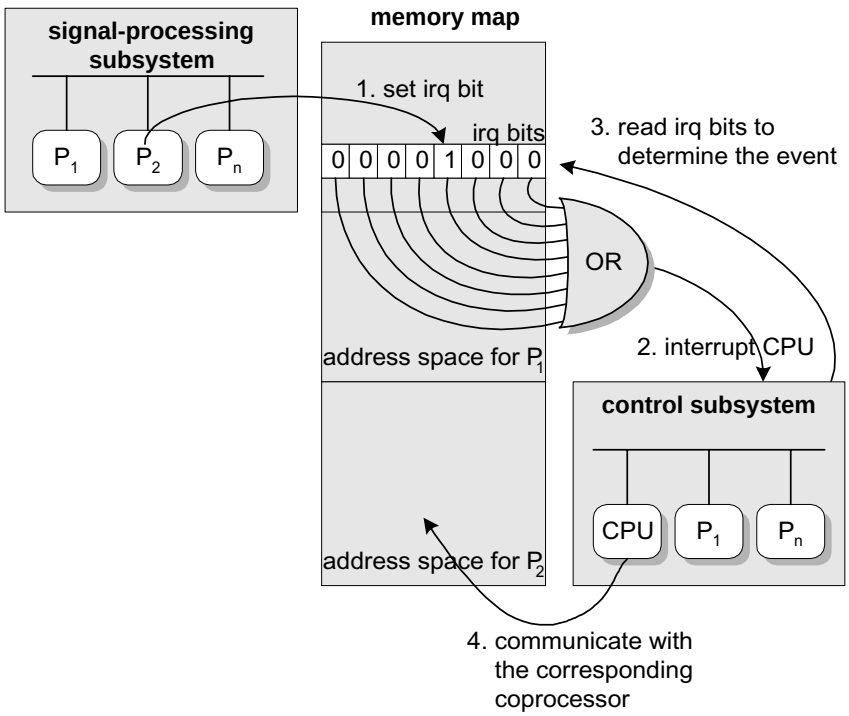


Figure 4.10: *Communication protocol for interactive processing.*

ple time for the microcontroller system to perform other tasks, such as e.g. modem communication or still-picture decoding.

4.6 Memory

4.6.1 Partitioning of internal versus external memory

In this section we focus on the memory aspects of the architecture in order to pursue a flexible programmable system for TV applications. In Section 4.5, the signal-processing model was presented and the required communication for keeping all coprocessors active simultaneously was explained. It is recalled here that the approach adopted is such that basically all coprocessors have local memory to avoid unnecessary background memory access for data packages that require only a small memory space. In the provided example signal processing task graph, the memory access concentrates on creating field/frame delays and alternative significant memory actions, like signal mixing and synchronization of independent video signals.

This approach is best illustrated by a few examples. The first example is the implementation of a vertical video scaler or also known as vertical sample rate converter (VSRC). The principle of the vertical converter is that a number of video lines are provided at the input, while a converted signal having more or fewer video lines is produced at the output. Note that in this concept, although one complete video signal is required at the input, a number samples of different neighboring video lines has to be supplied in parallel. This is because the filter in the converter requires information of various video lines to compute a new video line at the output. Thus the computation of one output signal requires several video streams at the input, corresponding to the number of filter taps (e.g. N). This input process represents a significant memory bandwidth, i.e. roughly N times the normal video bandwidth, if the required video lines are mapped into a background memory. Instead, we propose to insert the required video lines into a local memory unit that is combined with the vertical converter computation. The input of the memory unit requires one video signal, and the output of this unit simultaneously provides the video samples of all required video lines. The memory unit is organized as a line-based FIFO which produces a video output after each line delay. It is clear that this concept saves substantial bandwidth usage to the background memory, because each video line is read only once although it is required for N output lines. Since the amount of lines that are required simultaneously is limited, a local memory is clearly preferable.

The embedding of local memory is only sensible if the required storage space is not too large. This can be easily recognized for other processing units. For example, a 100-Hz converter can be seen as a temporal signal converter that uses field memories to convert a video signal for a temporal up-conversion towards 100 Hz field rate. Conceptually, the field memories belong to the temporal converter. Practically, the field memories require a significant memory capacity and thus silicon area. Additionally, they are of a size that re-usage of this capacity in case the temporal converter is not used is highly interesting. For this reason, such large memory functions are mapped into the background memory.

In this section, we deal with communication aspects of memory processing actions and we discuss several typical TV functions in which memory access and memory-based processing plays a key role. Finally, bandwidth budget and limitations are presented.

4.6.2 Communication between subsystems

As was mentioned in Section 4.4, a cost-effective solution requires one uniform background memory which is based on a standard off-the-shelf RAM. The RAM can be accessed by both the CPU and the coprocessors. An attractive property of the system architecture is that the control-subsystem and the signal-processing subsystem can be used as stand-alone devices, because they provide their own control for operation. Both systems are favorably combined into a powerful full-featured system by a connection via the memory bus, as depicted in Figure 4.11. This enables communication between the CPU and the signal-processing subsystem by means of *memory-mapped* I/O. One unified memory map is applied in which all parameter and operation-mode registers of the coprocessors are located. The CPU can access all the parameter registers to program the coprocessor settings and derive information from the processors and/or the signals being processed.

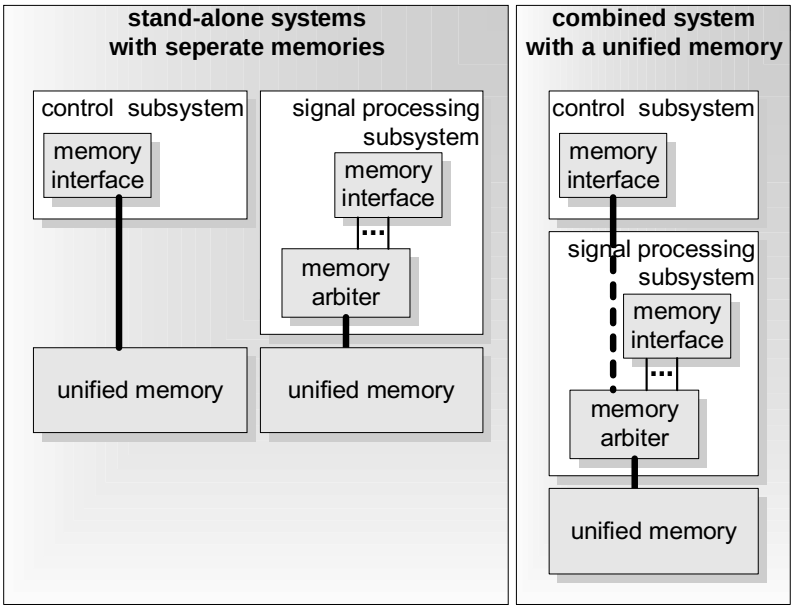


Figure 4.11: *Stand-alone and a combined system with unified memory.*

A complication of the centralized memory architecture is the arbitration of requested access by all processors. The implemented memory arbiter distinguishes two types of memory requests. One from the micro controller, requiring high-priority access with a low-latency and one from the copro-

cessors, requiring a guaranteed throughput. To establish this, the arbiter slices the memory cycles into groups of e.g. 64 cycles. To guarantee sufficient throughput for the coprocessors, a fixed number of cycles within the 64-cycle time slices is devoted to the coprocessor. For example, let us assume that 54 cycles of each slice are reserved for the coprocessors, leaving 10 cycles for the micro controller. If the memory arbiter receives a sequence of requests, first the requests from the micro controller are served. However, if the micro controller has occupied the 10 memory cycles, the priority is given to the coprocessors. Hence, the requests from the micro controller are still served, unless outstanding requests from the coprocessor are available. This straightforward memory arbitration scheme offers a combination of low latency for the micro controller and a guaranteed throughput for the coprocessors, but requires a static assignment of the memory bandwidth budget for the coprocessors.

Since in the architecture proposal a single memory is used, bandwidth limitations can become a serious problem. Some measures have been taken to relax this so-called von-Neumann bottleneck [25]. As an example, the frequency of the memory bus is twice the frequency of the CPU, whereas the bus width is equal to the word width of the CPU (32 bits). Furthermore, coprocessors can be programmed such that memory access can be reduced at the cost of some quality. Even the video application itself can be modified for memory reduction at the expense of a picture quality deterioration. These aspects are discussed in the following subsection.

4.6.3 Memory resources versus quality

The introduction of new TV applications for the system and their corresponding mapping onto the architecture is mainly limited by the computing power of the resources, the bandwidth of the coprocessors, and the bandwidth of on-chip and external memory. Consequently, special attention should be paid to the architectural design of the communication network. As was mentioned at the introduction of this section, communication for video signals via the memory should be limited to large memory functions only, e.g. the field delays for temporal noise reduction and field-rate conversion. Additional key issues in the design of the communication architecture that have a direct impact on the memory resources are listed below.

- *Local FIFO buffers* – These buffers are relatively small and located at the input and output stage of all coprocessors (see Figure 4.5). The local FIFOs enable date exchange between coprocessors without access to the external background memory. This significantly reduces

the bandwidth requirements of the external memory. This contrasts with processor systems that communicate all streams via the background memory.

- *Mixing or juggling* – The video juggler that mixes video streams to create e.g. a Picture-in-Picture (PiP), is designed such that it requires a minimum amount of memory bandwidth. In the background memory, two field blocks (for interlaced video) are allocated to construct the composed video frame for later display. These memory blocks are filled with the odd and even fields of the picture, except for the pixel positions where another overlapping video window will be positioned. This unused memory area in the memory is used by a second video path (corresponding with a different task and signal) to write the overlapping video window. Consequently, the total amount of data stored corresponds to one complete picture instead of two. Similarly, the total required bandwidth approximately equals the bandwidth for writing one complete stream instead of the sum of the bandwidths of the individual video streams.

- *Bandwidth equalization* – This is used to decrease the required peak bandwidth. Equalization in memory read/write tasks is obtained by spreading the data transfer of an active video line over the time of a complete video line including the horizontal blanking. Common video signals contain 15 % horizontal line blanking, leading to a 15 % reduction of the peak bandwidth.

The following aspects deal with individual video functions in which memory can be exchanged against picture quality. This relates to an additional system aspect which aims at realizing *scalable* processing power for a trade-off between computer resources (here memory) and picture quality. Some examples are given below.

- *Vertical scaling* – Vertical sample rate conversion at high quality requires that the interlaced input video is converted to progressive video prior to sample rate conversion. As a result, a memory access is necessary to write interlaced video fields and to read the odd and even video lines progressively. Subsequently, de-interlacing should be applied prior to the scaling. After scaling the progressive frames, they are interlaced again, by removing the odd or even lines before the final result is communicated to the rest of the system. The memory requirements for such a vertical scaling task are significant. At least one field memory is necessary and a memory bandwidth of three times the bandwidth of one single interlaced video stream is required to

write interlaced and to read progressive video. To save some memory resources, it is possible to scale the interlaced fields, thus to perform intra-field processing instead of inter-field processing. This decreases the picture quality but it saves the memory resources for progressive reading of video lines.

- *Graphics* – Graphics generation in the background memory requires a field or frame memory, depending on the desired quality. When a field memory is used and the content is read for both odd and even fields, the amount of memory is reduced at the cost of some loss in vertical resolution. Since synthetically generated graphics may contain high spatial frequencies, the use of a frame memory may result in annoying line flicker when the memory is displayed in interlaced mode. Moreover, it is also expensive in terms of memory size. For 50-60 Hz interlaced video, a field memory is most attractive, whereas for field rates higher than 70 Hz, a frame memory could be used for high-resolution graphics.

There is an important issue that remains after the discussion of the previous memory function aspects. When mapping a video application onto the proposed architecture, the application engineer has to evaluate and balance the total bandwidth of the application considered. This is due to the hard bandwidth limitation of the memory really applied. For example, a dual 16 Mbit SDRAM configuration of 16-bit width per device running at 64 MHz clock frequency offers a maximum theoretical bandwidth of 256 MByte/s. In practice, this will be somewhat lower as a result of inefficiency of the addressing (see Chapter 5). Thus, for a complete video application, the memory requirements of the individual video functions have to be added and compared to the achievable practical maximum. Then a memory budget planning divides the available bandwidth over the functions and trades-off quality and bandwidth. Also the bandwidth of the general-purpose computing units has to be considered in the planning. In the proposed system, a minimum budget is specified for the control processor CPU to ensure controlled operation of the TV system under all circumstances, even if the processing would abort due to e.g. a deadlock. The construction of such a budget and planning is discussed in Section 4.7 using a typical TV application as an example.

4.6.4 Interfacing with the real-time world

The Kahn process network as presented above provides automatic activation of tasks and synchronization of the communicated data. Moreover, it

enables the processors to operate in independent clock domains. For example, the clock rate at which the data is processed can be asynchronous with respect to the rate at which the video signal was digitized. However, in a TV application the display is synchronized to the input signal. Hence, if a 50-Hz TV set receives a 60-Hz video signal, it might not be able to display the video because it cannot synchronize to the signal. The synchronization is achieved with accurate information that indicates the position of the video signal on the screen. For example, at the input of the video processing system the digital video signal is accompanied with a clock signal synchronous to the pixel rate, an H-signal indicating the first pixel of the video lines, and a V-signal indicating the first lines of a video picture. However, due to the data-flow model, the notion of time is lost in the system. The model only guarantees a maintained sequential order of pixels but cannot relate a pixel position to an absolute moment in time. Therefore, when going from the real-time domain to the data-flow domain, special precautions have to be taken so that the relation between the pixel position and time can be recovered.

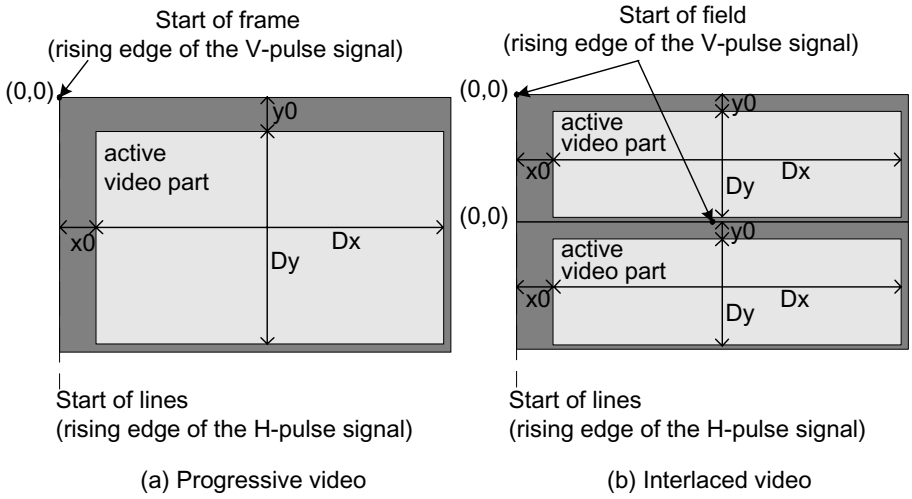


Figure 4.12: *The blanking and the active part of a video signal for progressive (a) and interlaced (b) video.*

We propose an efficient method that maintains the timing information, when data streams from a real-time environment are converted to a data-flow environment and/or visa versa. Using a fixed-sized picture format (pixels × lines), is the most straightforward approach to maintain knowledge on the pixel positions without conveying internal synchronization signals.

Counting pixels is sufficient to determine the exact position of a pixel. Note that this does not recover the timing. Extracting a fixed-sized picture format from a video stream is possible, because only a part of a video signal is used for visible video display. This part is also called the *active* video part. Almost 20 % of the pixels on a video line and 10 % of the video lines in a video frame are not displayed and this is called the *blanking*. For example, a CRT display uses this blanking time to reposition the electron beam to the start of the next video line. Evidently, only the active part of a video stream is of interest for the processing units. Figure 4.12 shows a part of the active video signal that is captured for processing. The left-hand side of the figure shows the capturing in case of progressive video, whereas the right-hand side shows capturing for interlaced video.

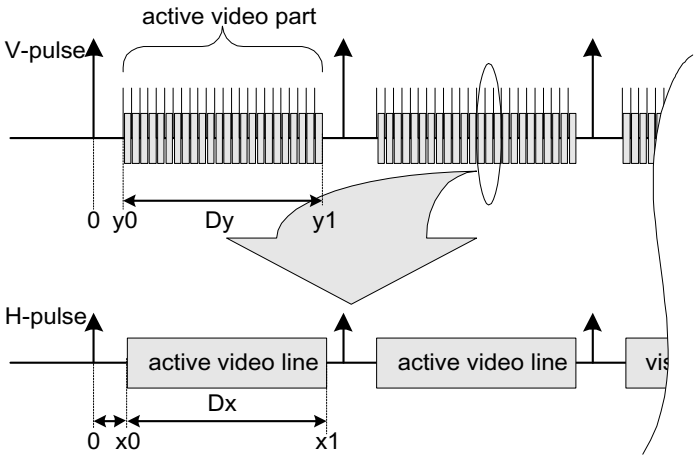


Figure 4.13: *A video signal with H- and V-pulses indicating the active part.*

Figure 4.13 clarifies the active part in the one-dimensional representation of the signal. In the horizontal direction, the active part of a video line starts *x0* pixels after an H-pulse. In the vertical direction, the active part starts *y0* lines after a V-pulse. Because a fixed-sized picture format is required, $x0 + Dx$ and $y0 + Dy$ (see figure) are constant. The remaining pixels of the video lines and the remaining video lines of the frame may vary (besides the right and bottom borders of the active video part in Figure 4.12). Only the fixed-sized active part is conveyed through the processing systems. The format may be regarded as the format of a standard signal (for a standard signal also time-stability constraints are involved). Obviously, the Video Input Module has to retrieve the fixed-sized active part ($Dx \times Dy$) from the video stream in all circumstances. A more elaborate overview of how this is accomplished is described in [14].

After processing the input video signals by the system, the Video Output module of the video processing system is responsible for making a transition from the Kahn process network to the real-time domain. Timing information and synchronization signals for the display have to be recovered and synchronization with a video signal at the Video Input module should be maintained if appropriate. For example, a video PC card displaying a 50-Hz PAL or 60-Hz NTSC signal onto a 72-Hz monitor is an example application where no synchronization is applied. Consequently, some of the video frames are repeated to display 72 pictures per second, resulting in video display that suffers from motion judder. This problem can only be solved by applying motion-compensated frame-rate conversion, which is an expensive video processing function. However, for TV applications, synchronization should generally be maintained. This is provided by the Video Output module which conveys a new output picture, a fixed delay after the picture was captured by the Video Input module. This is achieved by communicating the input V-pulse via a programmable delay to the Video Output module as depicted in Figure 4.14.

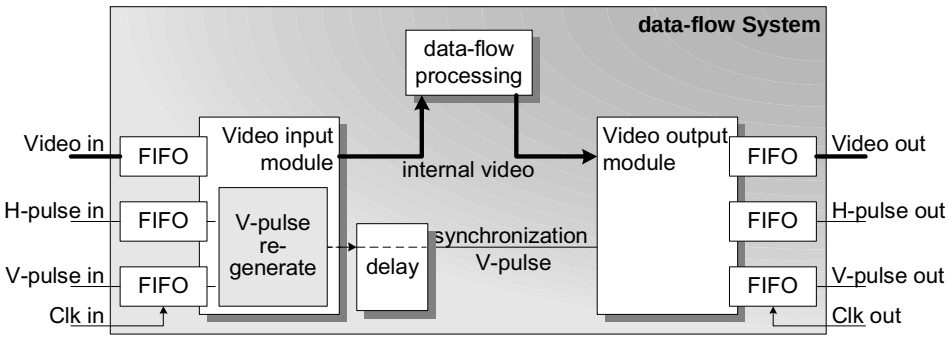


Figure 4.14: *Block diagram of a processing component.*

Note that under normal operation, the sequential order of the input pixels is maintained and that no pixels can be lost in a Kahn process network. Consequently, for every input V-pulse a picture is also received at the output of the system. Although they may seem trivial, the following issues have to be covered by the system.

- A V-pulse that precedes an input picture may only be communicated internally from the Video Input to the Video Output module if the fixed-sized picture capturing can be guaranteed. Hence, even if irregularities or corruptions in the input signal occur, the Video Input module has to convey fixed-sized input pictures into the system to

prevent unpredictable behavior of the system. For example, if a V-pulse is received at the output while no corresponding picture was captured, the Video Output module would drain the system, causing underflow. Similarly, when a picture is captured while the corresponding internal V-pulse is absent, the system will overflow since the picture will not be output for display.

- When 50-to-100 Hz conversion is applied by the system, one input V-pulse corresponds to two pictures at the output. Hence, the Video Output module needs to generate additional synchronization signals while the original signals remain synchronized with the video signal.

- For correct reproduction of the output signal, the pixel data should be available for the Video Output module as soon as they need to be output for display. Even if the data are not yet available due to e.g. erroneous settings in the software, the Video Output module is not allowed to skip pixels or to output pixels more than once. Due to the Kahn process network, this would cause a phase shift between the active video pixels and the V-pulse which cannot be detected and hence cannot be recovered.

Some of these issues are hard to prevent. For example, capturing fixed-size pictures for every input V-pulse cannot be achieved if the input is not connected and the input voltage is determined by noise. However, to deal with malfunctions both the Video Input module and Video Output module can perform measurement on the input signal and can communicate its status to the microcontroller by means of interrupts. Hence, if an input signal cannot be captured or displayed correctly, this can be signalled. Subsequently, the microcontroller can respond to the system by reprogramming coprocessors. The following items represent some example actions that the microcontroller can take.

- The video processing system can be reset and the Video Input module can be programmed to discard its inputs and to generate a blue screen. Moreover, graphical text can be blended onto the video indicating for example a missing input signal.

- If an underflow or overflow occurs, the subgraphs as described in Section 4.3 can be decoupled in the background memory. This means that the output of the predecessor subgraph is written into the memory and the input of the successor subgraph reads this memory data, asynchronously with respect to each other. Consequently, the speed of the predecessor subgraph is driven by the input video rate, whereas

the speed of the successor subgraphs is driven by the output video rate. Once the signal flow through the system has been stabilized again, synchronous communication via the background memory can be established again for optimal picture quality.

- The Video Input module is able to measure the number of pixels per line and the number of video lines per picture and can make the results available for the microcontroller. This information can be used to automatically capture the correct resolution at the correct frame rate. Moreover, this mechanism can also be used to capture only part of the input in able to free system resources or to quickly lock the Video Input module to the input signal.

The above-presented mechanisms only give a overview of the functionality to make the system more robust, i.e. guaranteed stable operation under all circumstances, synchronized to the input signal. For a more detailed description of the Video Input and output modules see [14].

The delay between the Video Input and output module determines the amount of data that is buffered into the processing system. A small latency of the processing units and a large delay of the internal V-pulse from the Video Input to the Video Output will require a large data buffer, whereas a large latency and a small delay of the internal V-pulse requires little buffering. Obviously, the latency of the total processing system should be constant to provide a regular video stream for the display. Because the processing may vary over time, also the latency of the processing changes. For example, the user may activate a vertical sampling-rate converter to do aspect-ratio conversion in a widescreen TV set. This will increase the latency of the processing chain depending on the scaling factor, the order of the filter, and the position of the video window to be scaled. The variation in the latency has to be compensated by buffering. With a delay unit for the internal V-pulse signal (see Figure 4.14), the exact time the Video Output module receives the V-pulse can be set such that for a maximum latency application, the buffer requirement is minimal. The application with the minimum latency will then determine the maximum buffering for compensation that should be available. This is illustrated in Figure 4.15

The television processor system as presented in this chapter is optimized for minimal communication requirements. Consequently, a tight coupling of the tasks on the coprocessors is provided by means of small FIFO buffers. Even the communication via field or frame buffers in the background memory for e.g. de-interlacing or field-rate conversion, is synchronized at pixel granularity. Hence, a read action can track the write pointers with one pixel

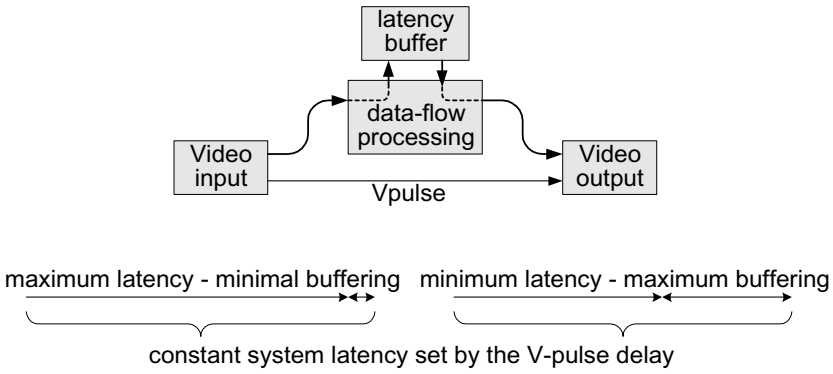


Figure 4.15: *Relation between the processing latency and the V-pulse delay.*

distance. This means that the transition from the Kahn process network domain to the real-time domain is particularly critical in such a system. Just a little too many or few pixels captured or displayed causes an undesired phase shift of the video signal with respect to the synchronization signals which cannot be recovered any more. Hence, the system has to be robust to all possible signal changes at the input or output. In conventional data-flow systems, robustness for the input signals is provided by first writing the input video streams into frame buffers. Subsequently, the video pictures are read and processed by the system. Even if the input signal is corrupted and only part of a picture is captured and written into the frame buffer, a complete picture is read for processing. It can be concluded that the frame buffers decouple the Kahn process network model from the real-time world. This robust decoupling using a frame buffer is not provided in the presented system of this chapter. This system features a tight coupling of the input and output tasks with the remaining video processing tasks by means of small FIFO buffers, thereby saving scarce memory bandwidth. Consequently, a robust interfacing between the real-time world and the Kahn process network model should be provided within the input and output processors. This required special attention for the design of the Video Input and Output modules.

4.7 Implementation example of the architecture

4.7.1 Introductory system presentation

In this section, we will focus on the functional possibilities of the architectural concept presented in the first part of this chapter. For this purpose,

we present an instantiation of the architecture in the form of a chip set containing two chips. The chip set was developed for experimental purposes. One of the chips was commercially released (known as the SAA7430)in a later stadium, whereas from the other, modules were extracted for alternative implementation projects.

The purpose of this example is to put the concepts of the first part of this chapter into practice and to learn from the mapping of video applications onto such an architecture. At the time of writing of this chapter, already new instantiations of programmable video architectures are emerging but they apply similar principles. The scale of operations and functions progresses with the IC technology.

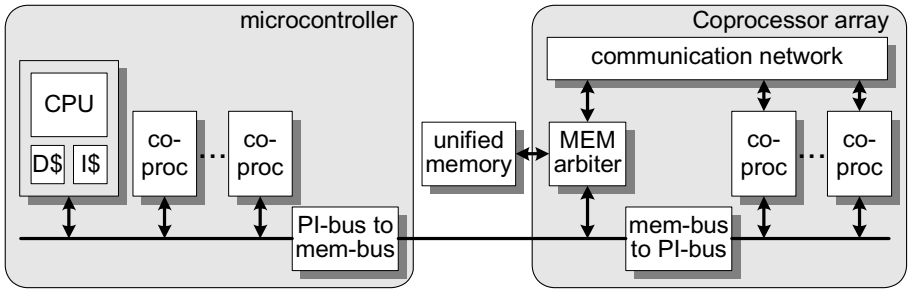


Figure 4.16: *The experimental two-chip digital video platform.*

Figure 4.16 shows the hardware architecture of the programmable video processor chip-set. The system consists of two chips: a micro-controller which has been presented in [64] and a coprocessor array. The former chip is for Teletext (TXT), graphics generation, system and chip control and various communication features. The latter chip executes the necessary video tasks in weakly programmable hardware. Both ICs are autonomous units which communicate with each other via a central bus. Internally, this bus operates according to the Peripheral Interconnect (PI) bus protocol [65], whereas the external part of the bus uses a synchronous DRAM (SDRAM) memory protocol. This enables stand-alone operation of both ICs, while for the case that both ICs are interconnected, they make use of one single unified memory.

The novelty of the system for TV applications is the high flexibility in signal processing. The *task graph*, i.e. the order of signal processing func-

tions through all coprocessors, is programmable by means of a flexible communication network [66]. The processing of data by several coprocessors is achieved without the need to access the bandwidth-limited memory for communication purposes. The coprocessors, which are interconnected via a communication network, are synchronized by means of data-driven processing of the pixels [62]. This implies that each task in a task graph or kahn process network is executed when data is present at the input, and if storage space is available at the output stage. This run-time synchronization mechanism, provides autonomous processing without interference of a micro-controller or a hardware scheduler. More details can be found in [6].

Let us now discuss how the aforementioned architectural properties are exploited for flexibility and new features. Two examples are illustrated briefly. An overview is provided in a following section.

- The two chips share the background memory. This enables the possibility to assign the available memory space to applications in a dynamic way. For example, if extra graphics data are generated for user control interface, the quality of one of the video applications, such as vertical scaling, can be scaled down temporarily to release resources.

- The order in which video functions are carried out is programmable by constructing a task graph. Furthermore, most coprocessors can execute more than one task at the same time. Consequently, more than one video input stream can be processed simultaneously. The tasks that are applied for each stream may be different and/or have different settings. For example, for two video windows, the noise reduction can be carried out for the window where it is needed most.

Note that the re-programming of coprocessors requires that specific aspects need to be covered. Firstly, if task switching is performed within an ongoing task, the processor should store that *state* or *context* of the process required to resume processing from the point where it stopped. Secondly, the processor should be informed about the data identity, thus the video stream type that is processed. This can be either communicated by the control PI bus, or by labelling the data with a data header packet that tells the origin, destination and nature of the video stream. Thirdly, if processes are dependent on each other and executed on the same processor, a so-called *deadlock* could occur. This is avoided by implementing separate input and output FIFOs for each possible task on a coprocessor (see Subsection 4.5.2). Let us now list the video functions of the system.

4.7.2 Overview of hardware functions

In this section we briefly describe the hardware blocks integrated in the micro-controller and the video processor chip. The micro-controller contains a 32-bit R3000 reduced instruction set computer (RISC) core for control of the video coprocessor array, and TV-set control. Moreover, blocks are integrated for graphics generation, Teletext decoding, and modem functionality for Internet connection. The peripherals in the micro-controller, as depicted in Figure 4.16, are an interrupt controller and a number of standard communication modules (UARTs, Infrared support for remote control, JTAG for testing, etc.). The controller chip also contains an I^2C block for generic control of neighboring chips. Important is the graphics output processor, which enables pixel-based graphics with a resolution ex-

Table 4.2: *Hardware functions in the micro-controller chip.*

Category	Hardware support
Software execution	R3000 RISC with I- and D-cache, interrupt control, timers, watchdog timer.
Teletext	Input for TXT front-end IC SAA5284.
Control communication	2 × high-speed UART (230 kbit/s) 2 × I^2C, IR connection.
Data communication	Serial Interconnect Bus (SIB) for UCB100H modem IC.
Memory connection	SDRAM controller for a.o. 16 Mbit and 64 Mbit memory devices, general-purpose ROM interface.
Testing, miscellaneous	enhanced JTAG interface, general I/O pins, software ADC pins.

ceeding conventional TXT images. Finally, an SDRAM memory controller supports the connection of a large external SDRAM memory for executing all software tasks and exchanging data with the coprocessor array. Table 4.2 gives an overview of the various hardware functions.

The video coprocessor array (see Figure 4.17) performs all video signal-processing functions of the chip set (see Table 4.3). It contains a set of coprocessors for TV functions which are typical for a high-end TV set or set-top box. For video image resizing, it contains a horizontal scaler and a vertical scaler with de-interlacing capability. This de-interlacing prevents aliasing artifacts and maintains optimal resolution when using interlaced

video signals. The input signal quality may be improved using the integrated adaptive temporal noise reduction, which analyzes both the video signal noise and the motion. The sharpness may be augmented with the contrast-adaptive local sharpness enhancement [9].

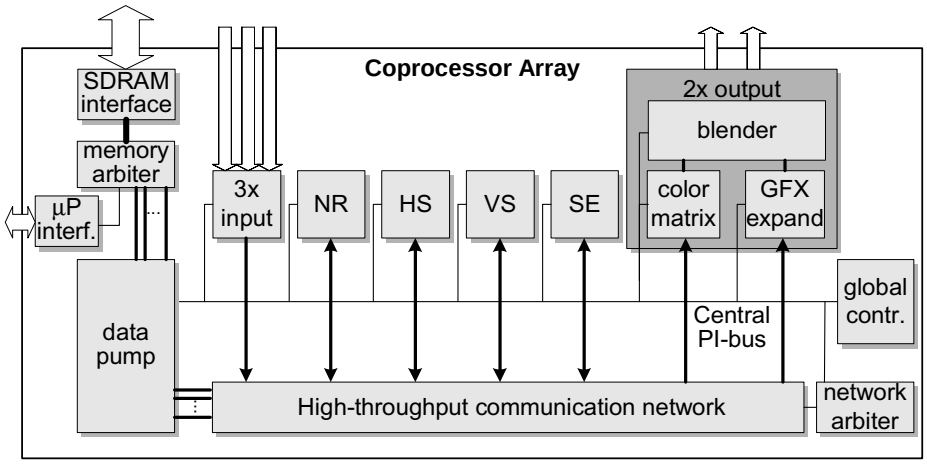


Figure 4.17: *Block diagram of the coprocessor array.*

There is also support for mixing several video streams. Firstly, a graphics blender features "alpha"blending of graphics and full-motion video. Secondly, a mixer can combine up to three moving video streams. Finally, a color-space converter transforms YUV to RGB signals. Most coprocessors are implemented only once, but have capabilities for parallel processing of multiple tasks. In this flexible approach, each task is independently programmable, e.g. the scalers can down-scale or up-scale an image to any size with arbitrary filter coefficients.

4.7.3 Video applications

Because functionalities of TV and PC are merging increasingly, it is likely that the user-interfaces of these devices are going to show similarities. For the TV, this implies a multi-window environment with multi-tasking capabilities. The flexibility of the described system enables this feature and provides a broad range of applications which are new in a consumer TV environment. From the previous sections, it can be seen that programming of an application is straightforward and is limited to programming of the task graph and the individual setting of the tasks. In the following subsection, the functionality and the settings of the individual coprocessors are

Table 4.3: *Hardware functions in the coprocessor-array chip.*

Category	Hardware support
Signal quality	Temporal noise reduction, locally adaptive Y peaking, Transient Improvement of UV signals.
Scaling	Horizontal scaling (up to 3 streams), vertical scaling (2 streams).
Graphics, mixing	Alpha blending, resolution > TXT, up-conversion of TXT signals, mixing up to 3 signals.
Video Interfaces	3 YUV inputs, 2 YUV/RGB outputs, CCIR-656 support.
Memory connection	SDRAM controller for a.o. 16 Mbit to 64 Mbit memory devices.
Testing, miscellaneous	JTAG interface, I^2C for stand-alone operation.

described. This description together with a few application examples, gives an impression of the application possibilities.

Functionality of the video coprocessors

The video coprocessor array (in the sequel mostly referred to as CPA), performs the computationally expensive regular video processing and contains coprocessors with carefully selected functionalities. Some coprocessors contain a cluster of video functions, since these functions are always used together. Flexibility in the processing order of those functions would not make sense. The following items describe the functionality of the individual coprocessors.

Vertical sampling-rate converter (VS) for up-scaling and down-scaling of images in vertical direction to any size. It is based on a 6-tap 32-phase polyphase filter and has a median filter to perform de-interlacing (optional). The number of applied filter coefficients is programmable as well as the magnitude of the coefficients. The de-interlacing is optional. Also programmable are the scaling factor (-4 .. 64) for down-scaling and up-scaling and the resolution of the input image. A local memory for eight full-color video lines is available and can be divided over the several vertical-scaler tasks which can be executed simultaneously.

Horizontal sampling-rate converter (HS) for up-scaling and down-scaling of images in horizontal direction to any size. This scaler is based on a 6-tap 64-phase polyphase filter and can be switched in a transposed mode. In this mode, only one single Nyquist filter has to be selected to down-scale the video pictures to any arbitrary size with high picture quality. The scaling factor (-64 ..64) for horizontal up-scaling is variable and according to a programmable parabolic curve as a function of the pixel position. This enables the commercially available "super zoom" option for aspect-ratio adaptation. A more detailed description of the sampling-rate employed conversion technique can be found in [10].

Advanced dynamic noise reduction (NR) to perform temporal noise reduction, adaptive to the amount of noise and the amount of motion in the picture. A special low-pass filter ensures that noise reduction is performed in the frequency area where the human visual system is most sensitive to the noise, while high-frequency detail is preserved. The strength of the noise reduction and the motion adaptivity is programmable by means of a programmable look-up table.

Adaptive sharpness enhancement (SE) for subjective sharpness improvement of the luminance as well as the chrominance. For the luminance signal, a 2-D high-pass filter is used to create an enhancement signal which is added to the original (peaking). A large extent of programmable control logic provides suppression of the enhancement on those places where edges are absent and noise is visible. Suppression also takes place on image locations where enhancement would introduce aliasing artifacts and edges are already sharp, e.g. synthetically generated graphics in the video. A more detailed overview of the adaptive peaking function is given in Chapter 3. For the chrominance signal, a non-linear algorithm is implemented that increases the steepness of the edges by means of pixel translations.

Graphics (GFX) with video blending for graphics formats up to 16-bit resolution. The formats are converted to a 24-bit RGB format and the video is converted via a programmable color-space matrix from a YUV 4:2:2 or 4:4:4 format to a RGB or YUV 4:4:4 format. For higher video resolutions, it is possible to up-convert the GFX by pixel repetition with a factor 1–8 to limit the CPU load for the GFX generation. This preserves a high graphics bandwidth and gives subjectively the best picture quality. The fully digital blending of the GFX into the video is performed with an alpha factor (α), describing the fraction with which the GFX is blended into the output signal. For mapping GFX pixels onto the signal components, RGBα, a color

look-up table (CLUT) is included. This allows the following GFX formats: CLUT8 (8 bits/pixel), RGBα 4:4:4:4 (α through LUT), RGBα 5:5:5:1 (α through LUT) and RGBα 5:6:5:0. Furthermore, the blender enables color keying for the video and/or the graphics.

Input processor for retrieval of three real-time video sources at resolutions up to HDTV and VGA. The processor supports several signal protocols for synchronization, e.g. with H- and V-pulse, with active video identifiers or according to the CCIR-656 recommendations [67]. The digital input signal may have an 8-bit or 16-bit bus, depending on a programmable time-multiplex mode. In addition, the input processor may select a capture window in the video to select the pixels to be processed by the rest of the system. This could be useful to reduce the bandwidth when only a part of the video is required for further processing, e.g. in the case that a part is zoomed in by the scalers.

Output processor to output one or two video signals with similar features as the input processor. Additionally, the output processor also contains a multiplex mode to display a 24-bit YUV or RGB signal. Both the input processor as the output processor have been developed such that synchronization of the video streams is independent from the internal processing speed. This requirement was achieved by introducing separated clock domains separated by FIFO memories [14]. This video I/O model is basically asynchronous and yet enables to synchronize a video display to one of the video sources. This is achieved by applying the data-flow concept for writing data into FIFO, i.e. the writing is blocked if the FIFO is full to prevent buffer overflow. The output of the FIFO for display is read with a constant rate. Hence, if the data is written sufficiently fast into the FIFO also buffer underflow cannot occur. This requirement is extremely important for constructing a well-defined video display signal model with appropriate delays for the primary and secondary video signals.

A memory input and output port to store, or shuffle video data at multiple positions in the video task graph. It can access a programmable cyclic memory block in a sequential order and has the additional option to skip video lines in order to access a progressive frame in an interlaced manner.

An advanced address generator (juggler) to write video data into the memory at any position with an arbitrary shape, e.g. to create a circular or alternative shaped Picture-in-Picture (PiP) [68].

Table 4.4: *Task resources of the coprocessors.*

Horizontal scaler	$1\times <$ 64Mpixels/s or $2\times <$ 32Mpixels/s or $2\times < 16 + 1\times <$ 32Mpixels/s or $1\times < 16 + 1\times <$ 48Mpixels/s
Vertical scaler	$2\times <$ 32Mpixels/s or $1\times < 16 + 1\times <$ 48Mpixels/s
Sharpness enhancement	$1\times <$ 32Mpixels/s
Noise reduction	$1\times <$ 16Mpixels/s HQ or $2\times <$ 16Mpixels/s MQ
Color conversion, GFX up-scaling, GFX blending	$1\times <$ 64Mpixels/s
Video Inputs	$2\times <$ 64Mpixels/s
Video Outputs	$2\times <$ 64Mpixels/s
Memory Inputs Memory Outputs	8 inputs + 12 outputs total < 192Mpixels/s

Resource requirements

To create an application, several video tasks as mentioned in Table 4.3 have to be executed in parallel. Because some tasks in the application may even be similar, some coprocessors should be able to execute more tasks simultaneously. As a consequence, the data rate in the coprocessors and the data bandwidth to the memory increases for more complex applications and is limited by the physical clock rate of the coprocessors (in this implementation 64 MHz) and the memory. Summarizing, the complexity of a complete application is limited by the available task resources of the individual coprocessors and the memory capacity and bandwidth. The task resources of the available coprocessors are shown in Table 4.4.

The current high-end TV-sets show features like Picture-in-Picture (PiP), dual-screen and Internet TV. However, these features are just a subset of the application space which is available with the introduced chip-set. Let us start with a straightforward example and consider a PiP application. Figure 4.18 shows the task graph of this application.

Bandwidth analysis
For wide-screen SD signals in the PiP application, the task resources as mentioned above are more than sufficient. To calculate the memory band-

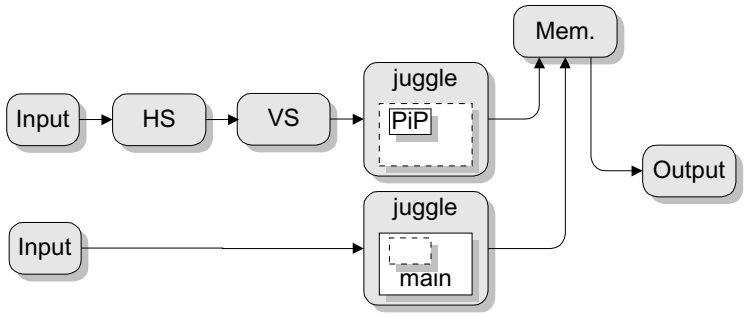


Figure 4.18: *The task graph of a PiP application.*

width, it is assumed that the video data rate is $f_i = 16$ MHz[1] . The system memory device runs at a clock rate of 96 MHz and has a bus width of 32 bits. For 16-bit pixels, this means a total memory bandwidth of 384 MByte/s. All communication between the CPU and the coprocessors for control and e.g. graphics generation, is performed via memory. For this type of memory communication, a part of the memory bandwidth is reserved and thus cannot be used for the remaining video processing. Assuming 30 MByte/s of memory bandwidth for control and communication between the CPU and the coprocessors, a bandwidth of 354 MByte/s remains for video processing. For the simple PiP application, only memory access for the mixing is necessary, thus the amount of available memory bandwidth is only used for a small part. This mixing or juggling of the video streams is designed such that it requires a minimum amount of memory bandwidth.

Memory capacity
In the background memory, two field blocks (for interlaced video) are allocated to construct the frame pictures. These memory blocks are filled with the subsequent odd and even fields of the picture, except for the pixel positions where the PiP window is located. This unused area in the memory is occupied by the second video path to write the PiP window. Therefore, the total amount of data stored, is equal to the data of one complete picture and similarly, the total required bandwidth equals the bandwidth for writing one complete video stream. With 2 Byte/pixel, the amount of memory becomes 0.98 MByte and the bandwidth becomes 64 MByte/s (reading and writing).

[1]For 4:2:0 and 4:2:2 sampling in DTV standards, the luminance sampling rate is usually 13.5 MHz, but in many television systems 16 MHz is commonly applied to compensate for widescreen displays.

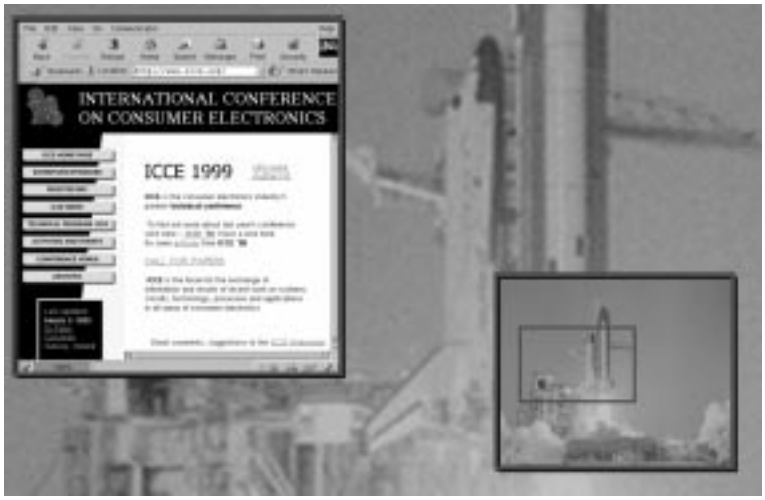


Figure 4.19: *A multi-window application with video and Internet.*

Since the two input video sources are generally not *synchronous*, the output image should be synchronized to one of the input video sources by means of a parameter setting. For the PiP application, the output video signal is usually synchronized with the input video signal of the main background picture. As a result, the input video signal of the PiP is written asynchronously into the field memories.

The following two application examples will discuss the budget assignment of the memory memory resources in more details.

Application example I: Zoomed Multiwindow

A more advanced example of a television application is a PiP with a video background containing a zoomed-in part of the PiP. The part of the PiP to be zoomed in on can be selected by the consumer by means of a small graphics square that can be moved and resized. In addition, an Internet browser, executed on the CPU, is shown on the display (see Figure 4.19). For generation of the graphics in the picture, no task resources other than the CPU and a memory output port are necessary. Therefore, the following calculations consider the video signal processing only and assume that the graphics are available in the background memory.

The task graph of the application is portrayed by Figure 4.20 and contains noise reduction (NR), scaling (HS, VS), mixing (juggle), sharpness

enhancement (SE) and graphics blending prior to picture display. The lower part of the figure shows that two video streams are processed: one for the zoomed-in background including noise reduction and one to create the PiP. After combining the separate video signals in the memory, sharpness enhancement is applied. At the output stage, the video is blended with the graphics that is generated by the CPU.

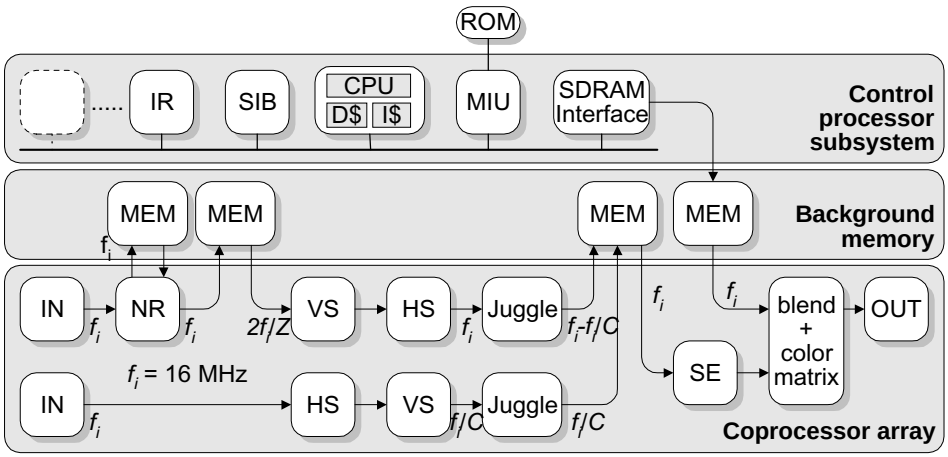


Figure 4.20: *Task graph of the multi-window application with Internet.*

Table 4.5 presents an overview of all memory accesses, including the required background memory capacity and the number of inputs and outputs to and from memory.

Table 4.5: *Background memory requirements for the multi-window application.*

	Connect. to/from Mem	**Memory (MByte)**	**Memory bandwidth (MByte/s)**
NR (up-scaling)	1/1	0.49	64
VS (up-scaling)	1/1	<0.98	<96
Juggling (write/ read, worst case	2/1	0.98	64
graphics	0/1	0.49	32
Total (worst case)	**4/4**	**2.94**	**256(218)**

Bandwidth for NR
First, the noise-reduction (NR) coprocessor accesses the memory to use a field delay for advanced adaptive temporal Infinite-Impulse-Response (IIR) filtering. For an SD image of 288 lines × 854 pixels with 2 Bytes/pixel, the required amount of memory equals 0.49 MByte. For a pixel rate of 16 Mpixels/s, the total memory bandwidth for writing and reading is 64 MByte/s.

Bandwidth for scaling
The memory requirements for the access prior to vertical scaling are different. The image is written with 16 Mpixels/s, but is read at $2 \times 16/Z$ Mpixels/s for inter-field processing, with Z being the up-scale factor. Because $Z > 1$, the required memory bandwidth is smaller than 96 MByte/s. If intra-field processing is used for vertical scaling, the data rate is even less than $16/Z$ Mpixels/s. The computation for the amount of buffering is less straightforward. If inter-field processing is used, a complete field of L_f lines has to be written in the memory and cannot be overwritten, because it has to be read out two times. Therefore, the required amount of memory for progressive video that is up-scaled equals:

$$Buf_{\text{inter}} = 2 \times \frac{L_f}{Z} B_l,$$

where B_l denotes the number of bytes per video line. For intra-field scaling, buffering is only necessary to compensate for the rate difference between the writing to and reading from the memory. In this case, the time for writing one field is equal to the time for reading one field. Writing of the video lines that will be up-scaled is performed at a higher rate than reading. The maximum distance in the memory between the read and write pointer is equal to the memory space that has to be buffered for the rate difference. This maximum distance occurs when the write pointer has just finished the field. The part of the field that has been read at that time is $1/Z$. Therefore, the part that is not read yet equals $1 - 1/Z$. As a result, the buffering that is required to deal with the rate difference equals:

$$Buf_{\text{intra}} = \frac{L_f}{Z} \left(1 - \frac{1}{Z}\right) B_l.$$

Since it is desired to have a static memory allocation, the maximum buffering can be found as follows:

$$\frac{d}{dZ}\left(Buf_{\text{intra}}\right) = -\frac{L_f}{Z^2}\left(1 - \frac{2}{Z}\right) B_l = 0 \Rightarrow Z = 2,$$

$$Buf_{\text{intra}} = \frac{L_f}{Z}\left(1 - \frac{1}{2}\right) B_l = \frac{L_f}{4} B_l.$$

For $L_f = 288$ and $B_l = 1708$ the amount of required buffering is Buf_{intra}= 0.12 MByte.

Bandwidth for Mixing and Graphics
Finally, the mixing or juggling of the video streams for image composition is performed. As explained in the previous subsection, the amount of data stored is equal to one frame and the required bandwidth equals to the bandwidth for writing one complete video stream. For generation of the graphics in the background memory, a field or frame memory could be used depending on the desired quality. When a field memory is used and the content is read for both odd and even fields, the amount of memory is reduced at the cost of some vertical resolution. Since synthetically generated graphics may contain high spatial frequencies, the use of a frame memory may result in annoying line flicker, when the content of the memory is displayed in interlaced mode. Therefore, a field memory is most attractive for 50-60 Hz interlaced video, whereas for field rates higher than 70 Hz and high-resolution graphics, a frame memory could be used.

Bandwidth and memory conclusion
Summarizing, the total amount of applied video memory is less than 3 MByte and the maximum memory bandwidth is 256 MByte/s. This required bandwidth is only used during the transfer of the active pixels. During the blanking, no data is transferred, thereby decreasing the average bandwidth significantly. In order to decrease the peak bandwidth, the data transfer rate can be equalized over time. To do this, the read and write tasks of the video processing system have the ability to spread the transfer of an active video line over the time of a complete video line including the horizontal blanking. Typical video signals contain 15 % horizontal line blanking time, so that the total amount of bandwidth can be reduced by 15 %. For this application, this leads to a net total memory bandwidth of 218 MByte/s.

Application example II: Dual screen 100 Hz

This subsection discusses an application where the memory bandwidth plays the principal role. Since the proposed architecture is mainly limited by the throughput bandwidth of the coprocessors and the memory, a large range of resolutions and frame rates can be generated. It may even provide 50-to-100 Hz conversion, making advanced use of the memory.

Figure 4.21 shows a simple dual-screen application which also provides 50-to-100 Hz conversion. The total task graph can be divided into several

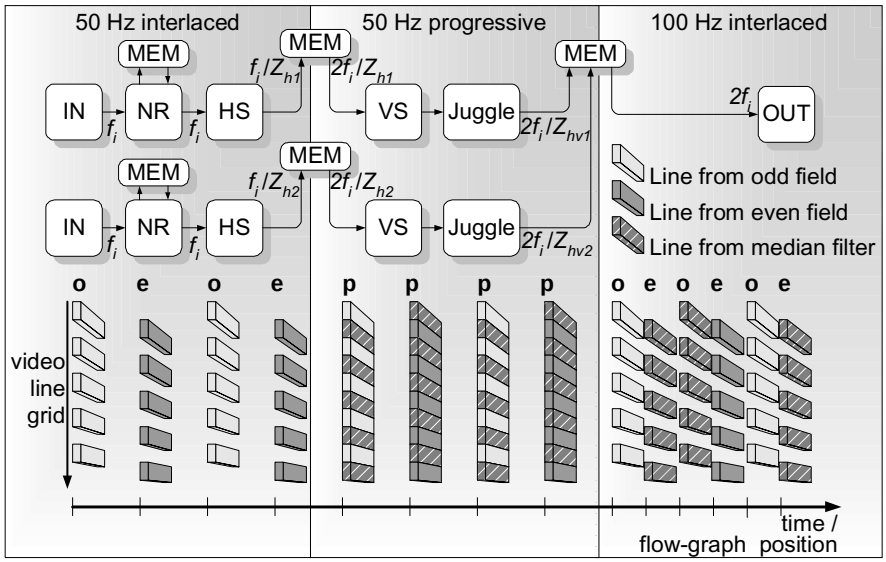


Figure 4.21: *An example application with 100-Hz conversion.*

independent subgraphs, separated by memory accesses. Because temporal scaling requires the use of field and/or frame memories, this can only be provided by the intermediate memory accesses. Therefore, it is not possible to perform temporal scaling within a subgraph. Only spatial scaling with relatively small local buffering can be applied. The subgraphs that contain the input processors (IN) should operate at the field rate of the (50 Hz) input signals, due to the data-driven concept. The bottom part of Figure 4.21 illustrates the position of the vertical video lines as a function of time. After temporal noise reduction (NR) and horizontal scaling (HS), the interlaced fields are written into the memory. In the succeeding subgraphs, the video data are read from memory again and scaled down in the vertical dimension to obtain the correct aspect ratio of the input images. As was mentioned in the previous subsection, the vertical scaler may read the data from the memory in a progressive-scan format to enable high-quality scaling. The vertical scaling is then applied to progressive video and is interlaced again at the output stage of the scaler. For this mode of operation, the interlacing at the output is not used and the video is written into the memory again in a progressive-scan format.

In the bottom part of Figure 4.21, it is shown that all missing lines of the interlaced fields are filled with video lines from the median filter. If further vertical processing of progressive video would be desirable (e.g. 2-D

sharpness enhancement), it would be obvious to perform it in this subgraph. The right-hand side of the figure contains the subgraph that reads the 50-Hz progressive frames in memory with an interlaced scan to do the field-rate up-conversion and to create the 100-Hz output signal. The figure shows a pair of subsequent fields containing both original video lines or video lines from the median filter. This type of 50-to-100 Hz conversion is commercially available in some TV sets and known as "digital scan". Let us finally consider the memory requirements for this 100-Hz dual-screen application. The necessary bandwidth equals 300 MByte/s and the amount of memory used is 2.94 MByte. These numbers are computed using similar assumptions as in the previous subsection, and include a 100-Hz graphics signal (stored in a field memory) in the final picture.

Overview of TV applications

The flexibility of the described system enables a broad range of features which are new in a consumer TV environment. Examples are given in Figure 4.22, which provides an overview of the application possibilities. Some of the features will be discussed briefly here.

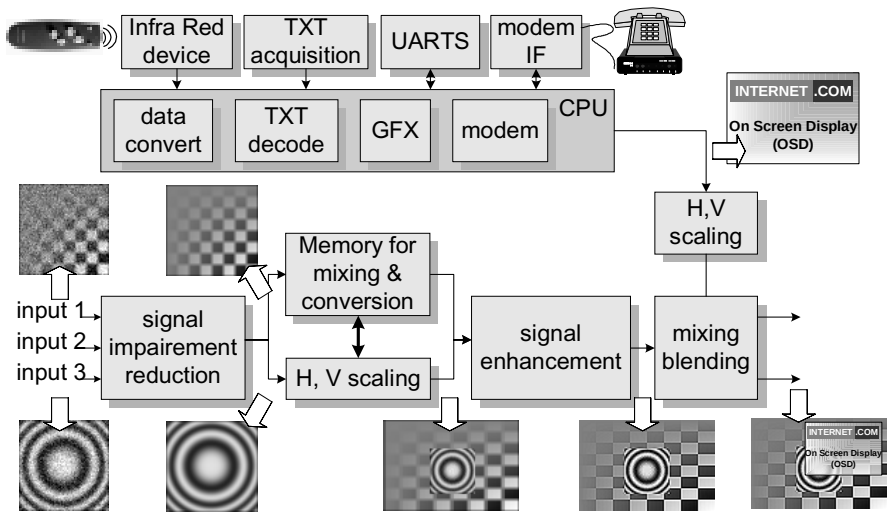


Figure 4.22: *Overview of system features.*

The top half of Figure 4.22 shows the application area of the telecommunication and control processor (TCP). An infrared (IR) device accepts user control commands from the remote control. Furthermore, acquisition and decoding of Teletext information is carried out and data ports

such as UARTs are available. New are the integrated modem interface enabling glueless connection to an existing modem IC and the generation of pixel-based graphics (GFX). The latter feature also supports Internet applications as is indicated. The processor also provides control of the special video hardware, which is shown at the bottom half of Figure 4.22 and control of external additional hardware. This programmability is possible, because the video processing does not require a large cycle budget of the CPU. Control of the video processing is discussed later in this section. Let us first focus on the aforementioned features.

In the field of Teletext, On-Screen-Display (OSD) and GFX (graphics) processing, the TCP has to decode Teletext information and generate the GFX in the memory, without severe real-time constraints. However, for GFX refresh rates higher than 10 Hz, the load on the CPU cycle budget becomes significant. The following features are supported by the hardware:

- high-quality pixel-based 2-D graphics;
- all-page storage for Teletext and programmable fonts;
- integration of GFX with photos or images;
- user-defined GFX environment;
- electronic TV program guide application;
- automatic TV controller with user-dependent setting and support via messages and menus.

For the modem functionality of the TCP, the real-time constraints are much more demanding, since the majority of the modem data conversion is executed as a SW program on the CPU. This may result in less performance of the graphics, depending on the application and the software architecture. The modem extension in the TV system offers a new range of telecommunication features, such as

- fax-message and data-file reception, storage and display;
- Internet connection for a.o. program data retrieval;
- interactive TV communication;
- downloading of photos and images;
- execution of various computer (WinCE) applications.

In the bottom half of Figure 4.22, the application domain of the video coprocessor array (CPA) is shown. The design of the architecture is such that video processing can take place without continuous set control. Control is performed on a periodic basis only (field rate), although control on interrupt basis is also possible. Video signals are mostly noise reduced at the input stage in the coprocessor array (bottom left). Furthermore, fully programmable video scalers can be used for down-scaling or up-scaling of full-motion video signals. This enables virtually any type of scaling function with a large dynamic range, which results in a very flexible multi-window TV. The setting and control may also be defined by the individual user. The signal quality can be optimized over the several windows. The multi-signal processing capability is very important for composing pictures of various size and contents. In all of these modes, the TCP generates the high-level commands for programming and setting of the CPA coprocessor hardware, thereby enabling for example:

- aspect-ratio conversions (panorama, side-panel, wide-screen);
- PiP, dual-screen, multi-window (arbitrary sizes, variable shape);
- PiP record and PiP playback;
- mosaic screen for visual channel selection;
- flexible matching to various input/output resolutions;
- high-quality sharpness improvement;
- dynamically moving of video, menu's and graphics.

Finally, as indicated in the Figure 4.22, graphics and video are blended in fully digital form. For this purpose, some of the graphics can be up-converted to a higher resolution, if required. For more details about applications and the quality of the individual coprocessors, the reader is referred to [13].

The most flexible and interesting features are enabled by the configuration where both chips are connected to each other with sufficient SDRAM and the modem function is activated. Whilst looking to a TV program, an image can be retrieved from the Internet and the TV may signal the completion of the recovered image to the consumer. If extra memory is needed temporarily for Internet communication, some memory may be taken from the video processing (e.g. the quality of the 100Hz conversion), in order to boost the microcontroller performance. With existing chip sets for TVs,

such a Quality of Service (QoS) feature is unknown to the manufacturer and the consumer.

It is evident that the chip-set can also be used in other consumer products than TV sets, such as the display signal part of a set-top box. Generally, this device features MPEG decoding, electronic programming guide, and interactivity via telephone for retrieval of a descrambling key. Furthermore, the chip-set could be used as a unified display driver which converts standard-definition signals or VGA-resolution video to any arbitrary format required for various display types, e.g. Cathode-Ray-Tubes (CRTs), computer monitors, plasma/Plasma-Addressed-Liquid-Crystal (PALC) displays or LCD displays. It can be concluded that the programmable concept behind the applications and the cost-efficient and modular architecture of these ICs give a high degree of applicability for various systems in the consumer market.

4.8 Conclusions

In this chapter we have studied an experimental system architecture offering new ways of video processing in terms of flexibility and programmability. The chapter showed a overview of processors and an analysis of video functions that are typical for television systems. We conclude the architectural discussion on two different levels. In the first subsection, we conclude on the features and merits of the actual proposal. In the second subsection, we look back and comment on our system and criticize it in hindsight in a way similar to the discussion about media processors. The architectures as presented in the past chapters are not in the actual figures, because implementations grow old quickly in a fast moving consumer electronics world. The value is much more in the thinking about architectures and the aspects of processing in tasks and control together with their memory usage.

4.8.1 Concluding remarks about the proposal

This chapter has presented a chip-set for high-end TV or set-top box, consisting of a microcontroller with a plurality of extensions and a video coprocessor array.

General-purpose computing
The microcontroller consists of a RISC core with a number of peripherals with the following functions: set control, Teletext and graphics, modem and low-speed data communication. Further peripherals include the usual

interfaces such as a serial UART interface and the I2C interface used for chip control. The architecture of the control chip (referred to as TCP) is a classical computing architecture with caches and a single bus for communication. Since graphics are programmable, a peripheral was defined for this purpose to support the RISC engine. The controller contains a special bus that forms the interface to the second chip for video processing. This interface can also be connected directly to an SDRAM and therefore it represents a memory bus.

Video processing
The video coprocessor array (referred to as CPA) represents a video processing architecture with high parallelism, based on a programmable array of coprocessors and a communication network. Function-specific hardware coprocessors perform pixel-based processing and are programmable at functional and system level. A switch matrix enables parallel processing of tasks and enables high communication bandwidth. Since the matrix is programmable, the order of signal processing tasks can be modified (programmable task graph) to realize new TV applications. Individual coprocessor processing power and performance can be adapted and optimized for each TV application task graph. The individual coprocessors are able to process several video signals simultaneously and contain a control unit to autonomously load the coprocessor settings at the correct locations in the video streams. This enables programming on a high system level. The coprocessor array contains functional units for horizontal and vertical scaling, sharpness enhancement, motion-adaptive temporal noise reduction, graphics blending, mixing of video streams and 100-Hz up-conversion. Due to the programmable registers of the coprocessors and the programmability of the task graph, the application range not only covers the intrinsic integrated functions [11], but also aspect-ratio conversions, PiP, a mosaic screen, pixel-based 2-D graphics for PC-like applications, etc. In fact, the video processing chip combined with the microcontroller chip enables all these features in a multi-window TV environment with high quality. As an example, Figure 4.23 shows the resulting displayed picture of the the TV application corresponding to the task graph in Figure 4.1.

Combined autonomous subsystems
The implementation was based on two chips that may be used in standalone operation in combination with existing off-the-shelf external SDRAM. Furthermore, the combination of the two chips results in a highly versatile package of TV applications in which video processing quality and external interactive data communication can be interchanged. User-defined inter-

Figure 4.23: *Example of a multi-window application.*

faces regarding the use of several video windows combined with the display of side information can be optimized with respect to optimal quality in all circumstances.

Programmability and reusability
The proposed chip-set enhances both quality and programmability of existing TV-sets and it may be used in set-top boxes as well. Due to the ever increasing transistor density of single chips, modules can be integrated or reused in new technology generations. The programmable communication structure allows a very broad range of TV function combinations and applications.

Memory
The architecture uses an external background memory which combines all large memory functions. This memory is shared with a microcontroller. Each coprocessor has some local video memory, in correspondence with the function carried out, to avoid severe memory-bandwidth problems. The memory bandwidth was discussed for several example applications.

Extensibility
The modularity of the architecture concept allows other advanced functions to be added easily in the form of new application-specific processors, e.g. MPEG decoding and 3-D graphics rendering. For consumer systems, it is possible that the architecture can be used for a number of years to come, due to its cost-efficiency. As technology progresses and the demand for programmability increases (e.g. MPEG-4, MPEG-7), the low-complexity and

irregular processing functions will be implemented in software on advanced microcontrollers or media processors, so that corresponding coprocessors will disappear. Combining the proposed platform with such generic processors is attractive, because a hardware-software tradeoff can be made. At the same time, it is expected that new computationally expensive functions, requiring specific hardware support, can be introduced.

Dynamic data processing
A novelty of the architecture is the use of Kahn process networks for video processing, in combination with application-specific processors. In the concept, processors perform independent processing and data streams are exchanged via small local buffers. These buffers can start and stop the processors dependent on whether a buffer is full or empty while additional communication rules avoid deadlock problems. The advantage of this system is that video processing functions can be handled as tasks of any length. Thus video frames of various sizes can be applied in the system without low-level reprogramming of the hardware. If a task is terminated, processor hardware becomes available for alternative tasks or otherwise it is stopped, thereby saving power consumption.

Predictable behavior
By means of the Kahn process network model and the FIFOs at the inputs and outputs of all coprocessors, the video processing (computation) and the communication are completely decoupled. Both the switch matrix and the off-chip memory interface provide a sufficient and guaranteed amount of communication resources which are statically assigned. Hence, it can be determined *a priori* whether a particular application can be executed with the available system resources. As a result, unpredictable behavior of the system is avoided under all circumstances, leading to robust operation of the system.

The TVP was proven in Silicon. Table 4.6 shows the characteristics of the CPA and TCP chips, while Figure 4.24 shows their layout.

4.8.2 Modifications for future systems

This section aims at compiling a number of aspects of the TVP system that need to be reconsideration when applied for future video-processing designs. Naturally, this presentation is performed with hindsight, as technology continuously moves forward and the latest architecture proposals in literature for video processing are being presented.

Table 4.6: *Chip-set characteristics of the TVP system.*

	Coprocessor array (CPA)	**Telecommunication and control processor (TCP)**
Process	CMOS 0.35 μm	CMOS 0.35 μm
Die size	150 mm2	80 mm2
Clock	64 MHz	48 MHz
Package	SBGA 352	SBGA 352
Dissipation	5 W	1.2 W
Transistors	7.1 M	2 M
Interfaces	JTAG, I2C, SDRAM 3×Input: YUV 4:2:2 (< 60 HMz) 2×Output: YUV 4:2:2/4:4:4 or RGB	JTAG, 2×UART, 2×I2C, SDRAM Serial Interconnect Bus (SIB) remote control, general I/O, software AD converter, GFX OUT: RGBα/FB, H/V sync.

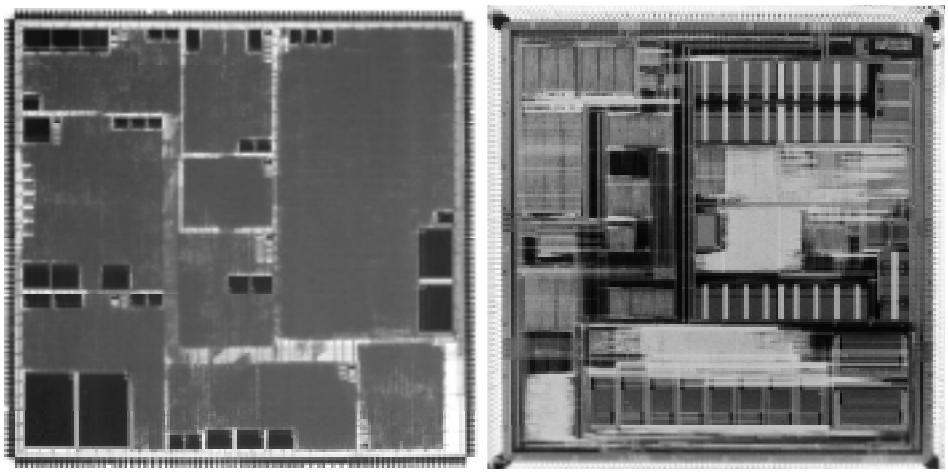


Figure 4.24: *Layout of the CPA (left) and the TCP (right) chips.*

Most of the critical points are well-considered choices that were made during the design of this particular system, performing its particular video-processing functions with a chip that is designed for a CMOS 0.35 μm process technology. Much more important than this, the purpose of presenting these remarks is to serve the learning process for future system design. This is also fully in line with Chapter 2 where we outlined the drawbacks of computing architectures as we developed the starting position for the design step made in this chapter. The most relevant modifications or extensions are listed below.

Communication and Interconnect
Though flexible with respect to programmability, the communication of the video processor system (CPA) has its limitations with respect to size of the interconnect and switching matrix. In the proposal, each coprocessor is connected to the other coprocessors giving fully programmable interconnections. But the number of connections grows quadratically with the number of coprocessors, so that the full switch matrix cannot be maintained in future generations. This holds in particular for the case that, due to the ever increasing integration density, the number of integrated processors will be in the order of hundred or more. To deal with this complexity, the communication network should become hierarchical. Thus clusters of coprocessors intensively communicate on a low level within a cluster, whereas the inter-cluster communication provided on a higher hierarchical level is limited. The implementation of such a hierarchical communication infrastructure is another concern, but to enable scalability towards more complex systems, the use of networks on a chip as formulated in Chapter 8 would be a logic step. In Chapter 7, where hierarchical communication is one of the central themes, busses are used for the implementation of the communication infrastructure.

Global external memory
The classical Von-Neumann bottleneck to the external memory has already been mentioned. The trend towards using large unified external memory in multimedia and video processing will proceed further in the coming years. As the number of integrated coprocessors further increases, the memory bandwidth bottleneck will become more strangling. Consequently, the system will need embedded memory. The most straightforward solution is to implement a memory caching hierarchy. However, a hierarchical communication infrastructure as suggested above, will lead to a physical distribution of the embedded memory which makes the architecture more complex.

Small local FIFOs
All coprocessors are separated by FIFO memories that decouple the computation and the communication and that may compensate for the dynamic behavior of each individual coprocessor. Because the video-processing functions of the system consist merely of regular processing, a relatively tight coupling between processing tasks is allowed, thereby reducing the required size of the communication FIFOs. Hence, these FIFOs have a size in the order of 32 Bytes. For systems with a more dynamic communication behavior (e.g. before or after a video scaling function), the data is conveyed

via relatively large buffers located in the external memory. However, since there is a clear trend towards more irregular and highly data-dependent processing, more decoupling of the processing tasks is preferred, thereby requiring larger communication buffers. Three options exist to provide a solution, depending on the requirements of the buffer. First, the size of the associated FIFOs at the input and/or output of the coprocessor could be increased. Secondly, if the size of the buffer resources are preferably allocated at run-time, an embedded memory unit would be connected to the switch matrix. Subsequently, inter-task communication could be streamed via the memory unit to provide the buffering. A third option is to implement an embedded cache between the off-chip memory and the switch matrix. Consequently, if large buffers are located in the off-chip address space and they would fit into the cache, data in the buffer is consumed before it is actually stored in off-chip memory (locality of data). Therefore, also for this third option the communication remains on-chip.

Lacking media processor
The programmability of the system is constrained. The external RISC-based controller can execute some non real-time tasks, that do not depend on the video processing. To increase the flexibility of the system, it would be desirable to enable partitioning of an application into hardware tasks on the coprocessors and multiple software tasks running on a programmable media processor. Although it is not inconceivable to provide an adequate solution, extensions in the architecture would be inevitable. First, a "standard" programmable media processor is not suitable for relatively small-grain multiple tasks that require communication with the hardware coprocessors at high synchronization rates via the small FIFO buffers. Secondly, the overhead of task switching depends on the state of the processor, thereby jeopardizing the predictability of the system. However, the embedding of a programmable core that is dedicated to a single task (i.e. not multitasking) is less problematic, since task switching is not required. Moreover, the operational clock of the CPU can be disabled and enabled to block and activate the processor, which can be used to implement a Kahn process network.

From the above-stated remarks we can conclude that the architectural concept has many attractive aspects. However, for future generations featuring more advanced video processing, some of the architectural elements need to be reconsidered. For example, the Kahn process network model proved to be very useful, but the switch matrix and its distributed communication buffers does not scale properly for a future system. Recently, options have

been found for a more hierarchical communication network that is scalable towards more complex future systems. This is combined with a hierarchy of distributed memories with a shared address space to dynamically allocate buffers, depending on the required throughput and buffer capacity. The architecture that is discussed in Chapter 7 partially addresses such an approach. Although it presents an implementation of a hierarchy with busses, also networks on a chip could be adopted to offer even more scalability.

Throughout the thesis, the communication aspects of the architecture appears to be one of the most important concerns and it has also a growing importance in general. This relates to many aspects, such as the network topology, the communication protocol and buffering. Particularly the memory is gradually becoming a dominant cost factor. Chapter 5 and 6 address various aspects for using memory efficiently in the video processing domain.

Qui non est hodie cras minus aptus erit
(Ovid, c.43 BC – c.17 AD)
He who is not prepared today
will be less so tomorrow

Chapter 5

Off-chip memory communication

FOR *video processing in systems-on-chip, it has become evident that off-chip memory communication is often a critical part for system performance and cost. Although the performance of general-purpose CPUs increases with 60 % every year, the bandwidth to the external memory increases with only 20 %. Because the desire for a shared centralized memory and flexible inter-processor communication further increases this bottleneck, Chapter 5 is devoted to application-specific memory usage and the bandwidth of data communication. It shows that significant gains can be achieved when matching the application-specific memory accesses with the behavior of the memory device. This substantial performance improvement is realized by introducing a new memory-communication model that incorporates the data-dependencies of the video-processing functions. The model is applicable to general multimedia processing, but this chapter concentrates in particular on an MPEG decoder as a realistic and worst-case example.*

5.1 Problem statement for off-chip memories

The previous chapters have shown how the requirements for future media processing systems in the digital TV domain lead to implementations with a large amount of parallelism, flexibility, and a high computation performance. It has also been indicated that flexibility is expressed in the design

of the communication infrastructure and significantly determines the system costs. Consequently, if a large amount of flexibility is required, the communication network may become the bottleneck in the design. Particularly the communication via off-chip memory is an important point of attention. An increasing amount of off-chip memory resources, such as memory bandwidth and memory capacity, results in wider data and address buses. This directly translates into more pins for the media processing chip, thus more power dissipation, a more expensive chip package, a larger printed-circuit-board (PCB) footprint, and a more complex PCB design with more layers. Due to the growing demand for more off-chip memory resources, this point in the communication infrastructure increasingly impacts the system costs. This system bottleneck was already recognized at an earlier stage in the design of general-purpose computing systems. Moore [69] predicted that the transistor density of systems on chip would double every 18 months. This has become particularly noticeable in the computer market where the performance of the CPU has increased proportionally with the number of integrated transistors on chip. Hennessy & Patterson [70] showed the performance increase of CPUs by measuring the performance of microprocessors that were developed in the last decades. The performance is defined as the time that is necessary to execute a well-defined benchmark set [71]. Unfortunately, the performance of off-chip memory communication has not evolved at the same speed as the CPU performance. For example, Figure 5.1 [70] showns that the CPU performance increases 60 % per year, whereas external memory bandwidth improves only with 20 % and latency is improved (reduced) with only 7 %. Concluding, there is an increasing gap between computational power and off-chip memory bandwidth.

For media processing in consumer systems, this performance gap between computation power and off-chip memory resources is also noticeable. However, the latency problem can be solved by using efficient pipelining and prefetching techniques [72]. This is possible due to the high level of *spatial locality*, meaning that references of data in the memory result with a high probability in subsequent references of data elements at neighboring memory addresses.

Because off-chip memory bandwidth is the most critical part of the communication infrastructure, this chapter will focus on reducing the off-chip memory bandwidth. Current data mapping into the memory is performed without application knowledge of the data. We have found that this leads to substantial inefficiencies and waste of scarce memory bandwidth. One of the main new elements in the chapter is that knowledge of the media

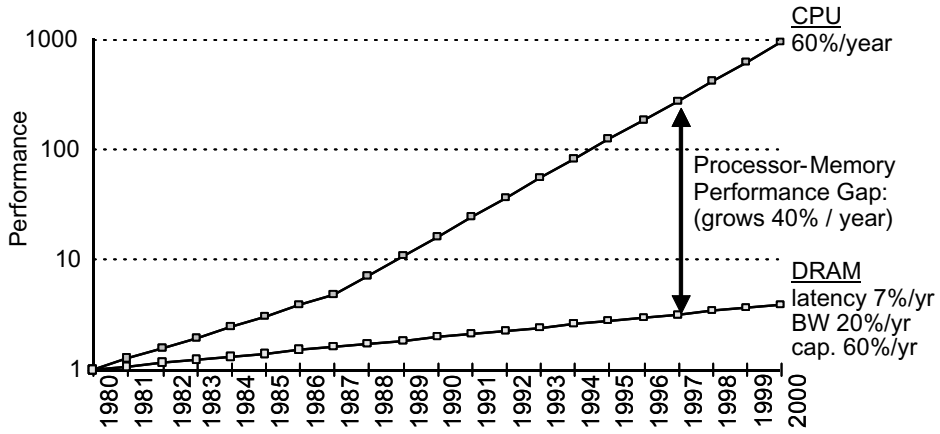


Figure 5.1: *The performance increase of external memory compared to the performance increase of CPUs.*

application is exploited to reduce the memory bandwidth. The reduction of the bandwidth is used to improve the performance of the architecture as a whole, because more video processing parallelism is enabled with the same architecture. In order to optimize the bandwidth usage of the memory, we have developed a practically feasible model of the memory read and write access procedure. In this model, we have included statistics of the data requests, such as the data-block sizes, their locations in the memory, etc. These statistics enable us to optimize the mapping of data onto the address space of the memory, so that the data bandwidth for memory access is minimized. We will show that the potential improvement is in the order of 30-40 %, which is considerable.

The chapter is subdivided as follows. First an outline of the memory technology, used for the main-stream off-chip memory devices, is given in Section 5.2, including a survey of promising new memory technologies. Subsequently, Section 5.3 discusses an MPEG decoder example for optimizing the mapping of pictorial data onto the physical memory space, thereby improving the bandwidth efficiency. Another dimension for optimization is to modify the video-processing algorithm or the memory architecture. Because we want to limit the scope of this thesis and because this other dimension for optimization is very application specific, it is not discussed in this chapter. However, the interested reader is referred to Appendix B, where an example is shown for an MPEG decoder. Moreover, Appendix C presents a technique to improve the efficiency by changing the memory interface.

5.2 Memory technology

Because memory functions determine a significant part of the system costs, the memory bandwidth and capacity are important optimization parameters in the system design. A system architecture with distributed off-chip memory such as shown in Figure 1.3 is not acceptable in future products. Such an approach requires more pins for the System-on-chip (SoC), thus increases costs. In distributed off-chip memory architectures, a more complex communication network is necessary to enable communication between processors and the multiple memory interfaces. To create a cost-efficient solution, one uniform off-chip memory, based on a standard off-the-shelf RAM is more beneficial. For a good understanding of memory size and bandwidth requirement let us first consider applied memory technologies in more detail.

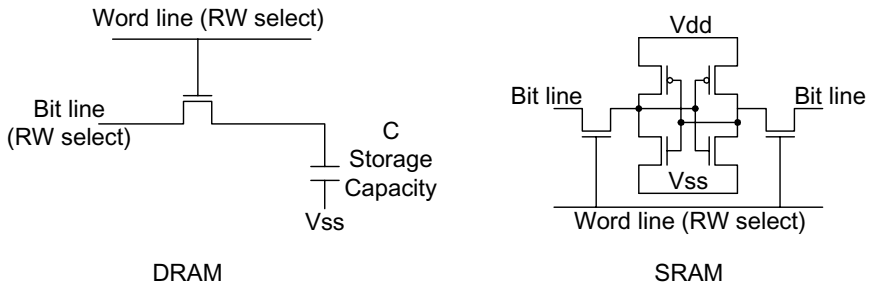


Figure 5.2: *Memory cell of a DRAM and an SRAM.*

With respect to the principle of a memory cell, we can distinguish static and dynamic random access memories (RAM). The memory cell of static RAM contains six transistors representing a latch. The dynamic RAM contains only one transistor and is therefore much smaller. However, it is based on the charging of a capacity. Because a capacity always has leaking currents, it has to be refreshed frequently. A small capacity can be charged and discharged much faster than a larger capacity, thus smaller memory structures obtain a higher speed. However, due to the leakage, a small capacity has to be refreshed more frequently. Consequently, the capacities have to comply a minimal size, thereby making it impossible to make the DRAM as fast as an SRAM. Because current processor systems require a large amount of memory, DRAM is generally used for external background memory. For high-speed embedded memories and caches, the faster SRAM are used. The conventional DRAM is asynchronous, meaning that the input and output signals are not gated with a clock. With synchronous DRAM (SDRAM),

all the interface signals are gated with a clock to optimize the timing for high bandwidth access to a large memory space. A similar but specialized version is the synchronous graphics RAM (SGRAM), which has a standard 32-bit interface, contains various special access features, and is used for video memory. However, this type of memory does not represent the main-stream market and is therefore more expensive. The increasing demand for more memory bandwidth requires the use of more sophisticated memory devices such as double-data-rate synchronous DRAM (DDR SDRAM) or Direct Rambus DRAMs (RDRAM). Because DDR SDRAM currently represent the lowest price per byte, which is an important parameter in the high-volume consumer products, it dominates the main-stream market for the coming years. The main addition of DDR SDRAM with respect to regular SDRAM is the transfer of data on both the rising and falling edge of the clock signal. Hence, the data throughput is doubled.

5.2.1 Prospects of emerging RAMs

The continuing memory-technology improvements has led to VLSI systems comprising ever growing memory capacities with smaller memory cells, higher speed, and greater ease of use. Hence, the improvements provide great benefits for system designs. The standard DRAM has increased its memory-chip capacity to 1 Gbit through the use of high-density 1-T cells (one-transistor one-capacitor cells). On the other hand, SRAM has achieved an extremely high speed for memories with a smaller capacity. However, apart from developments of these type of memories, breakthroughs in other types of memory technology have been established. For example, the market for flash memories has expanded because of their non-volatility feature, even though they contain a relatively small memory-chip capacity. Another important technology innovation is the use of embedded DRAM for a system-on-chip. Although these DRAMS have occupy an larger silicon area compared to standard DRAM, the speed has significantly improved and approaches the performance of SRAM. Table 5.1 shows how embedded DRAM trades-off the speed of SRAM and the capacity of standard DRAM memories [73].

Even more revolutionary type of memories such as ferro-electric RAM (FeRAM) and magnetic RAM (MRAM) are reported. Although their performance is promising, these new technologies are still in their infancy. They include all the advantages of the existing memories such as speed and capacity and in addition feature a nonvolatile behavior. Consequently, they could replace the use of DRAM, SRAM and flash memory by one MRAM memory device. This could potentially simplify the memory hier-

Table 5.1: *Comparison between standard DRAM, SRAM, and embedded DRAM using similar process technology.*

	Standard DRAM	**SRAM**	**Embedded DRAM**
Feature size	0.18 μm	0.175 μm	0.175 μm
Area / 8 MByte	50.8 mm^2	3 mm^2	19.3 mm^2
Throughput / data line	208 MByte/s	50 MByte/s	128 MByte/s

archy in systems-on-chip and could reduce the interface for programming the on-chip flash memory. Because these MRAM devices are just evolving, the remainder of this section will be limited to the mature DRAM-based memory devices.

5.2.2 Functioning of SDRAM-based memories

Dynamic memory cells consist of one transistor and one capacitor which can be charged or discharged, depending on the binary value it should represent. These DRAM cells are contained in an array of column address lines and row address lines, which can be separately addressed. Moreover, the memory device consists of several of such arrays, called memory banks, and can be addressed with separate bank address lines (input pins BA0 and BA1). With the Row Address Strobe (RAS) line, the Column Address Strobe (CAS), the Chip Select (CS), and the Write Enable (WE), commands can be issued like select row, read from a column address, write to a column address, etc. Figure 5.3 shows the block diagram of the DDR SDRAM architecture.

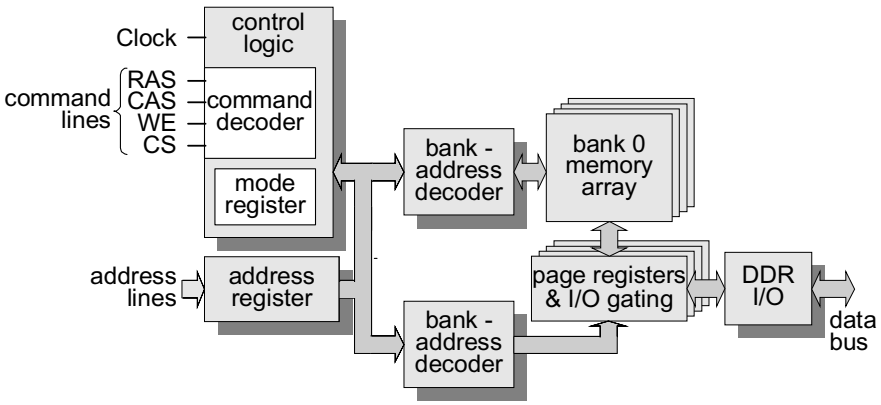


Figure 5.3: *Block diagram of a DDR SDRAM.*

DRAM-based memories provide a burst-access mode, enabling access to a number of consecutive data words by giving a single read or write command. Because the reading of dynamic memory cells is destructive, the content in a row of cells in the memory bank is copied into a row of static memory cells (the page registers). Subsequently, read and write access to this copied row is provided. The result after the required accesses in the row has to be copied back into the (destructed) dynamic cells, before a new row in the memory bank can be accessed. These actions of copying data into the page registers and back, are referred to as row-activation and precharging, respectively. During the copying, which takes considerable time, the associated memory bank cannot be accessed for data transfer. To prevent the loss of precious bandwidth, a multiple-bank architecture is used, where each bank can be accessed alternatingly. Hence, a bank can be accessed while other banks are activated or precharged. Furthermore, high throughput is achieved by dividing the device into stages using pipelining (at the expense of increased latency). When a memory row is activated, random access of the columns within the page registers can be performed. In each bank, only one row can be active simultaneously, but during the random access of the page registers, switching between banks is allowed without a penalty. Therefore, with a four bank device, four rows (one in each bank) can be addressed randomly.

Another important operational aspect of DRAM memories is the burst-oriented access to obtain a high bandwidth performance. This means that a number of consecutive data words are transferred to or from the memory by giving only one `read` or `write` command. Note however, that several commands are necessary to precharge a bank, to activate a row, and finally to issue a `read` or `write` command, i.e. three command-input cycles. Note furthermore that the data rate at the output is higher (DDR) than the rate at the input (command rate). Therefore, to exploit the available data bandwidth, the read and write accesses have to be burst oriented. Typically, the burst length used for DDR SDRAM devices is eight data words.

In the following section it will be shown how both the organization into memory banks and the burst-oriented behavior lead to efficiency concerns in the memory usage. Without going into more details it is concluded that the memory-bank architecture, requires a bank-interleaved access pattern for optimal efficiency of the throughput. Moreover, Section 5.3 shows how the burst-oriented access results in considerable transfer overhead. More details on the motivation and reasoning behind these access-dependent performance issues are elaborated in Appendix A.

5.3 Video-data storage in SDRAM memory

The previous section briefly outlined the architecture and operation of the SDRAM-based memory devices. It presented the internal organization into rows, columns and memory banks, and explained the burst-oriented accesses for high data throughput. Let us now take one step back and approach the memory usage from the application point of view, starting with a definition of the problem.

The architecture of present advanced video processing systems in consumer electronics generally contain various processor modules that communicate with an off-chip SDRAM memory (see Figure 5.4). For example an MPEG decoder requires a background memory to store reference pictures for prediction of successive video frames. When a large variety of processors desire communication with a standard off-chip memory configuration, a communication bottleneck will be materialized.

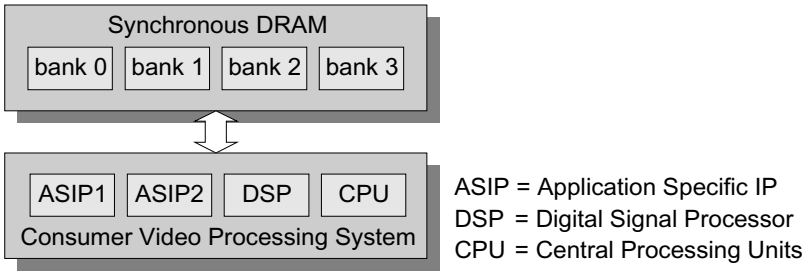


Figure 5.4: *Consumer system architecture.*

Let us focus on the bandwidth problem. In recently developed systems, this problem was solved by communicating to several memory devices in parallel. Currently, the smallest available double-data-rate synchronous DRAM (DDR SDRAM) has a 64-Mbit capacity with a 16-bit data bus or smaller, providing a peak bandwidth of 0.53 GB/s [74]. However, significantly more bandwidth is required for media applications, such as simultaneous High-Definition MPEG decoding, 3-D graphics rendering and field-rate conversion. The Imagine processor [72] features four memory controllers with a 32-bit data bus each. The Emotion Engine processor [37] contains a dual 16-bit memory bus at 800 MHz, providing 3.2 GB/s with 32 MByte in Direct Rambus DRAMs (RDRAM). However, this solution of parallel memory devices introduces more memory capacity than required by the system, leading to a lower cost efficiency. For the above-mentioned

systems, 256 Mbit of SDRAM memory is the minimal available capacity which is more than required by most consumer systems. In this section we focus on the reduction of the memory bandwidth, thereby contributing to a reduction of parallel memory devices. This is important because the use of parallel memory controllers leads to high systems costs such as, increased power dissipation, substantially larger silicon areas, and more expensive chip packages.

In the presented study we will concentrate on MPEG decoding as a pilot application. This application features block-based video processing and memory access. Such memory access to optimize bandwidth was already addressed in [75], where a mapping of fixed-size video data units into the memory is proposed. That work is related to analyzing the application software model only, without considering *data dependencies* such as the set of requested data-block memory transfers including their probability of occurrence. In this section, we determine an optimal mapping of the video into the memory by analyzing the actual memory accesses, so that data dependencies are taken into account. To design a memory system that can be reused for future memory technologies, we assume that a `burst stop` command is not available and burst interruption is not allowed. For the next generation DDR-2 SDRAM, these burst-interruption features are not provided to improve its potential speed performance.

5.3.1 The concept of data units

The increasing demand for more memory bandwidth requires the use of sophisticated memory devices like DDR SDRAM or RDRAM. To obtain high performance, these devices use two main features: the burst-access mode and the multiple-bank architecture. The burst-access mode enables access to a number of consecutive data words by giving a single read or write command. The multi-bank architecture provides the means to hide the time that is necessary to activate a row and to precharge it when all read and write actions in that row are performed. Let us now concentrate on the consequences of the burst-access mode, using the previously described SDRAM architecture. To optimize the utilization of the memory-bus bandwidth, data can only be accessed at the grain size of a data burst (e.g. eight words). As mentioned before, we assume that burst interruption is not allowed. If the memory configuration provides a 64-bit bus and is programmed for a burst length of eight words, one data burst contains 8×64 bits = 64 bytes of data. These data bursts represent non-overlapping blocks in the memory which can only be accessed as an entity. In the remainder of this chapter, these blocks are referred to as *data units*. Figure 5.5 shows

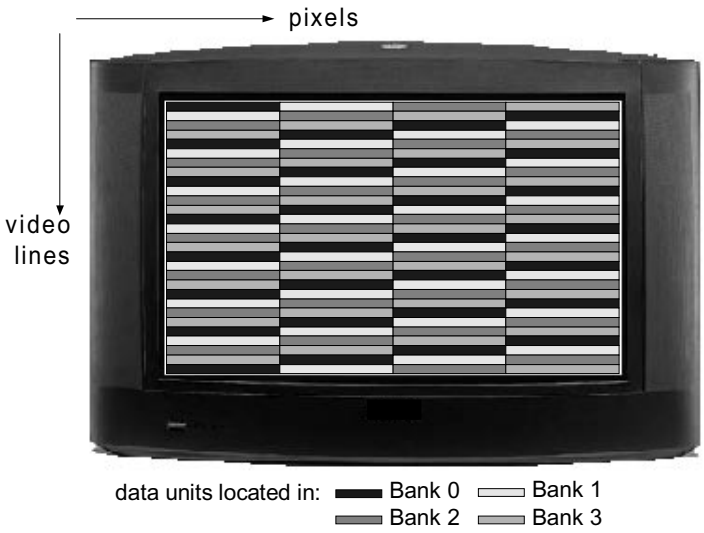


Figure 5.5: *Video pixels mapped onto data units, each located in a particular memory bank.*

an example how the pixels of a video picture are mapped onto data units in the memory. Each rectangle in the picture represents one data unit. A required group of pixels for processing may be partly located in several data units and requires the transfer of all corresponding data units. Hence, significantly more pixels than required are transferred. For example, Figure 5.6 indicates the transfer overhead for memory access to a macroblock in MPEG. In the sequel we call these extra pixels *pixel overhead*. This overhead becomes particularly noticeable if the size of the data units is relatively large compared to the requested group of pixels.

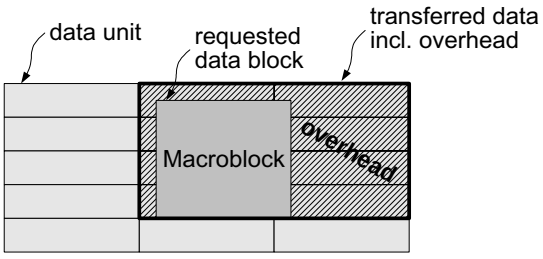


Figure 5.6: *Memory access of a macroblock including the transfer overhead.*

This section describes the partitioning of data units into memory banks and determines the optimal dimensions of the data units to minimize the pixel overhead, thereby increasing the bandwidth efficiency. The optimization includes statistical analysis of the data to be accessed. A mapping of video data units into the memory was already proposed in [75]. However, this paper proposes to analyze the video-processing algorithm only without considering data dependencies such as the set of requested data blocks including their probability of occurrence. For example, the type of data blocks that are fetched for motion compensation in an MPEG decoder, strongly depends on the motion-estimation strategy applied by the encoder. In this thesis, we determine an optimal mapping of the video into the memory by measuring and analyzing the actual memory accesses, so that data dependencies are taken into account. Another consideration that is important for bandwidth efficiency is the organization into memory banks, which is provided in all modern memory devices. It will become clear that both aspects contribute to a substantial improvement of the available memory bandwidth. Given the earlier-explained memory bottleneck, this leads to an overall performance improvement of the architecture.

Determine the optimal data-unit size

Let us now determine the best choice for the size of the data unit from the memory properties as mentioned in the previous section. Among other factors, it depends on the size of the burst length. To minimize the pixel overhead, the data units are preferred to be small. However, if the burst length is too small, the time that elapses after accessing all four banks does not exceed the minimal required row cycle time t_{RC} (see Appendix A) and causes some waiting cycles in which no data is transferred over the bus. Apparently there is a tradeoff between bus utilization and pixel overhead. To determine the minimal burst length BL for which a full utilization of the bus bandwidth can be achieved, the number of data words transferred within $2 \times t_{RC}$ (twice the cycles due to DDR output) is divided by the number of banks, thus:

$$BL \geq 20/4 \tag{5.1}$$

Since the burst length is a multiple of two, due to the double data rate and because the burst length can only be programmed for the values 2, 4 or 8, the burst length is set to $BL = 8$. Because this value is larger than the determined lower bound of 5 (see Equation (5.1)), it is not required to interleave all four banks before the first bank is accessed again. Note that access to three successive banks occupies 3×8 cycles, thereby already exceeding t_{RC}.

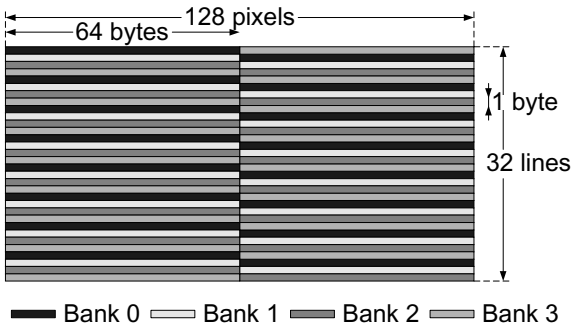


Figure 5.7: *Mapping of* 64×1 *adjacent pixels onto data units.*

5.3.2 The mapping of pixels into the memory

Let us discuss a few examples for data unit dimensions and the corresponding pixel overhead. For this purpose, we assume a system that requires a 64-bit bus SDRAM configuration to provide sufficient bandwidth. Consequently, the data units in the memory contain 64 bytes. For the mapping of pixels, several options can be recognized. The most straightforward way is to map 64 successive pixels of a video line into one data unit as depicted in Figure 5.7. The figure shows how each consecutive block of 64 pixels is interleaved in the banks in both horizontal and vertical direction. If for such interleaved mapping the pixel data is sequentially read or written (given by the application), the memory banks are accessed alternately. However, when a data block of 16×16 pixels is requested from the memory, the amount of data that needs to be transferred is much larger. If the data block is horizontally positioned within one data unit, 64×16 pixels are transferred. If the data block overlays two data units in horizontal direction, the amount of transferred data is 128×16, resulting in an excessive 700 % pixel overhead. Figure 5.8 shows a much more appropriate mapping of pixels onto the memory for this data-block request. Blocks of 16 pixels from 4 video lines are stored in a single data unit, resulting in less pixel overhead when accessing a data block of 16×16 pixels. However, when a data block of 128×1 is requested, Figure 5.7 provides a better mapping strategy.

Let us now discuss the effect of storing interlaced video frames. For several applications in a multi-media system, it is necessary to read the video data both progressively and interlaced, e.g. for frame prediction and field prediction in MPEG decoding. However, when subsequent odd and even lines are mapped onto the same data unit, it is not possible to access only odd

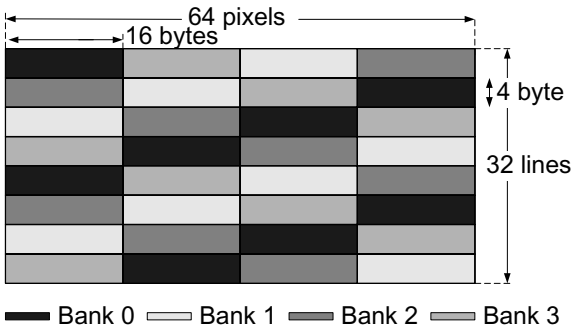


Figure 5.8: *Mapping of* 16×4 *adjacent pixels onto data units.*

or even lines without wasting memory bandwidth. Therefore, the odd and even lines are positioned in different banks of the memory. As a result, the data units are interleaved in the memory when the vertical size is larger than one. The resulting mapping strategy for data units of 16×4 pixels is shown in Figure 5.9. For this mapping, the 16×4 pixels are not adjacent.

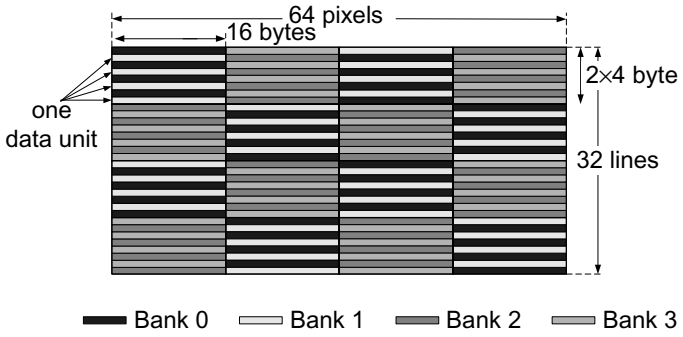


Figure 5.9: *Mapping of interlaced video onto memory data units.*

Four line pieces of 16×1 which are interlaced in the video frame are mapped onto one data unit. Note that for retrieval of data blocks with *progressive* video lines, the size of the smallest data packet to be accessed as an entity has become eight lines high (two vertically adjacent data units), whereas for access to data blocks with *interlaced* video, the size is four lines (one data unit).

For efficient access of interlaced video data, the mapping of odd and even video lines into odd and even banks is toggled after four units in vertical direction (a reversal of the bank parity when looking to the global checkerboard pattern in Figure 5.9). In the first group of 16 video lines, the odd

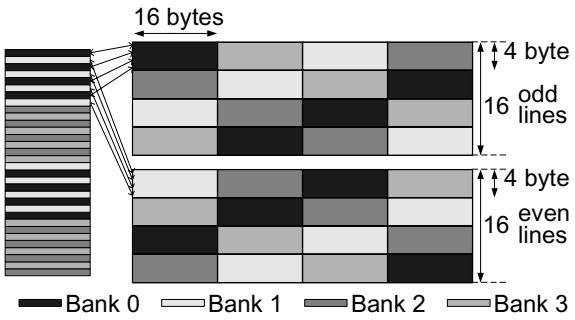


Figure 5.10: *Decomposition into mappings for separate video fields.*

lines are mapped onto bank 0 and 2, while the even lines are mapped onto bank 1 and 3. In the following 16 video lines (two checkerboard blocks lower), the odd lines are mapped onto bank 1 and 3 and the even lines are mapped onto bank 0 and 2. For progressive video this gives only a minor difference, but for interlaced video, this results in addressing of all banks instead of only odd or even banks. This is shown in Figure 5.10 where the mapping of Figure 5.9 is decomposed into separate video fields. The left part of the figure shows only one column of data units from Figure 5.9.

Concluding all above-mentioned system aspects from the previous examples in this section, the optimal mapping strategy depends on the following parameters (see Figure 5.11 for the definition of the size parameters).

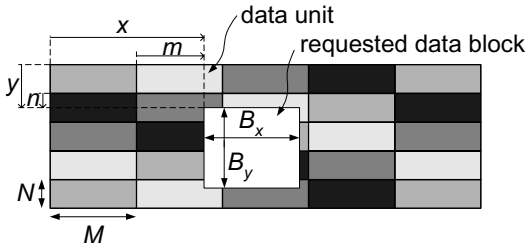


Figure 5.11: *Definition of the size parameters.*

- *The dimensions of the requested data blocks,* $B_x \times B_y$. MPEG-2 decoding contains a large variety of different data-block accesses: due to interlaced and progressive video, field and frame prediction, luminance and chrominance data and due to the sub-pixel accurate motion compensation (all these processing issues are addressed in the MPEG standard).

- *The interlace factor of the requested data blocks.*
  Progressive data blocks require accesses in pairs of data units in vertical direction. Consequently, the smallest data entity to access is two data units. Hence, the calculations are slightly different for progressive and interlaced video.

- *The probability of the data-block occurrence, $P(B_x \times B_y)$.*
  For example, if only 16×1 data blocks are accessed (100 % probability), the optimal data-unit dimension will also be very much horizontally oriented. Obviously, the probability of each data-block type depends very much on the application. Moreover, it depends on the implementation. For example, if the color components in an MPEG decoder are multiplexed before storage in the reference memory, some data-block types for chrominance and luminance are equal, thereby increasing their probability.

- *The probability function of data-block positions, $P_{B_x \times B_y}(m, n)$.*
  For this function, the position parameters $m = x \bmod M$ and $n = y \bmod N$, where M is the horizontal data-unit size and N the vertical data-unit size. Thus (x, y) are the global coordinates of the requested data block, and (m, n) denote the local coordinates within the corresponding data unit. If a requested data block is aligned with the boundaries of the data units, it overlays the minimum amount of data units, resulting in the minimum pixel overhead. Data blocks that overlay many data units cause significant pixel overhead. Note that the 16×16 macroblock grid for MPEG and the high probability of the zero-motion vectors have a positive influence on reducing the pixel overhead.

The last two parameters indicate that the statistics of the memory access are considered, because all requested data blocks (e.g. location and usage frequency) are retrieved with varying probability. The probability function introduced in this section will be inserted into an architecture model which is discussed in the next subsection.

5.3.3 Architecture model for simulation

To measure the statistics as indicated in the previous subsection, an MPEG-2 decoder was modelled including the main-memory interface that transfers data to and from the SDRAM memory when requested by the decoder. The interface between the MPEG-2 decoder and the memory interface is defined as follows:

```
void transfer(
  boolean       read,   // a read (TRUE) or write transfer
  integer       Bx,     // horizontal data-block size
  integer       By,     // vertical data-block size
  boolean       interl, // interlaced (TRUE) or progressive
  integer       x,      // horizontal data-block position
  integer       y,      // vertical data-block position
  integer       line,   // horizontal size of a video line
  u char        *mem,   // pointer to the memory
  u char        *buf)   // pointer to the read/write buffer
```

The implementation of the interface translates the input parameters to a set of corresponding data-unit addresses in the memory. Depending on the state of the memory banks, arbitration is provided between read and write requests from the MC unit, and the Video Output unit (VO) that displays the data. Subsequently, the memory interface translates all data-unit requests to memory commands and schedules the memory commands to satisfy all timing parameters for an optimal data-bus utilization. In addition, the memory interface generates function calls to the communication analyzer. The communication analyzer as depicted in Figure 5.12 analyzes the requests and updates the statistics which were mentioned in the previous section into a database. After decoding of a representative set of bitstreams, the optimal data-unit dimensions can be calculated from the statistics in the database.

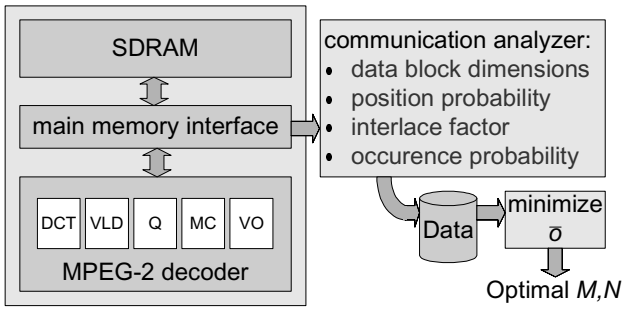


Figure 5.12: *Multimedia architecture model including an MPEG decoder for simulation.*

The optimal data-unit dimensions are calculated by minimizing the pixel overhead as function of the data-unit dimensions. The pixel overhead $\bar{o}_i$

for interlaced data-block requests is calculated as:

$$\bar{o}_i(M,N,V_i) = \frac{\sum\limits_{B_x \times B_y \in V_i} P(B_x \times B_y) H(M,N,V_i)}{\sum\limits_{B_x \times B_y \in V_i} P(B_x \times B_y) \cdot B_x \cdot B_y} - 1, \tag{5.2}$$

with

$$H(M,N,V_i) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} P_{B_x \times B_y}(m,n) \cdot M \cdot N \cdot \left(1 + \left\lfloor \frac{B_x + m - 1}{M} \right\rfloor\right) \cdot \left(1 + \left\lfloor \frac{B_y + n - 1}{N} \right\rfloor\right), \tag{5.3}$$

where V_i is the set of possible requested data blocks $B_x \times B_y$, $P(B_x \times B_y)$ the probability of the data block, M the horizontal size of the data unit and N the vertical size of the data unit (see Figure 5.11 for the definition of the parameters). The numerator in Equation (5.2) represents the amount of transferred data including the pixel overhead. The denominator represents the amount of requested data without the pixel overhead. Probability $P_{B_x \times B_y}(m,n)$ is equal to the probability that the upper left corner pixel of a requested data block $B_x \times B_y$ is positioned at any location (x,y) that satisfies the following condition: $x \bmod M = m$ AND $y \bmod N = n$.

For progressive data-block requests, the data has to be partitioned into two interlaced data-block requests. Therefore, the overhead calculation for progressive data-block requests is slightly different. V_i becomes V_p in Equation (5.2) and $H(M,N,V)$ in Equation (5.3) is defined as:

$$H(M,N,V_p) = \sum_{m=0}^{M-1} \sum_{n=0}^{2N-1} P_{B_x \times B_y}(m,n) \cdot M \cdot N \cdot \left(1 + \left\lfloor \tfrac{B_x+m-1}{M} \right\rfloor\right) \cdot \left(2 + \left\lfloor \frac{\lceil B_y/2 \rceil + \lfloor n/2 \rfloor - 1}{N} \right\rfloor + \left\lfloor \frac{\lfloor B_y/2 \rfloor + \lceil n/2 \rceil - 1}{N} \right\rfloor\right). \tag{5.4}$$

When a system that uses a combination of progressive and interlaced video has to be considered, the set of requested data blocks has to be separated into a set of interlaced data blocks and a set of progressive data blocks. Subsequently, Equation (5.2) has to be applied with both Equation (5.3) and (5.4). Thus

$$\bar{o}(M,N,V) = \bar{o}_i(M,N,V_i) + \bar{o}_p(M,N,V_p), \text{ with } V = V_i \cup V_o. \tag{5.5}$$

Note that Equation (5.5) is a non-weighted sum of averages, because each individual term covers only a part of the overall occurrence probabilities (thus already statistically weighted). For example, if the occurrence ratio between interlaced and progressive data-block requests is one over three, the value of $\bar{o}_i$ is only one quarter of the actual pixel overhead for interlaced data-block requests.

5.3.4 MPEG decoding as application example

As mentioned in the previous subsection, we modelled an MPEG-2 decoder as a pilot application. In our simulations, we consider the reading of data for motion-compensated prediction of macroblocks (MBs), the writing of reconstructed MBs and the reading of data for display.

Reading of prediction data

Let us consider the sets V_i and V_p that are used for prediction of the MBs.

$$\begin{aligned} V_p &= \{(16\times16),(17\times16),(16\times17),(17\times17),(16\times8),(18\times8),\\ &\quad (16\times9),(18\times9)\}\\ V_i &= \{(16\times16),(17\times16),(16\times17),(17\times17),(16\times8),(18\times8),\\ &\quad (16\times9),(18\times9),(17\times8),(17\times9),(16\times4),(18\times4),\\ &\quad (16\times5),(18\times5)\} \end{aligned}$$

The numbers 2^p+1 for the luminance data blocks originate from the desire to enable sub-pixel accurate motion compensation. For the chrominance, the components C_rC_b are multiplexed in the horizontal direction. Each odd sample is a value C_r, and each even sample is a value C_b. Therefore, the sub-pixel accurate motion compensation of chrominance data blocks may result in the numbers $2^p \pm 2$ for the horizontal direction. The probability function of the position of a requested block $B_x \times B_y$ that satisfies the condition $x \bmod M = m$ AND $y \bmod N = n$ was measured during decoding a representative test-set of MPEG-2 bit streams. Figure 5.13 and 5.14 show two examples of measured probability functions of the positions. Figure 5.13 shows high probabilities at the corner positions. At these positions, a progressive block of 17×17 is aligned with boundaries of the 8×8 data units and occurs when the block has a zero-motion vector (or half-pel). Apparently, zero or very low-speed motion macroblocks (MBs) have a high occurrence probability. If data blocks are aligned with the boundaries of the data units, the amount of pixel overhead is minimal. Consequently, the high probability of zero-motion has a positive effect on the transfer bandwidth. Figure 5.14 shows the position probabilities of an interlaced 18×4 block. From the zero probability of the odd horizontal po-

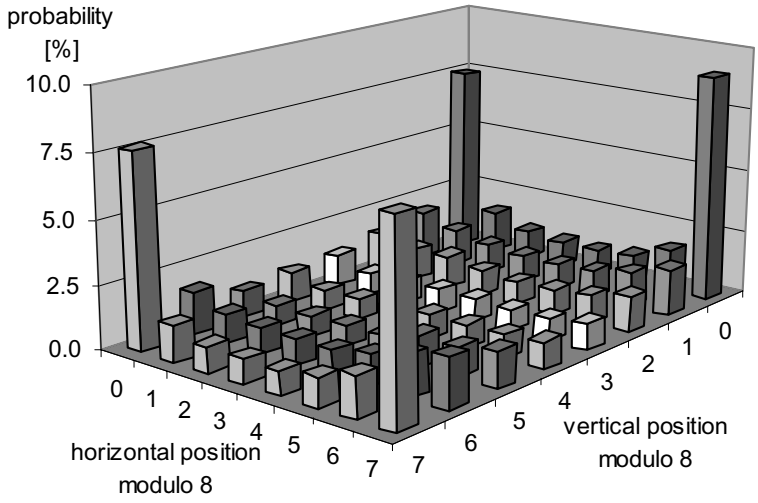


Figure 5.13: *Example of probability function for luminance of* $P_{17\times17}(n,m)$ *from set* V_p *with* $(M,N)=(8,8)$.

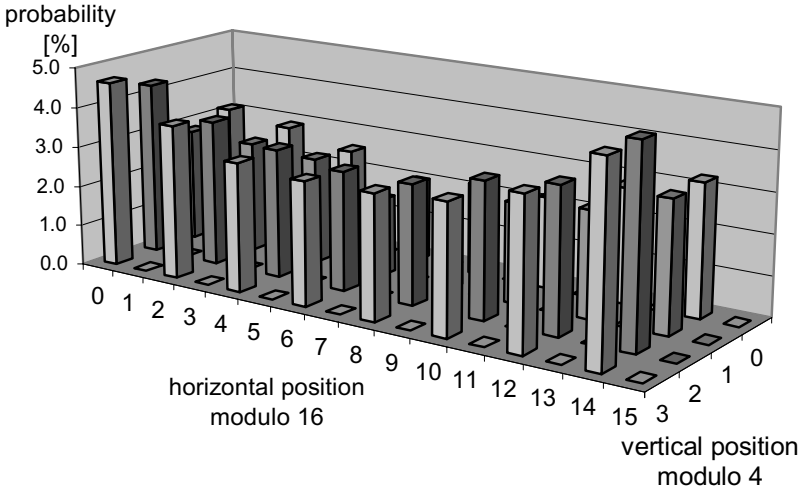


Figure 5.14: *Example of probability function for chrominance of* $P_{18\times4}(m,n)$ *from set* V_i *with* $(M,N)=(16,4)$.

sitions, it can be concluded that it concerns a chrominance block in which the C_r and C_b samples are multiplexed in the horizontal direction. Because the requested block contains interlaced video, the probabilities of the odd vertical positions are very similar to the probabilities of the even vertical positions.

Besides the probability function of the positions, also the probability of occurrence of all block types is measured (see Table 5.2). Note that the amount of block requests for luminance is equal to the amount of block requests for chrominance. Furthermore, the table shows that the blocks of $\{(16 \times 16), (17 \times 16), (16 \times 17), (17 \times 17)\}$ from set V_i are absent. This indicates that no field-based decoding is carried out by the MPEG decoder. Hence, only frame-based pictures are used. Since most commercially available MPEG encoders perform coding of frame-based pictures only, this is a realistic measurement. Because the motion vectors for the luminance and

Table 5.2: *Example probability of occurrence, $P(B_x \times B_y)$, of a set of data-block requests for motion-compensated prediction.*

Luminance frame predict.		**Luminance field predict.**		**Chrominance frame predict.**		**Chrominance field predict.**	
block type $\in V_p$	prob. [%]	block type $\in V_i$	prob. [%]	block type $\in V_p$	prob. [%]	block type $\in V_i$	prob. [%]
16×16	1.59	16×8	2.28	16×8	12.95	16×4	7.85
17×16	3.12	17×8	6.84	18×8	4.47	18×4	6.24
16×17	4.75	16×9	3.06	16×9	4.05	16×5	6.16
17×17	14.86	17×9	13.50	18×9	2.86	18×5	5.43
Total [%]	**24.32**		**25.68**		**24.32**		**25.68**

the chrominance in a MB are equal (apart from scaling), the probability of each chrominance block type can be computed from the probabilities of the luminance block types. However, this is not true for all applications. Hence, the occurrence probability of all block types was measured to generalize our optimization approach in this paper for all applications.

Writing of the reconstructed data

In an MPEG decoder application, the reconstructed pictures are written in memory for output display, or as reference pictures to predict new pictures using motion compensation. This writing is done on MB basis and consumes part of the memory bandwidth. Also for this kind of block-based access, the pixel-overhead considerations as discussed above are valid. However, the access for writing reconstructed pictures is very regular. The MBs of the pictures are written sequentially from left to right and from top to bottom at fixed grid positions of 16×16. Consequently, the probability function of the positions can be determined easily. Let us assume that the

16×16 grid of the MBs is always aligned to some boundaries of the $M \times N$ grid. With this restriction, the following probability function holds:

$$P_{(16\times16\in V_p)}(m,n) = \begin{cases} \frac{1}{\lceil\frac{M}{16}\rceil\cdot\lceil\frac{N}{16}\rceil} & ,m \bmod 16 = 0 \wedge n \bmod 16 = 0 \\ 0 & ,\text{elsewhere} \end{cases}, \quad (5.6)$$

with $m = x \bmod M$ AND $n = y \bmod N$. Because the bitstreams in the test set only contain frame pictures, the written MBs are only contained in the set V_p. Because the occurrence probability is a relative measure, it depends on the amount of data requests for the prediction. This is determined among other factors by the amount of field and frame predictions, the structure of the Group Of Pictures (GOP), the amount of forward, backward and bi-directional predicted MBs in a B-picture, etc. However, experiments have shown that the final results as presented in Section 5.3.5, are not very sensitive for minor differences of these individual aspects and are more or less similar for scenes that are encoded with any realistic encoder.

Reading of data for display

Besides the reading of prediction data and the writing of MBs, also reading of video data for display has to be taken into account. In contrast with the previous memory accesses, the reading of video data for display is performed line-wise instead of block-based. Conversion of the block-based data in memory into line-based data is another factor that influences the mapping strategy. To optimize the mapping strategy as a function of the pixel overhead calculated with Equations (5.2)-(5.5), the requests for display of the video have to be included into the data set. For the dimensions of the requested data for display, the following options are considered:

- reading of video lines by means of block transfers, thereby accepting a significant penalty in memory bandwidth;
- usage of embedded video-line memories in the architecture to convert data blocks into video lines.

For the first option, line-based requests are used with data blocks of size $M \times 1$ and are added to the set of data blocks. However, actually transferred are blocks of size $M \times N$. It is easy to derive that the pixel overhead for such transfers equals

$$o(M, 1, \{(M \times 1)\}) = \frac{M \times N - M \times 1}{M \times 1} = (N-1) \cdot 100\%. \quad (5.7)$$

The probability function of the position depends on the data-unit dimensions by:

$$P_{(M\times 1\in V_i)}(M,N) = \begin{cases} \frac{1}{N} & \text{for } m \bmod M = 0, \\ 0 & \text{elsewhere.} \end{cases} \tag{5.8}$$

Because the ratio between requests for writing the output MBs and reading for video display is fixed, the probability of occurrence for the line-based requests can be calculated as follows:

$$P(M \times 1 \in V_i) = \frac{16 \times 16}{M \times 1} \cdot P(16 \times 16 \in V_p). \tag{5.9}$$

When video-line memories are embedded, the size of the requested data blocks is $M \times N$ with the following probability function of the position:

$$P_{M\times N\in V_p}(m,n) = \begin{cases} 1 & \text{for } m \bmod M = 0 \bigwedge n \bmod N = 0, \\ 0 & \text{elsewhere,} \end{cases} \tag{5.10}$$

with $m = x \bmod M$ AND $n = y \bmod N$. The probability of occurrence is:

$$P(M \times N \in V_p) = \frac{16 \times 16}{M \times N} \cdot P(16 \times 16 \in V_p). \tag{5.11}$$

It is also possible to have a combination of the above-described options. For example, an MPEG decoder may use data units of 16×4 pixels instead of 32×2, thereby reducing the pixel overhead for block-based accesses. In this case, embedded video-line memories for $N = 2$ are used to convert the blocks into video lines. Consequently, the pixel overhead for reading video lines is not zero, but much smaller than the 300 % resulting from the case without video-line memories.

Overall occurrence probabilities

Table 5.3 shows the occurrence probability of each block type, considering all memory requests performed by the MPEG-2 decoder. The table results from using a decoder that performs line-based requests for the output display data and has a mapping of 16×4 pixels into data units. Note that the occurrence probability of the memory access for display is significant. Although it is much higher than the occurrence probability of the write requests for reconstructed MBs, the amount of data that is requested is equal. This is caused by the relation between the data-block size of $M \times 1 \in V_i$ for display of the video and $16 \times 16 \in V_p$ for writing the constructed MBs. Equation (5.9) shows that the size of the requested data-blocks times the occurrence probability is constant, thus:

$$M \times 1 \cdot P(M \times 1 \in V_i) = 16 \times 16 \cdot P(16 \times 16 \in V_p).$$

Table 5.3: *Example probability of occurrence, $P(B_x \times B_y)$, of a set of data-block requests for the complete MPEG decoding.*

prediction data-block requests			
block type	prob. [%]	block type	prob. [%]
$16 \times 16 \in V_p$	0.21	$16 \times 8 \in V_i$	0.30
$17 \times 16 \in V_p$	0.41	$17 \times 8 \in V_i$	0.89
$16 \times 17 \in V_p$	0.62	$16 \times 9 \in V_i$	0.40
$17 \times 17 \in V_p$	1.94	$17 \times 9 \in V_i$	1.76
$16 \times 8 \in V_p$	1.69	$16 \times 4 \in V_i$	1.02
$18 \times 8 \in V_p$	0.58	$18 \times 4 \in V_i$	0.80
$16 \times 9 \in V_p$	0.53	$16 \times 5 \in V_i$	0.81
$18 \times 9 \in V_p$	0.37	$18 \times 5 \in V_i$	0.71
write MB requests			
$16 \times 16 \in V_p$	5.12		
output read requests			
		$M \times 1 \in V_i$	81.85
Total [%]	**11.46**		**88.54**

It can be concluded that the pixel overhead is not merely determined by the occurrence probability of the data-block requests, but also by their size. However, the size may have an impact on the utilization of the data bus. As shown in Section 5.3, the scheduling of all memory commands is constrained by several timing parameters. Relatively small data-block requests (< 3 data units), will result in a decreased memory efficiency. Although memory command scheduling for small data block requests is beyond the scope of this paper, it can be concluded that a large amount of small data-block requests for display has a negative influence on the utilization of the memory bus.

5.3.5 Model simulation results

We have simulated the architecture of Figure 5.12 based on an SDRAM interface for determining an optimal mapping of the video into the memory with the objective to minimize the overall memory bandwidth. The mapping is optimized for reducing the transfer overhead by measuring and analyzing the actual memory accesses, so that data dependencies are taken into account. Another issue that is important for bandwidth efficiency is the organization into memory banks, which is provided in all modern memory devices. The proposed mapping strategy increases the memory efficiency, thereby contributing to a decreased memory-bandwidth requirement.

The experiments, which were conducted with a large test-set of bitstreams, were performed for architectures featuring a 32-bit and a 64-bit memory bus and for architectures with and without line-memories for conversion of block-based storage to line-based output. For each architecture configuration, the measured statistics were stored in a database for off-line calculation of the optimal data-unit dimensions. Subsequently, Equations (5.2)-(5.5) were applied to calculate the average pixel overhead for a given dimension (M, N) of the data units. The tables below show the simulated bandwidth numbers for various data-unit dimensions.

Table 5.4 shows the final bandwidth results for 32-Byte data units, where the requests for video display are line-based. The values in the tables are relative to the total data that was actually requested by the application excluding the transfer overhead. Therefore, 100 % equals 253 MByte/s for 25-Hz High-Definition video. From the table it can be concluded that the mapping of 32×1 results in the smallest pixel overhead. If the reading from memory for display of the video is $(M \times N)$-based, the optimal data-unit dimensions have a more vertical preference.

Table 5.4: *Bandwidth results for 32-Byte data units and line-based requests for output.*

data unit dimensions	**requested**[1] **data [%]**	**transferred**[1] **data [%]**
(32×1)	100	**100 + 58**
(16×2)	100	100 + 65
(8×4)	100	100 + 126
(4×8)	100	100 + 269

[1] 100 % equals 253 MByte/s for 25-Hz High-Definition video.

Table 5.5 shows the model simulation results for $(M \times N)$-requests for display as indicated in Equation 5.10 and 5.11. For this scenario, the 8×4 mapping outperforms the 32×1 mapping. Table 5.6 and 5.7 show the results for 64-Byte data units. For these systems, the usage of 32×2 and 16×4 pixels as data units provide the optimal solution using line-based and $(M \times N)$-based reading for display, respectively.

Apart for the optimal choices of block size, there is another important conclusion. All tables show that the penalty for applying arbitrary block sizes as is done in current practical systems is significant, considering that 100 %

Table 5.5: *Bandwidth results for 32-Byte data units and ($M \times N$)-based requests for output.*

data unit dimensions	requested data [%]	transferred data [%]
(32×1)	100	100 + 58
(16×2)	100	100 + 32
(8×4)	100	**100 + 27**
(4×8)	100	100 + 40

Table 5.6: *Bandwidth results for 64-Byte data units and line-based requests for output.*

data unit dimensions	requested data [%]	transferred data [%]
(64×1)	100	100 + 123
(32×2)	100	**100 + 102**
(16×4)	100	100 + 144
(8×8)	100	100 + 282

Table 5.7: *Bandwidth results for 64-Byte data units and ($M \times N$)-based requests for output.*

data unit dimensions	requested data [%]	transferred data [%]
(64×1)	100	100 + 123
(32×2)	100	100 + 69
(16×4)	100	**100 + 46**
(8×8)	100	100 + 53

represents a large value (e.g. 250 MByte/s) in bandwidth. A careful consideration of their efficiency is therefore no luxury. In recent proposals for multimedia computing architectures (e.g. [76][26][77]) video data is written line by line into the address space. This can be regarded as block-based data units with a vertical size of one ($N = 1$). For such systems, the results of the first row in the tables apply. Hence, the system with 64-Byte data units consumes a factor of 2.23 more memory bandwidth than requested. Note that we assume systems without burst interruption and access to memory

by means of 64-Byte data entities. The proposed mapping with the optimal data-unit dimension reduces the amount of memory bandwidth for such a system with 35 %. For systems with 32-Byte data units, the bandwidth reduces with 20 % (see Figure 5.15). For high-definition MPEG decoding, these numbers result in a reduction of 195 MByte/s and 80 MByte/s, respectively. This substantial performance improvement corresponds with a bandwidth magnitude of a complete function or application such as the display of a secondary standard-definition channel or the addition of an advanced 2-D graphics application. Moreover, the presented results can also be exploited to reduce the continuously growing gap between required computational power and memory bandwidth.

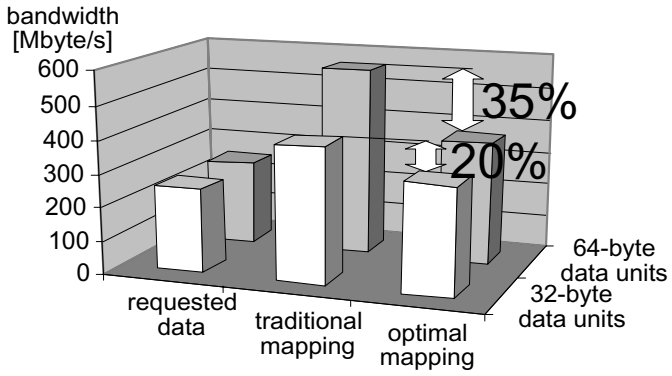


Figure 5.15: *Obtained bandwidth gain of the proposed mapping strategy.*

5.4 Concluding remarks

From video processing in SoCs it has become evident that off-chip memory communication is often the most critical part for system performance. Although the performance of general-purpose CPUs increases with 60 % every year, the bandwidth to the external memory increases with only 20 %. Partly, this can be solved by advances in technology that enable the embedding of an increasing amount of memory. However, from the application point of view the amount of necessary memory space also grows. Where analog TV systems required only a limited amount of memory for functions like Teletekst, current high-end TVs consume many MBytes of memory for temporal noise reduction, frame-rate conversion, flat-screen processing and rendering of several video sources together with graphical information onto a single display. Hence, the design of both embedded and off-chip memory remains a critical part of the system design.

Section 5.2 has explained that access to the data in SDRAM-based memories is provided in burst mode, resulting in a relatively large communication granularity. Furthermore, the addressing of data in the rows and columns of memory banks is constrained by the timing parameters of the internal memory operations. Consequently, the performance of the these memories such as the effective data bandwidth and latency, depend on the mapping of data onto the memory addresses and the order in which the data is accessed. Conventional systems apply a linear mapping, meaning that consecutive memory addresses coincide with the scanning order of the video pictures, i.e. from left to right and from top to bottom. Hence, the pixels of a video line are typically located sequentially in a memory row. However, if the order of data access is different than the scanning order, e.g. block-based access, the memory behavior becomes inefficient. It can be concluded that the mapping of the pictorial data into the memory should be optimized to the access statistics of the requested data accesses. This section has presented a memory communication model to optimize the mapping of video data onto the burst-oriented data entities in the memory. A novelty of this model is that it even takes the data-dependencies of the applications and the statistics of the data requests into account. Section 5.3 shows how this optimization is achieved by means of an example MPEG decoder. The results show a reduction of the total memory bandwidth with 35 %, corresponding to 195 MByte/s for high-definition MPEG decoding.

Apart from optimizing the mapping of pixel data into the memory, it is also possible to modify e.g. the application algorithm, thereby modifying the calculation model for further optimization. Hence, Appendix B shows how the memory bandwidth for high-definition MPEG decoding can be reduced with 100 MByte/s by writing the output of the MPEG decoder twice into the memory. Another example improvement in which the optimization model is modified is presented in Appendix C. In this example, the alignment grid of the memory transfers is refined by implementing a different the memory interface.

It can be concluded that the main achievement for optimal memory communication is to finding the best match between the characteristics of the data requests and the translation into the physical addressing of columns, rows, and memory banks. The following chapter discusses some additional bandwidth reduction techniques that are tuned to both the application and the memory configuration. However, because these techniques concern different functionality than the memory, they are discussed in a separate chapter.

Rem tene, verba sequentur
(Cato, c.234 BC – c.149 BC)
Keep to the subject and
the words will follow

Chapter 6

Communication bandwidth improvement

MEMORY bandwidth to the external memory devices(s) is an important and scarce resource in systems-on-chip. Apart from using the memory at high efficiency to optimize bandwidth utilization, complementary techniques can be applied to further reduce the memory bandwidth. This chapter discusses some of these techniques which are transparent for the application. First, a straightforward embedded compression technique is presented, followed by a section on a video-specific cache that exploits the two-dimensional locality of video. Because the MPEG-2 decoder from Chapter 5 represents one of the most critical functions for memory access, it is again adopted as an example function. The last section of the chapter reflects a collage of several discussed techniques, such as double writing (Appendix B), embedded compression (Section 6.1), and caching (Section 6.2). It shows that most techniques are complementary and can be used in combination, potentially resulting in tremendous memory bandwidth reductions.

6.1 Embedded compression

The previous section discussed how burst-oriented memory communication is required for efficient memory communication, even though it results in an overhead of data transfer. This section will show how this constraint is exploited to make embedded compression feasible for reduction of memory

bandwidth. Moreover, it explains how the calculation model as presented in Section 5.3 is extended for the communication of compressed video data.

6.1.1 Related work in compression

Although many compression techniques have been introduced, their usage for reducing the data bandwidth of the memory is not trivial. For example, a problem in computer systems is that the throughput of an application can degrade significantly when the working space of that application does not fit in the main memory. This results in an increased number of memory page faults so that the background memory on hard disk is accessed more often. As a possible solution, Roy *et al.* [78] implemented a concept of compressed memory pages to reduce the amount of disk accesses for the computer system. Although this offers a valid solution to decrease the number of disk accesses, the compressed memory pages need to be decompressed and written back into the memory when they are requested by the application. This process consumes extra memory access, so that the potential memory-bandwidth reduction is influenced in a negative way. Moreover, it will be shown later that a large grain size of compressed data packets such as the proposed memory pages, does not result in bandwidth reduction. Another application for embedded compression is presented in [79]. This approach applies lossless compression to relatively large data packets of 1 Kbyte in a high-complexity memory-controller chip to increase the memory capacity and is mainly intended for general-purpose computer servers. This functionality is located between the main memory and a large cache memory. When data is accessed at a small granularity ($\ll$1 KByte) the compression gives a penalty in bandwidth. However, due to the relatively large-grain data packets between the memory and the cache there may be sufficient net gain. In this paper, we concentrate on stream-based media processing with the objective to develop an inexpensive solution without the need for substantial cache memory.

Lossless compression has been studied primarily for reducing the required memory space. Its use for memory bandwidth reduction is not straightforward, because compression techniques are usually applied to a group of pixels to exploit correlation. For example, when nine MBs are encoded as one entity [80], the entity has to be partially decoded before the value of a certain pixel can be determined. For retrieval of an arbitrary data block in the picture, several compressed data entities may have to be decoded because they all contain part of the requested data block. Obviously, this does not help in the reduction of the memory bandwidth. On the other hand, many publications can be found on the reduction of necessary memory space in

an application. For example, in [81], a simple embedded Differential Pulse Code Modulation (DPCM) technique is applied for memory reduction in an high-definition MPEG decoder. Although this paper presents the measurement of bandwidth to derive the utilization of the memory bus, it does not further analyze the reduction in bandwidth. Van der Schaar *et al.* [82] proposed similar low-cost compression algorithms without bandwidth optimization and showed later [83] that the obtained compression factor only partially aids in bandwidth reduction. In this Chapter, we quantify results on the feasibility of low-cost compression schemes (e.g. [84]) for reduction of the memory *communication*. Furthermore, we propose an optimization technique to find all feasible compression factors that reduce the bandwidth between the memory and the video processing. This reduction, which can be as high as 67 % for a compression ratio of four, can be exploited to enhance the system quality, reduce costs and/or add extra functionality.

6.1.2 Feasibility of bandwidth reduction

Let us consider an example that shows why embedded compression for bandwidth reduction is not easily obtained. For this, we have adopted an MPEG-2 decoder as an example video function that requires access to the main memory. For the motion-compensated prediction in this coding standard, so-called reference frames (I and P) are stored in memories to provide the correct video data for either motion estimation/compensation. These reference frames are usually stored in an external background memory, leading to an increase of the memory communication. The architecture of the aforementioned video codecs usually includes various processors, which are communicating with a shared external SDRAM memory (Figure 6.1).

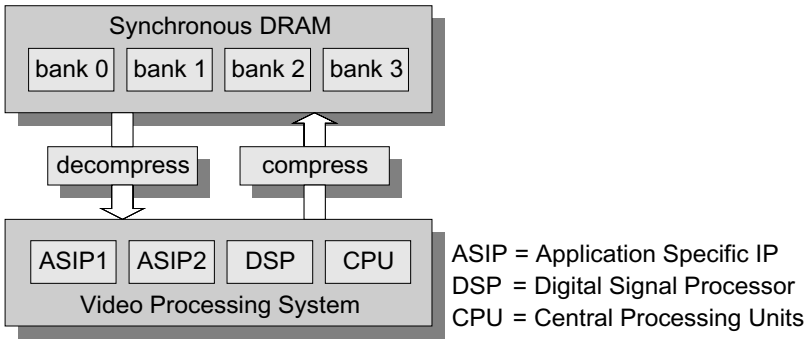


Figure 6.1: *A video system with embedded compression.*

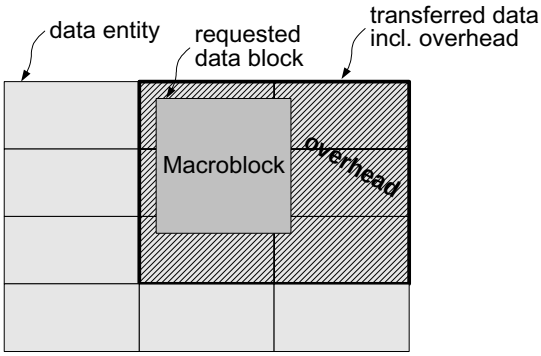


Figure 6.2: *Memory access of a macroblock including the transfer overhead.*

Chapter 5 has shown that access of the motion-compensated prediction data results in a transfer overhead, because more pixel data is transferred than is actually required for the decoding process. This is visualized in Figure 6.2. If for example, groups of 16×8 pixels are clustered as data entities, 78 % more data is transferred than was strictly required for the decoding process. Consequently, a compression ratio of 1.78 is necessary to accomplish the break-even point and even more for a net bandwidth reduction. However, when using an SDRAM-based memory, an overhead is already present due to burst-oriented storage. Table 6.1 indicates this overhead, dependent on the size of the communicated data bursts. For these numbers, the shape of the data entities, referred to as data units, are optimized for minimum transfer bandwidth. The bandwidth percentages show the transfer bandwidth relative to the bandwidth that is strictly required for the decoding. In the sequel of this subsection we call these numbers the *relative transfer bandwidth* (RTB). Since the data-burst size is

Table 6.1: *RTB requirement for data units with increasing sizes and the optimal dimensions.*

size [Bytes]	**optimal dimension**	**requested data [%]**	**transferred data $\bar{o} \cdot 100$ %**
16	(16×1)	100	117
32	(8×4)	100	127
64	(16×4)	100	146
128	(16×8)	100	178

given by the architecture, compression becomes more feasible, i.e. only the additional overhead due to compression needs to be compensated. For example, if data units of 16×8 (128 bytes) are compressed into 64-Byte data bursts, the RTB increases from 146 % to 178 %. Therefore, the break-even compression ratio equals $1.78/1.46 = 1.22$, which is feasible.

Up to this point, we discussed the properties of an SDRAM memory that result in block-based (bursts) storage of data and therefore increase the bandwidth requirement significantly. Furthermore, we have explained how this block-based storage can be exploited to reduce the bandwidth requirement by means of compression. However, the suitability of compression schemes is bounded by the constraints we have developed so far. Firstly, we found that the burst size determines the size of the compressed data entity. Secondly, the data must be easily accessible at regular address positions, thereby leading to a fixed compression ratio. Hence, this leads to fixed-sized input data units (data-units size S) and output blocks (data-burst size B). Let us now discuss a compression algorithm that satisfies the aforementioned constraints, presented by Bayazit *et al.* [84]. This technique is based on an adaptive DPCM algorithm, which can achieve a 50 % memory compression ratio with no distinguishable loss of visual quality. The paper describes the independent compression of one data block consisting of one luminance block of 16×2 samples and two chrominance blocks of 8×1 samples, thus 48 bytes in total. Because our system assumes separate storage of the luminance and chrominance data, it is required to independently compress luminance and chrominance components. This requirement does not limit the suitability of the algorithm and no performance degradation is expected for this reason. Another difference for the applicability of the proposed algorithm is the size of the data units. Data units of 48 bytes are rather small for obtaining sufficient compression. Section 6.1.4 will show that most suitable data-unit sizes for 32-Byte data bursts ($B = 32$) are $S = 64$ Bytes (and larger). For larger data bursts the most suitable size is even larger. Consequently, the algorithm will be able to exploit more correlation when applied for our purpose.

The experiments as described in the above-mentioned paper show that the compression scheme was not matched to the size of the memory bursts. The 48-Byte data blocks were compressed with a factor 1, 1.33, 1.6, and 2, resulting in compressed data entities of 48, 36, 30, and 24 Bytes, respectively. Because the sizes of these data entities do not match with the alignment grid of the data bursts, these compression ratios lead to sub-optimal solutions; i.e. more data bursts are transferred to access a compressed data entity.

The results of the experiments show a bandwidth reduction of 12.5 % for a compression ratio of two. The corresponding results on picture quality show a degradation of 0.97 dB for a MP@ML MPEG-2 video decoder, when decoding a high-quality 9−Mbps bitstream. For a bitstream that was coded at 4 Mbps, the quality degraded with only 0.34 dB. Subjective tests revealed high-quality video with imperceptible artifacts. For our system we target to decorrelate larger data blocks and hence an even better picture quality can be expected. Moreover, the results in the next section will show a significantly higher bandwidth reduction than in [84].

6.1.3 Bandwidth Calculations

In our experiments we again used the implementation of an MPEG-2 decoder to statistically analyze the behavior of the data communication, similar to the experiments as described in Subsection 5.3.4. To derive the memory bandwidth, the calculation model as proposed in [85][15] is used. The calculated result from this model represents the transfer bandwidth relative to the bandwidth that is strictly required for the decoding (RTB). However, in this model, embedded compression is not considered. Consequently, the size S of a data unit is assumed to be equal to the data-burst size B ($S = B$). In the following, we extend the above-mentioned model for the use of embedded compression. For this purpose, we introduce the compression ratio c_S, where S stands for the data-unit size. The value of the compression equals the ratio between the data-unit size and the data-burst size, thus:

$$c_S = \frac{S}{B}. \tag{6.1}$$

Below, we list the parameters on which the calculations for the RTB depend, including the compression ratio c_S:

- the dimensions of the requested data blocks, $B_x \times B_y$;
- the dimensions of the data units, (M, N);
- the interlace factor of the requested data blocks;
- the probability function of their occurrence, $P(B_x \times B_y)$;
- the probability function of their positions, $P_{B_x \times B_y}(m, n)$;
- the compression ratio, c_S.

For the bandwidth calculation, the set of possible data-block requests V has to be divided into a subset of progressive data block requests V_p and a subset

of interlaced data block requests V_i, such that $V = V_i \cup V_p$. This separation is necessary because the calculations for both contributions are slightly different. We denote the average RTB of progressive data-block requests by $\bar{o}_p(M, N, V_p)$ and for interlaced data-block requests by $\bar{o}_i(M, N, V_i)$. Both contributions are already probability-weighted, so that the total average RTB is:

$$\bar{o}(M, N, V) = \bar{o}_i(M, N, V_i) + \bar{o}_p(M, N, V_p). \tag{6.2}$$

To achieve the minimal bandwidth requirements for a given application, the dimensions of the data units (M, N) are optimized.

As mentioned earlier, we again consider an MPEG-2 decoder as an example for our experiments. For this application the set of data blocks is:

$$\begin{aligned} V_p &= \{(16\times16),(17\times16),(16\times17),(17\times17),(16\times8),(18\times8),\\ &\quad (16\times9),(18\times9)\}\\ V_i &= \{(16\times16),(17\times16),(16\times17),(17\times17),(16\times8),(18\times8),\\ &\quad (16\times9),(18\times9),(17\times8),(17\times9),(16\times4),(18\times4),\\ &\quad (16\times5),(18\times5)\} \end{aligned}$$

The large variety of data-block requests is caused by the MPEG standard and depends on field/frame prediction, luminance/ chrominance data and the sub-pixel motion-compensation accuracy. In our experiments, we consider the reading of prediction data for motion compensation, the writing of the motion-compensated result, $(16\times16) \in V_p$, and the reading for interlaced display of the video, $(M \times N) \in V_i$. For the last aspect, it is assumed that the display unit contains line memories to read the block-based data units of $(M \times N) \in V_i$ and to display the video lines sequentially. To acquire representative results for an average MPEG-2 bitstream, a large set of test bitstreams is used to derive the optimum data-unit dimensions.

To determine $P(B_x \times B_y)$, the numbers of occurrences of each type of data-block is measured at the memory interface, thereby feeding one of the data dependencies into the model. The probability function of the positions of the data blocks $P_{B_x \times B_y}(m, n)$, is defined as the probability that the upper-left corner pixel of a requested data block $B_x \times B_y$ is positioned at any location (x, y), satisfying the condition: $(x \bmod M = m)$ AND $(y \bmod N = n)$. Hence, a low-complexity bookkeeping of the occurrences at position $(x \bmod M, y \bmod N)$ is used to determine $P_{B_x \times B_y}(m, n)$. This probability function highly depends on the dimensions of the data units. Large differences may occur in the result, due to the 16×16 MB grid for MPEG and the high probability of the zero-motion vectors. For example, if $(M, N) = (32, 2)$, the probability that $(x \bmod 32 = 0)$ AND $(y \bmod 2 = 0)$

is relatively high. However, for data-unit dimensions that are not aligned with the MB grid, e.g. $(M, N) = (12, 5)$, the probability function of the positions of the data-blocks is totally different (see both examples in Figure 6.3).

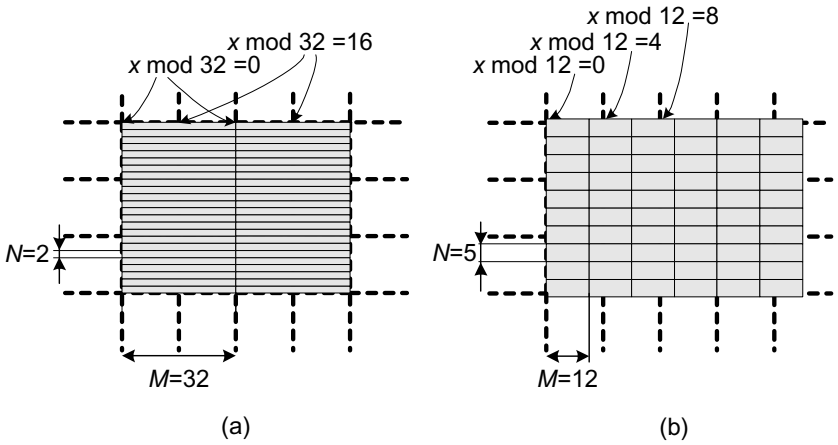


Figure 6.3: 32×2 *(a) and* 12×5 *data units (b) overlaid on a MB grid.*

At this stage, we have discussed the parameters that are necessary for the calculations and can derive the RTB. For the set of interlaced data-block requests, the following equation applies:

$$\bar{o}_i(M, N, V_i) = \frac{\sum\limits_{B_x \times B_y \in V_i} P(B_x \times B_y) H(M, N, V_i)}{c_S \cdot \sum\limits_{B_x \times B_y \in V_i} P(B_x \times B_y) \cdot B_x \cdot B_y}, \tag{6.3}$$

with

$$H(M, N, V_i) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} P_{B_x \times B_y}(m, n) \cdot M \cdot N \cdot \left(1 + \left\lfloor \frac{B_x + m - 1}{M} \right\rfloor\right) \cdot \left(1 + \left\lfloor \frac{B_y + n - 1}{N} \right\rfloor\right),$$

The summation in the numerator of Equation (6.3) represents the amount of transferred pixels including the overhead, whereas the summation in the denominator represents the amount of pixels that is strictly required for the decoding without the overhead. c_S indicates the compression ratio. Note that the calculation without c_S resembles the amount of transferred pixels

relative to the amount of pixels that is strictly required for the decoding. Since this is directly proportional to the bandwidth, the equation with c_S points to the effective bandwidth relative to the bandwidth that is strictly required for decoding without compression. The RTB calculation for the set of progressive data-block requests is similar to Equation (6.3) but V_i becomes V_p and $H(M,N,V)$ is defined according to:

$$H(M,N,V_p) = \sum_{m=0}^{M-1}\sum_{n=0}^{2N-1} P_{B_x\times B_y}(m,n)\cdot M\cdot N\cdot\left(1+\left\lfloor\tfrac{B_x+m-1}{M}\right\rfloor\right)\cdot \left(2+\left\lfloor\frac{\lceil B_y/2\rceil+\lfloor n/2\rfloor-1}{N}\right\rfloor+\left\lfloor\frac{\lfloor B_y/2\rfloor+\lceil n/2\rceil-1}{N}\right\rfloor\right).$$

6.1.4 Extraction of feasible solutions

In this subsection we derive the memory bandwidth by adopting the above-mentioned calculation model. The model exploits 64-Byte, 32-Byte, and 16-Byte data bursts, thus $B = 64$, $B = 32$, and $B = 16$, respectively. The method to determine systematically the optimal data-unit configurations for minimum bandwidth, comprises of the following steps:

- determination of the optimal data-unit configuration without using compression (in this case data-unit size S equals burst size B);
- determination of the optimal data-unit dimensions for an incremental value of the compression ratio;
- pruning of the non-feasible data-unit configurations.

Step 1- The RTB results without compression are presented in Table 5.4. The optimal dimensions (M_B, N_B) for a data-unit size $S = B$, are defined formally by:

$$(M_B,N_B) = \left(\exists_{(m,n)} : m\in[1..B], n\in[1..B] : \downarrow \bar{o}(m,n,V)\wedge m\cdot n = B\right), \tag{6.4}$$

where $\downarrow$ denotes the minimum and V is the set of requested data blocks. For example, the minimum RTB for 64-Byte data units equals $\bar{o}(M_B,N_B,V) = 146$ %.

Step 2 - The optimal data-unit dimensions are determined for increasing compression ratio. Note that the compression ratio c_S is given by $c_S = S/B$. Thus, for a fixed data-burst size (e.g. $B = 64$) the data-unit size is incrementally increased from the size of the data burst up to four times the burst

size. This resembles an incremental increase of the compression factor. For each $S \in (B..4B]$ the optimal data-unit dimensions are determined by:

$$(M_S, N_S) = \big(\exists_{(m,n)} : m \in [1..S], n \in [1..S] : \downarrow \bar{o}(m, n, V) \wedge m \cdot n = S\big) \quad . \tag{6.5}$$

Applying the optimal dimensions (M_S, N_S) for each S results in the minimum RTB value $\bar{o}_t(M_S, N_S, V)$. Figure 6.4 shows these values for $B = 64$ as function of the data-unit size. Because most solutions do not result in less data transfer than in the case without compression, they have been removed for an improved visualization of the results. Consequently, all shown solutions satisfy the following equation:

$$\bar{o}_t(M_S, N_S, V) < \bar{o}_t(M_B, N_B, V) \text{ with } S \in (B..4B]. \tag{6.6}$$

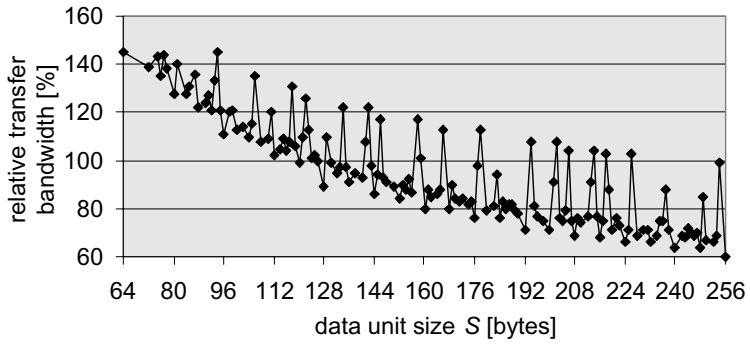


Figure 6.4: *Minimal RTB for $B = 64$ as function of the data-unit size S.*

Hence, incorrect dimensioning of the data-units can result in a sub-optimal solution or may even increase the memory bandwidth requirements.

Step 3 - In the third and final step, pruning of the found solutions can be applied. This pruning means in Figure 6.4 that, when starting at the left side and going to the right, only those solutions are adopted that give a lower RTB than the previously selected point. This means that all solutions with a larger compression ratio than other solutions while having less bandwidth-reduction gain, are removed. Since the picture quality as function of the compression ratio is generally a monotonic decreasing function, it can be concluded that the removed solutions have a lower picture quality than the remaining solutions while consuming more bandwidth. After the

pruning process all remaining solutions satisfy the following condition:

$$\Big(\forall_{\bar{o}_t(M_{S_1},N_{S_1},V),\bar{o}_t(M_{S_2},N_{S_2},V)} : S_1 \in (B..4B], \\ S_1 < S_2 \leqq 4B : \bar{o}_t(M_{S_1}, N_{S_1}, V) > \bar{o}_t(M_{S_2}, N_{S_2}, V)\Big). \tag{6.7}$$

The remaining data-unit configurations represent feasible solutions and are shown in Figure 6.5 for $B = 64$. The label at each point indicates the optimal data-unit dimension giving the result. This picture enables a tradeoff between compression ratio and bandwidth reduction. Note that data units of 16×8 reduce the relative transfer bandwidth from 146 % to 89 %. Up to this compression ratio the amount of RTB decreases more or less linearly per unit of compression. For $c_S > 2$, the gain of compression is saturating. Furthermore, because for the bending point $c_S = 2$ the data-unit dimensions are powers of two, this configuration is also particularly attractive from an implementation point of view.

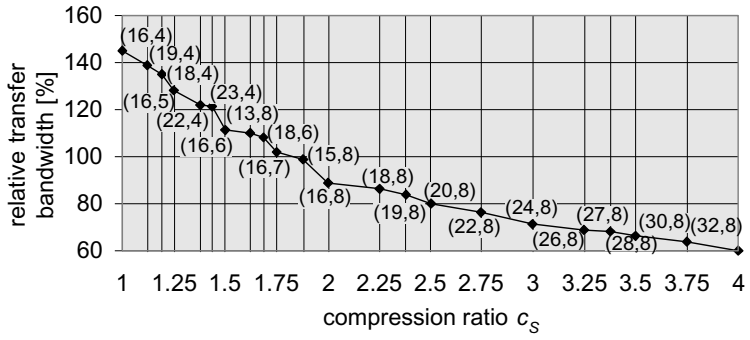


Figure 6.5: *Feasible data-unit configurations for compression into 64-Byte data bursts.*

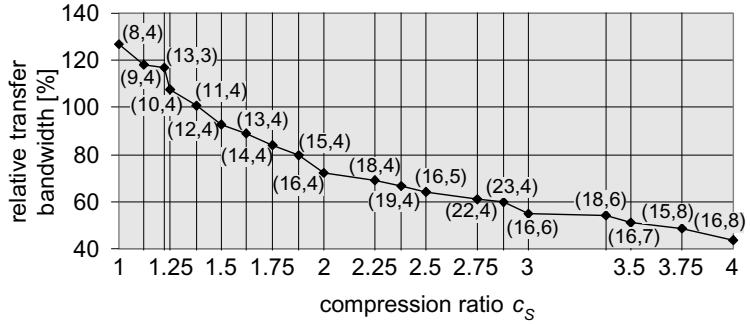


Figure 6.6: *Feasible data-unit configurations for compression into 32-Byte data bursts.*

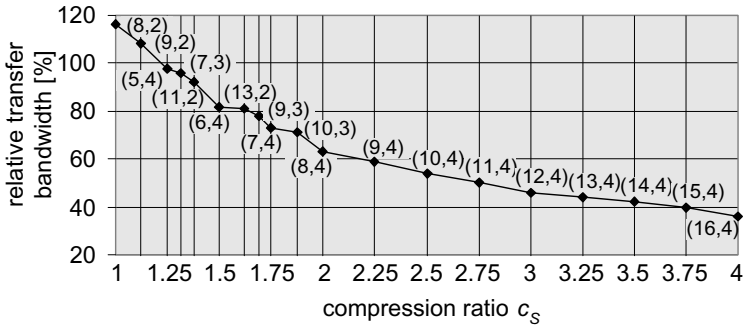


Figure 6.7: *Feasible data-unit configurations for compression into 16-Byte data bursts.*

Figure 6.6 and Figure 6.7 show the results for compression into 32-Byte and 16-Byte data units, respectively. Notice that the graphs all look very similar. All figures show an attractive design point at $c_S = 2$. For this compression ratio, the optimal data-unit dimensions are aligned with the MB grid in both horizontal and vertical direction, thereby enabling memory requests with low overhead at relatively high probability. However, also the size of the data unit is an important parameter for feasibility. A data unit of 16×8 can easily be compressed with a factor two, while maintaining a high picture quality. For data units of 8×4 this is less straightforward. Given the previous benefits, we have performed the picture-quality assessment as outlined in the next subsection for $c_S = 2$ and data bursts of $B = 64$.

6.1.5 Picture quality assessment

From the results presented above, it can be concluded that embedded compression is feasible for reduction of the memory bandwidth. However, these numerical results do not quantify the picture quality. Therefore, this subsection discusses some experimental results of a low-complexity compression scheme. The considered MPEG-2 decoding system represents a system that can measure the performance under worst-case conditions. This system applies both line-based and block-based accesses, while the order of accessing the picture data can be random. Additionally, any signal degradation caused by the lossy compression/decompression stage, is amplified due to the recursive nature of MPEG decoding (motion-compensated prediction loop). To test extreme cases, MPEG-2 bitstreams are used with a group-of-pictures (GOP) comprising one intra-coded picture followed by 14 pictures that are recursively predicted from the previous picture (IPPPP...). As a result, the quantization-error signal that is caused by the embedded

compression/decompression stage propagates 14 times through the MPEG decoding loop. Furthermore, the used video sequences contain highly detailed textures with slow smooth motion to maximize the visibility of the quantization error. The adopted compression algorithm divides the video signal into a DPCM-based lossless part and a lossy bitstream that can be truncated at any position, thereby enabling a scalable compression ratio without the necessity of a bitrate controller [86][87]. Figure 6.8 shows a peak signal-to-noise ratio (PSNR) plot of such a bitstream at 9-Mbps bitrate. The figure shows that the quality of the signal tends to drift away from the intrinsic MPEG decoder quality. The average PSNR difference is about 1 dB, whereas PSNR differences of more than 2 dB are measured for the last recursively predicted picture of the GOP. However, it has been found that the PSNR loss is rather content dependent and is not necessarily correlated with the subjectively perceived picture quality. For example, the experiments show that the PSNR loss caused by the embedded compression reduces for smaller bitrates of the bitstreams. Unfortunately, the effect of the embedded compression on the perceptual quality is more visible for such low bitrate sequences. This can be explained by the uncorrelated

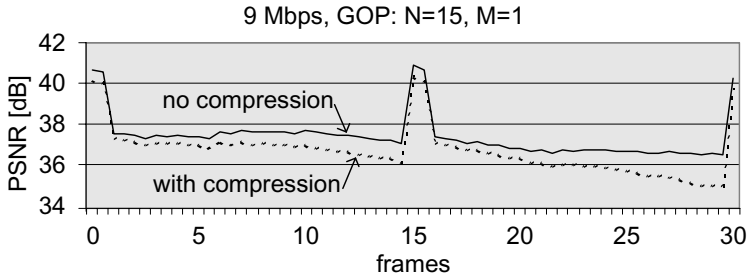


Figure 6.8: *PSNR : 9-Mbps bitrate and worst-case GOP structure.*

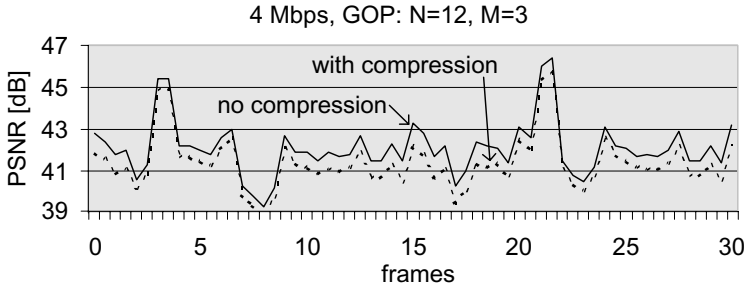


Figure 6.9: *PSNR : 4-Mbps bitrate and typical GOP structure.*

character of quantization noise from the MPEG coding, which is inherently more present in low-bitrate bitstreams. This is shown in Figure 6.10, where a part of the picture is enlarged containing considerable quantization noise from the MPEG coding. The picture is coded at 9-Mbps and contains high-contrast details which are difficult to encode with a high quality.

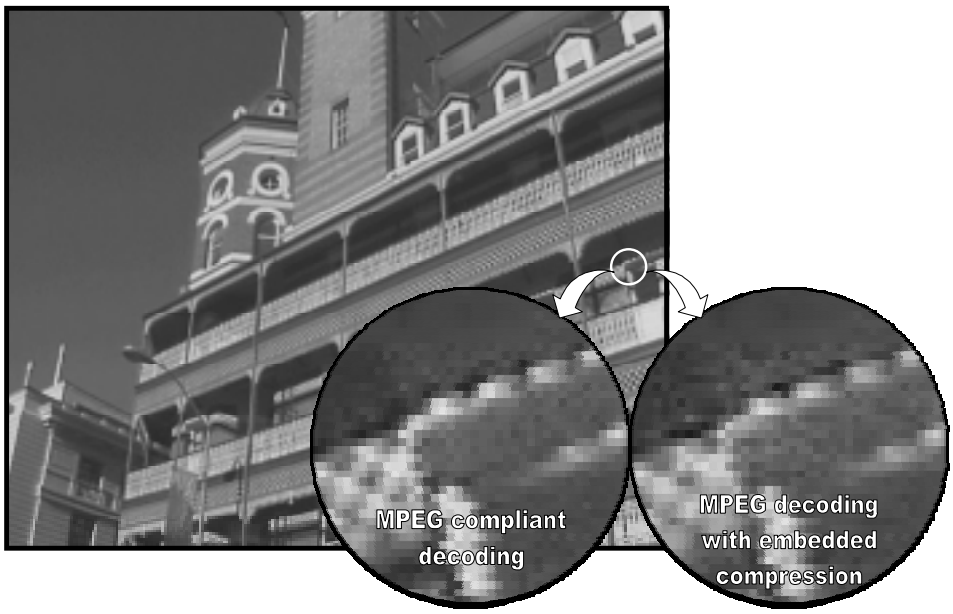


Figure 6.10: *Quality degradation from embedded compression due to MPEG quantization noise.*

To evaluate the picture quality for more practical MPEG bitstreams, several experiments were conducted with typical bitstreams at 4-Mbps and a more realistic GOP structure. With these bitstreams, the error propagates only four times through the decoding loop. Even though the earlier-mentioned experiments have shown a lower subjective picture quality for lower bitrates, no perceptual loss in quality could be observed. Figure 6.9 shows some typical results of the experiments of which the average PSNR difference is about 0.5 dB.

With future work, we expect to further enhance the picture quality by optimizing the applied compression algorithm. Bayazit *et. al* [84] obtained a 0.34 dB loss for a compression factor two and smaller 48-Byte data units, but their results were all based on typical GOP structures (no worst case) and less critical test material.

6.1.6 Conclusions

Many current multimedia systems intended for applications such as e.g. MPEG coding and 3-D graphics rendering, feature double-data-rate (DDR) SDRAM having a bus width of up to 64-bit. These expensive memory configurations are adopted to obtain sufficient communication bandwidth, which is demanded by the continuous increase of computational power. This section shows the feasibility of embedded compression for reducing the critical bandwidth bottleneck.

Our experiments with a conventional MPEG-2 decoder (with $B = 64$) without compression show that the amount of data transferred to and from the memory is 146 % of the data that is strictly required for the decoding, using the optimal mapping of groups of pixels (data units). However, in most currently used systems, a straightforward linear mapping of 64×1 pixels into data bursts is applied, resulting in a relative transfer bandwidth of even 223 % [15]. Due to the trend of increasing memory hierarchy, the size of the data bursts grows, leading to even more transfer overhead. Fortunately, larger blocks can be decorrelated more efficiently, which makes the use of embedded compression for memory bandwidth reduction increasingly attractive.

Another important observation is that compression does not lead to reduction of the memory bandwidth for all compression factors. The block-based storage of compressed data entities, may even lead to an increased bandwidth. Moreover, it has also been found that certain data-unit sizes offer less bandwidth reduction than others, while giving less picture quality. It can be concluded that the compression ratio and the data-unit dimensions have to be carefully matched with the size of memory data bursts. Hence, a selection of attractive data-unit sizes and corresponding compression factors has been reported.

For consumer applications, a low-cost compression scheme is required to reduce bandwidth without sacrificing visual picture quality. For a compression factor four, a bandwidth reduction is established of 58 % for 64-Byte data bursts. For a compression ratio of two, the bandwidth reduces with 39 %. Summarizing, the optimal grouping of pixels into data units decreases the relative memory bandwidth of the MPEG decoder from 223 % to 146 % and the embedded compression accounts for a further reduction to 89 %, thereby gaining a total bandwidth reduction of 60 %. A side benefit of using an embedded compression scheme with a fixed compression ratio is that it reduces the required memory size proportionally to the compression

ratio. Thus, the three picture memories in MPEG decoding are halved for a compression factor two.

The proposed techniques for reducing the bandwidth bottleneck of external memory can be generalized to a broad class of block-based video processing applications. When adopting embedded compression for decoding of high-definition video pictures, the bandwidth reduction is as high as 142 or 212 MByte/s for a compression ratio of two or four, respectively. Since this substantial improvement in memory bandwidth is guaranteed, it can be exploited easily for reducing the system cost, to improve the picture quality, or to extend the system functionality.

6.2 Caching

The previous subsection has discussed a low-complexity embedded compression scheme to reduce the memory bandwidth. A major advantage of the technique is the guaranteed reduction of the memory bandwidth. However, due to the fixed compression factor this approach cannot be achieved without loss of picture quality. This subsection discusses caching as another technique to reduce memory traffic of the MPEG decoder. A key difference with embedded compression is that caching is completely transparent for the MPEG decoder, i.e. the cache can be omitted without loss of functionality. Another difference is the obtained picture quality. Caching maintains the full picture quality, whereas embedded compression introduces small visual losses. A drawback of caching is that the bandwidth reduction cannot be guaranteed under all circumstances, although in general large bandwidth savings are achieved.

The main intriguing problem to solve is the potential gain in memory bandwidth and the associated costs. This section analyzes a similar MPEG decoding application as with embedded compression to determine the architectural requirements for the cache. Consequently, the cache implementation can be balanced to achieve a high performance at an acceptable design complexity.

Prior to discussing our experiments with caching, we briefly summarize a few concepts and techniques for video-specific caching. The principle of a cache is based on temporal or spatial locality of data. *Temporal locality* means that frequently accessed data is buffered locally (in the cache) near the processing engine. *Spatial locality* means that adjacent data items have high probability of being accessed successively. Hence, a complete data

unit as described in Section 5.3, is buffered in the cache by means of a single efficient transfer. For the location of the data units in the cache, we can distinguish set-associative caches and direct-mapped caches. In a set-associative cache, each data unit in the memory can be stored in several locations (i.e. cache lines) of the cache. Consequently, each data request requires to search all entries in the cache that qualify for a possible location. In addition, when the cache is full, it provides the flexibility to replace that data in the cache that is least relevant. In a direct-mapped cache, each data unit can be associated with exactly one cache line and thus it is not required to search through the cache. Although this implementation is less complex than set-associative caches, it does not offer the flexibility of selecting the least relevant data for replacement. Consequently, its performance will generally be poorer. To optimize the performance for such a cache, it is necessary to derive a suitable relation that associates a data unit in the higher-level memory to one of the cache lines. Basically, this means that the probability should be minimized that an accessed cache line is replaced before it can potentially be accessed again. This relation is straightforward for general-purpose processing and is well-known for CPUs. In this chapter, we reuse the newly introduced data dependency concept of the previous chapter. When using caches, this leads to the novel idea of exploiting the *two-dimensional locality* of video data. Without going into details, we want to explore the most important design parameters, such as the potential performance and the associated cost. The details on the implementation of video-specific caches and the mapping of data units onto cache lines can be found in Appendix D.

6.2.1 Experimental results for MPEG decoding

As stated above, we want to determine the gain in bandwidth reduction and the cost of the video cache. Up to this point, we know how to construct a cache for video-processing functions. However, to determine the performance potential and the cost of an effective cache, we need to properly dimension the following design parameters:

- associativity of a cache,
- optimal data-unit dimensions,
- cache size,
- relation for associating data with a cache lines, and
- replacement strategy.

As a case study, we explored the design of a cache for a motion-compensation unit in an MPEG decoder. The following constraints are used to limit the case study for this chapter:

- Since writing of the reconstructed macroblocks and reading of video data for display can be achieved without transfer overhead (see Section 5.3), a cache to exploit the spatial locality is not required. Hence, these memory transfers are directly conveyed to and from the background memory without accessing the cache.
- Data from the higher-level memory can be accessed by means of 64-Byte data bursts which can only be accessed as an entity. The size of the cache lines is matched to this burst length.
- The motion-compensation unit requests only that data that is strictly required for the decoding (no overhead). The bandwidth of these data-block requests are used as a reference to measure the actual bandwidth efficiency. Hence, the requested data bandwidth is considered to be 100 %.
- Timing issues are not included in the simulation model.

The design of a cache depends on the architecture of the higher-level memory and the functional processing unit. The background memory constrains the size of the cache lines, the size of the tags, etc. The functional unit that uses the cache determines the total cache size, the number of sets, the 2-D spatial dimensions of the cache sets, the replacement strategy, etc. Even if all such memory-system parameters and application parameters are known, the design space is still too large for complete exploration. Therefore, for each set of experiments in this section we will fix those parameters that are expected to be independent.

Cache associativity

In a set-associative cache, each data unit can be associated with several cache lines. This provides the flexibility to replace the least relevant data in the cache. A direct mapped cache does not offer this freedom, but it can be implemented with a lower complexity. Instead of searching through the cache for a requested data unit, a fixed relation associates the memory address of a data unit to a unique location in the cache. For MPEG decoding, several independent video buffers are adopted to separately store the luminance and chrominance pixels of both forward and backward pictures. For a direct-mapped cache, care has to be taken that data units from different video buffers but at the same pixel locations are not associated with the

same cache line. This would replace valuable cache data. As a solution, we can separate the video buffers by means of separate subsets in the cache (see Appendix D), or we can increase the amount of set-associative cache lines.

With the following experiment, we investigate the performance difference of subsets for forward/backward reference frames and luminance/chrominance data instead of more set-associative cache lines. For each simulation, the data units contain 16×4 pixels (64 bytes), the size of the subsets for luminance are 2×8 cache lines (32 pixels $\times$ 32 video lines) and 2×4 cache lines for chrominance.

- *Simulation 1* - Forward and backward reference pictures for luminance and chrominance have different subsets, four in total. For the chrominance, the subsets are twice as small in vertical direction compared to the subset for luminance. The cache is two-way set-associative. Consequently, the size of cache is 6 kByte.

- *Simulation 2* - Forward and backward reference pictures have different subsets, but no distinction is made between luminance and chrominance. Instead, a four-way set-associative cache is used. Consequently, the effective set size for chrominance is twice as large as for Simulation 1. Therefore, the size of the cache is 8 kByte.

- *Simulation 3* - Luminance and chrominance data are associated with different subsets, but no distinction is made between forward and backward reference pictures. Instead, a four-way set-associative cache is used. The cache size is 6 kByte.

- *Simulation 4* - No distinction is made between forward and backward reference pictures, nor between luminance and chrominance. Instead, an eight-way set-associative cache is used. Hence, an 8-kByte cache is used.

Figure 6.11 shows that all simulations of the above-defined experiment results in a bandwidth reduction between 1.56 % and 32.23 %. Therefore, it can be concluded that due to the regular behavior of video processing, the use of a direct-mapped cache (using subsets) for different streams, instead of set-associative caches, results in similar performance. However, with increasing associativity, the cache always improves its performance. Note, that it is more beneficial to have separate subsets for Y and UV components instead of separate subsets for forward and backward reference frames. Although the size of the cache in Simulation 3 is only 6 kByte due to the

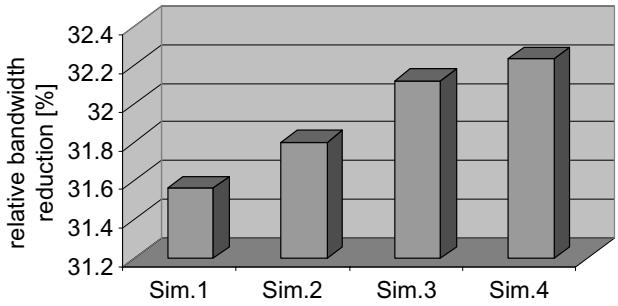


Figure 6.11: *The results of four different simulations, indicating the performance difference between a direct-mapped cache and a cache with more set-associative cache lines.*

smaller subset size for the chrominance, it outperforms the 8-kByte cache of Simulation 2. A drawback of set-associative caches is the more complex design due to the additional compare operations, which are required to search through all cache lines that can be associated with a data unit.

To limit the amount of experiment for design-space exploration, all following experiments will be performed without subsets to determine the upper bound of performance. For a final design, it can be easily verified whether subsets can be implemented for cost reduction without performance losses.

Optimal data-unit dimensions

Another design parameter that needs to be optimized is the shape of the data units, i.e. the data blocks that are stored in the cache lines. From the previous chapter, we have learned that data units of 16×4 result in the minimal bandwidth. However, when a cache is added to the system, this might be different. For example, if the cache performance is significantly better for data units of 32×2, this may influence the final result. Figure 6.12 shows the result of an experiment where the data-unit size is changed. For the experiment we assume that the amount of associativity does not have a large impact on the results. Therefore, we performed all simulations with a full-associative cache. Hence, the amount of associativity is equal to the cache size divided by the cache-line size (64 Byte). The figure shows that for the tested bitstreams the optimal data-unit dimensions is 16×4 for all cache sizes. It results in the largest bandwidth reduction for the used MPEG bitstreams. Note that this does not imply optimality. Figure 6.13 zooms in on the results of an 8-kByte cache. The upper figure shows that the performance of the cache (hit rate) decreases for increasingly

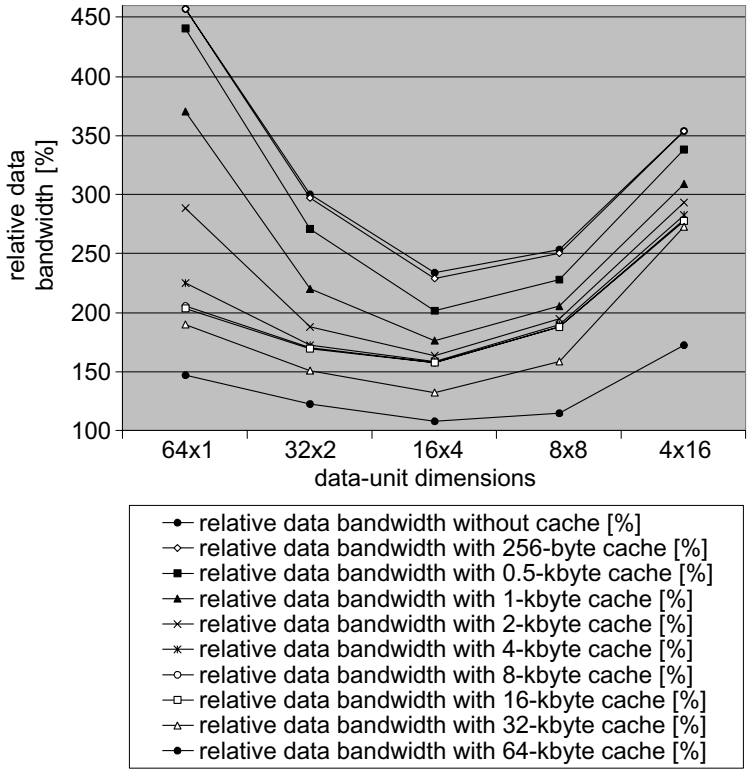


Figure 6.12: *Data bandwidth reduction as function of the data-unit dimensions.*

vertically-oriented data units. The bottom figure portrays that for the tested bitstreams, a minimal data bandwidth is required for 16×4 data units, when no cache is applied. Moreover, the lower figure shows that the addition of a cache reduces the bandwidth consumption, particularly for more horizontally-oriented data units. However, as already concluded before, the 16×4 data-unit remains optimal for the tested bitstreams. It is interesting to note that traditional systems writing the video data in line scanning order from left to right and top to bottom, benefit most from caching. The data burst to and from memory for these systems can be considered to contain data units of 64×1 pixels. Hence, if these systems use 64-Byte data bursts and do not feature burst interruption, the data bandwidth for motion compensation is 457 % of the data bandwidth that is strictly required. When applying the 8-kByte cache as described above, the data bandwidth reduces to 205 %, which provides a substantial gain of 55 %.

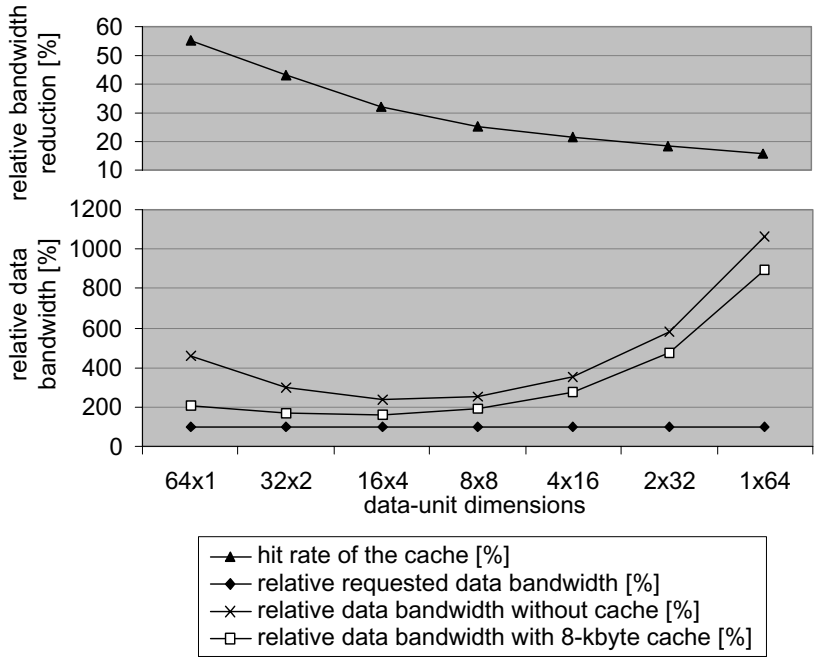


Figure 6.13: *Cache performance as function of the data-unit dimensions.*

Cache size

For the following experiments, we will concentrate on data-unit dimensions of 16×4 pixels. Although Figure 6.12 shows the results for different cache sizes, it does not give much insight in the relation between the cache size and cache performance. Therefore, we conducted simulations to measure the influence of an increasing cache size and plotted the relative data bandwidth (see Figure 6.14). For very small cache sizes ($<$ 512 Bytes) reuse of transfer overhead cannot be exploited due to the small lifetime of the data in the cache. Because the size of the cache is even smaller than the size of a macroblock, data that could be reused is already replaced when requested for the second time. From 512 to 2048 Bytes, the cache clearly exploits the spatial locality in horizontal direction. Because motion vectors of adjacent macroblocks are generally highly correlated and show a consistent behavior, the reuse of data in the horizontal direction is limited. Therefore, the bandwidth reduction saturates between an 8 to 16-kByte cache size. Beyond the size of a 16-kByte cache, complete stripes of macroblocks can be contained in the cache. Consequently, also the vertical spatial locality can be exploited in the cache. Up to a cache size of 64 kByte, which is

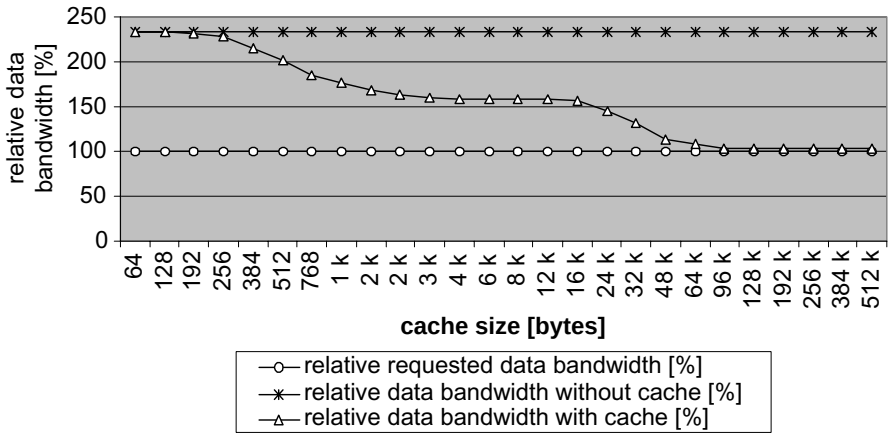


Figure 6.14: *Cache performance as function of the cache size for standard-definition video.*

equivalent to a memory containing 60 video lines, an increasing amount of vertical spatial locality is exploited. After this point, the influence of an increasing amount of cache memory is small. Note that the size of a cache exploiting the vertical spatial locality depends on the horizontal resolution of the picture. The adopted experiments where all performed on standard-definition (SD) picture material. Consequently, to exploit vertical spatial locality for high-definition (HD) video we expect the required cache size to be 1920/720=2.67 times as large. The experimental result in Figure 6.15

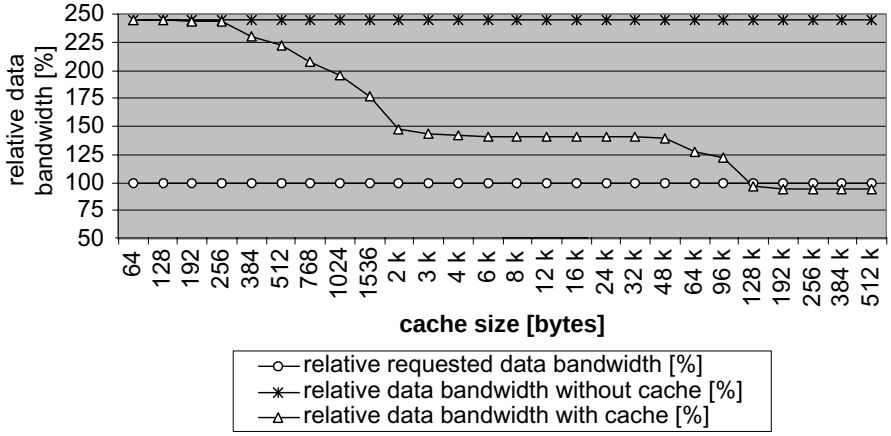


Figure 6.15: *Cache performance as function of the cache size for high-definition video.*

acknowledges this. Note that the cache size for exploiting the horizontal spatial locality in an HD video stream is similar to the size for SD. Thus a 3 to 4-kByte cache would be sufficient. Figure 6.15 also shows that there is an upper bound on the potential bandwidth reduction. Even with an unlimited cache size, the bandwidth does not decrease much further than 100 % of the bandwidth that is strictly required for the motion compensation. Apparently, the actually requested data hardly contains any temporal locality. Only the transfer overhead is reused. Consequently, we can conclude that caching for motion compensation in an MPEG decoder is only beneficial for decreasing the transfer overhead. Hence, for MPEG decoder systems with a 90 % memory efficiency or higher, caching is not very useful for reduction of the memory bandwidth.

For the MPEG decoder system that is considered in this thesis, a modest 4-kByte cache is sufficient to regain the memory bandwidth of the transfer overhead in horizontal direction. However, even for such a small cache, a full-associative implementation is not very practical due to its high design complexity.

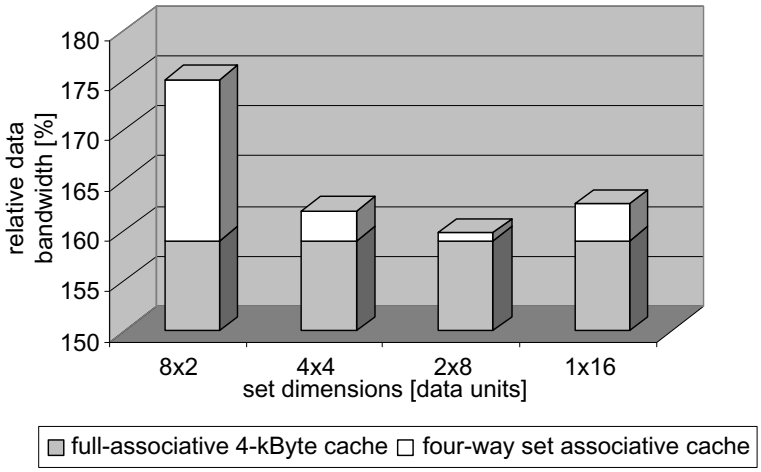


Figure 6.16: *Relative data bandwidth of a 4-kByte full-associative and four-way set-associative cache as function of spatial arrangement of the cache lines.*

Complexity reduction

The following experiment is conducted to reduce the complexity of the cache by reducing the amount of cache-lines that can be associated with each data unit. Instead, we can associate the cache lines with the data

units depending of their spatial position in the picture. Figure 6.16 shows how the performance of a 4-kByte cache decreases when a four-way set-associative cache is used instead of a full-associative cache. Moreover, its shows how different spatial arrangements of the cache lines in a set (for more details see appendix D) lead to different cache performances. When a set of 2×8 data units is provided, equivalent to a set size of 32 pixels by 32 lines, the performance is comparable to the full-associative cache. The bandwidth increases only slightly from 159 % to 160 %.

To bring this simplification even one step further, the experiment of Figure 6.11 was reproduced for a 4-kByte cache to determine the influence of using a direct-mapped cache instead of a four-way set-associative cache. The new results are very similar and present a bandwidth reduction between 30.13 % and 31.48 %. Applying the direct-mapped cache of Simulation 1 with 3-kByte memory, the bandwidth reduces from 233 % to 163 % of the data bandwidth that is strictly required for motion compensation.

Replacement strategy

Finally, the influence of the replacement strategy is investigated. The least-recently-used (LRU), the least-recently-written (LRW), and the largest Manhattan distance are evaluated. Both the LRU and the LRW perform better than the largest Manhattan algorithm. No difference in performance could be noticed between the LRU and LRW. Therefore, we propose to adopt the LRW algorithm due to its low complexity.

6.2.2 Conclusions

Experiments were conducted to design a cache for a motion-compensation unit of an MPEG-2 decoder. In these experiments, only the prediction data read from the reference frames are conveyed via the cache. The higher-level memory is assumed to have an SDRAM behavior. Hence, data can only be accessed at an alignment grid. Moreover, due to the burst access of the memory and the width of the communication network, data can only be accessed as entities of 64 bytes (burst interruption is not provided). The experiments reveal that data units with a dimension of 16 pixels by 4 lines are optimal for the tested bitstreams, independent of the size of the cache. Moreover, it can be concluded that a cache of 3-4 kByte reduces the bandwidth from 233 % to 159 % of the data bandwidth that is strictly required for the motion compensation. This corresponds with a cache performance (hit-rate) of 1-(159/233)=32 %. To further reduce the bandwidth, vertical spatial locality should be exploited, requiring a cache of 48 to 64 kByte for

SD video and about 128 kByte for HD video. Such a cache size reduces the relative transfer bandwidth to more or less 100 %, providing a hit-rate of more than 55 %.

For the experiments that were conducted for MPEG decoding at main profile - main level (MP@ML), several 25-Hz standard-definition MPEG bitstreams were decoded containing 732 video frames. In total 479 MByte (excluding the overhead) of video data was processed for motion compensation. Using these numbers, an average bandwidth of (479 MByte / 732 frames) $\times$ 25 Hz = 16.36 MByte/s is required. All experimental results presented are relative to this bandwidth requirement. Therefore a direct-mapped 3-kByte cache, reducing the relative bandwidth from 233 % to 163 %, saves 11.5 MByte/s for motion compensation during decoding of a standard-definition video sequence. The experiments with high-definition video reveal a reduction of even 80 to 100 MByte/s, depending on the used bitstream.

Due to the regular access behavior of the motion-compensation unit, a direct-mapped cache shows almost equal performance compared to the set-associative variants. However, the different video streams for forward and backward prediction data and luminance and chrominance data, require independent caches, or separate subsets, as explained in Subsection D.2. Because for motion compensation the chrominance prediction data is generally twice as small in the vertical direction as the luminance prediction data, the subsets for chrominance can be twice as small without significant loss of the cache performance.

The experimental results furthermore show that only the transfer overhead contains temporal locality. Therefore, MPEG decoder systems with a high bandwidth efficiency do not gain much from caching. It can also be concluded that SDRAM-based memory interfaces that do not feature burst interruption, can benefit significantly from caching, particularly if the video data are not very efficiently mapped onto the memory space. For example, if the video data are sequentially written into the memory in a scanning order from left to right, line by line, the relative transfer bandwidth is 457 % of the data bandwidth that is actually requested. In such a case, a small cache reduces the bandwidth to a slightly more than 200 %, resulting in a bandwidth gain of at least 55 %.

Apart from reduction of the bandwidth to the higher-level memory, an advantage of caching is the fixed-sized data-burst requests to the background

memory for the transfer of data (the cache misses). For systems that feature burst interruption, the memory interface is more complex and more critical to design, due to the more advanced memory-command scheduling. Let us clarify this statement. Burst interruption often leads to some loss in unutilized data-bus cycles and results in a higher probability of bank conflicts (see Subsection 5.2.2). Thus, for systems featuring efficient memory communication, caching can be used to simplify the memory controller of the background memory and to relax its memory-command scheduling.

6.3 Combined techniques

In this last section of the chapter we show how a collage of the following techniques can be combined in an MPEG decoder architecture that is again used as an example:

- block-based storage of video data into memory bursts (Subsection 5.3);
- a block-to-line buffer for line-based access of data that is stored block wise;
- double writing of reconstructed output pictures to optimize the video storage separately for motion-compensated prediction and for video display (Appendix B);
- embedded compression (Subsection 6.1);
- caching (Subsection 6.2).

Because the third technique is beyond the scope of the thesis, it suffices to understand that the reconstructed output of the MPEG decoder is written twice into the background memory: once with an optimal mapping for block-based reading to do motion-compensated prediction, and once for line-based reading by a display unit. Although the double writing of the output requires additional memory traffic, the amount of transfer overhead for reading reduces significantly, leading to a net bandwidth gain. A more detailed elaboration on this technique is presented in Appendix B.

As a reference implementation, we use a traditional MPEG-2 decoder system with line-based storage of the video data, without caching, and without embedded compression (see Figure 6.17). A separate video-out unit for display reads the video lines sequentially from the memory and does not contain any buffer memory that could be used to convert block-based data access to line-based output for display. We assume a 64-bit bus and

a burst length of 8 words, i.e. 64-Byte data bursts. These bursts cannot be interrupted. For the experiments, a representative set of SD bitstreams (720×576 @ 25 Hz) is used, containing 732 frames in total. Let us now dis-

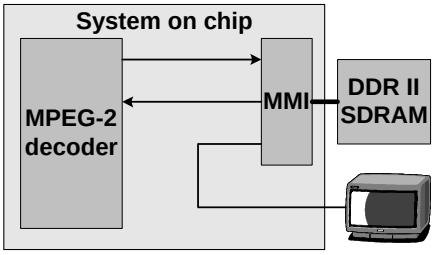


Figure 6.17: *Reference MPEG-2 decoder system.*

cuss all possible architectural options that result from the above-mentioned techniques. At this point, it should be clear that the applicability of these techniques depends on the well-known tradeoffs between cost, flexibility, picture quality, etc. We can combine caching and embedded compression, implement a block-to-line buffer in the video output unit, etc. To determine the optimum for each architectural option, we need to determine the transfer bandwidth for writing of the reconstructed macroblocks, the reading for display, and the read access for prediction. For reading of display data and writing of reconstructed macroblocks, the bandwidth requirements are straightforward. These actions consume the bandwidth of one video stream without any overhead due to the sequential access pattern. Thus, for 25-Hz standard-definition video the bandwidth for writing and for display equals $720 \cdot 576 \cdot 1.5 \cdot 25 = 16$ MByte/s. Also the combinations with double writing of the reconstructed output video to optimize separately for prediction and display, and the use of a block-to-line buffer can easily be derived, since they do not depend on the data content. By contrast, the bandwidth for reading of the prediction data is very data dependent, so that it has to be measured. Concerning the data-dependent bandwidth reduction techniques for prediction data, we can distinguish four different combinations:

- without cache and without embedded compression;
- with cache and without embedded compression;
- without cache and with embedded compression;
- with cache and with embedded compression.

The results of the measurements are presented in Table 6.2. The numbers in the table only show the measured average bandwidth numbers. However,

Table 6.2: *Transfer bandwidth for MPEG decoding.*

Access function	data unit ($M \times N$)	requested [Bytes]	req./transf. ratio [%]	P/B-pics. ratio [%]
Reading for prediction without cache, no compression	(64×1)	478871794	457	38/62
	(32×2)		300	39/61
	(16×4)		**233**	40/60
	(8×8)		253	40/60
Reading for prediction with cache, no compression	(64×1)	478871794	205	37/63
	(32×2)		170	37/63
	(16×4)		**158**	38/62
	(8×8)		188	37/63
Reading for prediction without cache, compression	(128×1)	478871794	418	38/62
	(64×2)		256	37/63
	(32×4)		181	40/60
	(16×8)		**163**	37/63
	(8×16)		207	39/61
Reading for prediction with cache, compression	(128×1)	478871794	139	38/62
	(64×2)		112	38/62
	(32×4)		**101**	37/63
	(16×8)		109	40/60
	(8×16)		151	36/64

as a function of time these bandwidth numbers for MPEG decoding vary dynamically. For example, I-pictures do not require any memory traffic of prediction data and because B-pictures are bi-directionally predicted, they generally require more bandwidth than P-pictures. If the decoder system contains sufficient buffering and can perform the decoding at a sufficiently high speed, the dynamics in the memory bandwidth can be smoothed out, e.g. I-pictures are decoded at high speed whereas for B-pictures the decoding is performed slowly to lower the memory bandwidth. However, in more practical systems, the decoder is designed for the average required throughput and with minimal buffer requirements. In such a case, the dynamical behavior of the memory bandwidth has to be studied for worst-case situations. Therefore, the last column in the table shows how much of the transfer bandwidth is consumed for P-pictures and how much for B-pictures. Within such pictures the bandwidth of data communication still changes dynamically. Hence, systems that cannot smooth out these dynamics, need to be analyzed with respect to the memory bandwidth at an even smaller granularity, e.g. at slice level. Without going into this level of detail for all architectural options, Figure 6.18 shows an example histogram of the memory traffic at slice level for decoding of B-pictures in a SD-resolution video sequence. The numbers are based on data units of

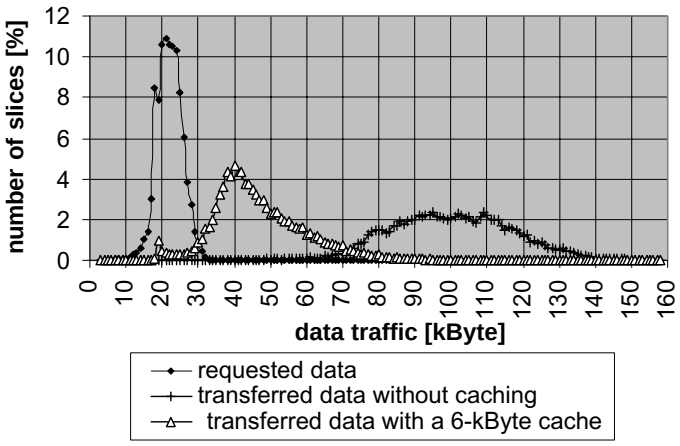


Figure 6.18: *Histogram of the data traffic for decoding of an high-definition MPEG-2 slice.*

(64×1). Note that the worst-case memory traffic for a slice is:

$$720/16 \cdot (64 \times 1) \cdot (2 \times 17 + 2 \times 9) \cdot 2 = 300 \text{ kByte},$$

where $720/16$ denotes the number of macroblocks in a slice, 2×17 the number of data units that might be used to retrieve a luminance block of 17×17, 2×9 indicates the number of data units for a multiplexed chrominance block of 18×9, and the factor 2 indicates the bi-directional prediction. Although the measured worst-case memory traffic for a slice is 158 kByte, there is a small probability that the memory traffic becomes almost double of that (300 kByte). Therefore, if the decoder system does not contain any speed margin and buffering, the communication infrastructure has to be dimensioned for this theoretical maximum throughput. Note that the cache considerably reduces the average data traffic for motion-compensated predictions. However, although the probability of the theoretical maximum throughput is reduced when applying a cache, it cannot become zero.

To discuss the worst-case bandwidth conditions at picture level, Table 6.3 shows the average transfer bandwidth to the higher-level memory for decoding of the B-pictures. The table shows the results for all possible architectural options. R_p in the table denotes the reading for prediction. The parameters W and R_d denote the writing of reconstructed macroblocks and the reading for display, respectively. For all options we have assumed 64-Byte memory bursts that cannot be interrupted and can only be accessed at a 64-Byte alignment grid. Architectural Option 1 represents a hypothet-

Table 6.3: *Options with different combinations of bandwidth reduction techniques (compression factor = 2, cache size = 4 kByte).*

architectural options		access type	BW [MByte/s]	total [MByte/s]
1	**without overhead**	R_p	20	52
		W	16	
		R_d	16	
2	**reference architecture writing in** (64×1)	R_p	93	124
		W	16	
		R_d	16	
3	**no cache, no compression write in** (32×2)**, no block-to-line buffer**	R_p	60	107
		W	16	
		R_d	31	
4	**no cache, no compression write in** (64×1) **and** (16×4)**, no block-to-line buffer**	R_p	46	77
		W	16	
		R_d	16	
5	**no cache, no compression write in** (16×4)**, with block-to-line buffer**	R_p	46	77
		W	16	
		R_d	16	
6	**with cache, no compression write in** (64×1)**, no block-to-line buffer**	R_p	42	74
		W	16	
		R_d	16	
7	**with cache, no compression write in** (64×1) **and** (16×4)**, no block-to-line buffer**	R_p	32	63
		W	16	
		R_d	16	
8	**with cache, no compression write in** (16×4)**, with block-to-line buffer**	R_p	32	63
		W	16	
		R_d	16	
9	**no cache, with compression write in** (64×2)**, no block-to-line buffer**	R_p	52	75
		W	8	
		R_d	16	
10	**no cache, with compression write in** (128×1) **and** (16×8)**, no block-to-line buffer**	R_p	32	48
		W	8	
		R_d	8	
11	**no cache, with compression write in** (16×8)**, with block-to-line buffer**	R_p	32	48
		W	8	
		R_d	8	
12	**with cache, with compression write in** (128×1)**, no block-to-line buffer**	R_p	28	44
		W	8	
		R_d	8	
13	**with cache, with compression write in** (128×1) **and** (32×4)**, no block-to-line buffer**	R_p	21	36
		W	8	
		R_d	8	
14	**with cache, with compression write in** (32×4)**, with block-to-line buffer**	R_p	21	36
		W	8	
		R_d	8	

ical architecture that is able of transferring only the data that is strictly required for decoding and display without any overhead. The second option shows the result for the reference architecture which is a traditional system that does not provide any of the bandwidth reduction techniques and stores the video data line-based into the memory. For this system, the transfer overhead results in a total bandwidth consumption of 124 MByte/s. All other architectural options apply the data-unit dimensions that result in the minimal memory bandwidth. Option 3 only applies block-based storage, thereby reducing the total bandwidth for B-pictures to 107 MByte/s. Option 4 shows the bandwidth results for B-pictures when using the double writing technique as described in Appendix B. Note that B-pictures only require writing of the display data because B-pictures are not further used as reference. Consequently, this technique is particularly suitable to reduce the peak bandwidth, i.e. the memory bandwidth for B-pictures. The same bandwidth result is obtained by using Option 5. Here, a block-to-line buffer in the display unit is used to transfer the block-based data for line-based display. As opposed to Option 4, this solution is more expensive in terms of silicon area due to the buffer, but for P-pictures Option 5 consumes less memory bandwidth. Option 6, 7, and 8 are similar to Option 3, 4, 5, respectively, extended with a cache for motion-compensated prediction. Again, Option 9, 10, and 11 are similar to Option 3, 4, 5, respectively, but now extended with embedded compression. Option 12, 13, and 14 are again similar to 3, 4, and 5, but now extended with both caching and embedded compression. In these configurations, we assume that caching is done in the compressed domain. Consequently, the effective size of the cache is doubled leading to an additional performance gain. Option 13 and 14 feature all presented techniques and reduce the bandwidth of the reference architecture from 124 MByte/s to an astonishing 36 MByte/s, a notable reduction of more than 70 %.

This chapter has shown that when a careful analysis of the video memory traffic is made, substantial reduction of the scarce memory bandwidth can be obtained. We have presented a number of options to be considered for achieving a large reduction. Table 6.3 shows that the external memory bandwidth is scalable by selection of the architecture options, depending on the desired bandwidth, constraints from the architecture, and the picture quality. For example, if the architecture is predetermined (this happens regularly), then only a subset of the architectural options may be feasible. Once the possible options are known, a suitable mix of the presented bandwidth-reduction techniques can be chosen.

Facilius per partes in cognitionem totius adducimur
(Seneca, c.4 BC – c.65 AD)
We are more easily led part by part
to an understanding of the whole

Chapter 7

System study on MPEG-4 decoding

FROM *The main observation from this thesis is that communication is the most critical part in the design of future systems which are become increasingly software oriented due to the demand for more programmability. The rationale on what functionality of an application is implemented in hardware, software, or a hybrid mixture of hardware and software depends on many technical and non-technical arguments. This chapter describes the process of hardware-software partitioning by means of an MPEG-4 decoding example and discusses a suitable architecture called* Eclipse. *For understanding of its concepts, we start with discussing the potential use of communication networks from existing media processing systems and discuss their novelties and shortcomings. Subsequently, the chapter discusses the adopted concepts to address the communication problem with an adequate amount of flexibility for future consumer systems. Then, the functionality and the characteristics of each task in the MPEG-4 decoder is presented, leading to hardware-software partitioning. The chapter concludes with a system proposal for a heterogeneous system.*

7.1 Introduction into hybrid systems

Chapter 1 through 6 have presented the challenges for the design of video processing functions into multimedia systems for consumer devices such as

digital television and set-top boxes. They discuss the tradeoff between system costs on one hand and sufficient flexibility on the other hand. Clearly, there is a trend for an increasing amount of software in SoC solutions. Consequently, future multimedia systems will contain an increasing amount of programmable media processors and some dedicated hardware to considerably accelerate the performance of the system at relatively low cost. Hence, the design strategy of a new product starts with adopting a flexible platform of programmable processor cores, followed by tuning of the architecture and extending it with application-specific hardware units. Although the system as described in Chapter 4 offers a flexible solution with high computation performance, the presented implementation does not execute an application with a mixture of tasks hardware and software that intercommunicate without going to the off-chip memory. It only executes functions in hardware and software that are clearly separated in distinct subsystems. For flexible hardware-software partitioning of fine-grain inter-communicating tasks, the system requires a mixture of CPUs and application-specific hardware that can highly interact. Moreover, a single communication bus or switch matrix as described in Chapter 4 becomes rather complex for the communication-intensive application that we target for. Hence, it is the aim of this chapter to analyze a versatile application that requires hardware-software (HW-SW) partitioning with high flexibility, for a system that features an effective concept for high-throughput communication.

As a representative application for a realistic future system, we discuss MPEG-4 decoding due to its characteristic properties. MPEG-4 is the first coding standard that particularly addresses the trend of converging television and computer technologies and comprises a very diverse set of computational tasks within one standard, leading to a versatile System-on-Chip (SoC). High-throughput stream-oriented data processing is combined with highly data-dependent irregular processing with complex control flows and requires an integral design of both the hardware and the software. Video and still-texture coding, 3-D wire-frames, animations, VRML-like descriptions of the scene, and interactivity are combined with 3-D video rendering (and 3-D rendering of MPEG-4 audio).

To determine the critical issues for an MPEG-4 decoder system and to learn from the past, Section 7.2 reports on existing or proposed LSI implementations of media processors. Moreover, it will work towards the concept of a novel communication infrastructure to meet the high-throughput requirement, combined with flexibility at reasonable costs. Subsequently, Section 7.3 presents the adopted design template and indicates how the

platform architecture can be extend and tuned for the application. Prior to discussing the actual HW/SW partitioning, Section 7.4 gives an overview of the functionality of an MPEG-4 decoder. Then, Section 7.5 outlines the analysis of the processing tasks within the decoder to categorize the tasks by different processing characteristics. In the successive step made in Section 7.6, each task is partitioned for implementation in hardware or software, depending on its category. To show how the partitioning is applied for the adopted design template, Section 7.6 illustrates an example implementation. Finally, Section 7.7 draws some conclusions and provides an outlook towards future research.

7.2 Analysis of various MPEG-4 architectures

This section discusses the concept of some existing or proposed media processing systems that could potentially be adopted for our targeted MPEG-4 decoder. For the aim of this chapter we will particulary discuss the characteristics of the communication infrastructure, because this is an important criteria for its applicability in the MPEG-4 decoder system. Also we will deal with the amount of flexibility and the amount of parallelism. The first system represent the television processor system (TVP), which was thoroughly discussed in Chapter 4. The secondly discussed system is the TMS320C6x, a popular and powerful Texas Instruments media processor. This media processor consists of a VLIW architecture and does not have a global communication infrastructure as compared to the other systems. However, because the mapping of MPEG-4 decoding onto the processor has been reported, it remains an interesting system to consider. The thirdly evaluated system is the extended M-PIRE, a flexible system that is dedicated for MPEG-4 encoding and decoding [88]. The fourth system, Imagine, is a highly parallel media processor [72]. Finally, we discuss the system as finally applied for our targeted MPEG-4 decoder.

- *TVP* – As mentioned before, the TVP system as described in Chapter 4 is particularly efficient in terms of parallelism. This is mainly caused by its dedicated processors. Nevertheless, this system provides an adequate amount of flexibility. The task graph can be programmed and the Kahn process network enables execution of functions independent of resolution, frame rate, regularity of the processing, etc. and without the necessity of constructing static schedules for all possible modes of operation. However, the presented implementation only features communication between tasks on a CPU and dedicated hardware by decoupling them via large buffers in off-chip memory

and therefore limits the flexibility. Moreover, the switch matrix does not scale properly towards more complex and dynamic video processing functions such as MPEG-4 decoding. This requires a hierarchical communication network as will be explained later.

- *TMS320C6x* – Budagavi *et al.* [89] 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, of which the latter is required for our application, shows a factor of 21 increase for the required decoding throughput in macroblocks/second, assuming an optimal utilization of the DSP. Extrapolating this number results in a required performance of 33,000 MIPS, without considering the increasing complexity that is required for grayscale alpha shape, interlaced video and sprite decoding. Bauer *et al.* [90], show an even higher MIPS requirement for the implementation on a DSP. Although a fully programmable system offers maximal flexibility, it lacks the computational performance compared to dedicated hardware. Parallelism can only be exploited at instruction level and the process technology of VLSI limits the speed for sequential processing. Because the TMS320C6x consists of a single VLIW processor, it does not contain a global communication infrastructure.

- *Extended M-PIRE* – Although an implementation in software results in maximal flexibility, we concluded that the required computational power is nowadays not sufficient for real-time decoding of high-definition video. To solve this problem, Berekovic *et al.* [91] proposed some extensions by adding function-specific blocks into the data path of a RISC processor. Their proposal addresses the instruction-level parallelism (ILP), which gives a partial solution to achieve an suitable 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. An example of such an approach is outlined by Berekovic *et al.* [92]. They propose an MPEG-4 codec, which is an extended version of the M-PIRE system [88]. It integrates the above-mentioned modified RISC core as Stream Processor for mainly bitstream parsing and variable-length coding, two DSPs for audio and acoustic echo cancellation, and a 64-bit dual-issue VLIW Macroblock Engine for all the block-based video processing. To relax the tremendous communication requirements between the Stream Processor and the Macroblock Engine, they added

a shared dual-port memory to interconnect both processors, next to a central AMBA AHB bus [93] to off-chip memory. This system is fully programmable and seems a very flexible solution. Although the processors are fully optimized for the MPEG-4 function, the system still offers all the programmability and thus flexibility by means of the software implementation. However, for high-definition video decoding of bitstreams based on the Main Visual Profile, the computational performance is still insufficient. To boost the performance, we propose to increase the amount of effective parallelism to an even higher level. Moreover, the communication network of the M-PIRE to connect the multiprocessor arrangement is based on a AMBA AHB bus which does not offer sufficient bandwidth performance for our high-throughput application.

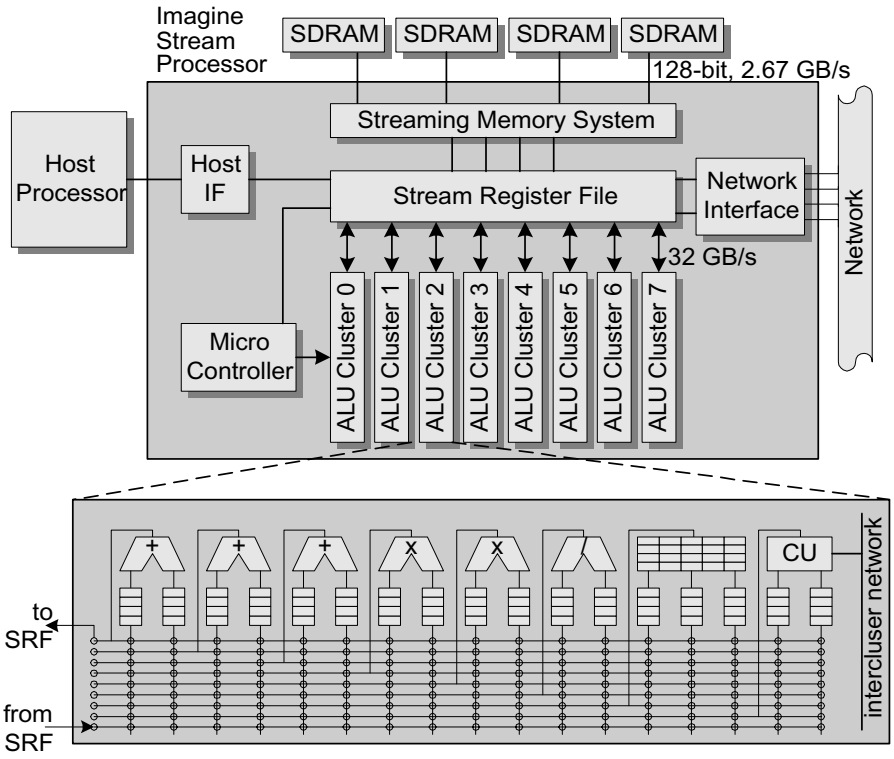


Figure 7.1: *Block diagram of the Imagine media processing system with its hierarchical communication infrastructure.*

- *Imagine* – Just as the M-PIRE, the Imagine is a fully programmable multiprocessor system [72] [94] (see Figure 7.1). The Imagine does

not contain any heterogeneous specializations, but instead consist of a homogeneous structure of eight parallel arithmetic clusters and is in that sense more flexible. Each arithmetic cluster contains eight functional units that are controlled with VLIW instructions, thereby enabling parallel operation of all functional units within a cluster. Due to the generic character and the high amount of parallelism, it is hard to efficiently utilize all this parallelism, leading to inefficient computational performance. However, the system comprises a novel communication approach. It consists of a communication infrastructure with three hierarchical levels to optimize the data bandwidth. The communication for a sequence of VLIW instructions, performing a task within an arithmetic cluster, is kept local by means of a *local register file* (level 1), and only the final results of the task are streamed to a larger *shared register file* (level 2). Only the streams that cannot be stored in the on-chip memory, such as the reference pictures of an MPEG-2 encoder, are communicated to off-chip memory (level-3). Thus, the hierarchical approach prevents exhaustive communication to the bandwidth-limited off-chip memory. Moreover, this efficient approach is scalable toward more complex future systems. The natural hierarchy in applications by means of operations, tasks, and functions can be exploited by such systems with hierarchy in the communication infrastructure. Thus, to scale the system for more complex applications, additional levels of communication hierarchy may be added.

Table 7.1: *Novelties of the various systems.*

System	Novelties
- TVP	- data flow communication instead of static scheduling - guaranteed communication throughput and latency, making the system fully predictable
- TMS320C6x	- fully programmable for maximal flexibility but lacks processing power
- M-PIRE	- heterogeneous mixture of application-specific programmable hardware for flexible yet powerful processing
- Imagine	- hierarchical communication infrastructure to improve data locality - fully programmable with a large amount of parallel general-purpose operations (ILP) - single thread of control

The aim of this chapter is to analyze a highly versatile MPEG-4 decoder that requires hardware-software (HW-SW) partitioning for a system that

features an effective concept for high-throughput communication. Therefore, we have adopted an architecture template that combines the novelties of the above-mentioned systems as indicated in Table 7.1. The adopted architecture provides a heterogeneous mixture of dedicated processors as contained in the TVP system, together with programmable media processors as provided by the M-PIRE. Hence, this heterogeneous mixture of subsystems provides an adequate amount of flexibility. In addition, it features the Kahn process network for easy programming, similarly to the TVP system. To provide an effective and high-throughput communication network, the adopted architecture template inherits the hierarchical communication concept of the Imagine. Section 7.3 gives an overview of the adopted architecture template.

7.2.1 Grain size of parallelism and the amount hierarchy

The previous subsection has discussed various media processor architectures and presented the novel aspects that are combined for our MPEG-4 decoder system. However, at this point it may be difficult to comprehend the reasoning behind this architecture. Therefore, this subsection elaborates on the concept of the hierarchical communication infrastructure.

Chapter 2 has explained how programmable digital signal processors showed an increase of the amount of parallelism by going from a processor with single-instruction-single-data (SISD), to single-instruction-multiple-data (SIMD), instruction-level parallelism (ILP), and finally to task-level parallelism (TLP). Each type does not replace the successive type of parallelism, but rather adds an additional *hierarchical* level to the system. Consequently, currently deployed systems [77][95][96][6] comprise multiprocessor and coprocessor architectures with all types of parallelism. These systems typically contain special-purpose hardware to further increase the performance density (i.e. the application throughput per unit area and per unit power). Basically, this means that not all levels of the hierarchy are programmable.

To design a system with a large amount of parallelism with a high throughput communication network, it is most important to provide a proper balance between the granularity of the parallel functional units and its corresponding level in the communication hierarchy. For example, Bove and Watlington [97] presented an architecture in which task-level parallelism is provided with processors containing a large amount of relative fine-grain functions. All functional units were interconnected by means of a large crossbar. Although this is a flexible approach, the bandwidth requirement

of the communication network increases significantly for complex applications, requiring an expensive communication structure to solve the bandwidth bottleneck. Using more coarse-grain functional units [77] to increase the computational performance is usually undesirable, because the special-purpose character of coarse-grain functions limits the reuse for different applications and different platforms.

Alternatively, a *hierarchical communication* infrastructure offers an effective solution. For example, the above-mentioned coarse-grain functional units are replaced by a subsystem containing its own communication network, together with fine-grain functional units. Consequently, this subsystem can be programmed to perform a particular function, e.g. sharpness enhancement. With the higher-level communication network, several tasks or functions can perform e.g. all the video processing of a TV. To tradeoff the effective parallelism, the communication efficiency, and the amount of flexibility, the following design choices should be made:

- a flexible (programmable) versus hardwired interconnection of the functional units at each hierarchical level, e.g. a VLIW media processor or a dedicated noise-reduction function;
- the amount of parallel functional units at each level;
- the number of hierarchical levels;
- the deployment of communication memory in the hierarchy (see next subsection).

Properly balancing these design parameters will combine the benefits of flexible fine-grain functional units with the communication efficiency of coarse-grain functions.

7.2.2 Memory in hierarchy communication

Apart from different hierarchical levels in the communication infrastructure, the deployment of embedded memory into this communication infrastructure is another point of attention. Typically, computer and media architectures contain a hierarchical memory infrastructure going from small high-speed memories, locally embedded near the central processing units, to relatively slow and large memories located off-chip. Patterson and Hennessy [98] describe the classical techniques for 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 are reused (temporal locality) within the memory space that is contained at a lower memory level. However, for streaming applications the reuse of data is limited, thereby making a cache less effective. Instead, it is more appropriate to apply an alternative approach of exploiting memory hierarchy. For this approach, the streaming data is stored at a separate space in the memory hierarchy, depending on the size of the required memory size and the required bandwidth. For example, the frame buffers for picture reordering and the storage of reference pictures for MPEG decoding 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 relatively slow memories, but to use its large storage capacity to store large chunks of data. The communication of fine-grain data packets, e.g. DCT blocks or video lines, can be kept on-chip and hence, are never exposed to higher levels of the hierarchical communication infrastructure.

It can be concluded that streaming applications often require a different approach of exploiting memory hierarchy than caching. However, the designer should carefully consider this per individual case. For example, although MPEG decoding is considered a streaming application, the zig-zag scan and quantization tables are accessed for each DCT block (temporal locality) and this property can be well exploited by means of a cache.

7.3 Processor system overview

The purpose of this section is to outline a suitable processor architecture for MPEG-4 decoding, called *Eclipse*. With the term "suitable" we mean that all required architectural elements, such as communication, processing, control, are sufficient for the following MPEG-4 application. Therefore, this section serves as an bottom-up view for solving the design problem. The next section will discuss the MPEG-4 application and acts as a top-down approach for obtaining a useful architecture. In Section 7.6 both views will be combined into a single proposal.

Let us discuss the architecture template for implementing the MPEG-4 decoder. The template is based on the functional hierarchy that is inherently present in video processing functions. It distinguishes fine-grain operations (e.g. operations of a VLIW processor), tasks (e.g. a DCT), and

functions (e.g. 3-D graphics rendering). The hierarchy is expressed by the way these distinct functionalities communicate. *Fine-grain operations* intercommunicate within the task boundaries via e.g. hardwired operations or the internal data paths of a CPU. *Tasks* communicate within the boundary of functions via e.g. a centralized memory bus or a dedicated data path. In other words, the communication grain size matches with the grain size of the functionality. Figure 7.2 shows the conceptual block diagram of the architecture template which contains a *two-level* communication network, each with its own set of processors.

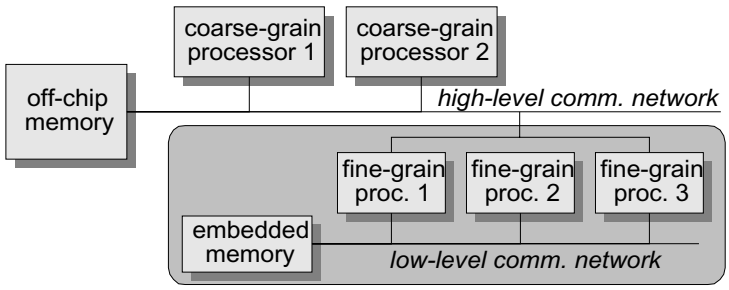


Figure 7.2: *Conceptual block diagram of the two-level system architecture.*

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 memory. For the low hierarchical level, the buffers are stored in the embedded SRAM memory. The following summarizes the advantages of hierarchical communication.

- As already discussed, it enables efficient communication.
- It separates the control of each part in the hierarchy. For example, the details of an MPEG decoder implementation can be hidden within the low-level communication infrastructure, whereas integration of the MPEG decoder in a total system can be applied independently.
- It gives the ability to optimize the design for each part of the architecture. For example, communication between fine-grain operations can be implemented with hardwired connects, whereas high-level communication can be implemented with a data bus for flexible routing of the data, controlled by software. Note that separately trading-off flexibility, cost, performance etc. for each part of the system, generally results in a heterogeneous systems.

Like in Chapter 1, also for the system that is described in this chapter, many design parameters are fixed due to specific constraints. For example, to prevent legacy problems, a common bus specification including the communication protocol, is adopted for the high-level communication network. To allow loosely-coupled independent operation of all processing units in the higher communication level, i.e. in a Kahn process network, communication is provided via large buffers. Consequently, these communication buffers are located in off-chip memory. Control of the high-level processor units 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, e.g. after processing a complete video picture from a media stream, a picture can be processed from another stream.

Because this chapter is mainly focussing on HW/SW partitioning at task level, i.e. the design issues within a function, we will more concentrate on the architecture of the lower hierarchical communication level. As stated before, this part of the system can be designed differently and independently, due to the separation property of the hierarchical architecture template. To give an overview of the characteristics and features of this system part, we will separately discuss the following items: the grain size of computation and communication, synchronization, multitasking capabilities and task scheduling, embedded programmable processors, and runtime reconfiguration of the task graph.

7.3.1 Grain size of computation and communication

At the second level of the two-level communication hierarchy, coarse-grain processors such as a graphics renderer, a media processor, and an MPEG decoder engine communicate and synchronize at the size of video pictures via off-chip memory. At the lower level, tasks such as DCT transformation, Motion Compensation and Estimation (MC/ME), Variable Length Encoding and Decoding (VLE/VLD), etc., communicate via relatively small buffers which are located in a shared on-chip memory.

The granularity at which the data communication is synchronized is relatively small. Thus, little data is written into the communication buffer by a producing task before it is released for reading by a consuming task. Therefore, only small communication buffers are required which can be im-

plemented locally on the chip. By using a *centralized shared* local memory, the system can adapt the sizes of the individual communication buffers at run-time, depending on the dynamic behavior of the implemented function.

The communication traffic that is generated by small-grain processing units is exposed to the low-level communication network, except for the communication to large memory buffers. Because the low-level communication is kept local on the chip, the cost for its flexibility is limited. Let us discuss an example for high-definition MPEG-2 decoding. For this example, four processing units are defined as follows:

- C1 - variable-length decoding (VLD);
- C2 - run-length decoding, inverse scan, and inverse quantization;
- C3 - inverse discrete-cosine transform (DCT);
- C4 - motion-compensated prediction and reconstruction.

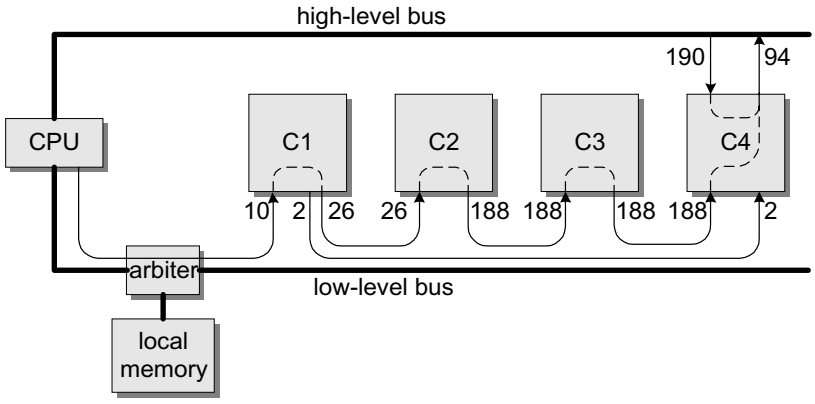


Figure 7.3: *Signal flow through MPEG-2 decoder processing units, including data transport bandwidths (MByte/s) for HDTV signals.*

Figure 7.3 depicts schematically how the data flows through the processing units. Starting as an MPEG bitstream, the data is conveyed from a CPU to processing unit C1 via the shared memory. According to the MPEG standard [7], the maximum bit rate at this point is 80 Mbit/s. Subsequently, the variable-length decoded output is transported to processing unit C2. The denoted 26 MByte/s data bandwidth for this data traffic was determined experimentally by means of measurements. In addition, 2 MByte/s of bandwidth is required for transport of meta data such as the

motion vectors. The output of the inverse quantization consists of 12-bit coefficients. For simplicity, we assume that these coefficients are packed into 16-bit scalars, thereby introducing a 33 % overhead. Hence, a coefficient bandwidth of 188 MByte/s is required. Next, inverse DCT is performed. Similarly, the 9-bit output is conveyed in 16-bit scalars. Finally, the decoder output is reconstructed by adding the motion-compensated prediction. Note that the data traffic for prediction and the reconstructed output is communication via the high-level communication bus to off-chip memory and is not exposed on the low-level bus. The total bandwidth requirement for the low-level communication bus serving this function is $10 + (2 \times 2) + (2 \times 26) + (4 \times 188) = 818$ MByte/s. The communication bandwidth on the high-level bus remains equal to the case when a rigid dedicated MPEG decoder would have been adopted. We can conclude that the high-level communication requirements are determined at functional level, whereas flexibility is provided at task-level.

7.3.2 Synchronization

The chosen architecture template contains hardware for the low hierarchical level to support Kahn process networks [62][99], enabling data-driven processing. To provide a *scalable* architecture template, all processing units provide autonomous operation without external control. Therefore, the communication control is distributed over the processing units. For example, each processor unit provides its own data synchronization, start and stops its processing tasks, etc. Due to the data-driven processing, activation of the tasks and synchronization of the data is performed at runtime by the hardware, thereby providing 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.

Implementation of buffer synchronization

A communication buffer in the shared memory can be observed as a FIFO in which the producer writes data and the consumer reads data. Both the producer and the consumer autonomously maintain some bookkeeping on the read and write pointers in the FIFO. To implement the typical FIFO behavior, cyclic addressing of the pointers is established. This is conceptually shown in Figure 7.4. Because both the producer and consumer task have the knowledge on access points A and B, synchronization can be provided easily. The essence of this proposal is as follows. If the consumer requests a buffer window that exceeds the producer access point A, the

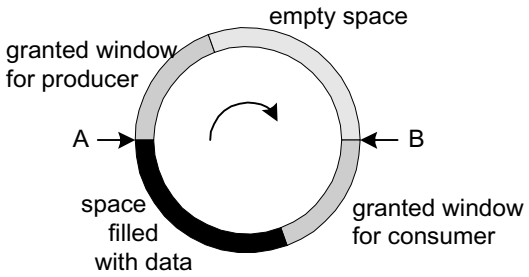


Figure 7.4: *Data communication via a cyclic FIFO buffer.*

request is not acknowledged because insufficient data for reading is available. In the TVP system that was discussed in Chapter 4, the absence of data causes the processor to block, i.e. a *blocking read*. However, in the Eclipse system the processor is not blocked. Instead it tries to see whether another task that is mapped onto the processor can be started (see next subsection on multitasking). If the producer requests a buffer window that exceeds the consumer access point B, the request is not acknowledged due to insufficient free memory space in the FIFO buffer. Also in this situation, the processor will try to progress with other tasks, instead of a blocking.

The novelty of this synchronization mechanism is that it separates the synchronization from the communication [100]. Thus synchronization by means of requesting and clearing a buffer window still allows to access the window byte-by-byte or even in a random order. Moreover, any buffer size or window size can be applied. This is an important feature, because the grain size of communication together with the buffer sizes mainly determine the allowed dynamical behavior of the processors.

7.3.3 Multitasking

Hardware sharing is provided in the architecture by its multitasking capabilities [101], i.e. to simultaneously reuse hardware resources for similar tasks. The main implications of multitasking are the need for task scheduling and task switching on each processing unit. Each processing unit is autonomous and responsible for control of its own multitasking.

Task scheduling for fine-grain tasks

The task scheduler decides when, which, and how often a processor unit should execute each task to attain sufficient progress. Typically, the workload of a task is data dependent and can vary dynamically. Consequently,

the scheduling is performed at runtime to manage the highly data-dependent workload effectively (e.g. for variable-length decoding). Because the task-switching rate for the relative fine-granular tasks is too high for runtime scheduling in software, this control function is performed by dedicated hardware. The scheduling algorithm is based on a round-robin scheme, extended with a weighting scheme for different resource requirements of the tasks.

Task switching and state saving

If the task scheduler decides to stop a current task and to subsequently start another task on the processing unit, a task switch needs to be performed. The *state* of a running task needs to be saved when a task switch is performed. If in a later stage the original task resumes execution, the state information needs to be recovered before proceeding its processing.

For some specific functions, tasks become stateless after processing a data packet. For example, if a DCT task has performed the transform of a complete DCT block, its state becomes void. By forcing the task scheduler to perform a task switch at these stateless moments, no state saving is required. Even though not all tasks are as simple as this example, it can be concluded that task switching is best performed at moments where the state is minimal.

7.3.4 Programmable processors

On both hierarchical communication levels, a DSP, media processor, or general-purpose processor can be included. Moreover, the architecture template allows to connect such a general-purpose processor to both levels of the hierarchical communication infrastructure. This is implemented by means of a special coprocessor connection that is often provided by the general-purpose processor and is used to access the second-level communication network. To allow a high-frequency and low-latency access from the CPU to the local embedded memory, the CPU has a private (and physically short) path to this memory to prevent stall cycles on the CPU. A dedicated arbiter for the local memory guarantees the throughput for all processing units and provides a low-latency access for a general-purpose processors. This is schematically depicted in Figure 7.5.

Because the embedded memory potentially has to serve several processor units and CPUs, it comprises static RAM. This minimizes the need for local buffering near the dedicated processor units and provides a low-latency access for the CPUs. During normal stream processing, the CPU can main-

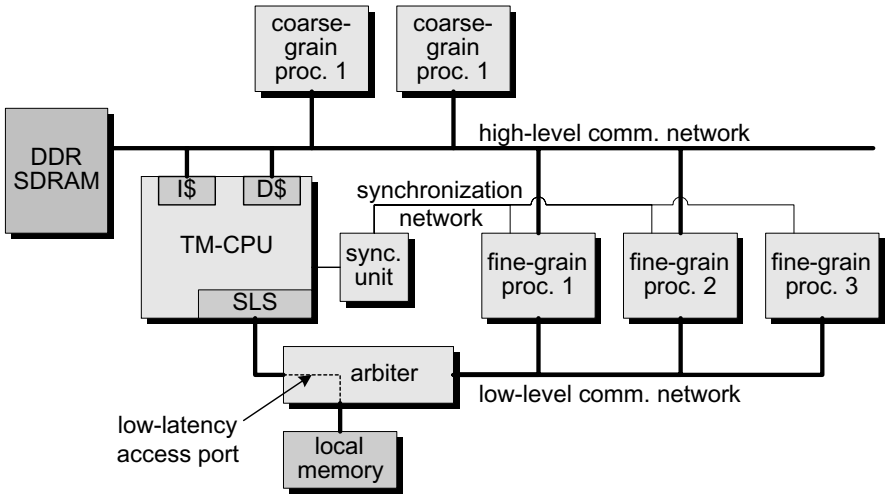


Figure 7.5: *Block diagram of a system containing a general-purpose processor with a connection to both levels of the hierarchical communication network.*

tain a high peak bandwidth for a couple of cycles until its register file is filled up. Subsequently, CPU cycles will be necessary for computations on the data. On a coarser timescale, there will be significant periods of time where the CPU is not at all involved with processing tasks, depending on other responsibilities of the CPU. Therefore, we can expect large variations in the processing rate of tasks that are mapped in software on a CPU. Consequently, if the processing units and the CPUs are tightly coupled with small communication FIFOs, the dynamical behavior of the CPU is passed over to the remaining chain of processing units. Hence, the throughput performance of all processing units needs to be dimensioned for the peaks of the dynamical behavior. Fortunately, the shared memory architecture enables the allocation of more communication buffering for the tasks in software, thereby smoothing the dynamical behavior of the data communication.

The synchronization and task switching of the software tasks is implemented somewhat differently than for the dedicated processing units. The main objective is to achieve this control without a significant cycle penalty for the CPU. Simply implementing the synchronization of relatively small granular data packets by means of interrupts would be highly inefficient. Therefore, an additional"sync. unit"of dedicated *hardware* checks the status for a set of tasks that run on the CPU. When there is enough workload

for a set of tasks, it signals the CPU to start processing these tasks. As a result of this solution, the CPU is responsible for its own task scheduling and synchronization for this set of tasks in *software*. Hence, part of the communication control for CPUs is implemented in dedicated hardware, whereas the remaining part is implemented in software on the corresponding CPU.

7.3.5 Programming of the configuration

A programmable host processor is required to configure the available processor units. This involves several tasks. First, it programs the task graph by means of buffer allocations and assignment of the associated pointers. Secondly, it programs the settings for the distributed task scheduling and the arbiter for the local embedded memory. Moreover, the host processor initiates each general-purpose processor to load the executable code of its application-specific tasks. Subsequently, the system can execute the application by enabling the tasks in the distributed schedulers.

Apart from static configuration of the system, it is also required to reconfigure at run time. To establish this, the configuration manager is able to block a specific task at a specific location in the stream. At run time, depending on the data in the bitstream, tasks are being inserted and/or deleted. An efficient way to perform configuration changes is to first block a producing task, wait until the consuming task empties the FIFO and then reprogram both the producer and consumer tasks.

A simple example how this can be used for the MPEG-4 decoding is as follows. It is possible to allocate new buffers and corresponding decoding tasks, when new visual objects appear in the MPEG-4 stream. Similarly, the buffers and decoding tasks can be released when the objects disappear.

7.3.6 Processor system wrap-up

The local buffer requirements are determined by grain size of communication, i.e. buffersize $\propto$ the bandwidth $\times$ the synchronization period. The presented synchronization concept for the autonomous processor units enables that the application programmer and the processor designers are not concerned with difficult scheduling issues, data synchronization, and buffer management.

The system offers easy tradeoffs between flexibility, cost and performance and effectively provides hierarchical communication for stream-oriented processing, which contains little temporal locality. Moreover, it provides

Table 7.2: *Properties of the adopted architecture template.*

Hierarchical aspect	Related issue	Hierarch. level	Examples and description
Computation	heterogeneity	high- and low-level	dedicated hardware
			general-purpose processor: MIPS
			media processor: TriMedia
	multi-tasking	high-level	centralized control by a host CPU
			at coarse-grain by multi-threading
		low-level	distributed control in hardware by the autonomous processors
			at fine data-grain and high-rate
Communi-cation	two-level hierarchy	high-level	inter-function communication
			coarse-grain synchronization
		low-level	inter-task communication
			fine-grain synchronization
	dynamic data-flow	high-level	run-time task scheduling by software
			data synchronization by software
			centralized control by a host CPU
		low-level	run-time task schedule by hardware
			data synchronization by hardware
			autonomously control by each processor
	data granularity	high-level	coarse grain to limit context switching
		low level	fine grain to limit on-chip buffer size
	configuration	high- and low-level	allows run-time reconfiguration at controllable locations in the stream
Memory	cache hierarchy	high-level	to reduce transfer overhead to reduce latency
		low-level	to enable prefetching to reduce latency
	hierarchy with distinct address spaces	high-level	storage of e.g. video pictures and coarse-grain communication buffers
			located off-chip
		low-level	storage of small data structures, task state, and fine-grain communication buffers
			located on-chip

data-flow communication as proposed by Kahn [62], similar to the system as described in Chapter 4. Hence, the architecture enables execution of functions independent of resolution, frame rate, regularity of the processing, etc. and without the necessity of constructing static schedules for all possible modes. Another important achievement of this architecture is the mixture of dedicated hardwired processing units, fully programmable CPUs, and any hybrid combination. For this arrangement of processing units, the control for synchronization and scheduling is fully distributed over the processing units to make them autonomous and hence scalable towards more complex designs. Obviously, this makes it particularly suitable for unconstrained hardware-software partitioning. More detailed descriptions on the implementation details of the architecture can be found in [21][22][101][102], which show the feasibility of the system for dual high-definition MPEG-2 decoding. Table 7.2 summarized the properties of the adopted architecture template.

7.4 Decoder functionality

Let us now work from the application point of view down to the architecture. 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"[103]. 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 7.6. Not all layers of this model are covered by the MPEG-4 standard; i.e. the Transport layer and Composition/Rendering have been left out explicitly. However, interfaces

between the Transport and Synchronization layers and between the Decoding and Rendering layers are specified.

In order to analyze our MPEG-4 application, Sections 7.4.1 through 7.4.3 outline the particularities of the various MPEG-4 layers, while scene-graph and resource management are discussed in Sections 7.4.4. As a starting point of the mapping of an MPEG-4 decoding system onto the architecture, Section 7.4.5 and 7.4.6 elaborate on the application domains including suitable MPEG-4 Profiles and Levels.

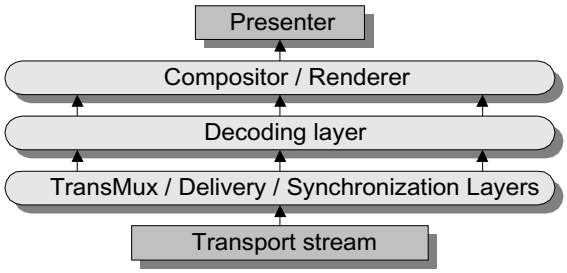


Figure 7.6: *Simplified view of the MPEG-4 decoding system layered model.*

7.4.1 TransMux, Delivery, and Synchronization Layers

The TransMux (Transport Multiplexing) layer offers transport services, while only the interface on top of this layer is specified by MPEG-4. The choice in transport protocol is left to the end user, 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 7.6 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 and composition time stamp, indicating the intended decoding time and composition time of the associated data. The TransMux, Delivery, and Synchronization Layers mainly concern stream parsing and management, and can be classified as event-driven processing.

Architectural essence

This type of processing is highly irregular and contains only a limited instruction-level parallelism (ILP). Moreover, the rate of control is relatively low.

7.4.2 Object decoding

The middle layer of the model in Figure 7.6 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 , etc. may be combined or separated into one or more ESs [104]. The audio objects are coded in separate elementary streams. Object Descriptors are used to relate ESs to media objects within a scene. Object descriptors (OD) themselves are conveyed in one or more ESs, since it is possible to add and discard streams (and objects) during the coarse of an MPEG-4 session. Similarly, the scene description (BIFS - Binary Format for Scene Description) [105] is also contained in an ES, allowing to modify the spatio-temporal layout of the presentation over time. Besides the scene description, the BIFS also contains 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 in 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, and motion compensation. 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.

Architectural essence

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 (data dependencies), and typically requiring irregular data access.

7.4.3 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 composed. Rendering of objects as described in the BIFS takes place in this layer. The

output of this layer drives the presentation devices (displays, speakers). Whether 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. Only the format of media objects and the BIFS are standardized, so that MPEG-4 does not depend on the presentation devices.

Architectural essence

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

7.4.4 Scene-graph and resource 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 ESs 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 control the visual renderer, i.e. to compose 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 3-D scene,
- updating the 3-D scene with temporal behavior (e.g. animation, scene updates), and
- generating visual renderer API calls to draw the scene.

Resource management comprises the control of the available resources in the different layers of Figure 7.6. 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 runtime 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, e.g. the video-object-plain decoder, still texture decoder, and mesh decoder.

Architectural essence

The scene-graph and resource management is event-driven processing and does not require severe computations. Many different routines are executed via data-dependent conditions at a relative low rate.

7.4.5 Application domain for the target architecture

With the architecture depicted in Figure 7.2, 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 [106] 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 will be considered. The MPEG standard allows for 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.

Table 7.3: *Characteristics of some MPEG-4 Visual Profiles @ Levels.*

	Simple Profile @ L3	**Core Profile @ L2**	**Main Profile @ L4**
a	352 × 288 (CIF)	352 × 288 (CIF)	1920 × 1088 (HD)
b	4	8	32
c	396	792	16320
d	396(=CIF)	792(=2×CIF)	16320 (=2×HD)
e	11880 (= 30-Hz CIF)	23760 (= 30-Hz 2×CIF)	489600 (= 30-Hz 2×HD)
f	not supported	not supported	65280
g	384	2000	38400
h	not supported	binary	binary, gray

a = typical Visual Session size
b = maximum number of objects
c = maximum number of macroblocks per video object plain
d = maximum amount of reference memory [macroblocks]*
e = maximum number of macroblocks/sec
f = maximum sprite size [macroblocks/s]
g = maximum bitrate [kbit/s]
h = shape

Architectural essence

The studied MPEG-4 decoder has to be mapped on mid-range to high-end media processors for Digital TV and set-top boxes. Therefore, high-quality digital TV (DTV) broadcast is the most important application domain.

7.4.6 Desired selection of 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 7.3 summarizes some characteristics of these Profiles at their highest Levels [107]. For the graphics part of the MPEG-4 standard, the Simple 2-D Profile is sufficient for the intended application. For the Scene Description, we comply the Simple Profile.

7.5 Analysis of the functions

7.5.1 Decoder framework

After having presented the functions in the previous section, let us now consider the task 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.

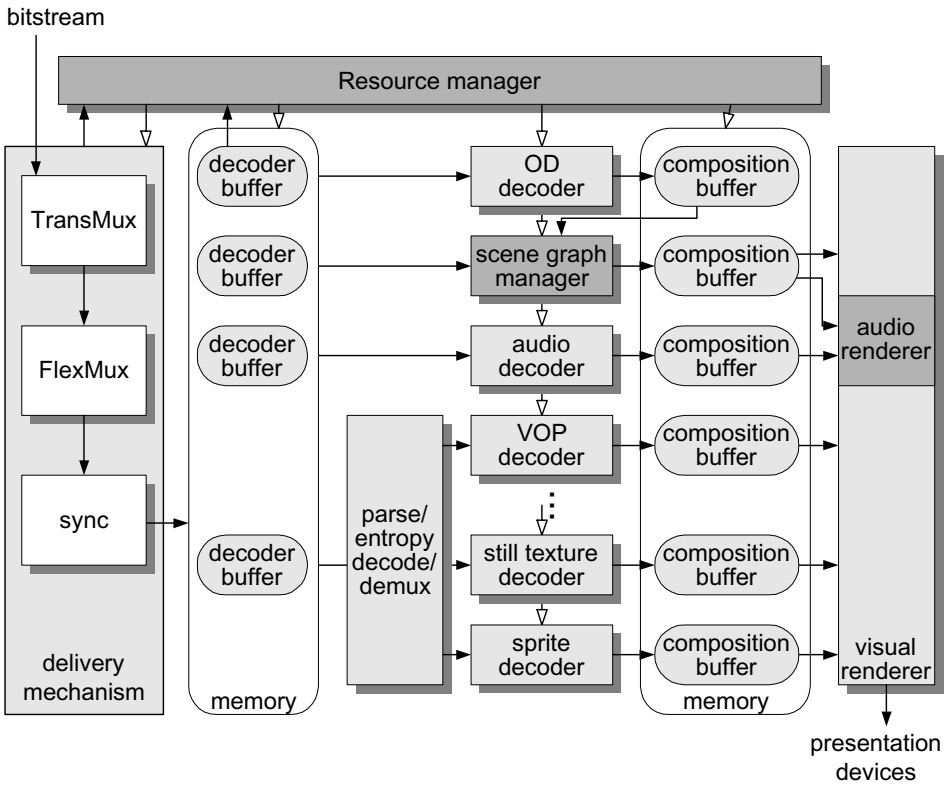


Figure 7.7: *Task graph of the MPEG-4 decoder functionality.*

Figure 7.7 shows the task graph of the complete MPEG-4 decoder system. The delivery mechanism comprises the TransMux, the Delivery and the Synchronization layers as described in Section 7.4.1. 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 extracts the necessary information from 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.

7.5.2 MPEG-4 processing task properties

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 7.4 shows an overview of these tasks, including their processing characteristics. As explained in Section 7.2.1, the amount of parallelism, the throughput, the flexibility and the complexity of control are related. However, it only discusses observations on system level 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 capabilities, and the design time to influence the time-to-market.

Key to the task-properties table

The first columns of the table are self explaining, therefore we focus on the right-hand side.

- *Column 4* – Note that parallel processing can be applied for parallel task graphs, but can also be applied to increase the throughput of a sequential task graph, using pipelining. Therefore, the fourth column in Table 7.4 indicates the throughput requirements of the tasks.
- *Column 5* – 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.
- *Column 6* – The reusability of tasks 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 design. A hardware DCT implementation may be preferred since this is most efficient and used in many cases.

Table 7.4: *Properties of the MPEG-4 decoding tasks that determine the mapping onto HW or SW.*

function	**tasks****	**parallelism*** (column 3)	**throughput** (column 4)	**complexity of control** (column 5)	**reuse over applications** (column 6)	**architecture** (column 7)
Delivery mechanism	a	sequential	low	low	low	RISC
OD decoder		sequential	low	medium	low	RISC
Scene-graph manager	c	sequential	low	high	low	RISC
Resource manger	d	sequential	low	low	low	RISC
Audio decoder		ILP & TLP	medium	low	medium	DSP / Media proc.
Parse/entropy decoder/demux		sequential	medium	high	low	RISC
Still Texture decoder	g	ILP & TLP	low	medium	low	Media Proc.
	h	sequential	low	high	low	RISC
VOP decoder, sprite decoding	i	sequential	high	high	low	RISC
	j	ILP & TLP	high	low	high	dedicated
Renderer, sprite warping	k	sequential	medium	medium	high	DSP / Media proc.
	l	ILP & TLP	high	low	high	dedicated

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

** a = transport protocol, transport stream demultiplexing, synchronization processing
c = BIFS parsing, 3-D geometric scene reconstruction, rendering command generation
d = (re)configuration of the hardware, memory management
g = DC prediction, ZeroTree decoding, inverse quantization, inverse Discrete Wavelet Transform
h = arithmetic decoding
i = Variable Length Decoding, Context-based Arithmetic Decoding
j = up/down sampling, inverse scan, inverse quantization, inverse DCT, padding, VOP reconstruction
k = BIFS browser, geometry, lighting
l = rasterizer set-up, rasterization

The hardware should still offer sufficient flexibility to enable its use in all desired applications. Moreover, if a CPU is already available in the system with sufficient resources, the additional hardware design requires more effort and can be a valid reason for a software implementation.

7.5.3 Hardware/software partitioning

Strategy

Let us now briefly discuss the choices of HW/SW partitioning for each part of the MPEG-4 decoder that is depicted in Figure 7.7. 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.

A. Delivery, OD decoder, scene-graph and resource manager

For the delivery mechanism, the OD decoder, the scene-graph manager, and the resource manager, the choice is straightforward. For these tasks, the computational requirement as well as the amount of parallelism is limited, thereby making these tasks very suitable for a general-purpose processor.

B. Audio

Audio decoding requires a medium throughput. For 6-channel audio, the decoded stream has a bandwidth of 0.5 MByte/s. Compared to video, the audio signal processing has a higher precision. 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.

C. Still textures

For Still Texture decoding a similar reasoning holds. In contrast with 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 parallelism that can be exploited with a SIMD or VLIW architecture, we have chosen a TriMedia [26] media processor.

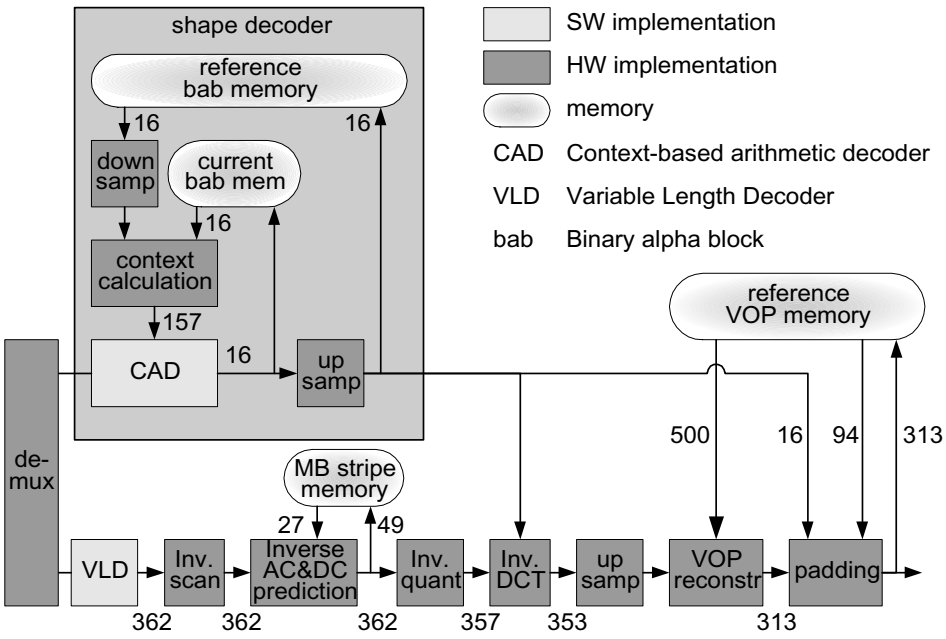


Figure 7.8: *Block diagram of Video Object Plane decoder, including data transport bandwidths (MByte/s).*

D. Video object planes and sprites

For Video Object Plain (VOP) and sprite decoding as shown as a single block in Figure 7.7, the mapping is less straightforward because this functionality consists of a diverse set of tasks, shown in Figure 7.8 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 7.4.6). 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.* [108]. The remainder of the functionality contains a significant amount of ILP and TLP. Moreover, the computational requirements are high, particularly for HD resolution. Note that the VOP decoding requires memory access to an off-chip memory at high bandwidth.

E. Rendering

The bottom row in Table 7.4 represents the rendering and composition of all objects into the final scene. This process is controlled by the scene-graph manager as shown in Figure 7.9. This scene-graph manager operates as a BIFS browser, analog to a familiar VRML browser. The output of the browser can operate at the 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 [109] 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 and lighting functionality in the renderer to create a 3-D geometric representation of the visual scene. Both the BIFS browser and geometry and 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. Since this task performs two-dimensional filtering of all polygons on pixel bases, it is most demanding in terms of computation and memory communication. To provide sufficient computational resources, both the rasterization and its setup are best performed with a dedicated hardware pipeline.

7.6 Mapping proposal

In this section we map the functions onto the overall system-architecture template as presented in Section 7.2, including the architecture of the separate processors in the system.

Figure 7.10 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 7.4.6). Because the memory requirements are mainly determined by the large bandwidth numbers, only these numbers are shown. Memory inefficiency, as described in Chapter 5 is not included. Moreover, the bandwidth indications are based on the worst-case MPEG-4 bitstreams,

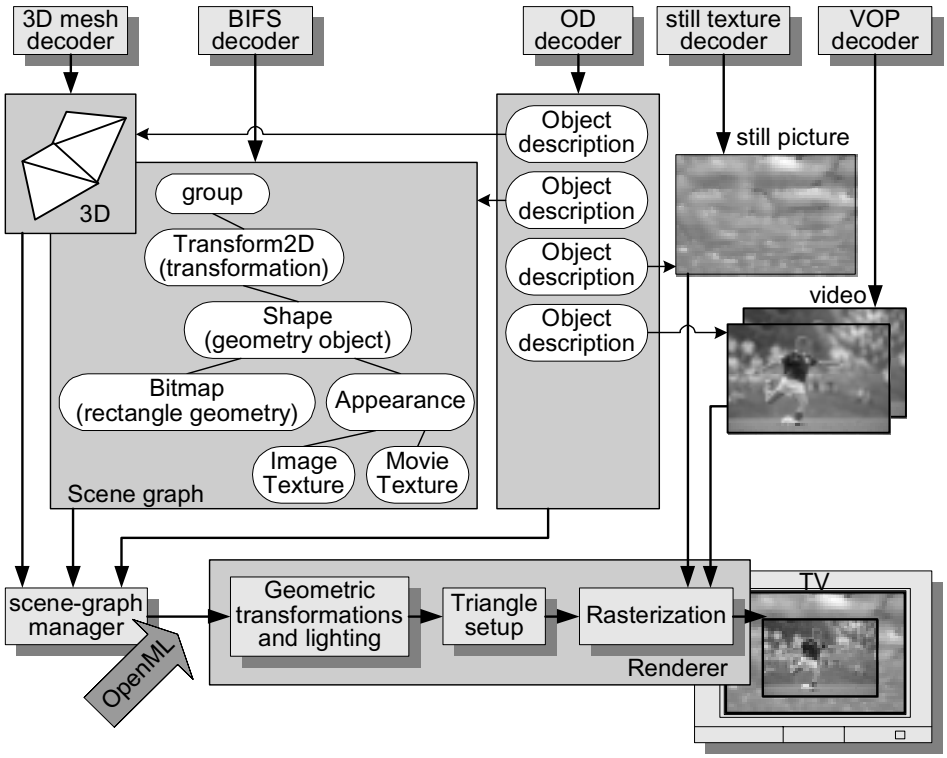
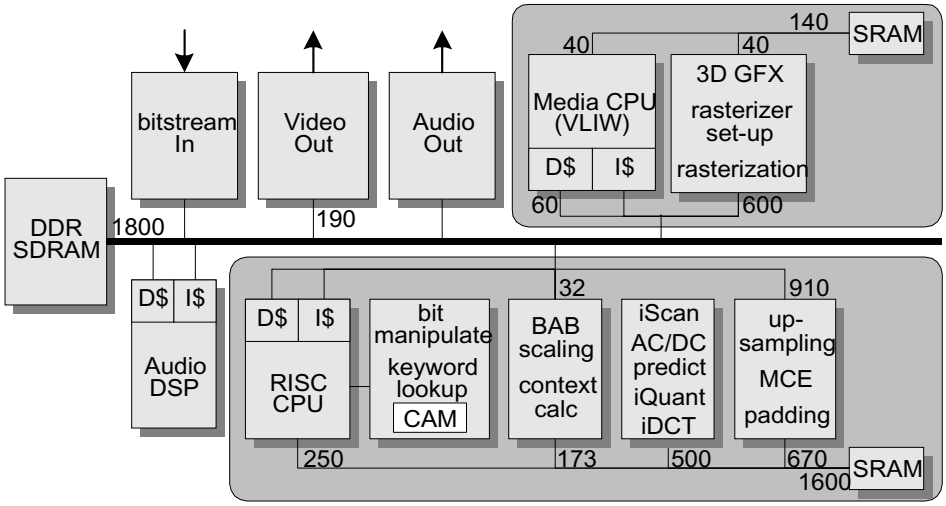


Figure 7.9: *The MPEG-4 rendering process, controlled by the scene-graph manager.*

the bandwidth numbers for rendering are typical, since the worst-case numbers for this function can be almost unlimited.

A. Hierarchy

The two-level hierarchical architecture can be clearly recognized in Figure 7.10. 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 relatively small memory size and communicate via small buffers, are connected to the high-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.



3D GFX = 3D graphics
DDR SDRAM = double-data-rate synchronous dynamic RAM
SRAM = static RAM
BAB = binary alpha block
CAM = context addressable memory
MCE = motion compensation / estimation

Figure 7.10: *Architecture implementation of the MPEG-4 decoding system, including data transport bandwidths (MByte/s).*

B. RISC core

The delivery mechanism in Figure 7.7 is performed by the RISC core. It parses the MPEG-4 bitstream from the off-chip memory, processes it, and writes the output ESs 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 coprocessor connection of 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 many powerful MIPS (e.g. PR3940) RISC cores are relatively inexpensive (e.g. 5.5 mm^2 consuming less than 200 mW).

C. Dedicated hardware

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 this 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 7.8), 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 multitasking capabilities, because a Main Visual Profile bitstream may contain up to 32 independent video objects. The multitasking capabilities described in Section 7.3 are provided in the architecture template.

D. Media processor

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 coprocessor connection without cache.

Besides Still-Texture decoding, the media processor also establishes the software tasks of the rendering functionality, i.e. BIFS browsing, geome-

try, 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.* [110] presented an image rendering coprocessor 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 here. 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). The access to this local memory is included in their results. 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.

E. Composition of the hierarchical communication network

Because the rendering functionality and the MPEG-4 decoding functionality do not share the same computational resources and only have intercommunication 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 would enable the implementation of additional tasks for the VOP decoder in software, thereby anticipating for future extensions.

7.7 Conclusions

When compared to other existing coding standards like MPEG-2 and H.26x, MPEG-4 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. Another observation is that 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. From these observations, together with the demand for a low-cost solutions in the competitive domain of consumer electronics, it can be concluded that hardware support is essential.

For the required combination of flexibility and a high computation performance, a hierarchical communication infrastructure is desired. This allows to tradeoff flexibility and performance at any stage in the hierarchy, while keeping the data bandwidth locally nearby the processing units. Besides a two-level hierarchical communication infrastructure, the adopted architecture template allows a mixture of dedicated hardware and programmable CPUs that can interact at relative small data granularity. Moreover, the systems provided communication according to Kahn process networks and offers multitasking capabilities for easy programming. These novelties enable effective partitioning of applications into hardware and software, without undesired constraints from the architecture.

For the adopted architecture template, an effective HW/SW partitioning was performed based on an MPEG-4 application analysis for aspects 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. From this mapping study, it can be concluded that sufficient flexibility can be provided without raising the required bandwidth to the off-chip memory.

Since the functionality of the MPEG-4 decoder that is mapped onto processing units can be considered as a superset of many other DCT-based coding standards, the MPEG-4 decoder can be designed for encoding and decoding MPEG-1/2 and Motion JPEG formats as well. In fact, the architecture template has already been exploited to perform MPEG-2 encoding and decoding. The mapping study of the MPEG-4 decoder demonstrates the flexibility and extensibility of the media-processing platform.

Longum iter est per praecepta,
breve et efficax per exempla
(Seneca Philosophus, c.4 BC – c.65 AD)
The way is made long through rules, but
short and effective through examples

Chapter 8

Conclusions

FOR many already existing video processing systems, the architecture and the video algorithms are designed independently. This thesis advocates the combined design of video processing algoritms and their corresponding architecture. For this reason, we have explicitly coupled design aspects such as bandwidth, programmability, etc., with an analysis of the video processing algorithms. The first section recapitalizes the most important conclusions of the individual chapters. Secondly, a section is devoted to an example state-of-the-art system design for the television domain. Subsequently, we present a novel design approach which results from the conclusions of the first section and use an example system that features similar functionality as the state-of-the-art system from the second section. Finally, we discuss the most important trends that impact the system design in future and emphasize the relevance of architectural analysis of the video processing functions.

8.1 Recapitalization of the individual chapters

Chapter 1 shows the growing need for more flexibility and programmability in video processing subsystems and more reuse of the design. Unfortunately, this generally leads to most costly solutions and a low computational efficiency. Yet at the same time, video functions keep on growing in complexity and require therefore more computational resources.

Chapter 2 presents a concise but relevant overview of computing architectures, which provide the most important input for the rest of this thesis. The overview of various processor systems briefly outlines ways for improving parallelism in video architectures and also discussing the associated control issues. The presented media processors have revealed the following primary aspects: computational power, communication and memory.

Chapter 3 addresses a detailed discussion of the resource requirements of several video-processing functions in the TV domain. The emphasis is on determining the computational complexity, memory bandwidth and the amount of memory. With respect to these architectural requirements, we have analyzed several video functions. The primary conclusion of this chapter is that implementation of the complete video-processing functionality in a television system cannot be achieved cost-effectively with general-purpose hardware. Instead, the requirement for low costs in consumer electronic domain and the strong heterogeneous nature of the video processing tasks lead to application-specific solutions. Finally, it is concluded that the design of the video processing algorithms and their architecture are interdependent and require an integral design approach, thus a codesign.

Chapter 4 presents a recently developed experimental TV chip set, proven in Silicon. This chapter serves as a clear design example that satisfies the requirements and analysis of the first three chapters. The key to the system is providing powerful video computing at low cost and at the same time introducing adequate programmability. Executing approximately 10 operations billion per second while dissipating 5 Watt, results in a power performance of 5 mW/MOPS for the CPA. The system features flexible ordering of video functions with optimization via programmable key parameters and on-chip inter-processor communication. The architecture is actually the first in the TV domain that applies a Kahn process network for the video processing functions.

Chapter 5 concentrates on the optimization of off-chip memory access for video processing functions in a system-on-chip. Generally, this part of the system causes a system bottleneck with respect to communication bandwidth and power dissipation. Up till now, multimedia architectures assume a transparent shared memory with unlimited bandwidth and a full efficiency. However, with many video processing tasks running in parallel, this assumption is not valid and actually yields considerable efficiency losses. Chapter 5 provided an example that shows an MPEG decoder, in which significant gains can be achieved when matching the application-

specific memory accesses with the behavior of the memory device. The results show that the total memory bandwidth reduces with 35 %, corresponding to 195 MByte/s for high-definition MPEG decoding.

Chapter 6 discusses embedded compression and video-specific caching as additional bandwidth-reduction techniques that are tuned to both the application and the memory configuration. Both techniques show that application-specific system optimizations considerably reduce the required system resources. For the MPEG-2 decoder function a memory bandwidth reduction of 40 % and 32 % was achieved with limited costs for compression with a factor 2 and a 3-4 kByte cache, respectively.

Chapter 7 provides a second architecture example in this thesis, where a set of video processing applications is mapped onto a heterogeneous multiprocessor system. This architecture applies a hierarchical communication network to offer scalability and reuse for future generations and to improve the communication throughput. It offers more local processor to processor communication via a shared embedded memory, thereby preventing exhaustive communication to the shared off-chip memory. Secondly, FIFO buffers for on-chip communication are mapped onto a centralized memory, so that larger effective buffer sizes are implemented, allowing more dynamic behavior of the processor system. Thirdly, a special synchronization unit enables the system to integrate fully programmable processors, thereby offering a heterogeneous mixture of DSPs, CPUs, and application-specific processors.

8.2 State-of-the-art system design

In the course of this thesis (Chapters 3, 4 and 7), it has become clear that conventional TV architectures with hardwired functions cannot fulfill the flexibility and programmability requirements of a future TV system. Recent more complex system-on-chip solutions contain the complete video processing chain between tuner output (CVBS) and RGB processing. Schu *et al.* [111] have presented a system that performs motion-adaptive deinterlacing and up-conversion including special cinematic source processing, picture-in-picture, split screen, 3-D spatial-temporal picture enhancement techniques, adaptive luminance peaking, transient improvement for both luminance and chrominance, as well as color decoding and all necessary A/D and D/A conversions.

Simultaneously with this SoC development of the TV processing chain, set-top boxes for digital reception of the TV signals from satellite, terres-

trial or cable contain an increasing amount of functionality. Apart from MPEG decoding, such a system also contains picture enhancement and processing algorithms for e.g. noise reduction, dual video inputs for picture-in-picture or split screen, graphics rendering, sharpness improvement, etc. Thus, high-quality video processing from the conventional TV broadcasting environment is combined with digital TV decoding. Also for the audio functionality and the system interfaces, the commonality between TV and set-top box systems is large. This merging trend, which is visualized in Figure 8.1, becomes even more true for digital and hybrid TV sets that are currently being developed.

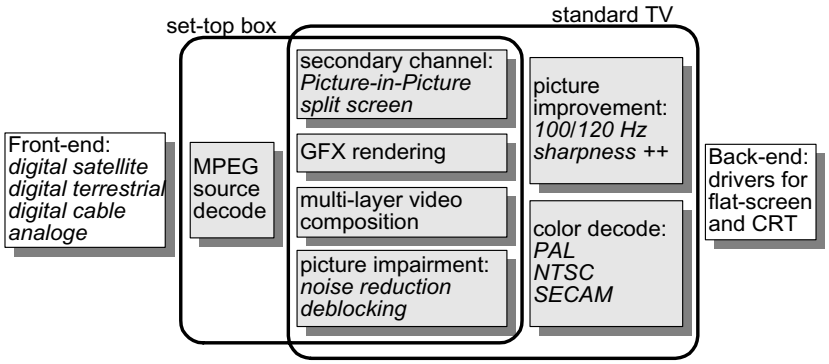


Figure 8.1: *Merging of set-top box and TV systems.*

For example, chip makers[1] offer hybrid solutions for set-top box or TV systems with an impressive list of features, indicating the large amount of TV functions enclosed. These solutions can handle conditional access for up to two digital transport streams, A/V decoding for up to two SD or one HDTV MPEG-2 programs, decoded from the IEEE-1394 interface, LAN or an attached IDE DVD drive, acquisition of up to two CCIR-656 based video sources, temporal noise reduction, temporal-spatial video improvement such as Digital Reality Creation or Digital Natural Motion, histogram modification, black-stretch, luminance and chrominance transient improvement, blending up to five video and one graphics image for an analog S-video or CVBS output, multi-channel audio processing and output via an IEEE-1394 interface, decoding of image files or audio content from a MemoryStick or MultiMediaCard, and a USB connection. Such multi-functional platforms featuring a large set of functions are applicable in a set of different products. The reusability is attractive from a manufacturing

[1]Philips Electronics has released a commercially available chip under the name PNX8525

point of view. The development costs and fixed production costs (e.g. cost of a mask set) can be distributed over a potentially high volume. Moreover, generic hardware solutions enable reuse of design and therefore further decrease development costs of the hardware.

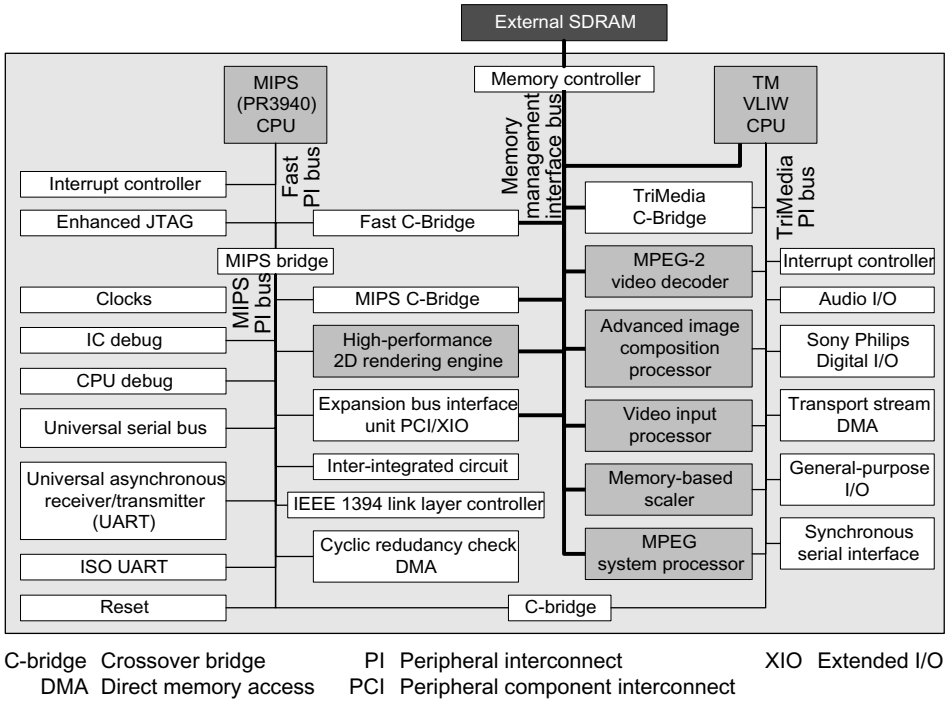


Figure 8.2: *Block diagram of the Viper system.*

Requirements for the previously discussed hybrid set-top box / TV system demand a high level of flexibility, together with a high computational performance at low costs. As a result, there is a clear trend towards systems offering high parallelism combined with a flexible infrastructure to enable communication between all included functional units. For example, a system implementation called the Viper [112] as shown in Figure 8.2, represents a fully interconnected system containing two communication networks: one costly high-throughput point-to-point bus and one scalable and low-cost tristate bus for low-throughput communication and peripheral control. The latter communication network is subdivided into three different segments to cluster the peripherals that require a different cost-performance tradeoff. Most functional units are not implemented for general-purpose processing. They are dedicated for special-purpose video processing, such as a 2-D image renderer, a 2-D sampling-rate converter for scaling, an

MPEG decoder, etc. This large amount of dedicated functional units results in a large computational performance for the domain of digital TV and set-top boxes. Moreover, it provides sufficient flexibility for reasonable silicon costs: 35 million transistors within less than 110 mm^2 of silicon in 0.18μm CMOS technology, consuming 4.5 Watts on the average.

Unfortunately, all video processing units in the system of Figure 8.2 are connected to the off-chip memory by means of one central memory controller. Communication between the processing units can only be achieved via this off-chip memory, thereby introducing a bottleneck in the communication network. As was concluded in Chapter 7, the architecture of complex systems as discussed above require a hierarchical communication infrastructure and a corresponding hierarchical memory architecture. In the following section we will present a hierarchical approach to effectively deal with the system complexity.

8.3 Future system design

In this section we attempt to combine all insights from the thesis to derive a template for future system design. As explained throughout the chapters we commence with the application. To determine the main changes in the final system architecture with respect to traditional systems, we first distinguish the main differences from an application point of view, i.e. the differences in characteristics of the functions (see Table 8.1).
From Chapter 7 we have learned that a hierarchical communication infrastructure can be adopted to effectively deal with the large amount of complexity in computation and communication of future SoCs. This is achieved by matching the hierarchy of the communication infrastructure to the natural hierarchy of the video processing applications. Such hierarchy enables the partitioning into subsystems with independent optimization of their architecture and provides locality of data processing for efficient communication. Figure 8.3 shows an example of a system-level architecture for a future system. From the figure we can recognize many of the following system aspects.

- The architecture is hierarchical, i.e. three different levels can be distinguished.
- At the lowest hierarchical level, clusters of processing elements may consist of both homogeneous structures with multiple DSPs, but can also contain a heterogeneous mixture of DSPs, CPUs and application-specific hardware blocks (ASIP). These ASIPs can be dedicated hard-

Table 8.1: *The main differences between the characteristics of future and traditional applications.*

Traditional video functions	Future video functions
- Mainly pixel processing	- heterogeneous mixture of pixel processing and control based processing
- Low-complexity processing loops without data-dependent branches and operations	- Complex control with many data-dependent branches and operations
- Mainly hard-real time processing	- Both hard-real time and soft real-time processing
- Constant resource requirements	- Dynamical behavior of the resource requirements
- Predictable system behavior straightforwardly implemented	- Predictable behavior under hard real-time constraints; not easily implemented
- Limited temporal processing, resulting in a low latency	- Memory intensive temporal processing, resulting in a large latency
- Relatively few communication and computation resources	- Significantly more communication and computation resources

ware or highly programmable processing units that are dedicated for a particular processing task.

- The number of hierarchical levels may be different per subtree. For example a subtree may comprise a single CPU or ASIP (a leaf), but may also consist of another communication infrastructure with several processing elements. This gradually leads to networks of computational units on a chip. To generalize the the interconnect of this hierarchy of computational units, a network on chip that is scalable toward more complex systems should be adopted, as we will see later.

- The highest level contains an interface to off-chip memory.

Let us now take a future TV set as an application for mapping onto the above-described system template. This is conceptually shown in Figure 8.4. The figure distinguishes three hierarchical levels, indicated by the grey levels. The hierarchy in the communication network is schematically indicated by the connecting lines. Later we will explain that these lines represent a sophisticated network and do not necessarily imply a traditional communication bus. Table 8.2 shows the main video functionality of the system. The partitioning of the TV application into a hierarchical structure as depicted in Figure 8.4, satisfies the following criteria.

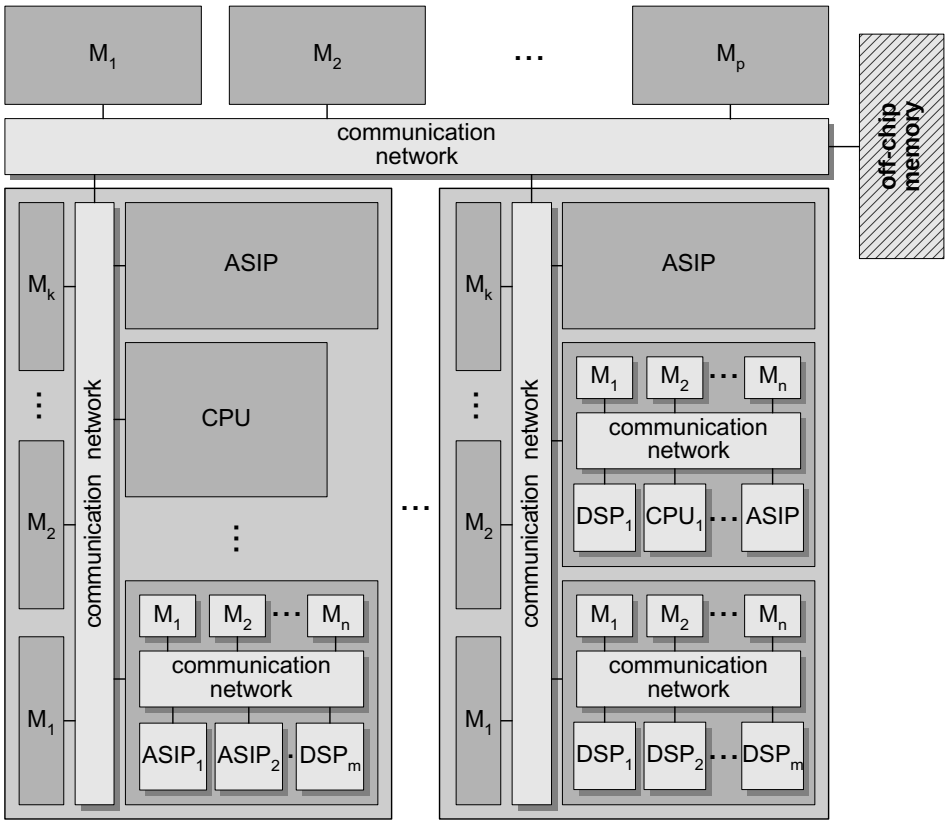


Figure 8.3: *A Future multimedia system, featuring hierarchical communication and a heterogeneous mixture of processing units.*

- Primitive tasks that can be shared by different functions are implemented as an independent subsystem. For example, the motion-estimation block for the display-dependent picture improvement can be used for inverse filtering of the motion-blur in LCD-based TVs, but can also be applied to do 50-to-100 Hz conversion, or to generate subfields for Plasma-based TVs.

- Locality of data is maximized by clustering functions in a subtree that mainly intercommunicate within the subtree. For example, the block denoted as "BAB scaling, context calculation", only communicates within the VOP/sprite decoder.

- Tasks or functions of which the position in the task graph is not fixed, should be implemented as separate subsystems. For example,

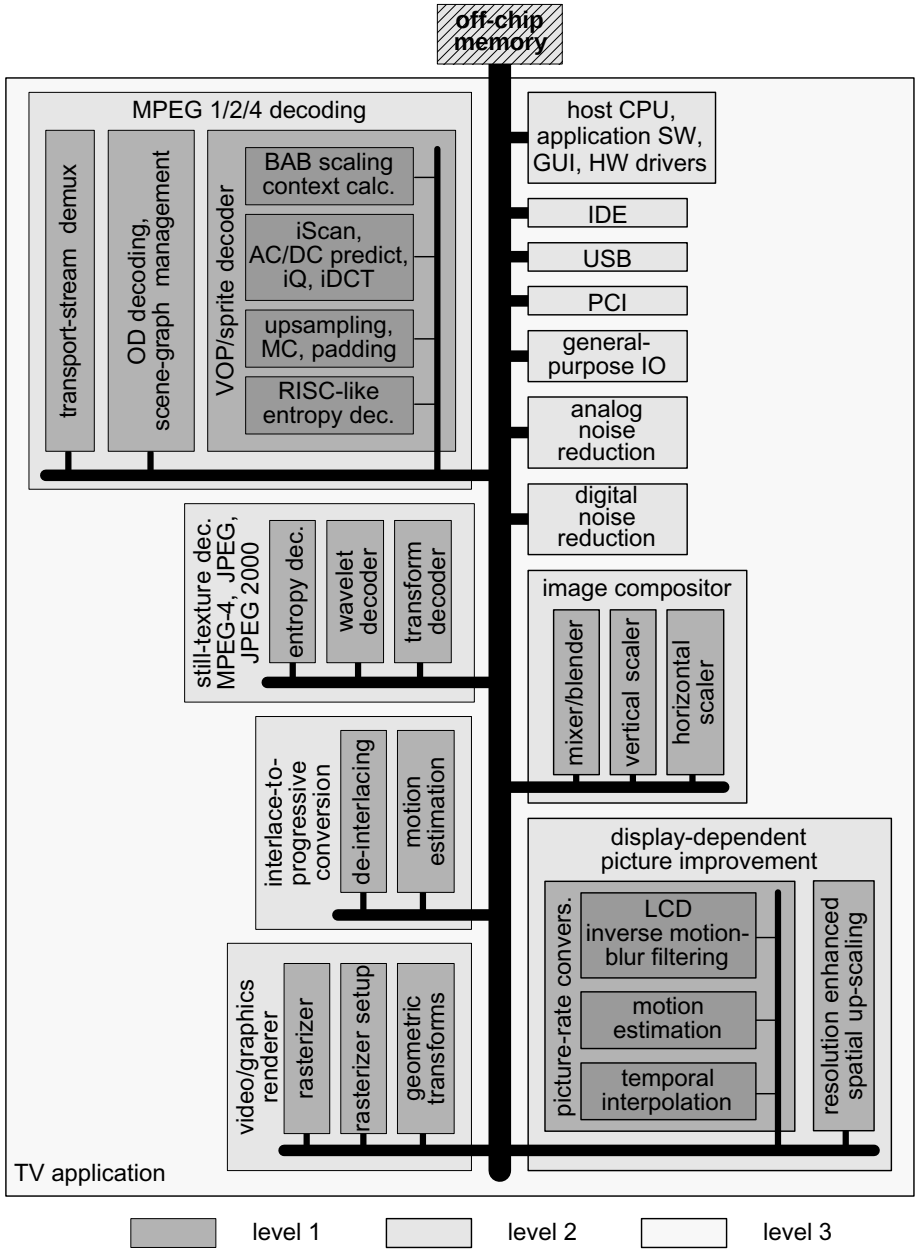


Figure 8.4: *Partitioning of a TV application for mapping onto a hierarchical system-architecture template.*

Table 8.2: *Overview of the main video functionality of a future TV.*

Functionality	Description
Host CPU	A CPU to control and execute the application, generate a graphical user interface (GUI) by means of the GFX renderer, and execute driver software for hardware abstraction.
Interconnections	Interfaces to for example a hard disk drive (HDD), PCI bus, USB peripherals, etc.
MPEG decoding	MPEG-1/2/4 source decoding of video from a tuner, a DVD player, or from the Internet (IP/TCP).
Noise reduction	Removal of the transmission impairments, e.g. analog broadcast noise or quantization noise from digital source coding.
Image composition	Scaling and mixing of several video streams or still pictures, to provide e.g. aspect-ratio conversion, Picture-in-Picture, multi-layer video blending, etc.
Rendering	The mapping video objects onto the display memory. For example, the projection of pictorial content in a 3-D world onto a 2-D screen, according to a 3-D description from the application.
Still-texture decoding	Decoding of still pictures that are used for graphics rendering (e.g. gaming), MPEG-4 decoding, or to enable photograph display from a digital photo camera.
Interlace-to-progressive conversion	Generates progressive video that is required for high-quality video processing in the vertical direction, e.g. vertical scaling and picture-rate conversion.
Picture improvement	Picture quality improvement, e.g. 50-to-100 Hz conversion for CRT, subfield generation for Plasma display, PixelPlus (Philips proprietary resolution enhancement), or Digital Reality Creation (Sony proprietary resolution enhancement).

the interlace-to-progressive conversion can be applied directly after decoding so that all further processing, such as scaling, blending, and picture enhancement, is performed on a progressive format for maximum picture quality. However, as a drawback more system resources are required to process the double amount of video lines. Hence, it can also be attractive to perform the interlace-to-progressive conversion at the final stage, just before the 50-to-100 Hz conversion. Similar reasoning also holds for the motion-estimation block within the image composition subsystem.

- Hardware sharing of ASIPs for multitasking is limited to reduce the complexity. The cost of an ASIP for future systems in terms of silicon

area is mainly determined by its state memory, of which the size is proportional to the amount of concurrent tasks. Moreover, for multitasking with real-time requirements, the throughput of the ASIP needs to be raised. Consequently, multitasking of the ASIP for hardware sharing does not gain much in silicon area but rather increases the complexity of the system. Note that the above-given guideline does not discourage hardware sharing of non-real time tasks, or tasks for different functions that do not execute simultaneously. For example, the motion-estimation block in the picture-rate converter can be used for temporal interpolation (CRT or Plasma-based display) and for inverse motion-blur filtering (LCD-based display), but is never applied simultaneously. Furthermore, still-texture decoding can be used by the MPEG-4 decoder and the graphics renderer, but is not time critical. Therefore, the still-texture decoding tasks can be performed after each other.

- Another criterium for partitioning is the amount of communication resources required for a subtree. For example, the highest communication network in the Figure 8.4 (level 3) connects 13 subsystems. Depending on the communication infrastructure the number of subsystems may be too high. As a solution, this hierarchical level can be subdivided into two separate subsystems, connected to an additional fourth level. However, this may be conflicting with the previously denoted criteria. For example, the highest level could be subdivided in a front-end processing part and a back-end processing part. However, this would be problematic because the still-texture decoder is a shared subsystem for both MPEG-4 decoding (front-end processing) and graphics rendering (back-end processing). Thus the network depth should be balanced with video-specific requirements.

At this point, we discussed the hierarchical partitioning of an application. As a next step we can define the corresponding communication infrastructure. Although the hierarchical structure provides scalability for throughput, it should also be scalable with respect to the physical length of the wires. Thus the lay-out design of the wires on the chip should not impact the timing of the overall system (timing closure). Moreover, besides the scalability issue also *predictability* of the system is an important requirement. Due to the growing complexity of SoCs and the diverse nature of future applications, this is becoming a difficult problem. Particularly for real-time systems such as a TV system, predictability can only be achieved by providing throughput and latency guarantees. Such guarantees make the system more predictable and enable independent design of the subsystems, because their communication do not affect each other. Both the scalability

and predictability issue can be solved by networks on a chip (NoC) consisting of routers and interconnections as shown in Figure 8.5 [113].

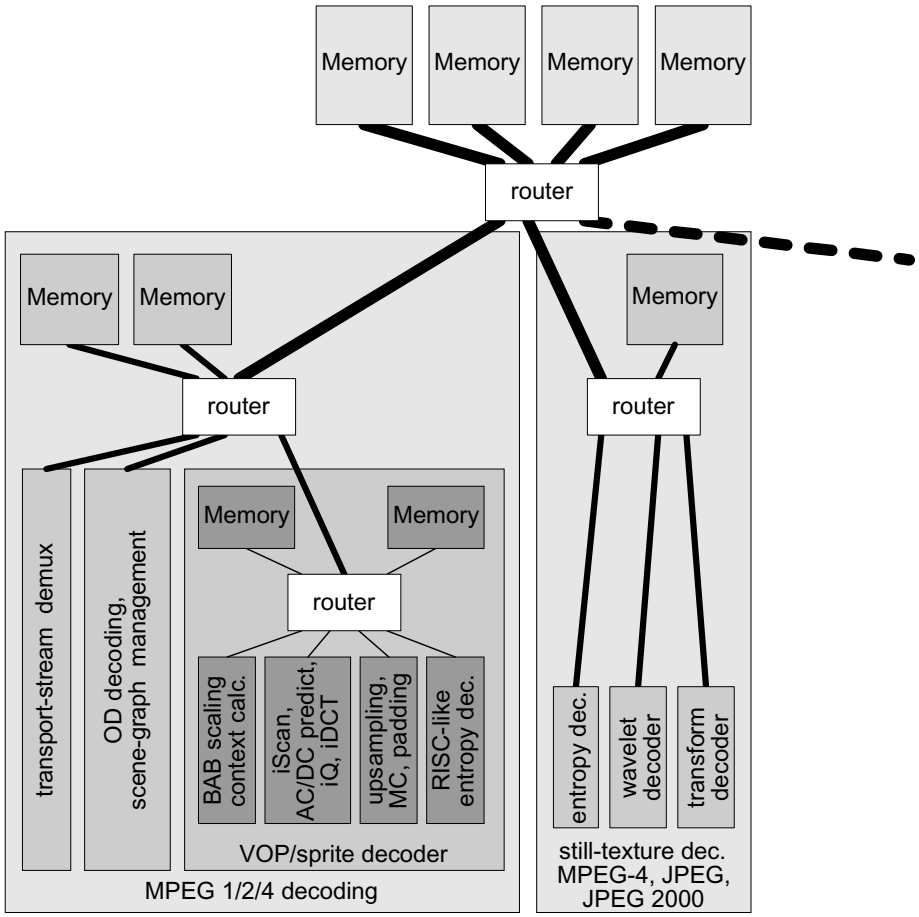


Figure 8.5: *The future TV system with a network on chip for hierarchical communication.*

For the implementation of the routers, we can distinguish circuit switching and packet switching. Circuit switching means that a physical point-to-point connection is established by control signals that configures the setting of the routers. For packet switching the control signals for the routers are coded into the header of each data packet.

For the design network, the amount of physical interconnections between the routers depends on the bandwidth requirements and the latency constraints of the connected functional blocks. Therefore, a traditional com-

munication bus or a switch matrix may still be satisfactory for a part of the hierarchical communication network in the system. In Figure 8.5 each node of the hierarchical communication infrastructure contains only one router. Consequently, between any two functional units in the system only one possible communication path exists. To guarantee sufficient communication resources, it is also possible to provide multiple routers per node. Such an approach enables the routing of data between any two functional units, via several possible communication paths at the cost of more complex routers and more wires.

When taking a bird's eye view on the above-discussed example, the architecture of high-performance multimedia video processing is becoming gradually mature as it increasingly reflects the nature of the underlying processing that takes place. Levels of granularity in computing and memory are also interconnected at distinct levels. This concept will lead to an architecture that can be well matched to the complexity of the tasks to be performed. By applying this approach, the cost efficiency of the system design can be optimized for the application area at hand and requires codesign of the system architecture and its functionality.

8.4 Concluding statement

For future design of consumer electronic systems, the main important starting point is that cost will remain a driving factor. It has become clear that the focus of design effort will shift towards the communication and memory part of the system. Solutions can be offered to make the product broader applicable by means of scalability. The developed solutions still require proper dimensioning of the infrastructure, synchronization of processing tasks, and so on, and once again, they depend heavily on the video functions that need to be executed. This means that the architectural analysis of the embedded video processing functions is indispensable in order to realize cost-efficient designs of SoCs.

Appendix A

Operation of SDRAM-based memories

SDRAM-based memories represent the main-stream market of stand-alone memory devices. Because these devices significantly impact the overall system cost, this appendix is dedicated to the operation of these devices. Dynamic memory cells consist of one transistor and one capacitor which can be charged or discharged, depending on the binary value it should represent. These DRAM cells are contained in an array of column address lines and row address lines, which can be separately addressed. Moreover, the memory device consists of several of such arrays, indicated as memory banks and can be addresses with separate bank address lines (input pins BA0 and BA1). With the Row Address Strobe (RAS) line, the Column Address Strobe (CAS), the Chip Select (CS), and the Write Enable (WE), commands can be issued like select row, read from a column address, write to a column address, etc. Figure A.1 shows the block diagram of the DDR SDRAM architecture.

DRAM-based memories provide a burst-access mode, enabling access to a number of consecutive data words by giving a single read or write command. Because the reading of dynamic memory cells is destructive, the content in a row of cells in the memory bank is copied into a row of static memory cells (the page registers). Subsequently, read and write access to this copied row is provided. The result after the required accesses in the

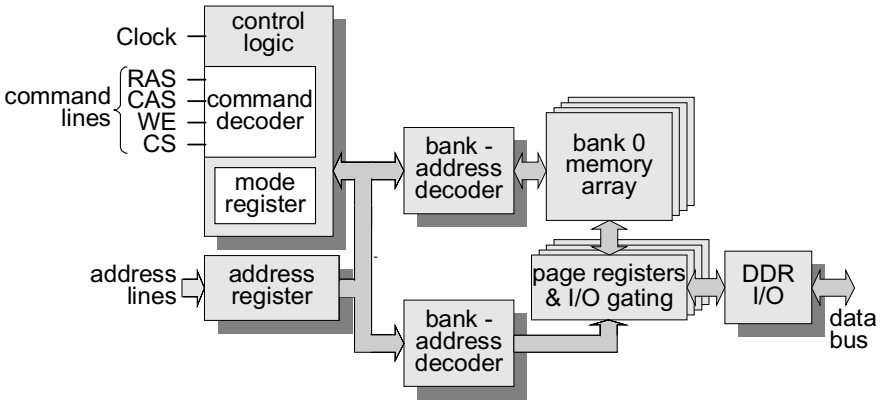


Figure A.1: *Block diagram of a DDR SDRAM.*

row has to be copied back into the (destructed) dynamic cells, before a new row in the memory bank can be accessed. These actions of copying data into the page registers and back, is referred to as row-activation and precharging, respectively. During the copying, which takes considerable time, the associated memory bank cannot be accessed for data transfer. To prevent the loss of precious bandwidth, a multiple-bank architecture is used, where each bank can be accessed alternatingly. Hence, a bank can be accessed while other banks are activated or precharged. Furthermore, high throughput is achieved by dividing the device into stages using pipelining (at the expense of increased latency). When a memory row is activated, random access of the columns within the page registers can be performed. In each bank, only one row can be active simultaneously, but during the random access of the page registers, switching between banks is allowed without a penalty. Therefore, with a four bank device, four rows (one in each bank) can be addressed randomly.

A.1 Memory commands

To explain the usage of SDRAM devices for media processing, this section discusses the memory commands to control the above-outlined memory architecture. Important to know is that SDRAMs provide burst access to obtain a high bandwidth performance. This means that a number of consecutive data words are transferred to or from the memory by giving only one `read` or `write` command. Note however, that several commands are necessary to precharge a bank, to activate a row, and finally to issue a `read` or `write` command, thereby requiring three command-input cycles.

Note furthermore that the data rate at the output is higher (DDR) than the rate at the input (command rate). Therefore, to exploit the available data bandwidth, the read and write accesses have to be burst-oriented. The length of the burst is programmable and determines the number of consecutive column locations that are accessed for a given `read` or `write` command. Burst lengths of 1, 2 or 4 data words or the length of the full page can be programmed for single-data-rate (SDR) SDRAM devices. For current DDR SDRAM devices, the possible burst lengths are limited to 2, 4, or 8 data words. For the DDR-2 SDRAM standard under development, the burst length will most probably be fixed to a burst length of 4. Once a burst length is programmed, the memory rows are divided into successive units equal to the burst length. When a `read` or `write` command is issued, only one of these burst units is addressed. The start of a burst may be located anywhere within the units, but when the end of the unit is reached, the addressing is re-initialized to the first word in the burst unit and subsequently continues until the complete burst is transferred. Starting a burst at another position than the beginning of a burst unit, is called *critical word first* access. For example, if the burst length is four, the two least-significant (LSB) column address bits select the first column to be addressed within a unit. Thus if the binary value of the these bits is 00 the sequential order of word access in the burst unit is 0, 1, 2, 3. If the value of the bits is 01, the order is 1, 2, 3, 0. If the value is 10, the order is 2, 3, 0, 1. And finally, if the value of the LSBs is 11, the sequential order is 3, 0, 1, 2.

The burst length can be programmed in the mode register by means of a `load mode register` command. Since reprogramming of the mode register takes 4 to 6 cycles and requires precharging of all banks, the burst length is usually only set once at initialization and remains equal during operation. Once a read or write burst has been started, it can be stopped with an explicit `burst stop` command or it may be interrupted by another `read` or `write` command. This enables to program a burst length of BL = 8 and read for example only six words. Also such a partial burst access cannot exceed the burst-unit boundaries. Moreover, due to timing constraints memory device may still need more cycles than the length of a short data burst. Hence, short data bursts usually lead to efficiency penalties. As a result, the burst-oriented access either causes transfer overhead when not all data in burst unit is required or decreases the effective memory bandwidth. However, stopping or interrupting a data burst is a particular interesting feature for the SDR SDRAM for which a full-page burst length can be programmed. It enables to access the row at any column position and to stop it when the end of the requested data is reached. Critical-word-first

Table A.1: *Memory command for SDRAM memories*

`desl`	-	Ignore command: When the Chip Select (CS) input is high, all inputs are neglected and the internal status is held.
`nop`	-	No operation: As long as this command is input, address and data input are neglected and internal status is held.
`act`	-	Row active: This command selects a bank and activates an addressed row by means of the address lines BA0 and BA1, and by A0 through A11 respectively.
`pre`	-	Precharge selected bank: This command starts a precharge operation for a selected bank.
`pall`	-	Precharge all banks: This command starts a precharge operation for all banks. This command is useful for refreshing of the device because all banks need to be precharge before a refresh command.
`read`	-	Column address strobe (CAS) and read command: This command starts a read operation from a selected column in the page registers of a selected bank.
`reada`	-	Read with auto-precharge: This command starts a read operation. After completion of the read operation, precharge is automatically executed. This command is equivalent to a `read` command followed by a `pre` command in the same bank.
`write`	-	Column address strobe (CAS) and write command: This command starts a write operation from a selected column in the page registers of a selected bank.
`writea`	-	Write with auto-precharge: This command starts a write operation. After completion of the write operation, precharge is automatically executed. This command is equivalent to a `write` command followed by a `pre` command in the same bank.
`bst`	-	Burst stop in read operation: This command stops a burst read operation and is disabled when auto-precharging is applied. It is not applicable for a burst write operation. Moreover, this operation is not available for DDR-2 SDRAM.
`ref/self`	-	Refresh: This command starts a refresh operation. There are two types of refresh operation.
`mrs/emrs`	-	Mode register set / Extended mode register set: The DDR SDRAM has a mode register and an extended mode register. These are used to program the mode of operation, e.g the burst length.

access does not re-initialize the addressing during the transfer, since the boundaries of burst unit in this case is the beginning and the ending of the complete memory row. In the next generation DDR-2 SDRAM standard which is under development, stopping or interrupting a data burst is not allowed. This decision was taken by the standardization committee to increase the potential speed performance. Consequently, transfer overhead may severely decrease the overall performance. Table A.1 summarizes the complete list of memory operations that can be issued by means of the command lines.

A.2 Timing constraints

To understand how certain data-block transfers can be performed efficiently, this section will discuss the most important timing parameters of DDR SDRAM devices. Table A.2 shows the meaning of these parameters. Some of them are a combination of parameters that can be found in the IC specification and depend on the CAS latency and the burst length. The CAS latency, t_{CL}, is a parameter that is used to define the delay between the start of a `read` command (rising clock edge) and the moment in time that the data from that `read` command becomes available at the outputs. The CAS latency is expressed in terms of clock cycles and is determined by the speed grade of the device and the clock frequency that is used in the application. This parameter is programmed into the mode register, once after power up. For Table A.2, the burst length is assumed to be `BL` = 8 and the CAS latency is chosen such that the device enables the highest possible clock frequency. As explained before, it is necessary to frequently refresh a dynamic RAM. Before a refresh command is issued, all banks in the device need to be precharged. When the refreshing is finished, all banks are precharged and may be activated. The DDR SDRAM can be refreshed by means of an automatic refresh command or by means of a self refresh command. For the first method, a refresh command has to be issued explicitly satisfying the maximum refresh period. In the self refresh mode, the clock signal is disabled, and the device remains refreshed until this mode is left. Refreshing the memory consumes only limited memory cycles (0.5 % of the cycle budget) and is therefore neglected in performance measurements throughout this chapter. Figure A.2 shows how the parameters of Table A.2 influences the timing of the device commands. To access a data unit in the memory, first a row-activate command also called Row Address Strobe (RAS) has to be issued for a bank to copy the addressed row into the page (static-cell registers) of that bank. After a fixed delay t_{RCD} (RAS to CAS delay), a `read` or `write` command also called

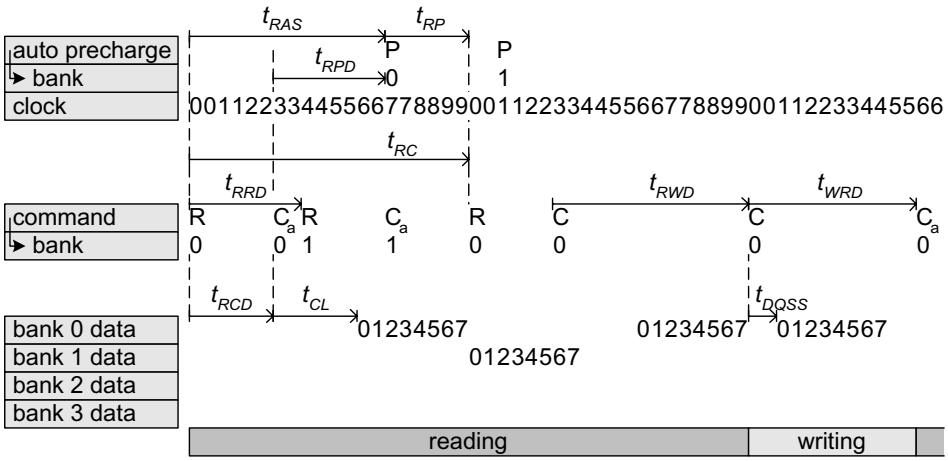
Table A.2: *Legend of timing parameters (minimal latency).*

t_{CL}	=	CAS latency - time between *read* command and first output data
t_{RC}	=	minimal time between subsequent row activates within the same bank
t_{RAS}	=	minimal time between a row activate and a precharge on the same bank
t_{RCD}	=	minimal time between a row activate and a column access (`read` / `write`) in the same bank
t_{RRD}	=	minimal time between row activates in different banks
t_{RP}	=	minimal time between a precharge and a row activate on the same bank (precharge time)
t_{RPD}	=	minimal time between a `read` command and a precharge
t_{WPD}	=	minimal time between a `write` command and a precharge
t_{RWD}	=	minimal time between a `read` command and a `write` command in the same bank
t_{WRD}	=	minimal time between a `write` command and a `read` command in the same bank
t_{CCD}	=	minimal time between a `read` / `write` command and a subsequent `read` / `write` command in different bank
t_{REF}	=	minimal time between a refresh command and the next activate command
t_{REFC}	=	maximal refresh period

Note:

t_{RPD}	=	half the burst length (BL/2)
t_{WPD}	=	half the burst length (BL/2) + time between the write command and the first data input (t_{DQSS}) + write recovery time (t_{WR} or t_{DPL})
t_{RWD}	=	half the burst length (BL/2) + the smallest integer number of cycles larger or equal to the CAL latency, e.g. $t_{CL} = 2.5 \rightarrow 3$ ($\lceil t_{CL} \rceil$)
t_{WRD}	=	half the burst length (BL/2) + time between the write command and the first data input (t_{DQSS}) + internal write to read command delay (t_{WTR}).

Column Address Strobe (CAS) for the same bank can be issued to access the required data units in the row. When all required data bursts in the row are transferred, the corresponding bank can be precharged, which takes t_{RP} time. The time from row-activate command until the precharge may not be less than the t_{RAS}, which is the minimum time a row is active. The precharging is required to prepare the bank for the subsequent row addressing in the same bank. Hence, the minimal time between two subsequent



Note: P is precharge
R is activate row command
C is read or write command
C_a is column command followed by an auto-precharge
t_{RC} is minimal row cycle time
t_{RAS} is minimal row active time
t_{RP} is precharge time
t_{RCD} is minimal RAS to CAS delay
t_{CL} is CAS to data latency

Figure A.2: *Timing diagram of a typical DDR SDRAM device.*

row-activate commands is the row-active time plus the precharge time t_{RP}, i.e. $t_{RC} = t_{RAS} + t_{RP}$. This so-called row cycle time is typically 10 cycles for current DDR SDRAMs (see Figure A.2).

The clock numbers are indicated twice because both positive and negative clock transitions are used to access data. The memory commands are provided at each positive clock transition only. The bottom of the figure shows how four bursts of eight data words are read by four successive accesses, each in a different bank. Obviously, the elapse time between the first data word from the first bank until the last data word from the fourth bank, consumed 32 double-data-rate (DDR) cycles and is equivalent to 16 single-data-rate (SDR) input cycles. At this point in time both bank 0 and bank 1 have exceeded the the row cycle time t_{RC}, a new row-activate command can be issued immediately for those banks, without wasting valuable memory-bus cycles. It can be concluded, that interleaved access of the memory banks provides optimal utilization of the memory-bus bandwidth.

Table A.3: *Worst-case timing parameter values in cycles for maximum throughput.*

t_{RC}	=10	t_{WPD}	=8
t_{RAS}	=7	t_{RWD}	=7
t_{RCD}	=3	t_{WRD}	=6
t_{RRD}	=3	t_{CCD}	=1
t_{RP}	=3	t_{REF}	=12
t_{RPD}	=4	t_{REFC}	=1117

The values of the most important timing parameters from Table A.2 are shown in Table A.3. The values are given in clock cycles. Note that the actual data rate is twice the clock frequency. For example, during the row cycle time t_{RC}, 20 data words can be transferred.

Appendix B

MPEG decoding with reduced memory access

This appendix proposes to slightly modify the MPEG decoding algorithm to further reduce the off-chip memory bandwidth on top of the bandwidth reduction achievement as shown in Chapter 5. For this modification, we assuming that for cost efficiency no embedded buffer memory in the output unit is used (see Table 5.5 and 5.7). As explained in Chapter 5, the burst access mode of SDRAM-based memory leads to an overhead of data transfer when accessing stored pictorial data. Bursts of data can be considered as non-overlapping blocks of data that can only be accessed as an entity. The previous subsection shows how pixel data can be mapped optimally into these burst entities, dependent on access characteristics of the application, e.g. the size and shape of the requested data block and its position in the picture.

Summarizing the required memory accesses for MPEG decoding, we distinguish three different video-data streams. First, motion-compensated prediction of pictures in MPEG. This stream requires the access of block-based data, which may be located at any position in the picture and is therefore not necessarily aligned with the data units in the memory (see Figure 5.11). The second stream writes the reconstructed macroblocks into the memory. These accesses are nicely aligned with the data units and do not cause any overhead. The third stream reads the video line by line from the memory

for display. Figure B.1 shows a block diagram of an MPEG decoder, including the accesses to the memory. Intuitively, it can be noted that the optimal shape (dimensions) of the data units for minimal transfer overhead are similar to the shape of the requested data blocks. For block-based accesses, the shape of the data units should also be block oriented. For line-based accesses, the data units are preferably horizontally oriented.

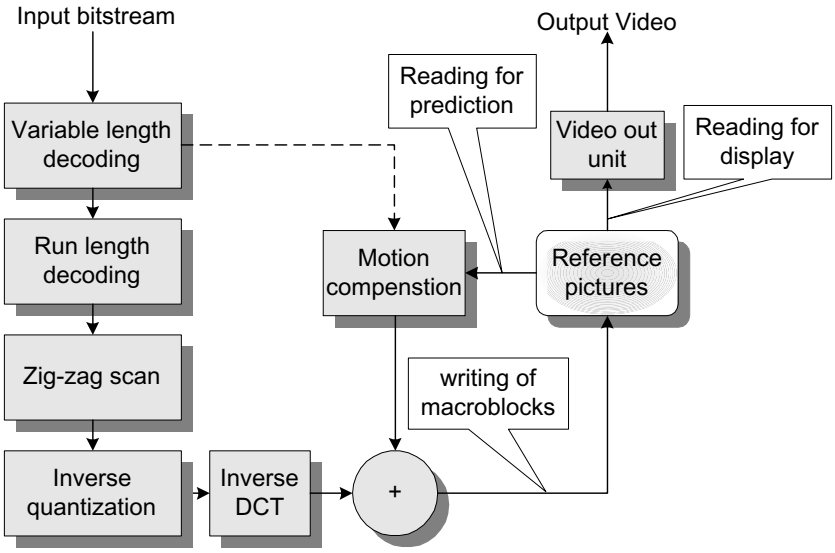


Figure B.1: *Block diagram of an MPEG decoder with indicated memory access.*

Note that the block-based reading for prediction and the line-based reading for display are contradicting with the optimization of the data unit dimensions. To determine the influence of the individual memory accesses to the overall bandwidth requirements, we outline the results of the previous subsection (Table 5.7) in somewhat more detail. Table B.1 shows the bandwidth requirements of the different memory streams for the total MPEG decoding process with data bursts of 64 Bytes. In this case it is assumed that the decoder system does not contain embedded memory for conversion of block-based data units that are transferred for display into a line-based output. The bottom four rows in the table show that data units with dimensions (32×2) result in minimum bandwidth requirements. If data units of (16×4) are used, the line-based reading for display will result in an overhead of 300 %, while the overhead for reading the prediction data will be minimal. If data units of (32×2) are used, the line-based reading for

display is achieved with no more than 100 % transfer overhead. However, the reading of data blocks for prediction is much less efficient.

Table B.1: *Transfer bandwidth for MPEG decoding for various data-unit dimensions.*

Access function	**data unit** ($M \times N$)	**requested** [Bytes]	**transferred** [Bytes]	**req./transf. ratio** [%]
Reading for prediction	(64×1)	478871794	2187744576	457
	(32×2)		1436512832	300
	(16×4)		1116667456	233
	(8×8)		1210167168	253
Writing of macroblocks	(64×1)	455362560	455362560	100
	(32×2)		455362560	100
	(16×4)		455362560	100
	(8×8)		455362560	100
Reading for display	(64×1)	455362560	455362560	100
	(32×2)		910725120	200
	(16×4)		1821450240	400
	(8×8)		3642900480	800
total	(64×1)	1389596914	3098469696	223
	(32×2)		2802600512	202
	(16×4)		3393480256	244
	(8×8)		5308430208	382

As an alternative solution, we propose to write the reconstructed macroblocks twice into the memory; one time for prediction and one time for display. Secondly, we propose to optimize the dimensions of the data units for each storage stream separately to reduce their individual transfer overheads that are caused during reading. Although the double writing of the reconstructed data causes additional data transfer, the transfer overhead for reading of the prediction data is reduced significantly, resulting in a net reduction of transfer bandwidth.

Following this for prediction, we store the reconstructed macroblocks in data units with dimensions (16×4). For display we store the macroblocks in data units with dimensions (64×1). Table B.2 shows the results. The relative bandwidth requirement has reduced from 202 % (Table B.1) to 178 %. Note that the reading for prediction and the reading for display consume minimal transfer bandwidth. Unfortunately, the transfer bandwidth for writing of the reconstructed macroblocks has doubled.

These numbers show the worst-case behavior because we assume that all

Table B.2: *Transfer bandwidth for MPEG decoding with double writing.*

Access function	requested [Bytes]	transferred [Bytes]	req./transf. ratio [%]
Reading for prediction	478871794	1116667456	233
Writing of macroblocks	455362560	910725120	200
Reading for display	455362560	455362560	100
Total	1389596914	2482755136	178

output data also has to be stored as reference data for prediction. This is the case for MPEG bitstreams that only contain I and P pictures. However, most commercially available MPEG encoders also use B pictures to achieve a higher performance (the product of compression ratio × picture quality). For example, the bitstreams we used for the experiments have the following sequence structure: I B P B P B P B I B .....

For such a sequence only half of the data has to be stored as reference data for prediction (only I and P pictures). As a result, the total request / transfer ratio further reduces to 163 %. Another commonly used sequence structure contains two B pictures in between the reference pictures, thereby even further reducing the bandwidth requirements.

Although the aforementioned solution writes the decoded data twice in separate frame memories, the required memory size does necessarily increase proportionally. For the conventional decoder, where the decoded data is only stored once, slightly more than three frame memories are used. In our proposed decoder implementation, four frame memories are needed instead of three, even though half of the output data is written twice. Thus 50 % more data is written, whereas only 33 % more memory is required. This is caused by the inefficient use of the three frame memories in the conventional decoder.

Comparing a conventional MPEG decoder and a newly proposed decoder, the reading for prediction, the writing of the decoded result and the reading for display are scheduled in the three or four frame memories, respectively. For the schedules as shown in Figure B.2 and B.3, the following considerations are taken into account.

- Reading for display and reading for prediction can be performed simultaneously in the same frame memory.
- Reading for display and writing of the reconstructed macroblocks can be performed simultaneously in the same frame memory. Both actions scan the picture in the memory from the left to the right and from the top to the bottom. The read action should be performed before the data is overwritten by the write action.

For the figures, the following glossary of symbols applies:

$W(n)$ - writing of the reconstructed macroblocks with n as the input frame number;
$W_p(n)$ - writing of block-based reference data for prediction with n as the input frame number;
$W_d(n)$ - writing of line-based video data for display with n as the input frame number;
$R_p(n)$ - reading for prediction with n as the input frame number;
$R_d(n)$ - reading for display with n as the input frame number.

frame memory1	*W(1)*	*R_p(1) R_d(1)*	*R_p(1)*	*R_p(1)*	*W(5)*	*R_p(5)*	*R_p(5)*	*R_p(5)*	*R_p(5) R_d(5)*	*R_p(5)*
frame memory2		*W(2)*	*R_p(2) R_d(2)*	*R_p(2)*	*R_p(2)*	*R_p(2) R_d(2)*	*R_p(2)*	*W(8)*	*R_p(8)*	*R_p(8)*
frame memory3			*W(3)*	*R_d(3) W(4)*	*R_d(4)*	*W(6)*	*R_d(6) W(7)*	*R_d(7)*	*W(9)*	*R_d(9) W(10)*
input picture types [I,P,B]	I(1)	P(2)	B(3)	B(4)	P(5)	B(6)	B(7)	P(8)	B(9)	B(10)

time [frame]

Figure B.2: *Schedule for accesses in the frame memories for a conventional MPEG decoding system.*

From left to the right, the figures show a specific read or write action in the frame memories for successive frame periods. To evaluate the validity of these schedules, the following observations can be made. First, the figures show that at every frame period a picture is written (necessary for subsequent reading for display). Secondly, every frame period a picture is read for display. As a third observation, it is shown that for every P-picture,

frame memory1	$W_d(1)$		$R_d(1)$ $W_d(3)$	$R_d(3)$ $W_d(4)$	$R_d(4)$ $W_d(5)$				$R_d(5)$ $W_d(9)$	$R_d(9)$ $W_d(10)$
frame memory2		$W_d(2)$				$R_d(2)$ $W_d(6)$	$R_d(6)$ $W_d(7)$	$R_d(7)$ $W_d(8)$		
frame memory3	$W_p(1)$	$R_p(1)$	$R_p(1)$	$R_p(1)$	$W_p(5)$	$R_p(5)$	$R_p(5)$	$R_p(5)$	$R_p(5)$	$R_p(5)$
frame memory4		$W_p(2)$	$R_p(2)$	$R_p(2)$	$R_p(2)$	$R_p(2)$	$R_p(2)$	$W_p(8)$	$R_p(8)$	$R_p(8)$
input picture types [I,P,B]	I(1)	P(2)	B(3)	B(4)	P(5)	B(6)	B(7)	P(8)	B(9)	B(10)

time [frame]

Figure B.3: *Schedule for accesses in the frame memories for the new proposed system.*

prediction data is read from the previously written I- or P- picture. Similarly, for every B-picture, prediction data is read from the two previously written I- or P-pictures. And finally, for the newly proposed decoder (Figure B.3) decoded data is written twice for I- and P-pictures and only once (for display) for B-pictures. The schedule of Figure B.3 shows that with four frame memories, we are able to perform the double writing action inside the MPEG decoder and only have 30 % extra memory capacity, even though 50 % more data is written.

Appendix C

Refinement of the access alignment grid

This appendix discusses another option to influence the optimization model as discussed in Section 5.3. In Appendix B, the application algorithm was modified to provide a different memory access pattern. This appendix proposes to change the memory interface for addressing the data.

As explained in Chapter 5, the granularity of the data units is determined by the width of the bus and the burst length, which is typically programmed at configuration time. These data units can only be accessed as an entity. For example, a burst length of eight and a 64-bit wide bus result in data units of 64 bytes. To meet the high bandwidth requirements in Systems-on-Chip, memory bus widths become larger. However, main-stream memory devices typically have 4, 8 or 16-bit data busses. Therefore, most systems include a memory configuration of several memory devices in parallel as shown in Figure C.1.

C.1 Single addressable memory

In conventional systems, all SDRAM devices typically share the same address lines and the data bus of each device (16-bit busses in the picture) is combined to a 64-bit wide data bus. As a consequence of the widening bus trend, the granularity of the data entities that can be accessed also

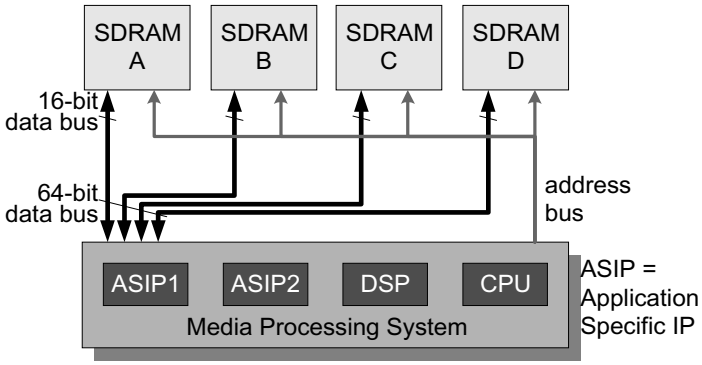


Figure C.1: *Traditional SoC with a memory configuration comprising four parallel SDRAM devices.*

increases. At least one word is accessed simultaneously, i.e. 8 bytes.

Figure C.2 shows how the memory addresses increment by eight for a memory map comprising a 64-bit wide memory configuration. This coarse-grain data access may have significant impact on the efficiency of memory transfers as outlined in Chapter 5. Figure C.3 shows an example of the organization of pictorial data in the above-mentioned memory configuration. For this example, the memory addresses sequentially increase when scanning the picture from the left to the right, and from the top to the bottom. To access a data block of 80 bytes (20 pixels from 4 different video lines) 128×4 bytes are accessed resulting in a transfer overhead of 540 %. Particularly for accessing small-grain data blocks, the transfer overhead increases significantly for increasing data-burst sizes. Although the size of the data bursts is inherent to the bus width and the burst length, part of the overhead is

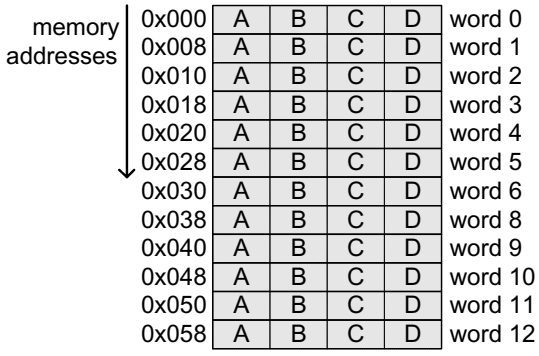


Figure C.2: *The memory map of a 64-bit wide memory configuration.*

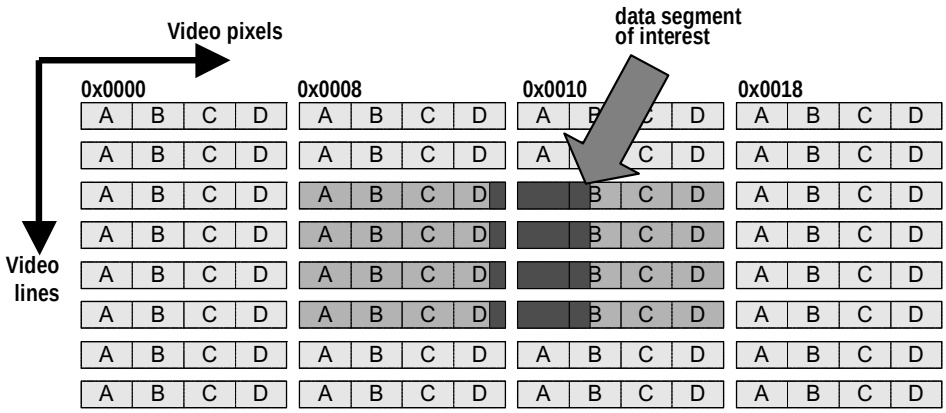


Figure C.3: *Transfer overhead for a requested data block from a traditional memory configuration.*

caused by the discrete locations of the data bursts. Memory accesses can only be applied at the alignment grid of the data bursts. For Figure C.3, the overhead would only be 270 % (instead of 540 %) if the 64-Byte transfers could start at the beginning of the requested data block.

To reduce the memory bandwidth, part of the transfer overhead can be reused with a local cache memory by exploiting this overhead data for subsequent data-block accesses (spatial locality of data). However, in such a system, the cache performance could be improve further when the start location of the data burst was not necessarily aligned with the 64-Byte memory grid. It would enable the system to capture that data in the transfer overhead that has a high probability of reuse for subsequent data-block accesses, i.e. a high cache-hit potential. Although the access of a data burst at arbitrary positions in the column would be optimal, any refinement in the alignment grid would improve the bandwidth efficiency.

C.2 Separately addressable memory

As mentioned before, most SoC contain a memory configuration with a shared address bus (see Figure C.1). However, by having multiple address busses, each device can be addressed differently while still providing the same total bandwidth. This subsection proposes a memory controller which provides different addressing for several memory devices to refine the alignment grid although the amount of bytes within a data burst remains equal. Part of the address lines are still shared by all memory devices such as the bank address lines and the row address lines. For the remaining part

of the address bus the SoC provides separate address lines for each memory device. The corresponding memory configuration is shown in Figure C.4 This proposal provides more flexibility in addressing to reduce the transfer

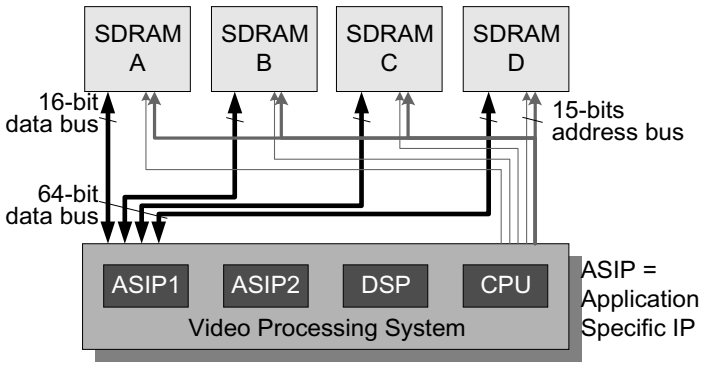


Figure C.4: *SoC with a memory configuration comprising four parallel SDRAM devices, each having shared and dedicated address lines.*

overhead and to control the memory location of the transfer overhead (for improvement of cache performance). The amount of non-shared address lines determines the granularity of the data entities and the amount of concurrent data entities. For the above-shown example configuration, four concurrent data entities are accessed, each containing 16 bytes. To access the data block of 20 pixels from 4 different video lines in the system of Figure C.4, 64×4 bytes are accessed (see Figure C.5) device B, C and D are addressed equally, but device A is addressed at the preceding memory space. Due to the finer alignment grid, only one data burst (per device) per row (64 bytes in total) is sufficient to access the requested data block whereas two data bursts per row were required in Figure C.3. Moreover, the location of a part of the overhead can be selected. In this case a selection can be made between the column in front of the requested data block or behind the requested data block. This flexibility can be exploited to improve the cache performance.

Obviously, the use of multiple address busses obviously adds cost to the design. This holds in particular when the memory is located off-chip. Multiple address busses require more pins on the chip device (more expensive device package) and increase the power. However, when only the column address lines are shared, only a relative small number of address lines need to be implemented for a multiple times. For example, for memory devices that have 256 columns within a row, only 8 address lines are implemented

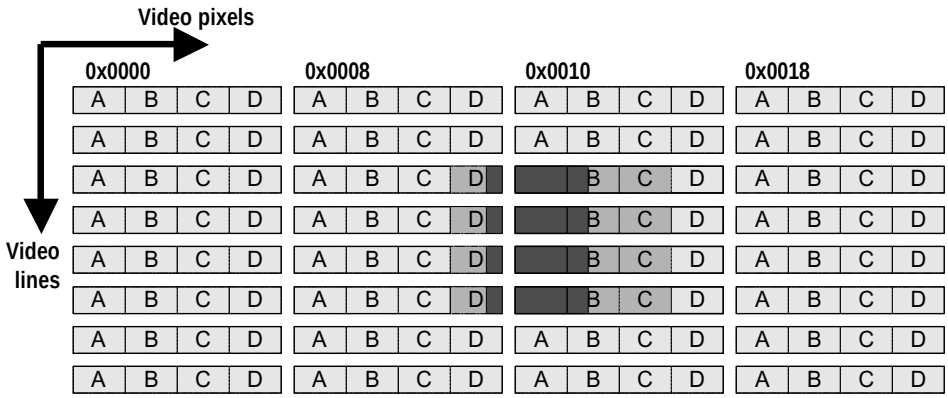


Figure C.5: *Transfer overhead for a requested data block from the separately addressable memory configuration.*

a multiple times. Besides the additional address lines, also a small part of address generation in the memory controller needs multiple implementations. However, for a system with separate column address lines only, this additional complexity is very small. Note that the additional costs for multiple address busses are only considerable for off-chip memory. For SoC with embedded DRAM, the additional costs are negligible.

Appendix D

Video-specific caching

Section 6.2 discusses caching as a technique for reduction of the memory bandwidth, while maintaining the picture quality. The discussed video-specific optimization as discussed in Section 6.2 assumes some basic knowledge of caching and its applicability for video processing, which is explained here.

D.1 Basics of caching

From the earliest days of computing, programmers have desired unlimited amount of high-speed memory. Von Neumann [25] indicated a memory bottleneck because one single operation in a CPU requires at least two memory fetches, i.e. one to load the instruction and at least one for accessing the operand data. Patterson [98] extrapolated the development of programmable processors and off-chip memories, using Moore's law. He indicated an ever increasing performance gap between the CPUs and its external memory. The performance of general-purpose CPUs increases with 60 % every year whereas the latency to off-chip memory reduces with only 7 % every year.

Although external memories (SDRAM) have a large capacity, they are relatively slow, and hence offer a limited data bandwidth and a larger latency than SRAM-based memories. Moreover, as explained in Section 5.3, the data can only be accessed efficiently on a coarse grain of data packets. This system bottleneck will continuously increase with the advancing

technology. Patterson and Hennessy [70] describe how large memory sizes and fast memory access can be exploited both using memory hierarchy. The first and lowest level close to the processor is a relative fast memory. These memories are more expensive per bit than the slower memories and thus are usually smaller. How higher the memory level, the slower and the bigger they become. In common practice, the low-level memories are implemented with SRAM (static random access memory) whereas the higher-level off-chip memories are implemented with DRAM (dynamic random access memory). In computer systems, an even higher level is often implemented by means of an magnetic disk. The goal of a hierarchical memory system is to present the user with as much memory as possible in the cheapest technology, while providing access at the speed offered by the fasted memory. In such a caching system, the memory content on a level close to the processor is a subset of the memory content at a level further away. The highest memory level contains all data stored. The principle of caching is based on the locality property of data. In [70], systems distinguish two types of locality.

- Temporal locality (locality in time): if an item is accessed, it is likely to be accessed again soon.
- Spatial locality (locality in space): if an item is accessed, nearby items with similar addresses are likely to be referenced soon.

If the addressing of subsequent accesses are randomly located in the memory address space, the locality of reference is poor. Hence the cache performance will be poor. However, if the access pattern is very deterministic, efficient caching can be implemented.

Let us now define some terms that will be used in the remainder. If the data requested by the processor appears in to be in the cache, this is called a *hit*. If the data is not found in the cache, the request is called a *miss*. Consequently, the *hit rate* is often used as a measure of the performance of the memory hierarchy, representing the fraction of the memory references that was actually present in the cache. Now let us start with a simple cache to explain the basics. First, how do we know if a data item is in the cache? And secondly, if it is, how do we find it? If each memory word can go in exactly one place in the cache, we will know to find it if it is in the cache. The simplest way to assign a location in the cache for each word in the memory is to assign the cache location based on the address of the word in memory. This cache structure is called *direct mapped*, and is usually simple to implement. Traditionally, almost all direct-mapped caches use the `modulo` operation. Thus (cache address) = (memory address) `modulo`

(size of the cache). As we will see later, this relation that associates data in the higher-level memory to a certain cache location is different for video-specific caches. Although direct mapping results in a unique cache location for each memory word, each cache location does not correspond to a unique location in the memory. The existence of a word in the cache is proven by means of an additional *tag* field for each cache location. The tag contains the address information required to identify whether a word in the cache corresponds to the requested word. The tag only needs to contain the upper portion of the address, corresponding to the bits that are not used as an index in the cache. Figure D.1 shows how the lower three bits of a memory

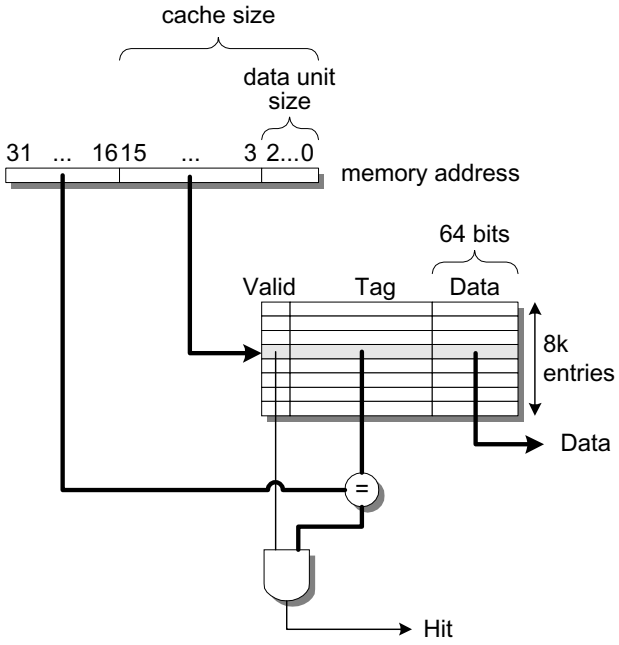


Figure D.1: *A basic cache implementation using a tag field for comparison.*

address are not used because they address individual bytes within one cache location. In the sequel of this appendix, one cache location is referred to as a cache line. Thus for the example as depicted in the figure, one cache line contains a 64-bit word, assuming that individual words can be transferred from main memory. The adjacent 13 bits are used to identify a unique position in the cache whereas the remaining bits resemble the number of possible memory locations that map onto the same cache line. Hence, the value of these bits is stored in the tag field of the cache to identify the exact memory location. If a cache line is empty, e.g. when the processor starts

up, we need to know that the tag should be ignored for such entries. This is accomplished with the so-called *valid bit*, which indicates whether a cache line contains a valid address. Figure D.1 shows that a FALSE valid bit in a selected cache line results in a FALSE value at the hit output, meaning that the requested data is not contained in the cache. To determine the size of the cache, the number of cache lines is multiplied with the number of bits for the cache line, for the tag, plus one bit for the valid. Thus for the above-depicted example, the cache is $8\text{ k} \times (64+16+1) = 846$ kbits.

Up to this point, we have only discussed a simple placement scheme: One block can go in exactly one place in the cache. Such type of cache is called a *direct mapped* cache. There is actually a whole range of schemes for placing blocks. For example, it is also possible to place a block in any location in the cache instead of one unique location. Such a scheme is called *full-associative*, because a block in memory may be associated with any entry in the cache. To find a given block in such a cache, all the cache lines in the cache must be searched because a block can be placed in any one. Commonly, such a scheme is implemented by simultaneously comparing the memory address tags with all tag entries in the cache. These comparators significantly increase the hardware cost, effectively making full-associative placement only practical for caches with small numbers of cache lines. In between the direct-mapped and the full-associative placements, there is a so-called *set-associative* placement. Such a cache consists of sets, where each set operates as a direct-mapped cache. However, each data unit can associated with each set. For example, a four-way set-associate cache consists of four sets. Thus, a data unit in the background memory can be mapped in one of four sets. Within the set, the data unit is associated with only one of the cache lines. Summarizing, with an 4-way set-associative cache, each data unit can be stored in four different cache lines in the cache.

D.2 Caching of 2-D video data

As mentioned above, the principle of caching is based on the locality property of data. However, does this property also hold for video data? Let us discuss an example. When applying video filtering in horizontal direction with dedicated hardware, indeed the same data samples are used to calculate successive pixels, i.e. for a six-tap filter each sample is used in the calculation of six different pixels. For each new filtered output pixel one new input pixel is read in the filter taps and replaces the least-recently read pixel of the six. This reading of the pixels is done in a sequential order.

The temporal locality is very high since each sample is used six times in a row, directly after each other. For vertical filtering and many more examples, a similar reasoning holds. The usage of memory for the filter taps limits the data traffic between the lower-level memory (integrated in the filter functionality) and the higher-level memory to a minimum, i.e. each pixel is read only once. However, it is still debatable to call these local integrated memories a cache. In this appendix we define a memory close to the processor to be a cache, if it is transparent for the processor, i.e. the system without the cache is still operational and has the same functional behavior. For the above-mentioned filter function this does not apply. The filter operations apply local addressing of the embedded memories or use hardwired connections to the memory cells to access the data instead of addressing the global address space.

Typical examples that require data-dependent access to a large memory space are MPEG encoding and decoding, field-rate conversion, and motion-compensated de-interlacing. In all these examples, the accessed memory locations are determined by means of calculated motion vectors. For rendering of video textures in a three-dimensional space, the accessed memory locations depend on the geometric transformations. Only for a few functions, such as block-based motion estimation, there is some temporal locality, because there is a probability that the same data will be accessed for matching successive blocks. However, for the majority of video functions, the temporal locality is limited. Video data appears to be hardly reused, so that caching is not suitable. On the other hand, there is usually a large amount spatial locality. For example, if a macroblock (MB) for MPEG decoding is read for prediction, the probability is large that the adjacent macroblock on the right side is read successively. This property can be exploited with a cache by means of prefetching to reduce the average latency of a memory request. Unfortunately, the bandwidth which is the critical parameter in streaming application, is not reduced.

As was discussed in Chapter 5, DRAM-based memories are accessed in burst mode, leading to transfer overhead. Due to the spatial locality of streaming applications, this transfer overhead often contains part of the data that is requests successively. Consequently, caching can be used to reduce this overhead. In the following section we gradually modify a cache to improve its applicability for video-data access. For example, the address of memory addresses is replaced by addressing via pixel coordinates and a picture address offset. Furthermore, data sets inside the cache are regarded as two-dimensional memory spaces.

D.2.1 Associating cache lines within the set of a cache

The main difference between a conventional cache and a cache for streaming video is the concept of two-dimensional data, i.e. there is not a one-to-one mapping of spatial locality in pictures and spatial locality of data in the memory space. For 2-D data, a relation exists between the coordinates of pixels (p_x, p_y) and the physical memory addresses A. If data is written sequentially into the memory in a scanning order of the picture from left to right and top to bottom, an address location A is calculated by

$$A = \text{picture offset} + p_y \times \text{line width} + p_x, \tag{D.1}$$

where the 'picture offset' is an address pointer to the start of the picture, and the 'line width' is the width of a video line in units of bytes. Due to this mapping, spatial locality of pixels in the picture does not necessarily mean that this locality is also present in the memory and hence is therefore not exploited by a "standard"cache. For the mapping as defined by Equation (D.1), two adjacent pixels in the vertical direction are *linewidth* bytes located apart from each other.

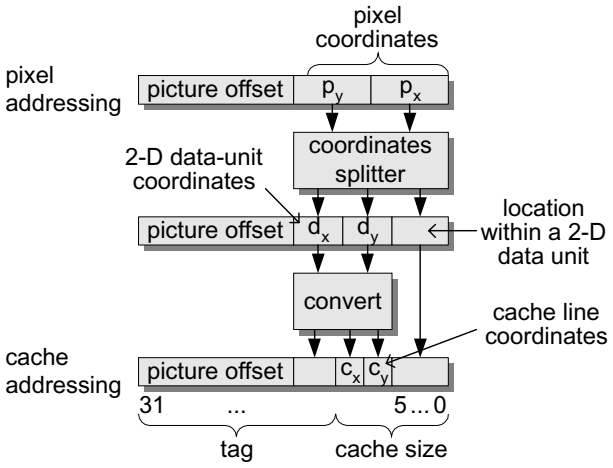


Figure D.2: *Association of memory addresses with cache-line coordinates.*

To understand the mapping of video data into sets of a cache, we first introduce the concept of 2-D sets. As mentioned before, the mapping of data within a set is equivalent to a direct-mapped cache. A 2-D set is a two-dimensional arrangement of cache lines that can be addressed by means of 2-D coordinates. Conventionally, a cache-line address c is associated with memory address A according to:

$c = (A)$ `modulo` (set size)

In a 2-D set, the associating is done according to:

$c_x = p_x$ `modulo` (horizontal set size), and
$c_y = p_y$ `modulo` (vertical set size).

Construction of the corresponding tag is less straightforward. For a conventional 1-D cache, the tag is simply calculated as follows:

$tag = A$ / (set size)

However, for the 2-D cache set this is not sufficient, since the positions of pixels in the picture are not related to a memory address. Therefore, it

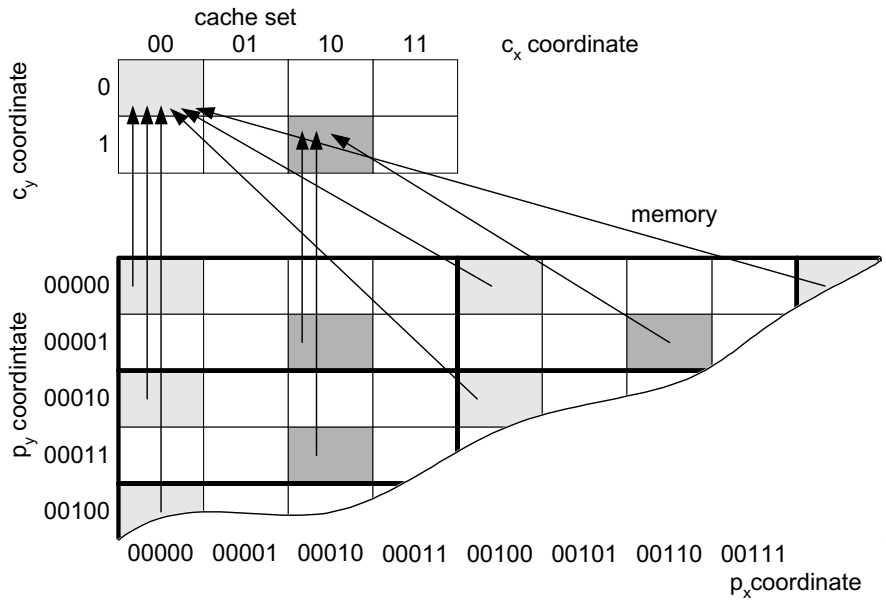


Figure D.3: *Addressing of a two-dimensional cache set.*

is required to include an offset address that indicates the start of a picture. Figure D.2 shows a schematic address translation that maps pixel data of a picture onto cache lines. First, the pixel coordinates are split into coordinates of the 2-D data units as discussed in Chapter 5, and a separate 6-bit address to access the individual bytes within the data units. Secondly, the picture offset and the data-unit coordinates are converted to a tag and coordinates that address the cache lines within the cache set. Conceptually,

this scheme can be considered as depicted in Figure D.3. The figure shows how the `modulo` operation divides the picture is into adjacent blocks of 4×2 data units. Moreover, it shows how these data units map onto the cache lines which are addressed by coordinates. Notice that a tag consists of two parts. One part addresses the corresponding block of 4×2 data units. The second part comprises the picture offset, which is similar for all cache lines. To reduce the space for storing the tags, the picture offset can be isolated from the tag and stored once per set.

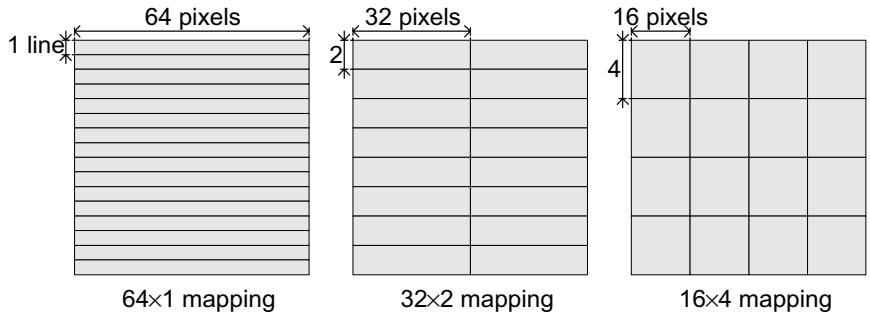


Figure D.4: *Equal set coverage for different cache line dimensions in a two-way set-associative cache.*

The 2-D size of a cache set in pixels and video lines depends on the dimensions of the data units and the amount of horizontal and vertical data units. This is depicted in Figure D.4, where equally-sized sets contain different amount of data units in horizontal and vertical direction. The width and height of a set is determined by the spatial locality of successive memory accesses, whereas the dimensions of the data units is mainly determined by the dimensions of the requested data blocks (see Subsection 5.3.2).

D.2.2 Two-dimensional set-associative caching

The previous subsection has presented the concept of 2-D sets and has shown how the cache lines can be associated with the pixel data. Figure D.5 shows some examples of a direct-mapped 2-D cache, a 2-way set-associative 2-D cache, and a full-associative 2-D cache, where the sizes of the depicted caches are equal, i.e. eight cache lines. For the direct-mapped cache example, the picture is divided into adjacent blocks of 4×2 data units, as discussed before. If the cache contains more sets of cache lines, e.g. a 2-way set-associative cache, each data unit can be associated with one cache line per set. Consequently, when accessing a data unit in the memory, the system will search in two cache lines for availability of the data unit. In

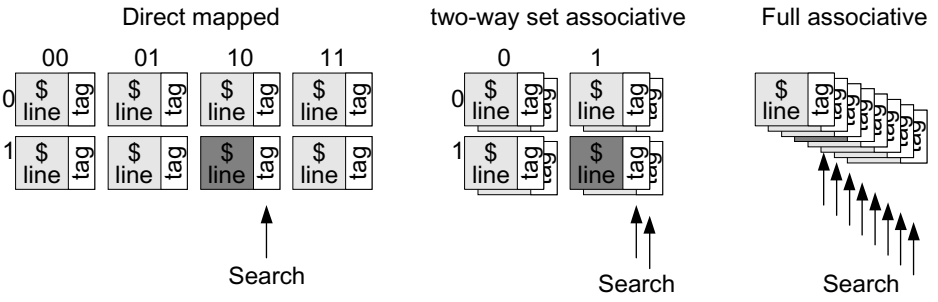


Figure D.5: *Equally-sized set-associative caches with different amount of cache lines per set in horizontal and vertical direction.*

case of a full-associative cache, all cache lines have to be searched to check the availability of the data unit.

The previous subsection already explained that set-associative caches are more complex that direct-mapped caches, due to the compare operations to search through all cache lines that can be associated with a data unit. However, the drawback of a direct-mapped cache is that written data can only replace the content of one cache line, even if it contains valuable data. In set-associate caches a data unit to be placed in cache can be associated with more cache lines, thereby enabling replacement of the least useful data in cache. Figure D.6 shows how a set-associative cache is implemented together with the address translation unit. This translation unit addresses the cache lines of each cache set by means of coordinates (c_x,c_y). Subsequently, the tag of these cache lines is compared with the tag of the requested data unit. If one of the comparisons is successful, it routs the content of the corresponding cache line to the cache port and reports a cache *hit*.

D.2.3 MPEG-specific caching

The mapping of pixel data onto cache sets as discussed in Subsection D.2.1 is limited to its use for storing one video picture. However, many functions exist that reference several pictures in the background memory. For example, if a cache is used by a motion-compensation unit in an MPEG decoder, prediction data can be referenced in a previous picture and a next picture. In such situations, a direct-mapped cache will mix the cache data of one of the reference pictures by accessing the other picture and vice versa. As a solution, we can increase the amount of sets to provide more cache lines that correspond with the same pixel coordinates. Alternatively, we can partition the sets of the cache into subsets, where each subset may contain a differ-

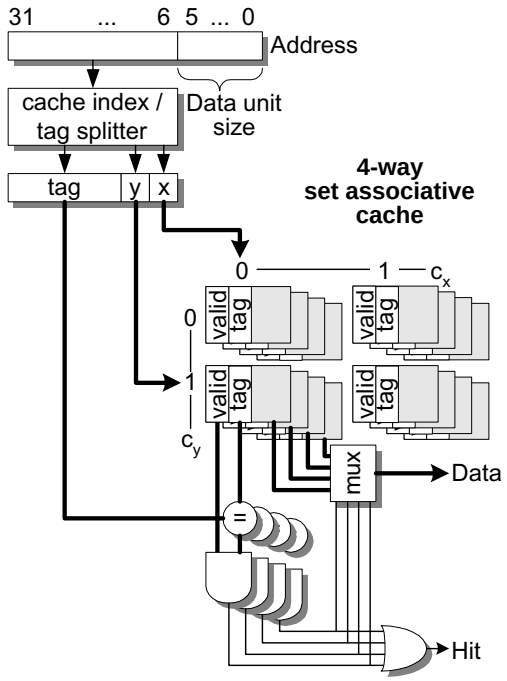


Figure D.6: *Implementation of a four-way set-associative cache.*

ent picture offset. This enables a less costly direct-mapped cache without the cache content mixing problem. An example of such a cache implementation for a motion-compensation unit in an MPEG decoder, is depicted in Figure D.7. In this cache, the references with different 'picture offset' address pointers will be associated with different 2-D subsets. The example in the figure shows subsets for referencing a forward reference picture and a

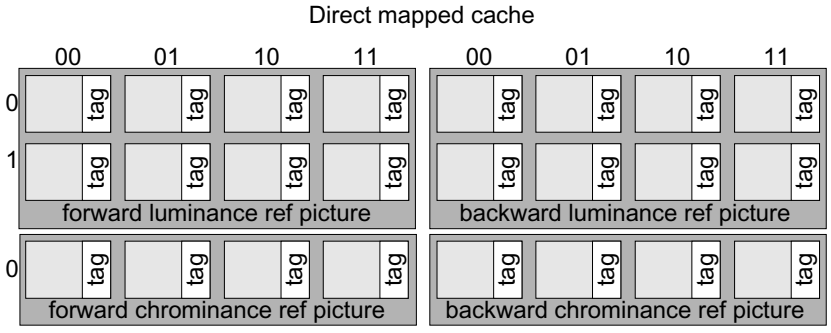


Figure D.7: *An example direct-mapped cache for multiple video streams.*

backward reference picture separately for luminance and chrominance data. Figure D.7 furthermore shows that the partitioning into subsets even allows unequal sizes. Because the chrominance data in an MPEG decoder is twice as small in vertical direction as the luminance, the size of the corresponding subset can also be twice as small. Also in the horizontal direction, the two color components are twice as small, but here it is assumed that two color components are located in the same picture memory where each C_rC_b duplet is located in subsequent bytes of the memory. Hence, in horizontal direction the size of the subset for chrominance is equal to the one for luminance.

D.3 Replacement strategy

Up till now, we have discussed the mapping of the global address space onto the cache sets. However, we did not discuss in which of the sets the data are written in case of a set-associative cache. The choice for this is determined by the so-called *replacement strategy*. As long as one or more of the associated cache lines in sets indicate invalid content, data can be stored without overwriting other valuable cache data. Otherwise, the replacement strategy determines the cache line of which its content will be replaced. Many replacement strategies have been proposed in the literature. For the case of MPEG decoding we have evaluated the following basic replacement strategies.

- *Least recently used (LRU)* – This algorithm is the most popular one. It requires bookkeeping of the sequential order in which data in the cache is used. For example, in an eight-way set-associative cache, a data unit can be associated with eight different cache lines. Hence, each cache line contains three bits to indicate the sequential order in time in which the eight cache lines are accessed. Thus, for such a cache each cache line consists of a tag, a valid bit, three LRU bits and a 64-Byte data buffer (if one cache line = 64 bytes). After each cache-line access, all associated eight 3-bit counters are updated. This LRU algorithm exploits temporal locality and assumes that the probability of accessing data that was already accessed before, decreases when in the mean time other memory location are accessed. Another well-known replacement policy that is quite similar is the Least frequently used (LFU) algorithm. [114].

- *Least recently written (LRW)* – This is a more simple replacement strategy [115] that is very similar to the LRU algorithm with the exception that it only measures the time from the first time it was

accessed. Because the first access of a data unit always start with a cache miss, the algorithm only needs to consider data that is copied from the memory into the cache. LRU assumes that the probability of reusing data is maximal after each re-access, even if it was already accessed several times before. As opposed to LRU, LRW assumes that the probability of reuse continuously decreases after it was written in cache, even after re-accessing this data. For the MPEG decoder function this might be a valid assumption, since temporal locality is limited. Furthermore, the spatial distance between prediction data for successive macroblocks continuously increases after it was accessed the first time. For the implementation of LRU, the cache lines that are associated with the same memory address are basically used as a FIFO. After data is written in the cache line, a pointer indicates the set which contains this most-recently written data unit. For each new data unit, the pointer is cyclically incremented. Consequently, the data that was written least recently is overwritten. This algorithm only requires a pointer for each set of cache lines that are associated with the same memory address and requires only one pointer update per cache write. Thus, an eight-way set-associative cache contains one pointer per eight cache lines.

- *Largest Manhattan distance* – The above-mentioned replacement policies are based on the temporal locality property. Another class of replacement strategies is more based on the spatial locality property. The largest Manhattan distance is an example replacement strategy of such a class. The cache lines with the largest Manhattan distance is defined as:

$$|d_x^{ref} - d_x^{cl}| + |d_y^{ref} - d_y^{cl}|, \tag{D.2}$$

  with (d_x^{ref}, d_y^{ref}) the coordinates of the data unit in the picture and (d_x^{cl}, c_y^{cl}) the coordinates of the data units in a cache line to be compared. This algorithm intuitively fits well to a motion-compensation unit which accesses data according to a motion vector. However, a drawback of this replacement policy is the distance calculations additional to the comparison as used for LRU.

Summarizing, the following results have been adopted for the experiments in Chapter 6. The LRW strategy was chosen as the best replacement strategy for our MPEG decoder system. However, it was shown that a direct-mapped cache offers similar performance as the more complex set-associative cache, so that finally a replacement strategy was required. The cache will be organized into two-dimensional cache data sets to associate

with two-dimensional data units. The optimal dimensions of the data units and size of the cache data sets will be discussed in Chapter 6.

References

[1] M.W. Maier, and E. Rechtin, *The Art of Systems Architecting*, p. 2, CRC Press LLC, 2002.

[2] E. Rechtin, *Systems Architecting, Creating & Building Complex Systems*, p. xiv, Prentice-Hall, Inc., 1991.

[3] F.J. Bingley, "A half century of television reception," *Proc. of the IRE*, vol. 50, pp. 799–805, May 1962.

[4] P. Lippens *et al.*, "A video signal processor for motion-compensated field-rate upconversion in consumer electronics," *IEEE Journal of Solid-State circuits*, vol. 31, no. 11, pp. 1762–1769, Nov. 1996.

[5] J. Veerhoek and A. Stuivenwold, "Economy picture-in-picture processor," *Philips woldnews, www.semiconductors.philips.com /news/publications/lists/worldnews/7/3/*, vol. 7, no. 3, pp. 3.27, July 1998.

[6] P.H.N. de With, E.G.T. Jaspers, J.L. van Meerbergen, A.H. Timmer, and M.T.J. Strik, "A video display processing platform for future TV concepts," *IEEE Trans. on Consumer Electronics*, vol. 45, no. 4, pp. 1230–1241, Sept. 1999.

[7] *Standard, ITU-T Rec.H.262 — ISO/IEC 13818-2*, 1995.

[8] *Joint Final Committee Draft of Joint Video Specification, ITU-T Rec. H.264 — ISO/IEC 14496-10 AV*, Sept. 2002.

[9] E.G.T. Jaspers and P.H.N. de With, "A generic 2D sharpness enhancement algorithm for luminance signals," in *Proc. of 6th Int. Conf. Image Processing and its Applications*, July 1997, vol. 2, pp. 269–273.

[10] J.G.W.M. Janssen, J.H. Stessen and P.H.N. de With, "An advanced sampling rate conversion technique for video and graphics signals," in *Proc. of 6th Int. Conf. Image Processing and its Applications*, July 1997, vol. 2, pp. 771–775.

[11] E.G.T. Jaspers, P.H.N. de With and J.G.W.M. Janssen, "A flexible heterogeneous video processor system for TV applications," *IEEE Trans. on Consumer Electronics*, vol. 45, no. 1, pp. 1–12, Febr. 1999.

[12] E.G.T. Jaspers, and P.H.N. de With, "Architecture of embedded video processing in a multimedia chip-set," in *Proc. of IEEE Int. Conf on Image Processing, ICIP'99*, Oct. 1999, vol. 2, pp. 787–791.

[13] E.G.T. Jaspers and P.H.N. de With, "Chip-set for video display of multimedia information," *IEEE Trans. on Consumer Electronics*, vol. 45, no. 3, pp. 706–715, Sept. 1999.

[14] E.G.T. Jaspers, B.S. Vissers, and P.H.N. de With, "Synchronization of video in distributed computing systems," in *Proc. of the SPIE – Visual communications and Image Proc. 2000, VCIP 2000*, June 2000, vol. 4067-3, pp. 1430–1440.

[15] E.G.T. Jaspers and P.H.N. de With, "Bandwidth reduction for video processing in consumer systems," *IEEE Trans. on Consumer Electronics*, vol. 47, no. 4, pp. 885–894, Nov. 2001.

[16] E.G.T. Jaspers, *Patent Application – Method for storing data elements*, PHNL020138.

[17] E.G.T. Jaspers, *Patent Application – Address space, bus system, memory controller and device system*, PHNL020098.

[18] E.G.T. Jaspers and P.H.N. de With, "Compression for reduction of off-chip video bandwidth," in *Proc. of the SPIE – Media Processors 2002*, Jan. 2002, vol. 4067-3, pp. 110–120.

[19] E.G.T. Jaspers and P.H.N. de With, "Embedded compression for memory resource reduction in MPEG systems," in *Proceedings of IEEE Benelux Signal Processing Symposium, SPS-2002*, March 2002.

[20] P.H.N. de With, E.G.T. Jaspers, A.H. Timmer, and J.L. van Meerbergen, "A flexible heterogeneous video processing architecture for powerful HQ TV applications," in *IEE Proc. Int. Conf. Image Processing and its Application*, July 1999, vol. 1, pp. 122–126.

[21] M.J. Rutten, J.T.J. van Eijndhoven, E-J. D. Pol, E.G.T. Jaspers, *et al.*, "Eclipse - heterogeneous multiprocessor architecture for flexible media processing," in *Proc. of the workshop on Parallel and Distributed Computing in Image Processing, Video Processing, and Multimedia, PDIVM'2002*, April 2002.

[22] M.J. Rutten, J.T.J. van Eijndhoven, E-J. D. Pol, E.G.T. Jaspers, *et al.*, "Eclipse - heterogeneous multiprocessor architecture for flexible media processing," *IEEE Design & Test of Computers*, vol. 19, no. 4, pp. 39–50, July/Aug. 2002.

[23] E.B. van der Tol and E.G.T. Jaspers, "Mapping of MPEG-4 decoding on a flexible architecture platform," in *Proc. of the SPIE – Media Processors 2002*, Jan. 2002, vol. 4674, pp. 1–13.

[24] E.G.T. Jaspers, *et al.*, "System-level analysis for MPEG-4 decoding on a multi-processor architecture," in *Proc. of the workshop on Parallel and Distributed Computing in Image Processing, Video Processing, and Multimedia, PDIVM'2002*, April 2002.

[25] J. Backus, "Can programming be liberated from the von Neumann style? a functional style and its algebra of programs," *Communication of the ACM*, vol. 21, no. 8, pp. 613–615, Aug. 1978.

[26] S. Rathnam and G. Slavenburg, "Processing the new world of interactive media," *IEEE Signal processing Magazine*, vol. 15, no. 2, pp. 108–117, March 1998.

[27] S. Purcell, "The impact of Mpact 2," *IEEE Signal processing Magazine*, vol. 15, no. 2, pp. 102–107, March 1998.

[28] R.J. Gove, G.J. Hewlett, and D.B. Doherty, "The MVP: A single-chip processor for advanced television applications," in *Proc. of the Int. Workshop on Signal Processing of HDTV, VI*, Oct. 1994, vol. 6, pp. 479–487.

[29] S. Rathnam, and G. Slavenburg, "An architectural overview of the programmable multimedia processor, tm-1," in *Proc. of COMPCON'96, Digest of papers*, Feb. 1996, pp. 319–326.

[30] D.W. Wall, "Limits of instruction-level parallelism," in *Proc. of the 4th Int. Conf. on Architectural Support for Programming Languages and Operating System (ASPLOS)*, April 1991, vol. 26, no. 4, pp. 176–188.

[31] G.M. Amdahl, "Validity of single-processor approach to achieving large-scale computing capability," in *Proc. of AFIPS Conf.*, 1967, vol. 30, pp. 483–485.

[32] R.J. Gove, "The multimedia video processor (MVP): an architecture for advanced dsp applications," in *Proc. of the 5th int. conf. on Signal Processing Appl and Technology*, Oct. 1994, vol. 1, pp. 854–859.

[33] H.J.M. Veendrick *et al.*, "A 1.5 GIPS signal processor (VSP)," in *Proc. of IEEE Custom Integrated Circuits Conf., CICC'94*, May 1994, pp. 95–98.

[34] V. Easwar, G. Campbell and R. Natarajan, "Multimedia system on a chip for set-top box applications," in *Digest of Technical Papers of IEEE Int. Conf. on Consumer Electronics*, June 1999, pp. 356–357.

[35] E. Roza, "Systems-on-chip: what are te limits?," *Electronics & Communication Engineering Journal*, vol. 13, no. 6, pp. 249–255, Dec. 2001.

[36] K.A. Vissers *et al.*, "Architecture and programming of two generations video signal processors," *Microprocessing and Microprogramming*, vol. 41, no. 5-6, pp. 373–390, Oct. 1995.

[37] M. Oka and M. Suzuoki, "Design and programming the emotion engine," *IEEE Micro (USA)*, vol. 19, no. 6, pp. 20–28, Nov. 1999.

[38] F.M. Raam, *et al.*, "A high bandwidth superscalar microprocessor for multimedia applications," in *Digest of Technical Papers of the IEEE Int. Solid-State Circuits Conf, ISSCC*, Feb. 1999, pp. 258–259.

[39] A. Kunimatsu *et al.*, "Vector unit architecture for emotion synthesis," *IEEE Micro*, vol. 20, no. 2, pp. 40–47, March-April 2000.

[40] H. Tago, *et al.*, "Importance of cad tools and methodologies in high speed cpu design," in *Proc. of Asia and South Pacific Design Automation Conf., ASP-DAC2000*, Jan 2000, pp. 631–633.

[41] M Suzuoki, *et al.*, "A microprocessor with a 128-bit cpu, tne floating-point mac's, four floating-point dividers, and an MPEG-2 decoder," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 11, pp. 1608–1618, Nov. 1999.

[42] A. Fosenfeld and A.C. Kak, *Digital Picture Processing*, vol. 1, pp. 237–250, Academic Press, 2nd edition, 1982.

[43] W.K. Pratt, *Digital Image Processing*, John Wiley & Sons, Inc., 2nd edition, 1991.

[44] R.H. Wallis, "An approach for the variant restoration and enhancement of images," in *Proc. Symp. Current Mathematical Problems in Imange Science*, Nov. 1976.

[45] J.D. Fahnestock and R.A. Schowengerdt, "Spatially variant contrast enhancement using local range modification," *Optical Eng.*, vol. 22, no. 3, pp. 378–381, 1983.

[46] Chen and V.-H. Tsai, "Moment-preserving sharpening," *Computer Vision, Graphics and Image Processing, an Int. Journal*, vol. 41, pp. 1–13, 1988.

[47] T.-L. Ji, M.K. Sundareshan, and H. Roehrig, "Adaptive image contrast enhancement based on human visual properties," *IEEE Trans. Medical Imaging*, vol. 13, no. 4, pp. 573–586, Dec. 1994.

[48] R.C. Gonzalez, *Digital Image Processing*, chapter 2, Addison-Weslay Publishing Company Inc., 1992.

[49] A. Luthra and G. Rajan, "Sampling-rate conversion of video signals," *SMPTE Journal*, vol. 100, pp. 869–879, Nov. 1991.

[50] L.B. Jackson, *Digital Filters and Signal Processing*, Kluwer, 1986.

[51] A.V. Oppenheim and R.E. Schafer, *Discrete Time Signal Processing*, Prentice-Hall Inc., Englewood Cliffs, NY, 1968.

[52] R.W. Schafer and L.R. Rabiner, "A digital signal processing approach to signal interpolation," *Proc. of the IEEE*, , no. 6, pp. 692–702, June 1973.

[53] A.H. Nillesen, "Non-integral delay circuit," *United States Patent*, vol. US 5,349,548, Sept. 1994.

[54] T.A.C.M. Claasen and W.F.G. Mecklenbräuer, "On the transposition of linear time-variant discrete-time networks and its application to multirate digital systems," *Philips Journal of Research*, vol. 33, pp. 78–102, 1978.

[55] A.J. Dalfsen, J.H.C.J. Stessen and J.G.W.M. Janssen, "Sample rate conversion," *European Patent Application*, vol. EP-A 96203035.9, Oct. 1996.

[56] B.G. Lee, "A new algorithm to compute the discrete consine transform," *IEEE Trans. Acoustics Speech and Signal Processing*, vol. ASSP-32, no. 6, Dec. 1984.

[57] N.I. Cho and S.U. Lee, "Fast algoritm and implementation of 2-D discrete cosine transform," *IEEE Trans. on Circuits and Systems*, vol. 38, no. 3, pp. 297–304, 1991.

[58] *et al.* E. Krishnan, "Design of a 2d dct/idct applicaton specific vliw processor supporting scaled and sub-sampled blocks," in *Proc. of the 16th Int. Conf. on VLSI Design*, Jan. 2003.

[59] G. de Haan, W.A.C. Biezen, "Sub-pixel motion estimation with 3-D recursive search block-matching," *Signal Processing Image Communication*, vol. 6, no. 3, pp. 229–239, June 1994.

[60] C. Hentschel and D. La Hei, "Effective peaking filtering and its implementation on a programmable architecture," in *Digest of Technical Papers of IEEE Int. Conf. on Consumer Electronics*, June 2000, pp. 56–57.

[61] K.S. Hosseini and A.D. Bovoloupus, "A simple and efficient bus management scheme that supports continuous streams," *ACM - Trans on Computer Systems*, vol. 13, pp. 122–140, May 1995.

[62] G. Kahn, "The semantics of a simple language for parallel programming," *Information Processing 74*, pp. 471–475, Aug. 1974.

[63] CCIR, *Recommendation 601-2, Encoding parameters of digital television for studios.*

[64] O. Steinfatt, P. Klapproth and H. Tichelaar, "Tcp: A next generation for tv control processing," in *Digest of Technical Papers of IEEE Int. Conf. on Consumer Electronics*, June 1999, pp. 354–555.

[65] *OMI/PI-Bus specification, OMI 324: PI-Bus Rev. 0.3d*, 1994.

[66] J. Leijten, *Real-time Constrained Reconfigurable Communication between Embedded Processors*, Ph.d. thesis, Eindhoven univ. of Technol., Eindhoven (NL), Nov. 1998.

[67] CCIR, *Recommendation 656, Interfaces for digital component video signals in 525-line and 625-line television systems.*

[68] S. Mietens, P.H.N. de With, and C. Hentschel, "Implementation of a dynamical multi-window tv system," in *Proc. of the* 22^{nd} *Symp. on Information and Communication Theory in the BENELUX*, May 2001, pp. 139–146.

[69] G. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38(8), pp. 114–117, April 1965.

[70] J.L. Hennessy and D.A. Patterson, *Computer Architecture a Quantitative Approach*, p. 374, Morgan Kaufmann, 2nd edition, 1996, ISBN 1-55860-372-7.

[71] www.specbench.org.

[72] B. Khailany *et al.*, "Imagine: Media processing with streams," *IEEE Micro*, vol. 21, no. 2, pp. 35–46, March-April 2001.

[73] K. Itoh, T. Watanaba, S. Kimura, and T. Sakata, "Reviews and prospects of high-density ram technology," in *Proc. of the 2000 Int. Semiconductor Conf., CAS 2000*, Oct. 2000, vol. 1, pp. 13–22.

[74] B. Davis, T. Mudge, B. Jacob and V. Cuppu, "DDR2 and low latency variant," in *Proc. of the Workshop on Solving the Memory Wall Problem*, June 2000, www.ece.neu.edu/wall2k.html.

[75] H. Kim and I.C. Park, "Array address translation for SDRAM-based video processing applications," in *Proc. of the SPIE – Visual Communication and Image Processing, ICIP*, June 2000, vol. 4067, pp. 922–931.

[76] S. Rixner, *et al.*, "Memory access scheduling," *Computer Architecture News*, vol. 28, no. 2, pp. 128–138, May 2000.

[77] S. Dutta, D. Singh and V. Mehra, "Architecture and implementation of a single-chip programmable digital television and media processor," in *Proc. of the IEEE Workshop on Signal Processing Systems, SiPs 99, Design and Implementation*, Oct. 1999, pp. 321–330.

[78] S. Roy, R. Kumar and M. Prvulovic, "Improving system performance with compressed memory," in *Proc. of Int. Parallel and Distr. Proc. Symposium - IPDPS*, April 2001.

[79] R.B. Tremaine *et al.*, "Pinnacle: IBM MXT in a memory controller chip," *IEEE Micro*, vol. 21, no. 2, pp. 56–68, March-April 2001.

[80] P.H.N. de With, P.H. Frencken and M. v.d. Schaar-Mitrea, "An MPEG decoder with embedded compression for memory reduction," *IEEE Trans. on Consumer Electronics*, vol. 44, no. 3, pp. 545–555, Aug. 1998.

[81] J. Tajime, *et al.*, "Memory compression method considering memory bandwidth for HDTV decoder LSIs," in *Proc. of IEEE Int. Conf. on Image Processing*, Oct. 1999, vol. 2, pp. 779–782.

[82] M. v.d. Schaar-Mitrea and P.H.N. de With, "Novel embedded compression algorithm for memory reduction in MPEG codecs," in *Proc. of the SPIE – Visual Commununication and Image Processing, ICIP*, Jan. 1999, vol. 3653-2, pp. 864–873.

[83] M. v.d. Schaar-Mitrea and P.H.N. de With, "Near-lossless embedded compression algorithm for cost reduction in DTV receivers," *IEEE Trans. on Consumer Electronics*, vol. 46, no. 4, pp. 923–933, Nov. 2000.

[84] U. Bayazit, L. Chen, and R. Rozploch, "A novel memory compression system for MPEG-2 decoders," in *Digest of Technical Papers of IEEE Int. Conf. on Consumer Electronics*, June 1998, pp. 56–57.

[85] E.G.T. Jaspers and P.H.N. de With, "Bandwidth optimization for video-based consumer systems," in *Digest of Technical Papers of IEEE Int. Conf. on Consumer Electronics*, June 2001, pp. 72–73.

[86] R.J. van der Vleuten, "Low-complexity lossless and fine-granularity scalable near-lossless compression of color images," in *Proc. of the IEEE Data Compression Conference*, April 2002, p. 477.

[87] R.J. van der Vleuten, *Device and Method for Compressing a Signal.*

[88] Stolberg, H. J., *et al.*, "The M-PIRE MPEG-4 DSP and its macroblock engine," in *Proc. of the IEEE Int. Symp. on Circuits and Systems, ISCAS2000*, May 2000, vol. 2, pp. 192–195.

[89] 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*, vol. 23, no. 1, pp. 51–66, Oct. 1999.

[90] 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*, vol. 23, no. 1, pp. 7–26, Oct. 1999.

[91] M. Berekovic, *et al.*, "Instruction set extensions for MPEG-4 videos," *Journal of VLSI Signal-Processing Systems for Signal, Image, and Video Technology*, vol. 23, no. 1, pp. 27–49, Oct. 1999.

[92] M. Berekovic, *et al.*, "Multicore system-on-chip architecture for MPEG-4 streaming video," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 12, no. 8, pp. 688–699, Aug. 2002.

[93] *(1999) AMBA Specification Rev. 2.0 ARM Ltd.*, http://www.arm.com.

[94] S. Rixner, *et al.*, "Media processors using streams," in *Proc. of the SPIE – Media Processors 1999*, Jan. 1998, vol. 3655, pp. 122–134.

[95] D.C. Wyland, "Media processors using a new microsystem architecture designed for the internet era," in *Proc. of the SPIE – Media Processors 2000*, Jan. 2000, vol. 3970, pp. 2–15.

[96] S. Sudharsanan, *et al.*, "Image anv video processing using majc 5200," in *Proceedings of 7th IEEE Int. Conf. on Image Processing*, Sept. 2000, vol. 3, pp. 122–125.

[97] 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*, vol. 5, no. 2, pp. 140–149, April 1995.

[98] Patterson, D. A. and Hennessy, J. L., *Computer Organization & Design, The Hardware/Software Interface*, Morgan Kaufmann Publishers Inc., 2nd Ed., San Francisco, CA, USA, 1998.

[99] E.A. Lee, and T.M. Parks, "Dataflow process networks," *Proc. of the IEEE*, vol. 85, no. 5, pp. 773–801, May 1995.

[100] O.P. Gangwal, *et al.*, "A scalable and flexible data synchronization scheme for embedded hw-sw shared-memory systems," in *Proc. on the 14th Int. Symp. on System Synthesis*, Oct. 2001, pp. 1–6.

[101] M.J. Rutten, J.T.J. van Eijndhoven, and E-J. D. Pol., "Robust media processing in a flexible and cost-effective network of multi-taksing coprocessors," in *Proc. of the 14th Euromicro Conf. on Real-time Systems, Euromicro RTS 2002*, June 2002, pp. 223–230.

[102] M.J. Rutten, J.T.J. van Eijndhoven, and E-J. D. Pol., "Design of multi-tasking coprocessor control for eclipse," in *Proc. of 10th Int. Symp. on Hardware/Software Codesign, CODES 2002*, May 2002, pp. 139–144.

[103] F. Pereira, *MPEG-4: Why, What, How and When?*, http://leonardo.cselt.it/icjfiles/mpeg-4_si/2-overview_paper/2-overview_paper.htm.

[104] C. Herpel, and A. Eleftheriadis, "MPEG-4 systems: Elementary stream management," *Signal Processing Image Communication*, vol. 15, no. 4-5, pp. 299–320, Jan. 2000.

[105] Signès, J. and Fisher, Y. and Eleftheriadis, A., "MPEG-4's binary format for scene description," *Signal Processing Image Communication*, vol. 15, no. 4-5, pp. 321–345, Jan. 2000.

[106] ETS 300 744,, "Digital video broadcasting (DVB), framing structure, channel coding and modulation for digital terrestrial television," in *European Telecommunications Standards Institute*, March 1997.

[107] 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.

[108] M. Berekovic, *et al.*, "A multimedia RISC core for efficient bitstream parsing and VLD," in *Proc. of the SPIE – Multimedia Hardware Architectures*, Jan. 1998, vol. 3311, pp. 131–141.

[109] S. Howell, "Openml v1.0 specification," in *http://www.khronos.org*, July 2001.

[110] M. Berekovic, *et al.*, "Architecture of an image rendering co-processor for MPEG-4 systems," in *Proc. 2000 Int. Conf. on Application-Specific Systems, Architectures, and Processors*, July 2000, pp. 15–24.

[111] M. Schu, *et al.*, "System-on-silicon solution for high quality consumer video processing - the next generation," *IEEE Transactions on Consumer Electronics*, vol. 47, no. 3, pp. 412–419, Aug. 2001.

[112] S. Dutta, R. Jensen, and A. Rieckmann, "Viper: A multiprocessor soc for advanced set-top box and digital tv systems," *IEEE Design & Test of Computers*, vol. 18, no. 15, pp. 21–31, Sept.-Oct. 2001.

[113] E. Rijpkema, *et al.*, "Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip," in *Proc. of the Design Automation and Test conference, DATE'03*, March 2003.

[114] L. Donghee, *et al.*, "Implementation and performance evaluation of the lrfu replacement policy," in *Proc. of the 23rd Euromicro Conf., EUROMICRO'97*, Sept. 1997, pp. 106–111.

[115] E.R. Altman, *et al.*, "A nove methodology using genetic algorithms for the design of caches and cache replacement policy," in *Proc. of the Int. Conf. on Genetic Algorithms, ICGA'93*, July 1993, pp. 392–399.

Summary

IN consumer electronics, applications such as television, Internet, games, and mobile communication, combined with the strong desire for connectivity, requires a flexible yet powerful media-processing solution. Such a system enables simultaneous execution of very diverse tasks, ranging from stream-oriented processing to highly data-dependent irregular processing. Cost-effective implementations in the consumer domain, can only be achieved by carefully matching the algorithms and the architecture within the boundaries of many constraints, such as cost, power consumption, scalability towards future products, flexibility to adopt future features, etc. The balancing of these constraints is a complex multi-dimensional problem. This thesis concentrates on this problem by exploring an integral design approach for all algorithms and the associated architecture on which they are executed.

From analyzing the trends in the television domain, it has become clear that, the complexity and the diversity of processing tasks increases continuously, more flexibility is required, and reuse of design is necessary for cost reduction. Although a fully programmable solution seems attractive, a rough inventory of the functionality in a mainstream analog television receiver shows that a solution with video-processing functions, performing several billion of operations per second, is not readily implemented on a general-purpose computer system, while providing a sufficiently high performance density (i.e. the performance per unit area and per unit power). However, a fully dedicated solution with hardwired functions, which generally offers two orders of magnitude performance-density improvement, lacks the flexibility to enable e.g. sharing of resources, scalability over a larger product range, and the embedding of future functions.

Having presented the design problem, we elaborate on the developments in multimedia and video computing architectures and present a chronological selection of programmable Systems on a Chip (SoCs) showing the

increase in parallelism. Starting with a general-purpose processor, parallelism is increased thereby making a tradeoff between costs and the amount of application-specific hardware. It is explained that the control of the parallel and flexible hardware can be decomposed hierarchically, going from the application, to functions, to tasks, and finally fine-grain operations. To actually gain some insights in system architecting, we discuss a flavor of example video processing systems and indicate their merits and shortcomings for future designs.

In order to define a new architecture for television systems, we estimate the required resources, such as computational performance, memory usage, and communication bandwidths. For understanding these estimates, typical video processing algorithms for sharpness enhancement and scaling are discussed thoroughly. From an application point of view, it becomes clear that a video processing function may be implemented with different algorithms, depending on e.g. the required picture quality and the degree of flexibility. From an architectural point of view, aspects such as the speed of the processors, the available memory, compliancy with standardized interfaces, protocols, etc., influence the algorithmic choices. An example about sharpness enhancement shows that algorithms for signal-processing functions may differ significantly, depending on the requirements and constraints. We conclude that only an integral design approach of algorithms and the architecture, and the use of a heterogeneous multiprocessor system are inevitable for a cost-efficient system.

Then we present a new architecture proposal for a high-end television system, called TeleVision Processor (TVP). The system provides a programmable task graph so that the order of video functions can be altered dynamically. This allows the dynamic transition between different features in a TV set such as moving Picture-in-Picture with arbitrary shapes, zooming-in of a user-selected picture region, and picture quality improvement on selected video signals. Instead of statically scheduling the hardware for each mode of operation, the system features a Kahn process network, which means that the hardware is activated by the communication infrastructure if data for processing and memory space for the output is available. Another novelty of the system is the clear separation of the communication and the functional computations, thereby realizing independent behavior of each coprocessor. The presented system consist of two chips: a microcontroller chip with peripherals to perform event-oriented tasks and tasks that are less time critical, and a coprocessor array chip for powerful video processing with high parallelism.

Due to the continuous integration of increasingly complex functionality on a SoC, off-chip memory remains a costly part of the design. Particularly, the desire for a shared centralized off-chip memory induces a memory bandwidth bottleneck. In order to reduce the required bandwidth, we have studied existing memory devices and constructed a realistic model for the memory communication. For this model, we introduce the concept of data units containing the video data. In a next step, the selection of pixels contained by each data unit is optimized for efficient memory communication. These selections of pixels depend on the memory requests of the specific video application. Experimental results from an MPEG-2 decoder implementation indicate a 35 % reduction of the memory bandwidth for SDRAM devices. On top of this optimal memory usage, we introduce additional memory bandwidth reduction techniques. First, a straightforward embedded compression technique is presented showing a memory-bandwidth reduction of almost 40 %. Second, a study on function-specific caching shows that a moderate direct-mapped cache of 3 kByte reduces the bandwidth for high-definition MPEG-2 decoding with more than 30 %. The main difference between the first and the second bandwidth reduction technique, is that the first maintains picture quality whereas the second ensures a guaranteed bandwidth reduction.

The afore-mentioned TVP system has a programmable communication infrastructure that does not scale to complex applications requiring a high number of processing units. This observation leads to a hierarchical design of the communication infrastructure and the memory architecture. We propose a close match of this hierarchical design with the hierarchy that is enclosed in video processing applications. Consequently, communication occurs locally near the functional units and only globally between clusters of functional units. Moreover, the hardware can be controlled independently at each level of the hierarchy. This hierarchical approach is a central theme of the so-called Eclipse system that is discussed for implementation of MPEG-4 decoding. This system consists of a high-level communication network of coarse-grain functional units in conjunction with an off-chip memory, and a second-level communication network that comprises an embedded on-chip memory. The second-level communication network contains a heterogeneous mixture of dedicated coprocessors, and allows the embedding of fully-programmable processors. Hence, part of the decoder tasks can be performed by dedicated hardware, whereas other tasks are executed in software. Similarly to the TVP system, the Eclipse system features dataflow communication for easy programming and ordering of functions.

For many already existing video processing systems, the design of the architecture and the included video functions are performed more or less independently. The main differentiating contribution of this thesis is the codesign of the video processing algorithms and the corresponding system architecture. This results from explicitly coupling design aspects such as bandwidth, programmability, etc., with the analysis of the video processing functions. The thesis concludes with a future outlook in which a multimedia system is presented, featuring hierarchical communication and a heterogeneous mixture of processing units ranging from ASIPs, DSPs to CPUs. For this hierarchical communication infrastructure, the clustering of functional units at various levels is in accordance with the natural hierarchy of the video functions.

Samenvatting

Architectuurontwerp van video-bewerkingssystemen op een chip

OP het gebied van consumentenelektronica openbaart zich een tendens waarin applicaties zoals televisie, Internet, spelcomputers en mobile communicatie naar elkaar toe groeien. Ook tekent zich een wens af voor het onderling verbinden van consumentenapparaten. Deze trends leiden tot een vraag naar een flexibel en krachtig mediabewerkingssysteem dat in staat is om zowel reguliere taken voor datastromen als onregelmatige, data-afhankelijke taken uit te voeren. Om voor dergelijke systemen een kosten-effectieve implementatie te maken, moeten de videobewerkingsalgoritmen en de architectuur goed op elkaar worden afgestemd binnen de beperkingen van o.a. kosten, vermogensverbruik, schaalbaarheid naar toekomstige producten, en flexibiliteit om toekomstige functionaliteit toe te voegen. Het maken van deze afwegingen is een complex multidimensionaal ontwerp-probleem. Dit proefschrift gaat over het onderzoeken van een integraal ontwerpproces van de videobewerkingsalgoritmen en de bijbehorende systeemarchitectuur.

Het analyseren van de trends in het televisiedomein maakt duidelijk dat de complexiteit van systemen alsmaar groeit, dat steeds meer flexibiliteit vereist is voor een grote verscheidenheid van taken, en dat ook hergebruik van het ontwerp voor toekomstige producten belangrijk wordt om kosten te reduceren. Hoewel een volledig programmeerbare en generieke oplossing aantrekkelijk lijkt, blijkt uit een inventarisatie van de videobewerkingsfuncties dat hiervoor miljarden operaties per seconde nodig zijn. Het is dan ook onmogelijk om dit kosteneffectief te realiseren met een volledig programmeerbare en generieke oplossing. Hoewel een rigide applicatiespecifieke hardwareoplossing een prestatiedichtheid (prestatie per eenheid siliciumoppervlakte en vermogen) levert die twee orden hoger is, biedt deze oplossing onvoldoende flexibiliteit voor o.a. het schalen van de oplossing over een grotere verscheidenheid van producten en het uitbreiden met toekomstige nieuwe functies.

Na een uiteenzetting van het genoemde ontwerpprobleem zal worden ingegaan op de ontwikkelingen van videobewerkingsarchitecturen, gevolgd door een historisch overzicht van programmeerbare systemen met steeds meer parallellisme. Hierbij wordt continu een afweging gemaakt tussen de hoeveelheid applicatiespecifieke hardware en de bijbehorende kosten. Het aansturen van dergelijke flexibele hardware kan hiërarchisch worden opgedeeld in applicaties, functies, videobewerkingstaken en fijnkorrelige operaties. Om inzicht te verwerven in systeemarchitectuur, worden enige videobewerkingssystemen geëvalueerd en worden de voor- en nadelen voor toepassing in toekomstige systemen uiteengezet.

Om een nieuwe architectuur voor televisiesystemen te definiëren, maken we eerst een schatting van de benodigde systeembehoeften zoals rekenkracht, geheugengebruik en communicatiebandbreedte. Om dit te begrijpen, wordt uitgebreid ingegaan op de algoritmen van een geavanceerde scherpteverbetering en een videoschaler. Daaruit blijkt duidelijk dat functies geïmplementeerd kunnen worden met verschillende algoritmen, afhankelijk van o.a. de beeldkwaliteitseisen en de gewenste flexibiliteit. Naast deze gebruikerseisen zijn er ook beperkingen die vanuit de architectuur worden opgelegd, zoals de snelheid van de processors, de beschikbare hoeveelheid geheugen, het voldoen aan gestandaardiseerde verbindingen, en communicatieprotocollen. Door middel van een scherpteverbeteringsfunctie wordt het verschil aangetoond tussen verschillende algoritmen, afhankelijk van de opgelegde eisen en systeembeperkingen. Hieruit mag worden geconcludeerd dat het integraal ontwikkelen van de videobewerkingsalgoritmen en de bijbehorende architectuur met heterogene multiprocessors onvermijdelijk is.

Vervolgens wordt het voorstel van een nieuwe architectuur voor hoogkwalitatieve televisiesystemen (TVP) beschreven. Het systeem beschikt over de mogelijkheid om het signaalpad door de set van bewerkingseenheden (coprocessors) met behulp van software dynamisch te wijzigen. Hiermee kunnen bijv. naadloze overgangen worden gemaakt tussen een 'Picture-in-Picture' (PiP)-functie, een zoomfunctie om geselecteerde uitsneden van een beeld uit te vergroten, en een functie die door een externe processor wordt uitgevoerd. Traditioneel moet voorafgaand aan de beeldbewerking, voor iedere instelling van het systeem een tijdschema worden gemaakt om alle hardware op de juiste momenten aan te sturen. Het TVP-systeem bevat een Kahn-bewerkingsnetwerk, wat inhoudt dat het communicatienetwerk zelf de videobewerkingsfuncties activeert op het moment dat er voldoende data aanwezig zijn op de ingang en er voldoende geheugenruimte is aan

de uitgang om het resultaat weg te schrijven. Een andere belangrijke toevoeging aan het systeem is de ontkoppeling van de datacommunicatie enerzijds en de functionele bewerkingen anderzijds, waardoor de coprocessors volledig onafhankelijk van elkaar opereren. De implementatie van het systeem bestaat uit twee chips: een 'microcontroller'-chip inclusief een aantal extra bewerkingseenheden voor de taken die minder tijdkritisch en meer irregulier van karakter zijn, en een matrix van coprocessors voor rekenintensieve videobewerking met veel parallellisme.

Door de continuerende integratie van steeds complexere functies op een chip blijft het externe geheugen een kostbaar onderdeel van het systeem. Vooral de behoefte aan een gecentraliseerd extern geheugen leidt tot een knelpunt voor de geheugenbandbreedte. Om de benodigde bandbreedte te reduceren, hebben we bestaande geheugenchips bestudeerd en een realistisch model geïmplementeerd voor het simuleren van de geheugencommunicatie. Dit model maakt gebruik van zogenaamde data-eenheden waarin de videodata zijn opgeslagen. Vervolgens wordt de selectie van pixels voor iedere data-eenheid geoptimaliseerd voor de meest efficiënte communicatie. Deze selecties van pixels zijn afhankelijk van de verzoeken die door de specifieke videoapplicatie gedaan worden aan het geheugen. Experimenten met een MPEG-2-decoder laten zien dat de geheugenbandbreedte met 35 % afneemt. Naast dit geoptimaliseerde gebruik van het geheugen worden ook nog enige extra technieken beschreven, die als doel hebben de geheugenbandbreedte te beperken. De eerste techniek betreft een relatief eenvoudige compressiemethode waarmee de bandbreedte met bijna 40 % afneemt. De tweede techniek omvat een studie over het gebruik van een functiespecifieke cache. Hiermee wordt aangetoond dat met een kleine cache van 3 kByte met een één-op-één-afbeelding ('direct-mapped') de geheugenbandbreedte voor hogeresolutie-MPEG-2-decodering met meer dan 30 % afneemt. Het grote verschil tussen deze technieken is dat de eerste het behoud van beeldkwaliteit garandeert, terwijl de tweede een gegarandeerde afname van de bandbreedte geeft.

Het genoemde TVP-systeem heeft een programmeerbare communicatie-infrastructuur, die niet makkelijk te schalen is naar complexe applicaties met meer bewerkingseenheden. Deze observatie leidt tot een hiërarchisch ontwerp van de communicatie-infrastructuur en de geheugenarchitectuur. Wij stellen voor om de hiërarchie van het ontwerp overeen te laten komen met de hiërarchie die van nature in de videobewerkingsapplicatie zit. Daardoor kan de communicatie lokaal en efficiënt bij de videobewerkingseenheden worden uitgevoerd en is er alleen globale communicatie via de hogere

lagen van de hiërarchie. Bovendien kan hierdoor de hardware onafhankelijk van elkaar worden aangestuurd op ieder niveau van de hiërarchie. Deze hiërarchische aanpak is een centraal thema van het zogenaamde Eclipse-systeem, dat uitvoerig wordt besproken voor de implementatie van MPEG-4-decodering. Dit systeem bestaat uit een hoogniveaucommunicatienetwerk voor grofkorrelige bewerkingseenheden in samenwerking met een extern geheugen, en een tweedeniveaucommunicatienetwerk dat bestaat uit een lokaal geïntegreerd geheugen en bewerkingseenheden die minder grofkorrelig zijn. Dit tweedeniveaucommunicatienetwerk bevat een heterogene mix van applicatiespecifieke coprocessors, maar staat ook volledig programmeerbare generieke processors toe. Hierdoor kan een deel van de MPEG-4-decodertaken worden afgebeeld op de applicatiespecifieke taken, terwijl de overige taken worden uitgevoerd in software. Net zoals het TVP-systeem bevat het Eclipse-systeem een Kahn-bewerkingsnetwerk om het programmeren eenvoudig te houden en om de volgorde van functies te kunnen veranderen.

Voor veel bestaande videobewerkingssystemen is het ontwerp van de architectuur en de videobewerkingsalgoritmen onafhankelijk gedaan. Het belangrijkste resultaat van dit proefschrift is het aantonen van de toegevoegde waarde van het gecombineerd ontwerpen van de architectuur en de videobewerkingsalgoritmen. Deze conclusie is ontstaan uit het expliciet koppelen van aspecten zoals de bandbreedte en programmeerbaarheid met het analyseren van videobewerking functies van het systeem.

Het proefschrift wordt afgesloten met een blik op de toekomst. Hierbij wordt een multimediasysteem gepresenteerd, dat hiërarchische communicatie biedt met een mix van heterogene bewerkingseenheden zoals ASIP's, DSP's en CPU's. Het clusteren van de bewerkingseenheden op de verschillende hiërarchische niveaus van het systeem is afgestemd met de natuurlijke hiërarchie van videofuncties.

Biography

Egbert Jaspers was born in Nijmegen, the Netherlands, in 1969. He studied *Electrical Engineering* at the Venlo Polytechnical College, which resulted in the B.Sc. degree in 1992. Subsequently, he joined Philips Research Laboratories in Eindhoven. For one year, he worked on video compression for digital HDTV recording in the *Magnetic Recording Systems* department. In 1993, he left Philips for three years to pursue a M.Sc. degree at the Eindhoven University of Technology, from which he graduated in 1996. In the same year he joined the Philips Research Laboratories in Eindhoven as a Research Scientist in the *Video Processing and Visual Perception Group*. He participated in the design of several video-processing functions for implementation in an advanced video processor system. Gradually, he refocused his research to the architecture-related aspects of video processing and contributed to projects, developing heterogeneous multiprocessor architectures for consumer systems. In 2000 he received a Chester Sall Award for the best papers of the IEEE CE Transactions in 1999. His current interest focusses on scalability and reuse aspects of system design for implementation of complex video-processing functions. He has published a number of papers on international conferences and was member of the review committee of the ICECS (Int. Conf. on Electronics, Circuits and Systems) and member of the program committee of the PDIVM (Parallel and Distributed Computing in Image Processing, Video Processing, and Multimedia) where he also attended as an invited lecturer.

Index

D

E

F

G

H

I

J

K

L

M

S

W

The work described in this thesis has been carried out at the Philips Research Laboratories Eindhoven as part of the Philips Research programme.