VIDEO SIGNAL PROCESSING

In digital video signal processing, an incoming data stream representing some form of video signal is continuously processed into an outgoing data stream representing some other form of video signal or some information derived from the incoming video signal. Intermediate processing between input and output generally comprises a limited set of tasks being executed repetitively on the streaming data in a periodic manner. Video signal processing tasks may involve operating both on individual data samples as well as on compound data samples interpreted as symbolic or object information. Typical applications in the rapidly growing field of video signal processing include digital TV broadcasting, visual communication, surveillance systems, object recognition and tracking, and many others.

In general, video signal processing applications are characterized by very high computational demands, resulting from complex sequences of operations to be performed on large

Table 1. Comparison of Source Data Rates for Various Signal Processing Applications (fps = frames per second)

Data Type	Parameters	Source Data Rate	Application
Speech	8 kHz, 8 bit	64 kbit/s	ISDN
Audio	44.1 kHz, 16 bit stereo	1.5 Mbit/s	CD-ROM
Video	352 × 240 pixel, 8 bit, 15 fps	30.4 Mbit/s	Video conferencing
—	1280 × 720 pixel, 8 bit, 60 fps	1.3 Gbit/s	HDTV

data volumes at high sampling rates. Table 1 compares source data rates of various signal processing applications. When assuming that the computation rate is roughly proportional to the source data rate, video applications yield orders of magnitude higher processing requirements than other signal processing applications. In order to meet human visual perception characteristics, real-time processing is frequently mandatory, which means processing speed has to keep pace with the display rate. Therefore, maintaining a high data throughput is of great importance. With increasing sophistication of applications, the number of operations per data sample also rises. These factors together are responsible for the extreme performance requirements in video processing. A prominent example to illustrate the high computational demands of video signal processing applications is given by current video compression algorithms that may involve up to several billions of arithmetic operations per second, depending on image format and frame display rate.

For practical applications of video signal processing, another issue frequently becomes important: the commercial success of new video services and applications fundamentally depends on the availability of compact hardware devices that are inexpensive both to purchase and to maintain. Therefore, video signal processing devices for such services call for low-cost implementations featuring low power consumption while providing the required high performance capacity. Examples of emerging devices with these demands are set-top boxes for digital TV or mobile phones for video communication, both targeting mass-market consumer applications with considerable economic potential. On the other hand, an increasing diversification of video signal processing applications and the growing demand for joint processing of various data types—for example, in the field of multimedia—require integrated solutions of high flexibility and versatility.

The specific requirements of video signal processing applications cannot be met by conventional processor architectures due to insufficient performance and/or high implementation cost. Therefore, digital video signal processors have emerged as new processing devices that are particularly well adapted to the characteristics and demands of video signal processing algorithms. The required high performance levels are attained with architectures providing a high concurrency of operations. Cost-effective very large scale integration (VLSI) implementation, on the other hand, is achieved by careful analysis of target algorithms followed by architectural adaptation to their characteristics in order to avoid unnecessary hardware overhead. The combination of these design strategies results in superior solutions for various video signal processing applications.

GENERAL DESIGN APPROACHES FOR VIDEO SIGNAL PROCESSORS

In this section, special characteristics of video signal processing algorithms are identified, and basic architectural alternatives for video signal processors are derived with consideration of these special algorithm characteristics.

Algorithm Characteristics in Video Signal Processing

Architecture design for video signal processors is fundamentally driven by the specific demands and characteristics of typical video signal processing algorithms. An outstanding feature of most video processing algorithms is their ample potential for parallelization on various levels. In general, parallelization opportunities can be classified into data parallelism, instruction parallelism, and task parallelism:

- Data parallelism denotes the identical processing of multiple data entities at the same time. For example, operations that have to be performed on each pixel of an image in an equal manner could be executed simultaneously for all data entities, provided that sufficient hardware resources are available.
- Instruction-level parallelism comprises the concurrent execution of multiple distinct instructions of a task. An example is given by performing a multiplication, a shift operation, and a data access in parallel, given the corresponding functional units are available simultaneously and can operate independently.
- Task parallelism stands for the independent execution of complete operation groups making up entire computation blocks of an algorithm. As an example, a filter algorithm and a segmentation algorithm could run in parallel on independent processing modules.

By exploiting the parallelization potential of video signal processing algorithms on multiple levels, performance and throughput capabilities of video signal processors can meet real-time processing demands.

Besides their parallelization opportunities, video signal processing algorithms exhibit more features that can be exploited by architectural adaptation in order to achieve higher computation efficiency. One of these features is the frequent operation on small integer data operands representing pixel values, which leads to poor utilization of the wide data paths typically found in conventional processors. By incorporating multiple data paths adjusted to the small word length, or by allowing several small data operands to be processed in a wide data path in parallel, processing efficiency can be enhanced considerably. Another feature to be observed, particularly in the most computation-intensive parts of video processing algorithms, is the limited variety of encountered operations. By restricting the choice of function units to a small number of highly optimized modules, a higher efficiency in terms of silicon area can be obtained for video signal processors, and a higher performance can be achieved.

In general, complex video processing algorithms are composed of various subtasks with different parallelization potential and computation characteristics, thus exhibiting a heterogeneous nature. The incorporated subtasks can roughly be classified into low-level and high-level type. Low-level tasks are characterized by data-independent, highly deterministic control flow but involve individual processing for all data samples and are, therefore, very computation-intensive. High-level tasks, on the other hand, operate on a lower number of symbols or objects and require much less computational power. However, their control flow strongly depends on intermediate computation results and cannot therefore be predicted in advance. As a consequence, the diverse computational characteristics of video processing subtasks make largely different demands on corresponding hardware modules.

As an example for a complex, heterogeneous algorithm, the hybrid video coding scheme underlying several recent video compression standards [e.g., ISO MPEG-1 (1), ISO MPEG-2 (2), ITU H.261 (3), ITU H.263 (4)] is considered. Figure 1 gives an overview of the encoder. The complete scheme comprises the tasks of discrete cosine transform (DCT), inverse discrete cosine transform (IDCT), motion estimation (ME), motion compensation (MC), variable length coding (VLC), variable length decoding (VLD), quantization (Q), inverse quantization (IQ), as well as some coder control. Of these subtasks, DCT, IDCT, ME, and MC can be classified as pure low-level tasks; they operate on individual data samples and account for up to 90% of the computational demands for hybrid video coding. The other subtasks—VLC, VLD, Q, IQ, and coder control—are less computationally intensive and show more high-level characteristics in varying degrees. These tasks together make up about 10% of the computational requirements but are more dependent on the actual video data that has to be processed. Architectural solutions for some hybrid video coding subtasks will be presented in a later section.

Basic Architectural Alternatives

For efficient video signal processor design, the forementioned algorithm characteristics have to be carefully considered. Depending on the targeted application field, two general approaches exist for video signal processing devices: dedicated and programmable architectures. Dedicated architectures allow a fixed algorithm to be implemented with highest efficiency by full architectural adaptation to the algorithm's computational characteristics. As they involve minimum control overhead due to a well-defined, fixed operation flow, dedicated architectures are able to deliver sufficiently high performance

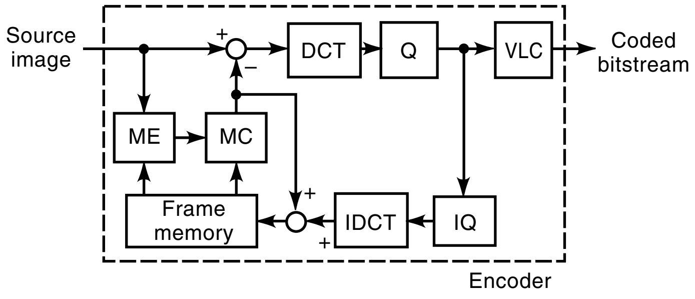

Figure 1. Overview of the hybrid coding scheme. The encoder comprises the subtasks ME, MC, DCT, Q, IDCT, IQ, VLC, and some coder control. The decoder operation is incorporated in the encoder.

at comparably low hardware expense. Programmable architectures, on the other hand, provide the flexibility to execute different algorithms on the same device by software modifications. Furthermore, they are able to execute program code involving highly data-dependent control flow. However, programmable architectures incur higher hardware cost for control functions and program storage, and their higher flexibility typically leads to lower utilization of hardware resources. The decision between dedicated and programmable architectures finally depends on the degree of flexibility required, on the predictability of the operation flow, and on the envisioned production volumes of an application.

A mixture of both dedicated and programmable modules within one device is possible. Such an approach makes sense when considering the heterogeneous nature of complex video signal processing algorithms. Following the description of low-level and high-level tasks given in the preceding, low-level tasks would preferably be implemented on dedicated modules, whereas high-level tasks appear to be executed more efficiently on programmable modules. Such a partitioning leads to a typical coprocessor architecture. Adequate design examples of coprocessor architectures reported in the literature are presented in a later section.

The opportunities for parallel processing in video algorithms can be exploited by both dedicated and programmable architectures. In the design of dedicated architectures, formal methods exist (5) for the mapping of regular algorithms to corresponding architectures that enable a tradeoff between processing speed and hardware effort by supporting different degrees of parallelism. The number of parallel computation units has to be matched to the real-time processing demands. A higher degree of parallelism is generally not desired as it would increase hardware cost without enhancing the targeted application. The automated design flow based on the formal mapping process yields short design times for dedicated architectures of regular structures.

Incorporating parallel processing into programmable architectures leads to the design of multiprocessor systems. There are a large variety of alternative approaches in this field. A detailed examination of design alternatives for programmable multiprocessors is presented in a later section. In contrast to dedicated architectures, the implementation of video signal processing applications on programmable architectures involves for the most part the design of software. While a single programmable architecture can be utilized for a variety of applications, software development for individual applications can assume considerable dimensions in terms of time and cost requirements. The productivity of code development can be enhanced significantly by the availability of high-level language compilers; however, highest code efficiency and best resource utilization is only achieved by low-level assembly language programming. With the growing complexity of applications, efficient high-level language support becomes increasingly important.

Besides utilization of parallel processing principles, a concurrency of operations can also be achieved by the introduction of pipelining (6). In a pipelined architecture, numerous function units are arranged in a cascade, each performing a single step of a complete operation. The input data are passed through the different stages of the cascade until they are readily processed. Between the pipeline stages, intermediate memories (registers) are required to store the partial results.

Pipeline implementations can range from suboperation level (micro-pipelining) up to task level (macropipelining). Although the actual computation time (latency) required for processing an individual data item cannot be decreased by pipelining, a significant improvement in the throughput rate can be achieved as processing of consecutive data items can be started at the short interval of a single pipeline stage. As the throughput rate is an important performance criterion in real-time video processing, pipelining is widely employed in both dedicated and programmable video signal processors. The combination of pipelining and parallel processing within an architecture design offers the opportunity to achieve highest performance and throughput levels.

After this general introduction into design issues of video signal processors, architectural measures for dedicated and programmable processors are surveyed in more detail in the following sections, and an overview of existing design examples is given later.

ARCHITECTURAL MEASURES FOR PROGRAMMABLE VIDEO SIGNAL PROCESSORS

Programmable architectures provide the flexibility to allow various algorithms to be executed on the same hardware. Different functionality can be achieved by software modifications without the need for hardware changes. This is important for systems aiming at wider application fields, or when later algorithm modifications have to be taken into account. Particularly, in rapidly evolving areas, such as video compression, numerous examples for late extensions of existing standards have been observed. On programmable architectures, extensions of already implemented applications can be incorporated simply by software update.

The possibility of applying software extensions for the implementation of individual product features is another advantage of programmable architectures. It allows system vendors to differentiate their products from those of other vendors, which would be difficult to achieve with standard devices of fixed functionality. The deviation from standardized procedures in applications is useful, for example, to restrict the accessibility of specific data and services to devices from a particular vendor. Such strategies increasingly gain economic importance for services incorporating digital video signal processing.

A further important point for programmability arises from the growing complexity and decreasing predictability of emerging video signal processing algorithms. Dedicated architectures are not applicable to algorithms with highly content-dependent operation flow; only programmable processors provide the flexibility to deal with arbitrary conditional execution encountered in new applications in large scale. Therefore, emerging applications in video signal processing will increasingly become the domain of programmable processors.

Flexibility and programmability are features already offered by conventional general-purpose microprocessors as found in workstations, PCs, or embedded applications. Continuing progress in VLSI technology leads to an ever-increasing computational power of these devices. Nevertheless, general-purpose processors fail to perform video signal processing tasks efficiently as they do not exploit the special characteristics of video algorithms. Generally, conventional processors in video processing suffer from poor utilization of available hardware resources and spend too many clock cycles for conceptually simple, but frequently recurring operations. As a consequence, they cannot reach the required high performance levels, and they are, moreover, too expensive for most video applications as they incorporate various hardware units that are not utilized in video signal processing.

The architectural approaches for programmable video signal processors to overcome the limitations and drawbacks of conventional processors in video signal processing can be classified into two main strategies: (1) parallelization on data, instruction, or task level yielding a massive increase of available processing power; and (2) adaptation to special algorithm characteristics by implementing specialized instructions and dedicated hardware modules, resulting in higher efficiency for a limited application field. In existing designs, a mixture of both directions is frequently employed. In addition to these approaches, pipelining is generally introduced in programmable video signal processors to increase clock frequency and data throughput. A number of architectural measures for programmable processors based on the two principles of parallelization and adaptation are examined in the following.

Parallelization Strategies

Single Instruction Stream, Multiple Data Stream. The data parallelism inherent in video signal processing algorithms can effectively be exploited by single instruction stream, multiple data streams (SIMD) multiprocessor architectures (7). As depicted in Fig. 2, SIMD processors are characterized by a number of identical data paths that execute the same operation on multiple data items in parallel. As the same instruction stream is issued to all parallel data paths, only a single common control unit is required, and most of the silicon area on a chip can actually be spent for a multitude of processing units to yield high data parallelism. While a high degree of parallelism can thus be achieved with little control overhead, pure SIMD architectures are not without problems for implementation of practical applications. The lack of flexibility in data path controlling limits efficient use of SIMD processors to algorithms with highly regular computation patterns, that is, low-level algorithms. In case of data-dependent operations or scalar program parts, a number of data paths may be idle

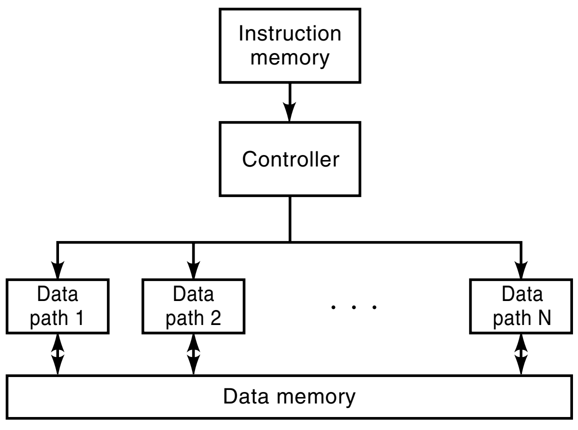

Figure 2. The SIMD multiprocessor architecture. The global controller provides a single instruction stream to all parallel data paths.

most of the time, and the processor utilization decreases rapidly.

Split-ALU. Based on a principle similar to SIMD, the split-ALU concept also targets data parallelism in video signal processing algorithms. This concept, also referred to as subword parallelism, involves processing of multiple lower-precision data items in parallel on a single wide ALU (8). On a 64-bit ALU, for example, eight 8-bit data items can be processed in parallel by executing a single instruction. As a prerequisite, minor hardware extensions are necessary to prevent carry signals arising during arithmetic operations from being propagated across the boundaries of separate data items. Figure 3 shows a possible split-ALU implementation. As for SIMD architectures, the benefit of a split-ALU is highest only for low-level algorithms comprising identical operations to be performed on large data volumes. Moreover, the obtainable data parallelism depends on the precision required for an operation: in case of higher wordlength demands, the degree of utilizable parallelism decreases. Fortunately, most computation-intensive low-level video signal processing algorithms involve operating on low-precision (8-bit) video data; therefore, subword parallelism can effectively be employed to speed up these program parts. By providing several split-ALU instructions for different data formats, the achievable data parallelism can scale with the precision demands of algorithms and operations.

The small incremental hardware cost for a split-ALU—provided a wide ALU is already available—makes this concept well-suited for the extension of existing general-purpose processors with respect to video signal processing. Likewise, a split-ALU may be preferable for pure video signal processors where the availability of a wide ALU can be appreciated for the execution of program parts with higher precision demands. Typical operations to be performed in a split-ALU include addition, multiplication, or compare. As a drawback, split-ALU instructions are generally not supported by high-level language compilers due to the lack of adequate language constructs to express the desired operations.

Very Long Instruction Word. An approach aiming at instruction-level parallelism is represented by the very long instruction word (VLIW) architecture concept. In a single long instruction word, several operations are specified to be executed concurrently. Multiple function units have to be implemented to enable concurrent execution. The assignment of individual operations to function units is generally achieved by static mapping via operation slots defined in the VLIW. Figure 4 illustrates the basic structure of a VLIW architecture (9).

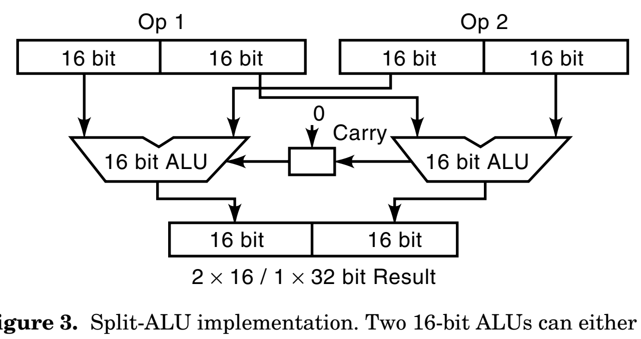

Figure 3. Split-ALU implementation. Two 16-bit ALUs can either be combined to a 32-bit ALU by propagating the carry signal or operate independently by blocking the carry signal.

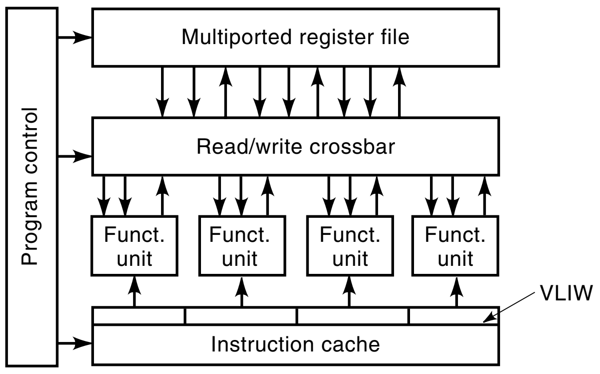

Figure 4. The VLIW architecture. Multiple function units are targeted by separate operation slots in a single very long instruction word. A multiported register file is required to provide simultaneous access for all function units.

In contrast to superscalar execution, which is another way of exploiting instruction-level parallelism frequently employed in high-end general-purpose microprocessors, VLIW architectures have to rely on static instruction scheduling at compile time to assemble the long instruction words. As an advantage, no additional hardware units for dynamic code reordering at run time—for example, reservation stations or reorder buffers as in superscalar architectures—are required. Therefore, more silicon area is available for multiple function units, enabling a wider variety of operations to be implemented and a higher degree of parallelism to be achieved. However, effective hardware utilization and actually achieved parallelism depend fundamentally on the compiler technology available. In essence, VLIW architectures shift complexity from hardware to the compiler.

Performance gains of VLIW architectures depend strongly on the degree of exploitable instruction-level parallelism inherent in the target algorithm. In order to enhance parallelization opportunities, sophisticated compiler techniques, such as loop unrolling or guarded execution, may be applied that aim at increasing the pool of instructions to be scheduled into the long instruction words. With a powerful compiler, a degree of parallelism even higher than in superscalar architectures may be achieved due to larger code units that can be inspected at a time. On the other hand, VLIW architectures are not well-suited for program flow involving frequent run time dependencies that would benefit from efficient branch predictions schemes, which thus remains the domain of architectures with dynamic scheduling.

Multiple Instruction Streams, Multiple Data Streams. For exploitation of both task level as well as data level parallelism, multiple instruction streams, multiple data streams (MIMD) architectures (7) are a possible solution. In contrast to SIMD processors, each data path of an MIMD architecture features a private control unit, as indicated in Fig. 5. Thus, each data path can execute an individual program, hence enabling the exploitation of task level parallelism. Likewise, several data paths can execute the same operation sequences as well, thus allowing exploitation of data parallelism. The high flexibility is the major advantage of MIMD architectures, equally en-

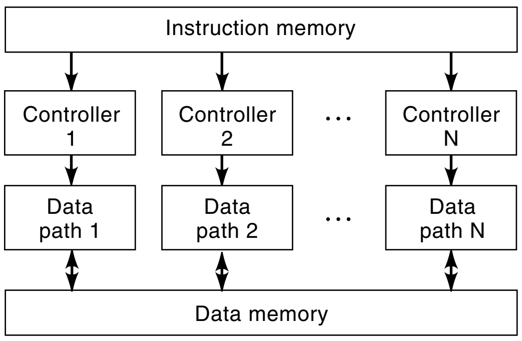

Figure 5. The MIMD multiprocessor architecture. Each parallel data path features a private control unit providing an individual instruction stream.

abling the execution of low-level and high-level tasks. However, the duplication of control units results in much higher silicon area demands for a single data path and thus limits the achievable parallelism on a chip. Moreover, the high demands on instruction memory bandwidth for continuously supplying instruction streams to the control units can easily become a performance bottleneck. Due to the high hardware cost associated with MIMD processors, they do not constitute efficient solutions for most video signal processing applications.

Other difficulties with MIMD processors include poor programmability and lack of synchronization support. Typically, high-level language compilers do not offer support for parallelization. Therefore, separate programs have to be developed for individual data paths, and synchronization between different data paths typically has to be achieved manually. These may be the main reasons why MIMD processors have not been in widespread use in commercial video applications so far.

Adaptation Strategies

Specialized Instructions. Adaptation of programmable processors to special characteristics of video signal processing algorithms can be achieved by introducing specialized instructions for frequently recurring operations of higher complexity. An example is given in Fig. 6 by the multiply-accumulate operation with subsequent saturation that would require a high number of conventional instructions to be executed. This operation is frequently used in many video signal processing applications. Thus, the introduction of a specialized instruction for this operation sequence reduces the instruction count significantly and results in faster program execution. The benefit of specialized instructions rises with the frequency of their use. Therefore, specialized instructions are not useful in general-purpose processors but offer a great benefit in video signal processors, which have frequent encounters with a limited number of special operational sequences.

The introduction of specialized instructions generally requires the implementation of additional function units, for example, multiply-adder. However, the design complexity of these additional units can usually be kept at modest levels due to high specialization and optimization. The decision about which instructions should be implemented finally has to depend on the probability of their use.

Coprocessor. Both parallelization and adaptation principles are utilized in coprocessor architectures. Typically, a flexible general-purpose processor module, for example, a standard reduced instruction set computer (RISC) processor core, is coupled with a dedicated processing element highly adapted toward a specific function, as illustrated in Fig. 7. Thus, the highly computation-intensive low-level algorithm parts of an application can be efficiently executed on the adapted coprocessor module, while the general-purpose processor core takes over execution of the data-dependent high-level parts having fewer computational demands. Specialized coprocessor modules can also be considered for subtasks with lower computation requirements if they exhibit other special characteristics that make them difficult to implement on standard processor cores.

Coprocessor architectures can be extended to heterogeneous multiprocessors by incorporating multiple adapted modules for different specific subtasks. The combination of parallel processing by providing two or more independent processing modules and adaptation by integrating a highly specialized coprocessing element allows sufficient processing power to be provided for the target video processing algorithms at increased efficiency. For algorithms that cannot make use of the specialized coprocessing modules, however, efficiency decreases rapidly. In consequence, coprocessor architectures offer less flexibility regarding algorithm modifications as significant changes in the target application may lead to a highly unbalanced resource utilization.

Memory System Design

Besides the measures already discussed to increase the processing performance of programmable architectures by paral-

```
Operation:    r2 = sat(r2 + sat(#imm × r1))

sri r3, #imm          r3 := #imm
mulcc r3,r3,r1        r3 := r3 × r1
jmpcc ov, _sat1       saturate, if overflow
add r2, r3, r2        r2 := r3 + r2
jmpcc ov,_sat2        saturate, if overflow

maci r2, r1, #imm     specialized instruction
```

Figure 6. Specialized instruction for multiply-accumulate with saturation. A longer sequence of standard instructions is replaced by a single specialized instruction. Branching to saturation subroutines is eliminated.

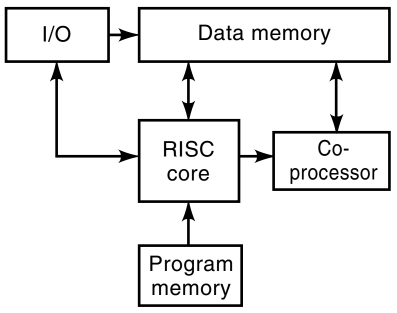

Figure 7. Coprocessor architecture. The efficiency of a dedicated module is coupled with the flexibility of a general-purpose RISC core. The RISC core performs global control functions.

lelization and adaptation strategies, memory system design for video signal processors deserves special attention. As video signal processing applications operate on large data volumes, the memory system has a considerable effect on overall performance. While general-purpose processors employ local on-chip caches to speed up data access times on average, conventional cache strategies have to fail for video applications because of the stream-like nature of the incoming data. Caches rely on the occurrence of frequent accesses to the same data items, which is not given in video processing where individual data items are continuously replaced by new data. However, for regular program parts, memory access patterns are typically predictable. Therefore, special stream caches have been proposed that employ prefetching techniques to access shortly needed data in advance (10). In addition to the streaming video data, other data structures of a nonvolatile nature may be involved in video signal processing applications, for example, look-up tables. These data may be best placed into on-chip memory—not cache—where they are accessible within the shortest time and are safe from being replaced by other data.

Instruction memory behavior in video signal processing differs fundamentally from data behavior. Video signal processing applications typically involve a limited set of tasks being executed periodically on the incoming data stream. As a consequence, the same instructions will be fetched and executed repetitively in the course of video processing. Thus, conventional cache strategies may be successful for instructions, provided the cache is large enough and mutual code replacement can mostly be prevented. To achieve this goal, code-positioning techniques can be applied to create a code layout in memory that minimizes cache replacement. Another possibility is the integration of on-chip memory for instructions, guaranteeing the fastest access to the most heavily executed code parts.

Software Issues

Of particular importance for programmable video signal processors is the support for software development. Although high-level language compilers are frequently available and suitable to enhance code development productivity, best hardware utilization and highest performance are generally achieved only by low-level assembler programming. Therefore, a typical software design flow for video signal processing applications may involve mixed high-level/low-level programming: after exploration of parallelization opportunities, the entire target application is described in a high-level language. Then, the program parts with the highest performance requirements are identified by execution profiling, and these code parts are optimized on assembly language level. This software design flow can iteratively be applied until existing performance requirements and/or real-time constraints are met. Increasingly, hardware vendors provide entire code libraries that contain optimized low-level implementations of specific tasks frequently encountered in video signal processing applications (11). Alternatively, program parts with the highest processing requirements may be offloaded to dedicated hardware modules by applying a hardware-software codesign approach (12), leading to a coprocessor/heterogeneous multiprocessor architecture.

The architectural measures for programmable video signal processors presented in this section differ widely in terms of hardware cost and flexibility. Individual measures are generally not applied exclusively; in most existing architectures, various approaches are combined to obtain sufficient processing power with efficient use of hardware resources. The best architectural mix depends on the respective targeted application field.

DESIGN APPROACHES FOR DEDICATED ARCHITECTURES

In contrast to programmable video signal processors, dedicated architectures are designed to perform one specific task. The specialization opens up opportunities for aggressive optimizations in terms of performance, cost, and power consumption. In the following, a number of dedicated solutions for specific video signal processing tasks are presented.

Due to its prominence among video signal processing algorithms, the hybrid coding scheme employed in video compression will be used in this section as an example algorithm to demonstrate dedicated implementations of various subtasks. Within the hybrid video coding scheme, potential candidates for a dedicated implementation include DCT, ME, and VLD. Both DCT and ME are characterized by very high computational requirements, and they exhibit algorithmical regularities that can be exploited for automated derivation of suitable architectures. For these algorithms, architectures are typically designed that utilize parallel processing and pipelining principles in order to break down execution time by achieving a concurrency of operations (13). In contrast, VLD is an example for a more irregular algorithm with little parallelization potential. However, as VLD cannot be implemented efficiently on most programmable general-purpose processors, dedicated implementation is advantageous for this task. In the following, function-specific architectural approaches for DCT, ME and VLD will be presented.

Discrete Cosine Transform

The two-dimensional DCT as a real-valued frequency transform plays a key role in image and video processing. Equation (1) expresses the transformation of an $L \times L$ image block, where $c_{i,k}$ and $c_{j,l}$ stand for the transform coefficients comprising cosine functions:

$$y_{k,l} = \sum_{i=0}^{L-1} c_{i,k} \cdot \left[\sum_{j=0}^{L-1} x_{i,j} \cdot c_{j,l} \right] \tag{1}$$

The core operation in DCT computation is the combination of multiplication and accumulation, commonly referred to as MAC operation. Equation (1) already indicates the possible decomposition of the 2D transform into two separate 1D transforms, thus reducing the number of MAC operations to be performed from L^2 down to $2L$ per pixel. Benefitting from the reduced operation number of the separated DCT, a possible hardware implementation comprises a cascade of two 1D transform blocks and an intermediate transposition unit. For efficient implementation of the 1D-DCT, several alternatives are known from the literature (14). A direct implementation of the underlying matrix-vector multiplication based on four

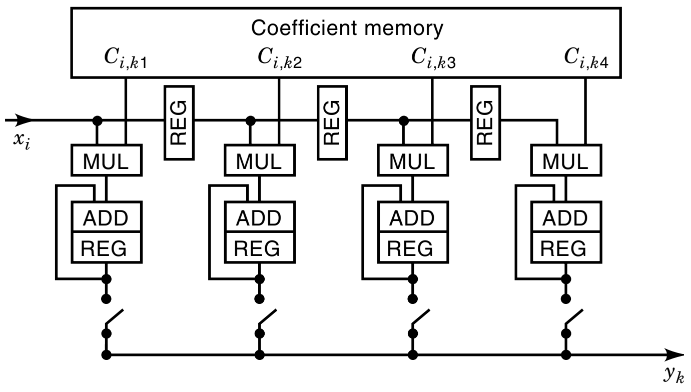

Figure 8. Direct 1D-DCT implementation based on four MAC units. The input sequence x is distributed to all MAC units in a delayed manner, and the MAC results are multiplexed to form the output sequence y. The structure is suitable for matrix-vector multiplication in general.

MAC units is shown in Fig. 8. It allows a trade-off between operation speed and hardware cost by multiple use of MAC units within the computation of one row or column of a block.

Several fast DCT algorithms have been proposed that lead to further significant reductions in operation number by decomposing the transform matrix C into simpler submatrices. The resulting architectures are typically based on efficient implementation of the butterfly structure—three input operands are combined by addition, subtraction, and multiplication to deliver two output operands—as the underlying computation pattern of fast DCT algorithms. While the number of required additions and multiplications can be minimized, the more irregular data flow of fast algorithms complicates the hardware design, and higher accuracy requirements for intermediate results may increase the hardware cost again. Nevertheless, dedicated implementations of fast DCT algorithms typically result in high-performance solutions (15).

An alternative approach for dedicated DCT implementation is based on distributed arithmetic. This technique avoids multiplications completely by replacing them with table lookups of precalculated sums of products stored in a ROM. In each processing cycle, the ROM is addressed by a bit plane of the input vector. The operations to be performed can be derived by substituting x_i in the 1D-DCT definition with its binary B-bit representation as given in Eq. (2).

$$y_k = \sum_{i=0}^{L-1} c_{i,k} x_i, \quad x_i = -x_i^{(B-1)} 2^{B-1} + \sum_{j=0}^{B-2} x_i^{(j)} \cdot 2^j \tag{2}$$

$$\begin{aligned} y_k &= -\sum_{i=0}^{L-1} c_{i,k} x_i^{(B-1)} 2^{B-1} + \sum_{i=0}^{L-1} c_{i,k} \sum_{j=0}^{B-2} x_i^{(j)} 2^j \\ &= -2^{B-1} \phi_{k,B-1} + \sum_{j=0}^{B-2} 2^j \phi_{k,j} \end{aligned}$$

with

$$\phi_{k,j} = \sum_{i=0}^{L-1} c_{i,k} x_i^{(j)} \tag{3}$$

Equation (3) shows the result of the substitution. The sums $\phi_{k,j}$ can be precalculated for each possible bit plane pattern of the input vector x. Thus, only accumulators and shifters are required in addition to the ROM to perform the transformation in a bit-serial manner. Due to the two's complement representation, the addition of the $\phi_{k,j}$ has to be reversed to a subtraction for the most significant bit.

The core operations of the distributed-arithmetic DCT can be implemented in a RAC cell comprising a ROM, an accumulator, and a shifter. A concurrent architecture for an L-point DCT requires L RACs (16). To reduce the size required for the ROM from $L \times 2^L$ down to $L \times 2^{L/2}$, a mixed flow-graph/distributed arithmetic architecture has been reported with a first stage of butterfly computation followed by two independent $L/2$-point distributed arithmetic implementations.

Motion Estimation

Motion estimation is another computation block well suited to dedicated implementation. In most cases, a simple block matching algorithm is employed to estimate the motion between consecutive frames by determining for each $N \times N$-block in the current frame the block in the previous frame that matches its contents most closely. As a result, a motion vector is assigned to each $N \times N$-block of the current frame. The match criterion typically used is the mean absolute difference (MAD) because of its computational simplicity. Thus, a motion vector is determined by the displacement between two blocks that minimizes the MAD for a given search area.

A straightforward, although computationally expensive, approach to find the motion vectors is the exhaustive search. It involves computing the MAD for each pel i,j and each possible position m,n of a candidate block y within a given search window of the range $\pm w$ with respect to a reference block x, as specified by Eq. (4):

$$D_{m,n} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |x_{i,j} - y_{i+m,j+n}| \tag{4}$$

After computing the distortion $D_{m,n}$ for all $(2w + 1)^2$ possible positions of the candidate block, the minimum distortion is determined, and its corresponding displacement is taken as the motion vector. The associativity of the operations involved in computing $D_{m,n}$ allows the block matching algorithm to be described by various different flow graphs. In consequence, several alternative block-matching architectures can be derived by applying a formal mapping process to translate the flow graphs into corresponding architectures (13).

Figure 9 shows a 2D-array architecture as a direct implementation of the exhaustive-search block-matching algorithm. It comprises an $N \times N$-array of absolute-difference processing elements (AD-PEs), each operating on a candidate block/reference block pixel pair. The reference block data are serially shifted into the array within N^2 clock cycles, and each pixel $x_{i,j}$ is stored in one AD-PE. In order to avoid a complete refill of the PE array with new candidate block data, a larger portion of the search area comprising $N(2w + N)$ pixels is stored in a 2D-bank of shift registers within the circuit. Thus, vertical candidate block displacement can simply be achieved by shifting the stored data within the array accordingly, whereas only one new column of $2w + N$ pixels has to be entered into the circuit for horizontal candidate block displacement. The latency for reference block loading can be hidden when double buffering is employed to allow operating on

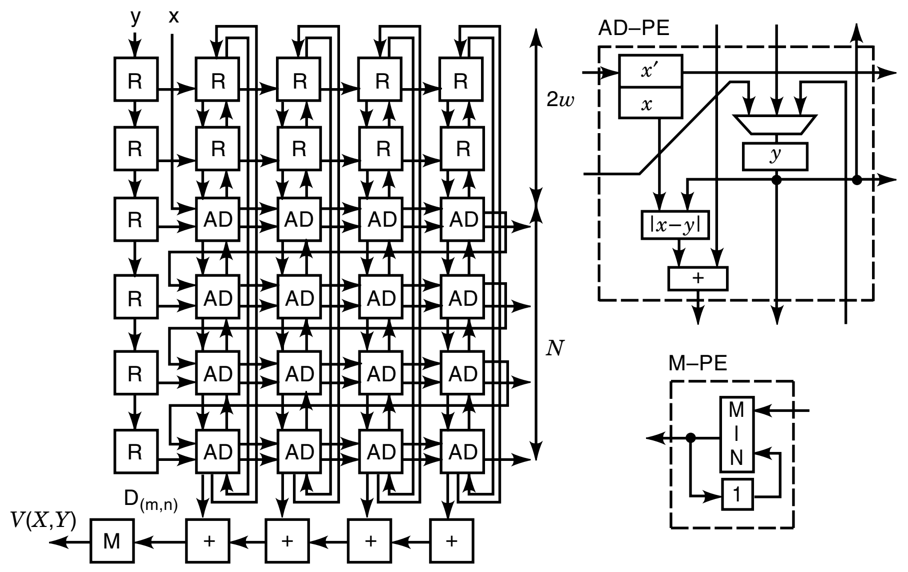

Figure 9. Direct BM implementation. The array consists of R cells (registers) to store the search area, AD cells to calculate and vertically accumulate the absolute differences of pixels, a chain of adders at the bottom for horizontal accumulation, and an M cell to determine the minimum of the consecutively computed values.

the pixels $x_{i,j}$ while the next reference block pixels $x'_{i,j}$ are already shifted into the array.

As shown in Fig. 9, the AD-PE computes the difference and absolute value of two pixels, adds the value to the partial result coming from the upper neighbor element, and transfers this sum to the lower neighbor. Thus, the partial results are computed in columns, and a chain of adders at the bottom of the array performs the horizontal summation to calculate the distortion $D_{m,n}$ for a given displacement m,n. Finally, an M-PE is responsible for finding the minimum among the consecutively computed distortion values.

The extremely high computational requirements of full-search block matching can be reduced substantially by subsampling of the image data or by employing hierarchical search techniques. Subsampling simply decreases the number of absolute differences to be computed per each candidate block. Hierarchical search algorithms typically determine the motion vectors within a number of successive steps involving an incremental refinement of the image data resolution. As a result, the number of candidate blocks is reduced. Various architectures for hierarchical block matching have been proposed (17).

Variable Length Decoding

Variable length decoding is an example for a higher-level algorithm that is difficult to implement with sufficient operation speed on programmable general-purpose processors. As processing requirements for VLD depend mainly on the input data rate, architectures for high bit rate decoding typically have to comprise dedicated VLD modules to ensure sufficient decoding performance.

In VLD, a sequential bitstream has to be parsed, and code words in the form of bit patterns as defined in a codebook thereby have to be identified. In order to exploit statistical properties for transmission rate reduction, the bit patterns are not of constant length: frequently encountered patterns are shorter than less frequent ones. The variable-length bit patterns have to be sliced off the bitstream and translated into the decoded symbols of fixed length. Due to the arbitrary bit pattern length, decoding of a code word can only be started after the previous code word and its associated length have been identified. This kind of feedback loop in the algorithm accounts for the VLD being a strictly sequential process.

In contrast to implementations of DCT and ME, the VLD algorithm cannot be decomposed into a set of elementary operations to be mapped onto an array of simple processing elements. The inherently sequential structure of the VLD algorithm and the dependency of operation flow on the processed data call for different design approaches. A possible implementation of a bit-serial decoding strategy comprises a finite-state machine that enters a new state with each received bit (18). The bit-serial approach however is not feasible for high bit rate decoding due to insufficient operation speed.

In order to provide decoding speed suitable even for high bit rate applications, a look-up table approach may be employed, which takes a complete bit pattern as input and delivers the decoded symbol as output at a constant rate. This approach involves simultaneously inspecting at least as many bits from the bitstream as contained in the longest possible bit pattern in order to guarantee instantaneous decoding. The look-up table may be realized as a ROM with the bit patterns as addresses and the decoded symbols as contents. However, large memory sizes may be required depending on the length of the longest code word. On the other hand, the variable length of the code words leads to sparse utilization of the ROM as almost all code words require multiple entries to ensure unambiguous decoding. For this reason, it is favorable to replace the ROM by a programmable logic array (PLA), which requires only one entry per code word independent from the code word length due to logic minimization (18). If later codebook updates cannot be precluded, the ROM may be replaced by a RAM in order to allow downloading of different code tables. Alternatively, the PLA can be substituted by a content addressable memory (CAM), which also requires only a single entry for each code word while allowing downloading of different code tables.

A dedicated architecture of a PLA-based variable-length decoder is shown in Fig. 10. Besides returning the decoded symbol, the PLA delivers the word length of the decoded bit pattern to a formatter and buffer unit as well as to a barrel

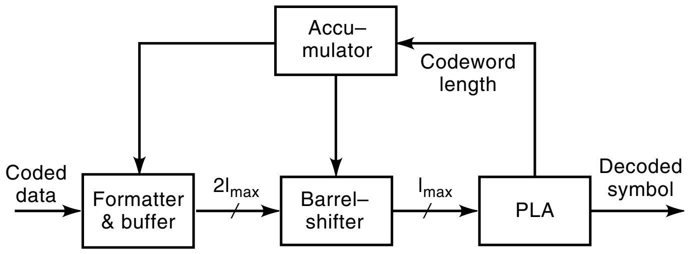

Figure 10. Example of a PLA-based VLD implementation. The PLA serves as a look-up table for decoded symbols and their associated word lengths. The word length is required to align the bitstream correctly to the beginning of the next code word, performed in the barrel shifter. The formatter and buffer unit provides the next piece of the coded bitstream.

shifter. The formatter and buffer unit updates the parsing window of the bitstream, and the barrel shifter aligns the contents of the current bitstream window to the first bit of the next code word before feeding the next bitstream chunk to the PLA as new input. The architecture in Fig. 10 allows a new symbol to be decoded each clock cycle.

Several dedicated solutions for individual tasks have been discussed in this section. They differ widely in terms of performance and hardware cost. With the rapid progress in semiconductor technology, the integration of complete systems on a single chip has become possible. When selecting among different architectural alternatives for individual subtasks, the throughput rates of different modules have to be matched, and constraints concerning overall chip size and power consumption have to be met.

OVERVIEW OF REPORTED VIDEO SIGNAL PROCESSOR DESIGNS

The different design approaches for programmable as well as dedicated architectures presented so far have been incorporated in existing video signal processor designs in various ways. Depending on the targeted application field, a multitude of architectural solutions exists. In this section, some examples of current video signal processors reported in the literature are presented.

Programmable Processors

Numerous programmable video signal processor designs have been reported in the literature. Most of them target the rapidly evolving market of multimedia. Therefore, the design of these processors has focused mainly on video compression, sometimes in combination with processing of other data types such as audio or graphics. In addition to specific video signal processors, several commercially available general-purpose microprocessors have recently undergone instruction-set extensions targeting video and multimedia processing, thus documenting the growing importance of this field. In the following, architectural measures for programmable video signal processors are identified in existing designs.

Texas Instruments MVP. The MVP (19) is an image and video signal processor employing an MIMD controlling concept. It features four parallel digital signal processors (DSP)

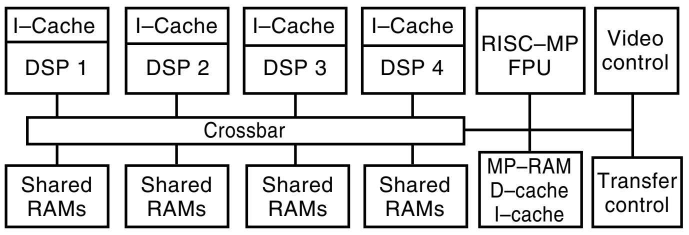

Figure 11. The Texas Instruments MVP video signal processor. The MIMD architecture features four independent DSPs and an additional RISC as master processor. A crossbar switch connects the processors with multiple memory units.

and a RISC master processor on a single chip. As shown in Fig. 11, a crossbar connects the parallel DSPs to four shared memory modules of 32 kbytes in total. In addition, each DSP has access to a private 2-kbyte instruction cache. Separate on-chip memory and cache modules are available for the RISC master processor. The RISC also connects to the crossbar in order to take over global control of the DSPs and to obtain access to the shared memory areas. An on-chip direct memory access (DMA) unit and a video interface complete the design of the MVP.

The parallel DSPs are controlled by 64-bit long instructions words (LIW) allowing multiple operations to be specified in parallel. Moreover, they feature 32-bit split-ALUs that can operate on two 16 bit or four 8 bit data entities in parallel to exploit data parallelism. Hardware multiplier, extensive hardware loop support, and a set of specialized instructions speed up frequent operation sequences in image and video processing. The RISC master processor includes a floating-point unit that is useful in audio and 3D graphics processing.

On one hand, the large variety of architectural measures is able to increase video signal processing performance; on the other hand, the numerous processing and memory units sum up to a considerable silicon area and the complex structure with the large crossbar may constitute a potential limit for the achievable clock rate. Due to the MIMD controlling concept employed in the MVP, software development becomes a demanding task as multiple programs have to be written and proper synchronization has to be ensured.

AxPe640V Video Signal Processor. The AxPe640V (20) as shown in Fig. 12 is an example of a less extensive video processor design based on the coprocessor concept. It couples a RISC control processor with an SIMD-style coprocessor mod-

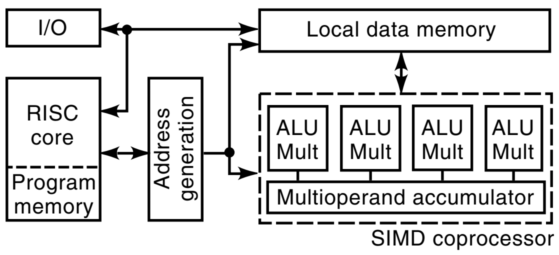

Figure 12. The AxPe640V video signal processor. A general-purpose RISC core is coupled with an SIMD coprocessor adapted to computation-intensive low-level tasks. The RISC functions as a global controller.

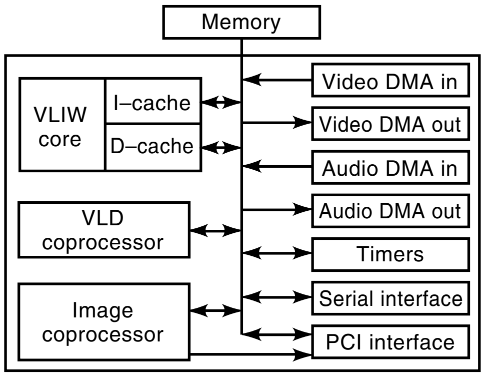

Figure 13. The Philips Trimedia processor. It features an extensive VLIW processor core, coprocessors for VLD and image processing, and various interfaces to facilitate system integration.

ule particularly adapted to low-level video algorithms. The SIMD coprocessor features four 8 bit data paths that can also operate in a 16 bit mode when higher precision is required. Integrated hardware multipliers and a common multioperand accumulator enable efficient execution of frequent video processing operations. An on-chip memory provides the coprocessor with the required operands. The RISC core offers specialized instructions useful for the execution of more irregular tasks, such as VLD or quantization. While both processors are able to execute different tasks independently, the RISC processor also functions as a global controller.

The restriction to two processing modules and the high adaptation of the coprocessor toward computation-intensive low-level video algorithms lead to a moderate design complexity of the AxPe640V and result in high efficiency for the targeted algorithm classes. For higher performance demands, several processors can be combined to form an MIMD multiprocessor.

Philips Trimedia. The Philips Trimedia (21) represents a typical multimedia processor targeting joint processing of video, audio, graphics, and communication tasks. Its central processing unit consists of a VLIW core comprising as many as 27 functional units. Up to five operations to be executed in the functional units can be specified within a single instruction word. In addition to the VLIW core, the Trimedia features a number of coprocessing modules including a VLD coprocessor, an image coprocessor, and various interfaces, as shown in Fig. 13. On-chip caches for data and instructions speed up access to frequently used items. No on-chip memories are included. Additional features of the Trimedia include floating-point support, specialized instructions, for example, for motion estimation, and split-ALU capabilities.

As a VLIW processor, the Trimedia relies on static instruction scheduling at compile time. Therefore, performance and efficiency depend fundamentally on the capabilities of the available compiler. Advanced features such as guarded execution to avoid frequent branching and instruction word compression techniques to relief bandwidth constraints are helpful in increasing both performance and silicon efficiency.

Multimedia Extensions. A new trend in general-purpose processor design is marked by the introduction of multimedia instruction-set extensions in order to speed up the computation-intensive tasks of video processing applications (22,23). They are essentially based upon two principles: the integration of specialized DSP instructions and the incorporation of a split-ALU. While a fast MAC instruction is contained in most extensions, advanced features include saturation capabilities or specific support for motion estimation targeting video compression. The split-ALU concept allows significant enhancement of the efficiency of the typically wide data paths (e.g., 64 bit) in general-purpose processors for video processing.

Even with multimedia extensions, general-purpose processors cannot compete with video signal processors in the domain of pure video signal processing applications, mainly because of their much higher prices. However, for desktop applications where a general-purpose processor is already available, multimedia extensions may be able to enhance the performance of video signal processing without having to incorporate additional devices.

Dedicated Implementations

For video signal processing devices targeting a single specific application, dedicated architectures may provide the best solution in terms of performance, cost, and efficiency. While previous semiconductor technology enabled the monolithic implementation of only subtasks, for example, DCT or ME as demonstrated in the preceding section, the continuing advances in VLSI technology put the implementation of complex applications on a single chip, such as complete video compression schemes, well within reach. Besides multiple dedicated modules for the diverse subtasks of an algorithm, in many cases a programmable control processor can also be integrated on the same chip. Two examples of dedicated implementations of complete applications are reviewed in the following.

Toshiba MPEG-2 Decoder. An example taken from video compression applications is the single-chip MPEG-2 decoder from Toshiba (24), shown in Fig. 14. Besides complete dedicated implementations of the MPEG video and audio decoder modules, it features a transport processor for demultiplexing, a programmable RISC core for control tasks, and exhaustive interface support, which allows for operation of the chip in

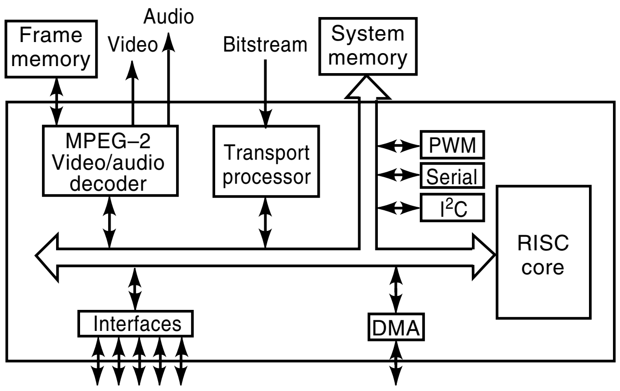

Figure 14. The Toshiba single-chip MPEG-2 decoder. A complete MPEG-2 video decoder, MPEG-2 audio decoder, a transport processor, a programmable RISC core, and various interfaces are integrated on a single device. Only external memory modules have to be added for a complete system solution.

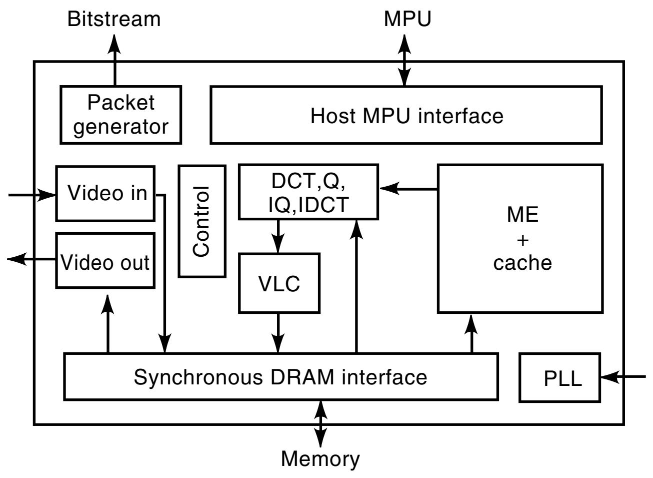

Figure 15. The NEC single-chip MPEG-2 video encoder. The chip contains all computation units necessary for MPEG-2 video encoding. A programmable control processor and memory are required as external modules.

various environments. When adding appropriate memory modules, the chip is a completely self-contained decoder solution targeting high-volume applications such as set-top boxes.

NEC MPEG-2 Encoder. Another example of a dedicated architecture of high integration density is the NEC single-chip MPEG-2 video encoder (25). Due to the motion estimation task incorporated into MPEG video encoding, the hardware requirements for video encoders are significantly higher than for decoders. As Fig. 15 shows, the NEC encoder implements all subtasks required for encoding on a single chip, including a packet generator responsible for packing the encoded bitstream. Besides memory, an additional programmable control processor is required as an external module. The chip integrates 3.1 million transistors on a 155 μm^2 die in a 0.35 μm CMOS technology, operates at a 54 MHz internal clock, and consumes about 1.5 W at 3.3 V.

FUTURE TRENDS IN VIDEO SIGNAL PROCESSOR DESIGN

The future trends in video signal processors are driven by the continuing development of new, sophisticated applications, on one hand, and by the rapid progress in VLSI technology, on the other. Both programmable and dedicated approaches will continue to play an important role in their respective application domains. However, as applications of higher complexity and less computational predictability evolve, programmable video signal processing architectures are likely to become increasingly preferred to dedicated solutions.

In the future, video signal processing applications can be expected to involve growing algorithmical complexity, for example, to improve real-life impressions generated from synthetic data. Higher algorithmical complexity translates into a higher number of operations per data sample, substantiating the need for higher performance levels of video signal processing architectures. On the other hand, sophisticated applications will contain considerably larger portions of high-level algorithms in order to realize intelligent functions. As a result, an increasingly irregular and data-dependent operation flow has to be expected, which complicates architecture design as automated derivation of processing modules for these algorithms is not easily possible. Additionally, the fine-grain parallelization potential as found in low-level algorithms is likely to decrease, and performance enhancements will have to be realized increasingly by exploitation of large-grain task-level parallelism, for example, through concurrent execution of independent scalar tasks.

Another trend in future applications will be the merging of video signal processing with processing of other data types, as encountered, for example, in multimedia environments. This development requires extending the design focus in order to enable joint processing of various data types within the same architecture. In contrast to video processing, audio processing, for example, involves operating on data samples with higher precision/word-length requirements, restricting the utility of split-ALU or similar concepts. Instead, integration of other function units such as floating-point modules may become necessary, which typically may not be found in pure video signal processors. In effect, the diversification of applications desired to be executed on a single device limits the potential for strong adaptation to special algorithm characteristics and accounts for a decrease in silicon efficiency.

While the opportunities to exploit fine-grain parallelization and adaptation principles to enhance performance as well as efficiency of video signal processors are likely to diminish, the implementation of sophisticated future applications will be enabled by the rapid progress in VLSI technology, which allows integration of an ever-increasing number of transistor functions on a single chip operating at increasingly faster clock speed. While current processor designs comprise up to around ten million transistors, monolithic integration of more than a hundred million transistors has already been announced for the near future (26). These new levels of integration density offer exciting opportunities in video signal processor design, but also require new architectural approaches to employ the enormous amount of transistors efficiently.

One way to utilize the additional hardware resources in future video signal processors is to extend the integration of specialized modules and system interfaces on-chip, leading to the design of complete systems on a chip. While the lower design complexity associated with the strictly modular structure is thus retained, the system cost can be reduced significantly by replacing multiple chips with a single device. However, the incorporation of more specialized modules decreases the range of applications, for which the video signal processors constitute an efficient solution, and the possibly resulting lower production volumes of such architectures may in turn absorb the cost advantage of higher integration density.

Alternatively, a more generic design with extended capabilities may offer greater advantages for future video signal processors. Considering the emerging characteristics of new applications, a parallel architecture with multithreading support may be a sensible approach. Novel controlling schemes for efficient execution of highly data-dependent operation flow have to be explored. Additional available silicon area may be spent for the integration of the larger on-chip memories necessary to alleviate the growing gap between processor clock speed and external memory access times. Hence, such an architecture may offer sufficient processing performance as well as flexibility for a variety of evolving applications in the area of video signal processing.

Regardless of the actual architecture of future video signal processors, effective support for software development will be of vital importance for commercial success. With the expected growing complexity of emerging video signal processing applications, realizing an implementation within competitive time frames depends fundamentally on the availability of an efficient, user-friendly programming interface. Time-consuming, error-prone low-level assembler programming will become less and less feasible for the implementation of huge software projects. High-level language support will be required for future video signal processors that provides the programmer access to all relevant machine features, helps to identify and exploit parallelization opportunities, and offers extensive support for multithreading.

BIBLIOGRAPHY

1. Video Codec for Audiovisual Services at p×64 kbit/s, *ITU-T Recommendation H.261,* 1993.
2. Video Coding for Low Bitrate Communication, *ITU-T Draft Recommendation H.263,* 1995.
3. Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s (MPEG-1), *ISO/IEC 11172,* 1993.
4. Generic coding of moving pictures and associated audio (MPEG-2), *ISO/IEC 13818,* 1993.
5. S. Y. Kung, *VLSI Array Processors,* Englewood Cliffs, NJ: Prentice-Hall, 1988.
6. M. J. Flynn, *Computer Architecture: Pipelined and Parallel Processor Design,* Boston: Jones and Bartlett, 1995.
7. K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability,* New York: McGraw-Hill, 1993.
8. R. B. Lee, Subword Parallelism with MAX-2, *IEEE Micro,* **16** (4): 51–59, 1996.
9. S. Dutta et al., Design issues for very-long-instruction-word VLSI video signal processors, *VLSI Signal Processing IX,* IEEE, 95–104, 1996.
10. D. Zucker, M. Flynn, and R. Lee, Improving performance for software MPEG players, *Proc. Compcon,* IEEE CS Press, 327–332, 1996.
11. J. Bush and Y. Kim, Developing software for C80-based machine vision systems, *EE Times,* July 1996.
12. M. Schwiegershausen, H. Kropp, and P. Pirsch, A system level HW/SW-partitioning and optimization tool, *Proc. Eur. Design Autom. Conf. (EDAC),* 120–125, 1996.
13. P. Pirsch, N. Demassieux, and W. Gehrke, VLSI Architectures for Video Compression—A Survey, *Proc. IEEE,* **83**: 220–246, 1995.
14. N. Demassieux and F. Jutland, Orthogonal transforms, in P. Pirsch, ed., *VLSI Implementations for Image Communications,* Amsterdam: Elsevier, pp. 217–250, 1993.
15. A. Artieri et al., A VLSI one chip for real-time two-dimensional discrete cosine transform, *Proc. IEEE Int. Symp. Circuits Syst.,* 1988.
16. M. T. Sun, L. Wu, and M. L. Liou, A concurrent architecture for VLSI implementation of discrete cosine transform, *IEEE Trans. Circuits Syst.,* **34**: 992–994, 1987.
17. L. De Vos, VLSI-architectures for the hierarchical block matching algorithm for HDTV applications, *Proc. SPIE Visual Commun. and Image Process. '90,* **1360**: 398–409, 1990.
18. M.-T. Sun, Design of high-throughput entropy codec. In P. Pirsch, ed., *VLSI Implementations for Image Communications,* Amsterdam: Elsevier, pp. 345–364, 1993.
19. K. Guttag, The multiprocessor video processor MVP, *Proc. IEEE Hot Chips V,* 1993.
20. K. Gaedke, H. Jeschke, and P. Pirsch, A VLSI-based MIMD architecture of a multiprocessor system for real-time video processing applications, *Jour. VLSI Signal Processing,* **5** (2/3): 159–169, 1993.
21. S. Rathnam and G. Slavenburg, An architectural overview of the programmable multimedia processor, TM-1, *Proc. Compcon,* IEEE CS Press, 319–326, 1996.
22. M. Tremblay et al., VIS speeds new media processing, *IEEE Micro,* **16** (4): 10–20, 1996.
23. A. Peleg and U. Weiser, MMX technology extensions to the Intel architecture, *IEEE Micro,* **16** (4): 42–50, 1996.
24. J. Turley, Toshiba, TI Roll Out Set-Top Box Chips, *Microprocessor Report,* **10** (7): 16, 1996.
25. Y. Ooi et al., An MPEG-2 encoder architecture based on a single-chip dedicated LSI with a control MPU, *Proc. Int. Conf. Acoust. Speech Signal Process.,* 599–602, 1997.
26. Anonymous, New TI technology doubles transistor density, *TI Integration,* **13** (5): 1996.

PETER PIRSCH
Universität Hannover
HANS-JOACHIM STOLBERG
Universität Hannover

VIDEO SIGNAL PROCESSING. See DATA COMPRESSION FOR NETWORKING.