DSPs for image and video processing

Klaus Illgner*

Texas Instruments, Inc., P.O. Box 660 199, MS 8632, Dallas, TX 75266, USA

Received 1 August 1999; received in revised form 12 December 1999

Dedicated to Prof. H.-D. Lüke on the occasion of his 65th birthday

Abstract

As communications get digital there is an increased use of image and video processing in embedded multimedia applications. Digital signal processors (DSPs) provide excellent computing platforms for these applications not only due to their superior signal processing performance compared to other processor architectures, but also because of the high levels of integration and very low-power consumption. This paper presents an overview of DSP architectures and their advantages for embedded applications. The specific signal processing demands of image and video processing algorithms in these applications and their mapping to DSPs are described. Recent results of successful implementation of two major embedded image and video applications – digital still cameras and video phones – on TI’s TMS320C54x DSP series conclude the paper. © 2000 Elsevier Science B.V. All rights reserved.

Zusammenfassung


Résumé

Les communications devenant numériques l’utilisation du traitement des images et des vidéos s’accroît dans les applications multimédia. Les processeurs de signaux numériques (DSP) constituent d’excellentes plate-formes de calcul pour ces applications non seulement du fait de leurs performances de traitement des signaux supérieures à celles obtenues avec d’autres architectures de calcul, mais aussi du fait de leurs hauts niveaux d’intégration et de leur consommation très faible. Cet article passe en revue les architectures de DSP et leurs avantages pour des applications immergées. Les demandes spécifiques en termes de traitement des signaux des algorithmes de traitement d’images et de vidéos dans ces applications et leur portage sur DSP sont décrits. Des résultats récents d’implantation réussie de deux applications

* Fax: +1-972-761-6967.
E-mail address: illgner@ti.com (K. Illgner).

0165-1684/00/$-see front matter © 2000 Elsevier Science B.V. All rights reserved.
PII: S 0 1 6 5 - 1 6 8 4 ( 0 0 ) 0 0 1 2 0 - 1
1. Introduction

Generating and displaying images in the past was restricted to analog processing only, requiring several, partially costly processing steps, e.g. chemical films or analog signal processing in TV broadcasting. Digital imaging, on the other hand, could be realized only in areas where size and power consumption and/or cost played merely a secondary role, e.g. medical imaging [1] or space exploration. Nowadays, increasing computational power of processors combined with very low prices made imaging applications feasible on workstations and desktop computers. As a result digital imaging alters existing applications and generates new applications. Increasing integration density combined with reduced power consumption allows to implement imaging applications even in small portable devices. Digital cameras which not only compete with film cameras, but also add new functionalities, are one example. Another one is mobile image and video communications, an application not possible a couple of years ago.

This paper analyses first basic processor architectures enabling imaging in portable and embedded devices, respectively. Realizing such, mainly consumer market, applications is not only constraint by the integration density of computational power, but cost and power consumption are equally important. Therefore, catalogue DSPs and DSP-based systems sustain and even gain market shares against specialised processor concepts involving a general purpose processor core (GPP) with accelerator units. Reduced time-to-market combined with the ease to add features favours programmable solutions over dedicated chip designs (ASICs). This paper aims in giving an insight into how DSPs can be used for certain imaging applications and video processing.

After discussing todays platform concepts and why DSPs are especially well suited, the fundamental operations of imaging applications are analysed. Section 4 discusses the feasibility and implementation issues of image processing algorithms on DSPs. Finally, two examples of implementing imaging systems on DSPs are introduced, a digital still camera and a video communications codec.

2. An introduction to DSPs

There exist 2 basic processor architecture concepts: the GPP and the DSP. At a first glance it might be surprising that DSPs still exist besides the competing and continuously higher performing GPPs. Pentium-based systems, as an example for a today’s GPP, are capable to run complex signal processing tasks like audio and video decoding in real time. However, this is achieved by adding DSP concepts (e.g. MMX [2]) and increasing the clock speeds [4,5,24]. Modern high-powered DSPs like the TMS320C6x series are able to even encode video in real time1, keeping DSPs for signal processing dominated applications ahead of GPP cores [27].

The resources of core processors may not be sufficient for dedicated applications like for instance high-resolution video processing (MPEG-2 encoding and decoding for DVB).2 As an alternative to pushing regular cores in terms of complexity and clock frequency, dedicated ‘processor’ architectures have been developed. An overview of design concepts can be found in [17]. Instead of using multiprocessor concepts [9], new concepts combine DSP and RISC concepts into a single architecture dedicated for multimedia [13,15,19,21]. Another approach integrates a standard core processor

---

1 Video encoding requires more computational resources than decoding, mainly for motion estimation.
2 Digital video broadcasting.
along with coprocessors or task specific accelerator units. An example is the TriMedia with a GPP VLIW\(^3\) core including DSP-like units and coprocessor units for filtering and scaling, colour space conversion, and Huffman decoding [20].

Before selecting a processor architecture the requirements of portable devices need to be analysed from a system perspective. Here, power consumption is a major issue. Power dissipation is primarily related to the number of recharging cycles in a time period. This translates into the aim of executing an operation in a minimal number of machine cycles involving a minimal number of gates. To penetrate into high-volume applications, e.g. portable consumer appliances, the chip must be cheap. Therefore, the design aims on minimising the chip size. For both reasons, the design must implement only the essential logic. However, the aforementioned constraints are not supposed to sacrifice computational power. Comparing GPP and DSP cores, the advantage of DSPs lies in the high MIPS/mW and MIPS/cost ratios, making them the first choice for signal processing dominated applications.

2.1. Overview of DSP architectures

As the name indicates DSPs are specifically designed for digital signal processing. One fundamental calculation in digital signal processing algorithms is filtering\(^4,5\)

\[
g[n] = \sum_{k=0}^{N-1} a[k]s[f(n,k)], \quad n, k \in \mathbb{Z}. \tag{1}
\]

This structure maps into a multiplication and an accumulation of the result to the previous result. Therefore, DSPs almost always implement a so-called MAC-instruction (Multiply and ACumulate) (Fig. 1).

To filter an image this sum is calculated very often (1024 \times 1024 pixels filtered with 3 \times 3 kernel \(\Rightarrow 9.4M\) MAC operations). For minimising execution time of the MAC-instruction parallel memory accesses are required. For that reason and despite the higher complexity, DSPs utilise the Harvard-architecture which provides multiple address and data buses rather than the von Neumann architecture, found in traditional GPPs (Fig. 2). Because the CPU speed of modern GPPs like the Pentium has increased much more than memory speed, GPPs utilise nowadays the concept of the Harvard-architecture as well [24].

To accelerate memory accesses cache memory is common in GPPs, which has been introduced to DSPs as well. Instructions and/or data items are stored locally in fast memory rather than in slower external memory. The placement of program and data segments in the cache is handled by a hardware unit. In case of cache misses, e.g. as a result of data-dependent branches, the execution time becomes context dependent and unpredictable. As DSPs are used very often in hard real-time applications like cellular phones or modems, execution time predictability is a must. Also, to optimise performance DSP programmers must have control over the cache utilisation. An alternative to cache may be dual-ported on-chip RAM (DARAM), provided by some DSPs.

---

\(^3\) Very long instruction word.
\(^4\) \(f()\) is a mapping function, because Eq. (1) applies to FIR/IIR filtering, correlation calculations, and transformations as well.
\(^5\) The notation distinguishes between discrete signals \(s[\cdot]\) and vectors \(s = (s_1, \ldots, s_M)\).
Memory bandwidth is only one aspect. DSP algorithms frequently loop a small block of instructions very often. In the FIR structure above the loop of \( N \) MAC-instructions calculating one output sample gets executed for each pixel (filtering 1024 \times 1024 \) pixels \( \Rightarrow 1 \text{M} \) loops). Therefore, DSP designs add zero-overhead looping and very low overhead branch instructions. Branch prediction utilised in GPPs reduces branch overhead only in average, while the prediction logic adds complexity.

The MAC-instruction can not really be executed in one cycle, because the adder has to wait for the multiplier result. For optimised utilisation of the hardware resources pipelines are utilised (Fig. 3). Although there exist very different pipeline designs, the basic functionality is to break up multi-cycle operations into several phases. Each phase is associated with dedicated hardware logic. With every machine cycle an instruction is initiated but multiple instructions may be in some phase of execution keeping all hardware units busy. By utilising this ‘instruction-level’ parallelism in the implementation of algorithms the performance improves significantly. The drawback is that programming gets more complex as the result becomes available delayed (potential pipeline conflict). Pipelines increase the penalty of branches as the pipeline must be flushed. Additional characteristics of DSPs are a broad variety of addressing modes: absolute, relative to a register, pre/post-increment and -decrement, increment using a register content, or indexed addressing. For filtering circular addressing in particular is powerful to realize ring buffers.

Adding a separate address generation unit allows the address modifications in parallel to the ALU (\( \rightarrow \) pipelines).

Enabling single-cycle execution for specialised instructions by providing hardware support results in irregular instruction sets and register files. Specialised instructions reduce code size, however, this comes at the cost of flexibility [22]. Also, binding registers, especially of small register files, to certain functions reduces flexibility. But, concentrating the architecture design on signal processing tasks allows to design fast low-power cores on a small chip area. Therefore, DSPs are very well suited for embedded and portable/mobile systems.

2.2. The TMS320C54x architecture

An example DSP TI’s TMS320C54x series is shortly introduced, because the system described in Section 5 is based on this DSP. The processor is not only a DSP but with respect to communication applications an ASIP.\(^6\) For instance, the butterfly function for Viterbi decoding is supported by instructions, such that it can be executed in a minimal number of machine cycles. The architecture schematic in Fig. 4 shows the Harvard architecture featuring 2 data memories [26]. There are 2 40 bit accumulators and a \( 17 \times 17 \) bit multiplier. The 7 stage pipeline provides an execution of most instructions in one cycle. The processor has a heterogeneous register set, some of the registers can be used for advanced addressing, e.g. indexed addressing, or post-modification. Besides instructions for Viterbi decoding the DSP comes with specialised

\(^6\)Application specific instruction-set processors: variant of a DSP specifically designed for a certain application by cutting the overhead of standard DSPs in terms of chip size, and adding instructions minimising the cycle count for core loops.
instructions, e.g. for symmetrical 1D FIR filtering, and least-squares fitting.

One of the lowest power cores is TI’s C5402, providing 100 MIPS@100 MHz while consuming just 58 mW. The performance in MIPS/mW is ahead of what can be reached with GPPs. Although this DSP was designed specifically for communications applications this DSP can efficiently run other applications as well. Most surprisingly even image processing applications can be efficiently implemented as will be shown in Section 5.

3. Digital imaging systems

There exists a broad spectrum of digital imaging applications, e.g. as an important component of WEB pages, digital video and video communications, surveillance systems, advertisement, movies, digital still cameras (DSC), medical imaging, and even toys. Except for artificially generated images, images captured by digital imaging sensors must be processed before they are available for further compression, manipulation, transmission, and display. This processing comprises all important image processing functions, and therefore is a suitable reference of how to map image processing task on DSPs. Although the paper focuses on still imaging (in a very generic sense) and image sequence processing (video), the results are applicable to a much broader range of applications.

A typical image processing chain for such systems consists of two major units, the processing after capturing the image and the processing unit for displaying (Fig. 5). The type and order of image processing steps within the processing and display unit may differ between applications, the basic flow is similar. In case of video communications the compression unit includes also frame predictive coding.

3.1. Image acquisition and processing

As imaging sensors mostly CCDs are used, but CMOS sensors are currently getting an attractive alternative. While CCD sensors provide excellent image quality at the expense of a costly manufacturing process, CMOS sensors can be integrated along with the processing circuitry [6].

In the first processing steps distortions incurred by the sensor and the analog frontend need to be reduced. Exposure adjustments taking the responsivity of the sensor into account are essential to obtain high-quality images. In case of colour sensors the colour components need to be adjusted with respect to each other, which is loosely termed...
‘white balancing’. The processing consists basically of point operations \( F(p): s[i,j] \mapsto g[i,j], \forall (i,j) \in A \). 
\( A \) denotes the image grid and \( p \) the parameter vector. Sensor inhomogeneities may call for spatially dependent parameters or even non-linear approaches.

Several noise sources distort the CCD output signal as well. Fixed-pattern noise reduction is mostly handled already in the sensor module. Uncorrelated white noise can be reduced by standard filtering approaches. Poisson-noise, however, requires conditional filtering, where the filter kernel depends on the pixel intensity at each image location.

To obtain colour images from a single imager, a colour filter is placed in front of the sensor such that each pixel senses the image spectrum only in a limited spectral range. For DSC applications typical are the three colour primaries red, green, and blue. There exists a broad variety of colour filtered array (CFA) patterns, where the Bayer pattern (Fig. 6) is the most popular one. In order to obtain a full resolution colour image the colour bands are interpolated to full CCD size. This processing step is also known as CFA interpolation. Although there exist quite a few different approaches, this step is again basically a filter operation. Depending on the size of the sensing element (pixel size) and the modulation transfer function (MTF) of the optical path the spatial sampling by the sensor may introduce aliasing. Therefore, the interpolation filter may include image restoration approaches.

The following step handles illumination correction, tone correction, and colour space conversion. In most situations the images are taken under different illuminations than they are displayed. To allow for colour correct reproduction this step transforms the input signal into a reference colour space under a reference illumination. In most cases a linear multiplication of each colour pixel vector \( s = (s_R, s_G, s_B) \) with a (pre-calculated) \( 3 \times 3 \) matrix \( A \) is sufficient \( g = A \cdot s^T \).

Finally, after correcting system related shortcomings the images may undergo enhancement operations such as contrast enhancement to improve the image dynamic, edge enhancement, or false colour suppression. The type of operation required may call for other approaches besides point operations and linear filtering.

Generally, for most of the image processing steps more sophisticated algorithms may be necessary, which often involve data-dependent selection of parameters, non-linear mappings (non-linear point operations or non-linear filtering), and even divisions. But because not all architectures efficiently support them, these types of operation require a careful analysis before considering implementation. Algorithms should be regular as branching is expensive. Unless the saving in calculations is higher than executing a branch, data-dependent processing should be avoided. In some situations algorithms may be regularised by always executing a calculation, but using neutral parameters (1 for multiplication, 0 for adders) whenever a condition should be ignored. Also divisions are not available as single cycle instructions on most processors. Non-linear processing can be handled efficiently by mapping it to table look-ups. The potential bottleneck, however, is the amount of memory available.

### 3.2. Image compression and displaying

Because compression of images is an important application in many systems, JPEG compression is addressed. As a transform-based coder followed by
entropy coding this standard can serve as a suitable reference for analysing the processing steps associated with image compression.

The basic encoding flow comprises a DCT-transformation stage, quantisation, zig-zag scanning to map the 2D image structure onto a 1D-vector and variable length coding (entropy coding). The DCT calculation has a structure similar to Eq. (1). This holds in general for linear transformations as they can be written as multiplication of a matrix with a data vector. The 2D transforms of an 8 × 8 image block \( S \) reads as \( S_{TR} = A_{TR} \cdot S \cdot A_{TR}^T \), with \( A \) the 8 × 8 coefficient matrix. Quantisation is a point operation, eventually involving a multiplier, and zig-zag scanning can elegantly be implemented with indexed addressing. The operations involved in entropy coding are counting (run-length coding) and bit manipulations forming a continuous bitstream. For Huffman coding the mapping of the symbol to be encoded to the codeword requires indexing the codeword table. However, most VLC encoding implementations contain data-dependent conditions and exceptions, which may call for more complex bit-manipulations.

Variable length decoding is a challenging task, since the a priori unknown and changing length of each codeword requires to analyse every bit. This characteristic prevents also parallelisation. An alternative to bitstream parsing, which leads to extensive bit manipulations, involves again table look-ups exploiting the prefix condition (no codeword emulates in the first bits shorter codewords). The remaining parts at the decoding side are not that different from encoding. The inverse zig-zag scan is again efficiently handled by indexed addressing. Also calculating the IDCT follows the identical approach as the DCT calculation.

A step frequently underestimated is to adapt the image to the display or printer characteristics. Otherwise the printed image or the image on the screen may look unexpected. Mapping an image to a printer involves complex operations as the printer dyes, the gamut, and device non-linearities must be considered. Basically, a colour triplet is mapped onto a \( N_c \)-dimensional colour vector. Since the mapping is non-linear matrix multiplications are only partially useful. Typical solutions condition matrixes on local statistics or use polynomial regression. Intelligent printers entering the market perform the conversions themselves, if the input image is characterised with respect to illuminant, colour space, and colour primaries (ICC profile [12]).

For displaying on TV or computer monitors, the colour characteristics are more homogeneous and have been standardised (Rec.709, sRGB). The correction reduces to a colour space conversion into the sRGB or YUV colour space [18], again a matrix–vector multiplication. Resizing the decoded image to fit to the screen format is essentially a filter routine. The step termed gamma correction, which is a point operation, pre-distorts the image to compensate for non-linear intensity characteristic of the monitor – and the human observer. This step is included in some colour space definitions (sRGB), otherwise it must be performed before display.

3.3. Video processing

The market sees nowadays one major standard for video communications — MPEG-4. The ITU standard H.263 for video conferencing only is less complex but restricted to this application, while MPEG-4 offers additionally a toolbox for a broad range of applications [23].

The first processing steps in video and still imaging applications are in principle identical. The raw CCD data has to be processed first before each frame can be compressed. The processing, however, can be compromised, as video sequences will mostly be viewed on small screens, single frames are not printed out, and the connection bandwidth of handheld devices for instantaneous transmission is very limited. As most processing steps are required, computational requirements can be lowered by using less complex functions, e.g. use shorter filters for interpolation and antialiasing filtering.

For video applications specific CCDs exist, e.g. [25]. The main difference is that they use YMC colour primaries and provide an interlaced read-out. This reduces the computational burden while with some signal processing allows to maintain a good quality [10]. High-resolution CCDs (> VGA resolution) provide a high frame rate readout mode at a reduced vertical resolution.
There are quite a few other techniques, but block-based techniques are by far the most common and perform reasonably well.

The well-known structure of today’s video codecs (Fig. 7) allows to compress each frame in a frame-predictive way. Compared to still image compression using JPEG the main additional task is (block-based) motion estimation\(^7\) and compensation, as DCT and entropy encoding (VLC) are already part of JPEG. The computational burden for motion estimation comes from minimising an error criterion for each block. The vector \( \mathbf{v}_{\text{opt}} \) minimising the MSE (or SAD) for a set of test positions \( \mathbf{v} \)

\[
\mathbf{v}_{\text{opt}} = \arg\min_{\mathbf{v}} \sum_{x} || g_{\text{act}}(x - \mathbf{v}) - g_{\text{ref}}(x) ||^2
\]

is transmitted to the decoder. Although the operations involved are simple, constraints are coming from the number of operations required to calculate one vector, and the fact that data from the last decoded frame must be stored. Fast algorithms improve throughput by either cutting down the number of test positions, reducing the number of pixels employed for calculating the criterion, or precalculating partial sums. The approaches defining the test positions may encode them as a fixed scan (e.g. full search), or condition them on the error criterion (e.g. 3-step).

Once the displacement is known motion compensation reduces to an addressing task. The feedback loop adds more computational load as for encoding, besides motion estimation, the dequantisation and the IDCT must be calculated as well. The decoder requires no new operations as the decoder is part of the encoder. The bitstream parsing and entropy decoding is with respect to the type of operations involved similar to JPEG decoding.

4. Image processing on DSPs

For an implementation and feasibility analysis of image processing on DSPs it is not sufficient to concentrate on the DSP alone. Rather, it is necessary to consider the speed at which the DSP can access memory and time constraints imposed by the applications.

4.1. DSP system designs

In still imaging the broadly available sensor resolution has reached 2 million pixels. The typical dynamic range of CCD modules is 10-12 bit. Hence, the amount of RAM necessary to store a single raw CCD image (\( \approx 5 \text{MByte} \)) exceeds already the address range of some processors. Moreover, DSPs provide typically only a limited amount of on-chip memory, which can be accessed at processor speed. Additional external RAM cannot be accessed in most cases at full speed and not in parallel with multiple accesses to internal memory.

Because CCDs typically clock out data in a line-sequential fashion, a solution avoiding extensive intermediate buffering is to stream the data as it is generated through several processing steps. In the example in Fig. 8a, in the first step each pixel is filtered horizontally. With each pixel clock tick the FIR unit must accept the next pixel and pass the result to the next unit. Hence, streaming is like a macro-level processor pipeline. This approach is frequently found in hardwired solutions (ASICs), e.g. for video processing. Streaming data requires the unit to \textit{complete} processing at the pixel clock speed, implicitly realizing real-time processing. Assuming a modest pixel clock of 12 MHz and 50 cycles to process the pixel, the processing unit must provide 600 MIPS. A drawback of this concept is that this requirement of DSP resources cannot be
relaxed in non-real-time applications, such as still imaging. Also, depending on the type of operation buffering cannot completely be avoided, e.g. in case of vertical filtering. In this case the vertical dimension of the filter kernel specifies the number of lines which must be stored. In case of 2M sensors this amounts already to > 10 kB of RAM (DSPs have usually around 32–64 kB of on-chip RAM), leaving little headroom for data buffers for the remaining processing steps.

A more flexible solution is to build a DMA unit into the system. The raw CCD data is streamed into a large external memory, and the DMA handles memory transfers from slow external memory to internal DSP memory. Because the DSP has no direct access to the external memory, this approach has an impact on how to organise the program flow. First, the algorithms must be organised to work localised on small blocks of data. Second, after issuing a data transfer request to the DMA the data transfer itself does not allocate DSP time. Therefore, data transfers and processing can run concurrently. Because in most system designs the processing time exceeds the data transfer time, this approach allows to slow down memory accesses saving cost and power. Another advantage is that this concept permits to scale the DSP’s computational resources. For real-time applications like the one previously mentioned a 600 MIPS processor must be used, whereas in a still imaging application a 100 MIPS DSP may be sufficient.

4.2. Implementation considerations for image processing operations

As a consequence of the limited memory available, image processing tasks must be designed for operating on small data units. Even more important, in order to limit expensive data IO to external memory, a data unit should be fetched only once at the beginning of the processing and written back after completion of the all processing steps. This requires to analyse the processing for data dependencies. In general, the implementation is a trade-off between speed and memory size, such that an efficient memory utilisation is key for fast implementations [8].

Basic linear point operations pose in general no problem on the design, since they are directly supported by instructions (MPY, ADD). Non-linear operations map more efficiently on look-up tables rather than on a set of conditional branches. On a TMS320C54x a table look-up takes about 6 cycles while a single conditional branch alone takes at least 3 cycles. However, table sizes can quickly amount to a significant memory allocation (12 bit data 4096 table entries). Also, using data values as address modifiers may cause pipeline latencies. Spatially dependent parameters add overhead for determining the parameter. Therefore, smoothing the algorithm by using the same parameters for several subsequent data items improves performance.

The sliding window of filtering requires to consider an overlap between neighboured processing units, e.g. for applying a 3 × 3 2D filter to a 16 × 16 block a block of 18 × 18 pixels must be accessed. The straightforward way to address this problem is to load the data of the overlapping area as each processing unit is loaded. For long filters this results in an excessive overhead (2.6 × read overhead for filtering a 16 × 16 block with an 11 × 11 kernel). An alternative is to rearrange data once it has been loaded and reload only new segments. Applying spatially dependent filter kernels on a pixel by pixel basis may add only little overhead, if only the
pointer addressing the filter coefficients needs to be set. Non-linear filtering can get very costly, e.g. the sorting involved in median-filtering requires several branches.

Another aspect is that most DSPs do not implement special instructions to handle 2D array processing. Although looping is efficiently handled in the DSP, multiple nested loops still add up to a considerable cycle count overhead. The reason is the DSP, multiple nested loops still add up to a considerable cycle count overhead. The reason is

\[
g(x', y') = \sum_{j=0}^{N} \sum_{i=0}^{M} c[i,j] s[x + i, y + j].
\]

For each output sample \( M \) pixels of size \( N \) must be fetched where the address offset between vertically neighboured pixels can be as much as one image line. After fetching \( M \) pixels of one line the data pointer must be advanced by a different amount. The overhead lies in lines 3,7,9, but also in resetting all inner loop counters in lines 2,4,5:

1. for \((y = 0; y < \text{DY}-N; y + + ))\{
2. for \((x = 0; x < \text{DX}-M; x + + ))\{
3. data_p = image_p + ;
4. for \((j = 0; j < N; j + + ))\{
5. for \((i = 0; i < M; i + + ))\{
6. tmp + = *filter_p + + *data_p + + ;//1 MAC
7. data_p + = line_offset-M;
8. *(output_p + + ) = tmp;
9. image_p + = M;
10. \}
11. \}
12. \}
13. \}
14. \}
15. *(output_p + + ) = *idx_p + + ;
16. *(output_p + + ) = *idx_p + + ;
17. image_p + = M;

An approach to reduce the overhead associated with the overlap for localised filtering interprets a 2D filter as a filterbank of \( N \) 1D filters of length \( M \). Each line is filtered by the \( N \) filters and the results are added to the results of filtering previous lines. Although this scheme requires \( 2N - 1 \) intermediate line buffers, which results in the same memory allocation as the vertical overlap area, these overlap lines need not to be processed again. Furthermore, filtering each line can be parallelised either in hardware or if the DSP provides parallelised units [14].

CFA interpolation should not be implemented as standard 2D filtering on deinterleaved data to avoid running a routine to deinterleave the CFA data into 3 separate colour bands (Fig. 6b). Instead, the DSP can access the pixels belonging to one colour band by taking advantage of the addressing capabilities. In the following code example one filter kernel is used for all three colour planes. Indexed addressing in combination with a circular buffer of 4 elements \( t[\ ] \) and forward/backward incrementing (icr) is used to assign the accumulated filter result to the correct colour plane (lines 14,15,16), dependent on the current position in the CFA pattern (phase()).

1. for \((y = 0; y < \text{DY}-N; y + + ))\{
2. for \((x = 0; x < \text{DX}-M; x + + ))\{
3. idx_p = &t[phase(x,y)]; register icr = tw_o compl(icr);
4. data_p = image_p + + ;
5. for \((j = 0; j < N; j + + + ))\{
6. for \((i = 0; i < M; i + + ))\{
7. \[0\] + = *filter_p + + *data_p + + ;
8. \[1\] + = *filter_p + + *data_p + + ;
9. data_p + = line_offset-M;
10. for \((i = 0; i < M; i + + ))\{
11. \[2\] + = *filter_p + + *data_p + + ;
12. \[3\] + = *filter_p + + *data_p + + ;
13. data_p + = line_offset-M;
14. *(output_p + + ) = *idx_p + + ;
15. *(output_p + + ) = *idx_p + + ;
16. image_p + = M;

This not only saves memory but also computational load, as multiplications with data samples
known to be zero are avoided. Because this operation has a major impact on the overall image quality, many different approaches have been developed. However, only regular filtering approaches map really well on DSPs.

The block-based structure of the DCT calculation maps nicely on the localised processing approach. Although various fast DCT implementations have been developed, the high performance in executing the MAC-instruction makes the direct implementation (matrix multiplication) almost as efficient as ‘fast’ DCT algorithms, especially if they involve highly irregular computations. Another advantage is, that the direct implementation allows to perform any real-valued transform by changing only the transform coefficients.

An issue not addressed so far is the representation of fractional numbers and the calculation precision. Floating-point units are more expensive both in terms of area and power compared to fixed-point DSPs. By appropriately scaling fractionals and taking into account that shifting can usually be handled without any overhead, fixed-point DSPs are sufficient. Important to note is, that the bit precision throughout the processing of CCD data has to be kept as high as possible ($> 8$ bit if the final dynamic range will be $8$ bit).

The entropy coding as outlined in the previous section is a task not mapping straightforward onto DSPs. At a first glance encoding may be programmed as a table look-up. Code tables, however, can get very large. A solution trading-off memory with execution speed compacts table based on the variable length nature of the code words. Most DSPs support conveniently bit manipulations within a register word (16 or 32 bit), but handling manipulations across word boundaries may demand multiple operations. Entropy decoding is even more challenging in the sense that the bitstream must be parsed. Basically, each bit must be analysed to identify the codewords with frequent bit manipulations across word boundaries. The second step, determining the values associated with the codeword, may again be realized as a table look-up, under the same constraints as encoding. As a consequence of efficiency drawbacks chip architectures often implement hardware assisting in entropy encoding and decoding.

For an efficient and fast implementation the processing steps may be rearranged to reduce the number of calculations and the address generation overhead. Examples are to combine multipliers, combine colour space conversion and tone correction, or merge different filters into 1 filter. Also, in case of CCD image processing point operations can be applied to CFA data before tripling the amount of data in the CFA interpolation stage.

4.3. Video

Video processing can be seen as an extension of the previously described basic imaging system, but which challenges system design by increasing constraints and computational load. While in still imaging applications the processing and compression time for a single frame has no hard real-time constraint, to deliver a certain frame rate in video communications the processing and compression time must not exceed a maximum time limit. The second major difference is, that every completely processed image is coded with respect to its predecessor (predictive coding), which considerably increases the bandwidth requirements for memory accesses.

To meet the real-time constraint the computational load can be reduced by implementing less complex algorithms. Advantageous is that the image resolution is restricted to CCIR601, or even smaller. Problematic are data-dependent conditional operations like the VLC encoding and decoding. The execution time for VLC can differ very much such that it would be inefficient to design the software for the worst case in order to guarantee a minimal frame rate (the DSP gets idle for most frames). Instead, implementations should take advantage of the variable frame rate capabilities of the codecs. Regarding the remaining processing steps the same statements as made in the previous section hold.

Because on-chip memory is typically too small to store 2 complete frames, frame predictive coding requires additional external memory accesses to reload (part of) the previous frame. These accesses are slow, so that buffers for motion estimation and compensation should reuse data as much as possible. Based on a block size of $8 \times 8$ the additional buffer for motion estimation needs to hold a block
from the previous frame of \((8 + 2vx_{\text{max}}) \times (8 + 2vy_{\text{max}})\), twice as large as the maximal displacement vector \(v_{\text{max}} = (vx_{\text{max}}, vy_{\text{max}})\). Also recalculating data can be more efficient than reloading. An example is subpel motion estimation which involves interpolated frames. A DMA transferring data in parallel to the DSP processing improves performances.

4.4. Software

A discussion of implementation issues should also review the aspect of software support. Earlier complains of little reusability of DSP code paired with time-consuming assembler programming do not hold anymore. Modern integrated development environments (IDE) exist for DSPs as they are known for GPPs. Furthermore, C Compiler efficiency has improved to a stage where it is necessary to code only very time critically sections in assembler. Modern DSPs are even designed under the constraints of newly developed compiler techniques, such that C-Compilers achieve a very high efficiency on DSPs [7]. Code reusability of assembler modules improves as the installed bases of DSP systems, such as C54x systems, increase. Finally, with operating systems getting more common, the careful planning of the implementation required for code reusability is guided and supported by standardised APIs.

5. Imaging solutions on DSPs

Based on the catalogue DSP C549 two imaging applications have been realized. The first example is a platform for digital still cameras, which utilises the C549 as processor for all image processing tasks as well as controller. The second example demonstrates video encoding and decoding on this DSP.

5.1. Digital still camera platform

The camera system comprises as its core the C549, a new DSC chip, and SDRAM memory [11]. The DSC chip provides the SDRAM controller, a programmable preview engine and includes peripherals like UART, USB, compact flash card IF (CFC) and the TVEncoder (Fig. 9). The system implements all basic operations necessary for a DSC, like preview, capture, decoding and playback. All image processing related tasks are implemented on the DSP, which also serves as the microcontroller. The complete software fits along with all data sections into the 32k of RAM provided by the C549. The fixed-point nature of this DSP poses no constraint on the processing as real-valued filter and DCT coefficients can be handled by scaling. The access to SDRAM is handled through the SDRAM controller such that the DSP needs only to initiate an access request to the SDRAM controller. The transfer is executed in parallel to DSP operations.

In the preview function, which is basically a digital view finder, the CCD image is processed in a preview engine, and stored in SDRAM. Concurrently, the processed data is fetched and streamed through the integrated TV encoder and made available as full-resolution NTSC/PAL signal. The DSP accesses the data in parallel to retrieve exposure control parameters used to reprogram the preview engine. For capturing images, the raw CCD data is stored in SDRAM and subsequently read by the DSP for processing and compression. Each processing units consists of a macroblock \((16 \times 16)\), however the input data size is larger, overlapping with neighbouring blocks to handle filtering. After reading in the CFA data all processing steps are performed (Fig. 5) and the compressed JPEG bitstream for that block is written back to SDRAM or directly to CFC. This structuring allows to run the program solely on on-chip data. The pipeline implements all processing steps as described in Section 3.1. Clocking the DSP at 100 MHz processing an image takes about 3 s for 1 M pixels, including JPEG baseline compression.
The compressed bitstream is stored on compact flash card from where it is read back for display (playback). This operation also scales the image to fit it into the TV resolution. The playback operation takes about 1 s for a 1 M pixel image.

5.2. Video

An MPEG-4 video codec has been implemented on a C54x based system at SQCIF resolution [3] as well as a H.263 codec at CIF resolution on a C6x [28]. This was made possible by localising the processing and minimising the data IO. Overlapping buffers and coding program parts in assembler allowed to fit program and data buffers onto on-chip memory. Both codecs have already processed image data in YUV format as input. The implementation followed the ideas as outlined in the previous sections. A performance of decoding a H.263 bitstream encoded at 700 kb with 70 fps (CIF 352 × 288) demonstrates the DSP capabilities.

6. Summary

This paper gave first a short overview of DSP architecture fundamentals. The requirements of image processing tasks were analysed in the following section before looking in more detail at the implementation on DSPs. Image processing and especially video coding poses very high requirements in terms of computational power and memory bandwidth on processors, challenging traditional DSPs. But it turns out that for numerous applications it is not necessary to design dedicated processors for image processing. Rearranging image processing steps and taking system constraints into account, catalogue DSPs can handle even complex image processing applications very well. Not only the DSP itself but the whole system design including the software determines the final performance [16].

References


