6.6

Embedded Video Codecs

Minhua Zhou and
Raj Talluri
Texas Instruments

1 Introduction

Digital video capture and playback has become a standard feature in many handheld consumer electronic products. Figure 1, illustrates some of the popular, handheld, portable digital multimedia products such as digital still cameras (DSCs), 3G cellular phones, digital camcorders, and personal digital assistants (PDAs). This feature allows the user to capture short video clips in a popular file format such as Apple QuickTime, Microsoft ASF, or MPEG mp4. These captured videos can then be instantly viewed, can be shared via e-mail, or used for later viewing on desktops or laptops. Cellular phones, traditionally designed for voice and data transmission, are becoming increasingly popular and are now providing options to capture still images and video clips and thus enabling applications such as video messaging, video-phoning, video conferencing, or mobile video broadcasting. The 3G cellular phone, for example, allows users to capture the exciting moments as video messages and share them with friends or loved ones via 3G MMS (multimedia message service). It is also now possible to downloaded and watch televised news, movies, or the satellite or terrestrial live television broadcasting on some mobile phones. Video camcorders that have traditionally used analog format to record to magnetic tape are now being replaced by digital format and can record to tape, optical disk, removable memory cards, and recently even hard-disk drives. Inexpensive removable memory cards and advanced video compression technology make it possible to recorder a huge amount video in digital format, which is easy for editing, encryption, storage and information sharing.

The tremendous amount of memory and bandwidth requirements of raw video necessitates the need for efficient compression techniques making video compression technology an integral part of these consumer products. To enable interoperability, most products use standards-based video compression techniques. The ISO MPEG-1 [1], MPEG-2 [2], MPEG-4 [3] and the ITU H.261 [4], H.263 [5], and H.264 [6] are some of the popular international video compression standards. The essential underlying technology in each of these video compression standards is very similar [i.e., motion compensation, discrete cosine transform (DCT), quantization, entropy coding, and deblocking (post or in-loop)] [34]. The standards differ in the applications they address. Each standard is tuned to perform optimally for a particular application in terms of bit rates and computational requirements: MPEG-1 for CD-ROM, MPEG-2 for DVB (digital video broadcasting) and DVD (digital video device), H.263 for videophones, H.261 for videoconferencing, and MPEG-4 for wireless and Internet. H.264, the most recent video compression standard jointly developed by MPEG and ITU-T, provides up to 50% more bit-rate reduction at the same

Copyright © 2005 by Elsevier Academic Press.
All rights of reproduction in any form reserved.

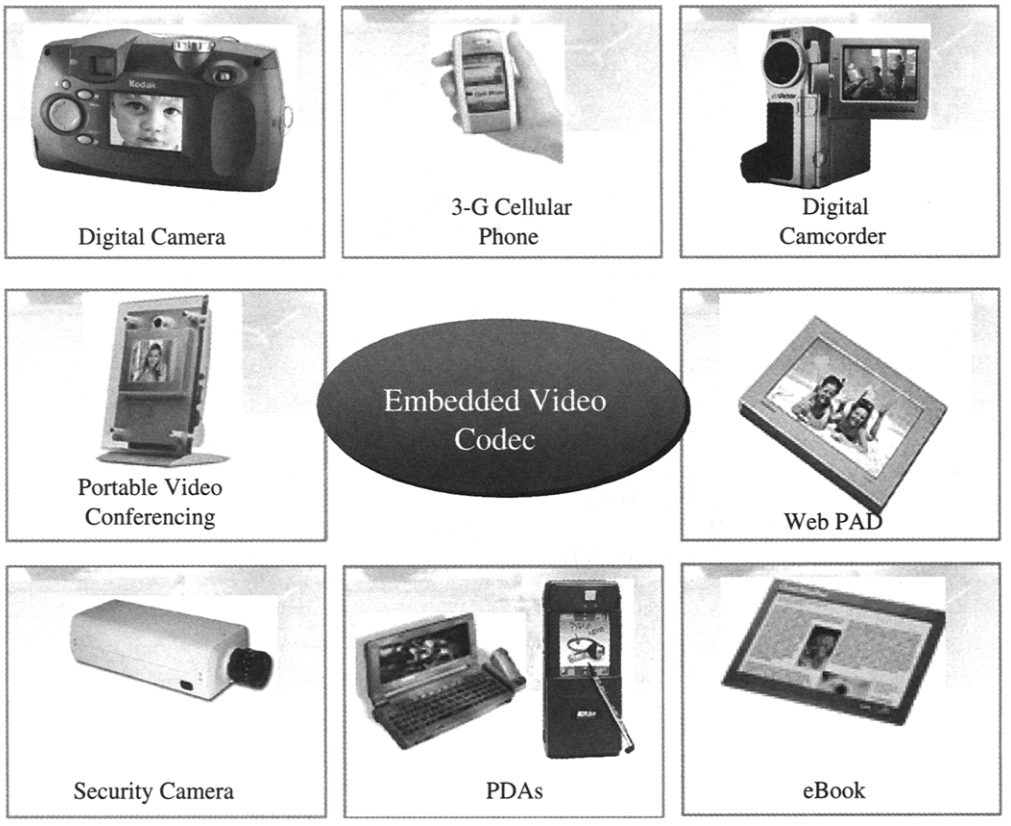

FIGURE 1 Embedded video codec aplication in handheld and portable multimedia products (see color insert).

quality of other existing standards. It will most likely be used in wide variety of applications due to this significant advancement in compression technology.

In the handheld products discussed above, the video functionality is provided along with other multimedia features that include speech, audio, and graphics. The consumer price points that these products need to meet call for a high level of integration in the silicon solutions used to power these products. These silicon solutions thus need to be able support these multimedia functionalities in a highly integrated and cost-effective manner. In addition, most of these products are battery operated and need to be compact. The long battery life demands and the portable nature of these products require the engines that power these products to be extremely energy efficient. In addition, with the ever-increasing image and video resolutions, and the complexity of the compression technologies, these silicon solutions need to be extremely high performance.

In the past few years, the rapid progress made in the very large scale integration (VLSI) technology has dramatically changed the embedded world. Today's high-performance, programmable signal processing chips can comfortably satisfy the high computational and memory bandwidth requirement of video processing while continuing to satisfy the low-power requirements and stringent cost pressures. Although it was hard to implement a cost-effective embedded video codec on programmable processors 5 years ago, it is fairly common now to realize high-quality video capture and playback along with other multimedia functionality on a low-cost digital signal processor (DSP) [35]. The programmable nature of the embedded solution makes it economical to support multiple audio/visual formats or add upgrades and new applications with rapid time-to-market.

However, implementing a good-quality, embedded video codec on these, highly integrated, low-cost, high-performance, low-power, programmable processors requires many algorithmic and engineering trade-offs. This chapter describes some of these trade-offs.

The rest of chapter is organized as follows: Section 2 presents an overview of the popular block-based video coding technique. Section 3 addresses the embedded video codec design requirements and constraints. An example embedded video codec development flow is presented in Section 4, followed by a summary in Section 5.

2 Block-based Video Coding

The essential technology in the MPEG (MPEG1, MPEG2, and MPEG4) and ITU-T (H.261, H.263, H.264) video coding standards is a block-based, predictive, differential video coding scheme. As mentioned in Section 1, the fundamental coding

techniques involved are motion-compensated prediction, transform (4 × 4 integer transform for H.264, and 8 × 8 DCT for the rest of the standards), quantization (Q), entropy coding, and loop-filtering (for H.264 only). In the block-based video coding [21], the sequence of pictures to be encoded is divided into several group of pictures (GOPs). A GOP normally contains an intracoded picture (I-picture) followed by several predictive-coded pictures [P- and B-pictures (bidirectional picture)]. The I-picture is independently encoded without any relation to the previous pictures (hence the name intracoded), whereas in an intercoded P-picture or B-picture, the current picture is predicted from other reference pictures, and the difference between the current picture and the predicted picture is encoded. A P-picture (predictive picture) can only reference the reference pictures in the past, where as a B-picture (bidirectional picture) can predict from reference pictures both in the past and in the future.

To encode a picture, it is first decomposed into a set of macroblocks. A macroblock contains a 16 × 16-luminance area and the corresponding chrominance areas of two chrominance components (for 4:2:0 chroma-format, a macroblock contains a 16 × 16 luminance block and an 8 × 8 chrominance block from each of the chrominance components). As shown in Fig. 2, on the encoder side, motion vectors are estimated for each intercoded macroblock then the motion-compensated prediction is used to reduce the temporal redundancy. The prediction errors are further compressed by using a transform technique, such as DCT, to remove spatial correlation. The transform coefficients are then quantized in an irreversible manner that discards the less important information. The motion vectors associated with intercoded macroblocks are differentially predicted. Finally, the differential vectors are combined with the quantized transform information, and are encoded by using an entropy coding technique such as variable-length coding (VLC). In the case of H.264, an in-loop de-blocking filter (loop-filter) is also applied on the reconstructed macroblock to reduce the blocking coding artifacts [36].

On the decoder side, all the necessary information for picture reconstruction, including motion vectors and DCT coefficients, are retrieved from encoded video bit stream by using variable-length decoding (VLD) technique. Inverse quantization and inverse transform are then used to reconstruct the residual macroblock. A similar motion compensation technique as on the encoder side is also performed in the decoder to obtain the motion-compensated macroblocks. The final reconstructed macroblock is formed by adding the residual and motion-compensated macroblock together and clipping all the pixels to [0:255] range. In the case of H.264, analogous to the encoder side, the loop filter is applied to the reconstructed macroblock at the decoder also.

This macroblock-based coding technique requires only a small amount of fast data memory, thus makes the video coding implementation friendly to programmable multimedia processors. In these processors, the high-speed internal memory size is usually very limited due to the low cost and low-power requirements. In addition, the basic algorithms involved video coding such as motion compensation, transform, and quantization offer a high level of parallelism, which enables the effective real-time implementation on multimedia chips.

3 Embedded Video Codec Design Requirements and Constraints

As discussed above, low cost and low-power consumption are two key requirements in designing a programmable multimedia engine used for handheld applications. To keep the chip cost compellingly low, the on-chip memory size of these multimedia chips needs to be small. Typically, the on-chip program and data memory sizes are in the order of

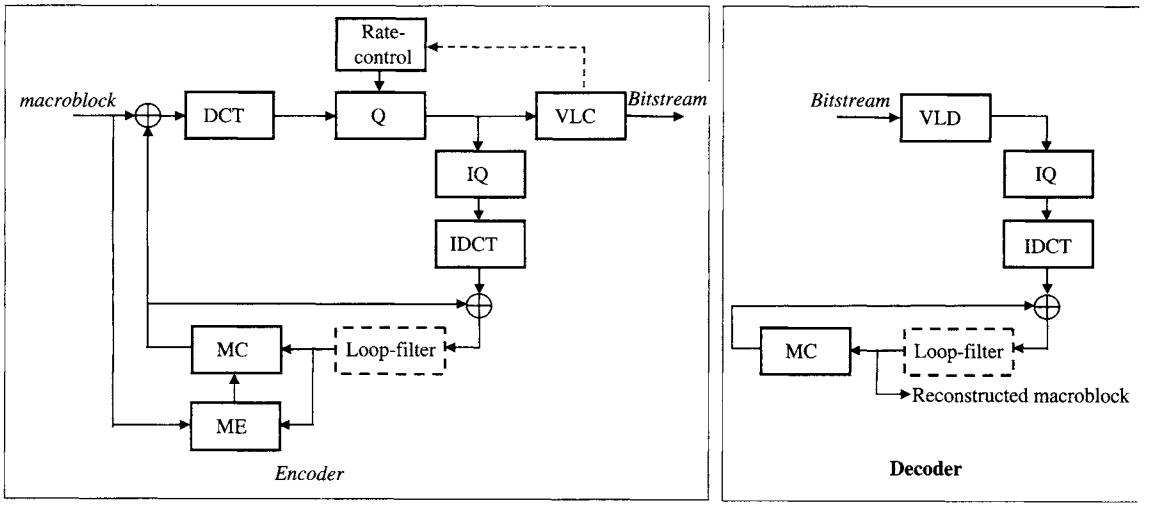

FIGURE 2 Block diagram of MPEG/ITU-T video encoder/decoder.

thousand bytes (e.g., 64 Kbytes, 128 Kbytes, or at best 192 Kbytes). Due to low-power consumption requirements, the clock rate of processors cannot be too high, which restricts the processing power [usually measured in millions instruction per second (MIPS)] and memory bandwidth of these chips. Sophisticated power management logic is also incorporated into these chips to manage power consumption at the lowest possible level.

In developing an embedded video codec, due to the limited memory and processing resources of the chip, particular attention should be paid to code size, code efficiency, and memory allocation. Code should be written in a way that the entire codec could fit into the on-chip program memory. For devices with on-chip instruction cache, the cache miss rate should be minimized. There is usually a trade-off between the code size and code efficiency. For example, for an iteration loop with several conditional cases, there are two different ways to write the code. One possible way is to keep a single execution loop that contains all the conditional cases. In this case, the code size is small but the code is less efficient because the in-loop conditional checks consume extra cycles. The other way is to use multiple loops with each loop dealing with a single case of processing. Here the code size is larger but the code executes faster because the conditional checks are now moved outside the loop. Other similar trade-offs should be made between the code size and code efficiency on the basis of the on-chip program memory or cache size.

Memory allocation is another key factor that has significant impact on code efficiency. Since the on-chip data memory or data cache is extremely limited, only a small amount of data can be buffered on chip. One common strategy is to allocate the most frequently accessed data to on-chip data memory while leaving the less frequently accessed data on off-chip memory. The on-chip data include the static tables used for VLC and VLD, motion vectors, DCT coefficients, reference blocks used for motion compensation, and reconstructed macroblocks. Off-chip memory is used to store less frequently used data such as decoded pictures. To minimize intermediate memory requirements, a memory-overlapping technique is often applied to reuse memory space that is freed up by the proceeding function blocks. To minimize the time and processor resources spent in data transfers from external memory to processor internal memory, these transfers are usually accomplished by an on chip direct memory access (DMA) controller. Attempts to directly access the off-chip data via general-purpose memory interface will significantly decrease the codec performance because such types of memory access are usually much slower.

For handheld multimedia applications, a cost-effective video codec should keep the program size, on-chip data size, and computational complexity in a reasonable range. To do this, compromises between the coding quality and complexity must be made in the embedded video codec design.

Developing a good embedded video codec is about finding a good trade-off between the coding quality and computational complexity. The best possible video coding quality is usually achievable only for non–real-time applications such as off-line video content productions for digital video broadcasting and DVD production. In these applications, the cost of the video encoder is not a critical issue. In these non–real-time applications, the video content can be encoded using sophisticated and computationally intensive video encoding algorithms. The encoder can even use brute force, exhaustive search methods to make more optimal decisions on things such as mode selection and motion estimation. However, for the embedded video encoders running on handheld devices because the memory and MIPS are quite limited, only lightweight algorithms can be used to achieve the target speed at the cost of certain quality loss. An embedded video encoder design is thus about achieving optimal video quality for a given memory/MIPS budget and encoding speed requirement.

Some of the most challenging parts of encoder implementation are motion estimation, mode decision, rate control, and rate-distortion (R-D) optimization. These tools have a significant impact on coding quality and speed. Motion estimation generates the motion vectors for the current macroblock so as to reduce the temporal redundancy with the motion-compensated prediction. The mode decision stage is responsible for choosing the best mode for the current macroblock from the all the possible modes supported the standard. Rate control ensures a video sequence is encoded at the target bit rate. R-D optimization optimizes the video compression by jointly considering the quality degradation and number of bits used. It can be combined with motion estimation, mode decision, and rate control to provide significant quality enhancement in video coding.

Of the encoding tools, motion estimation is usually the most time-consuming part of the encoder in terms of memory access and computation. The complexity of motion estimation depends on the extent of the search range and the number of vectors to be estimated for each macroblock. Search range also determines the size of reference blocks that need to be loaded from off-chip memory to on-chip internal data memory of the processor. In embedded video codecs, it is not realistic to use a full-search, block-matching algorithm to estimate motion vectors for prediction because of the extensive memory and computational requirements of this algorithm. Instead, a fast, lightweight motion estimation algorithm is preferable to achieve a good trade-off between encoding speed and video quality.

For sequences with small motion or small picture resolutions, the N-step (e.g., $N=3$, 4) search motion estimation algorithm [11, 12] or diamond search algorithm [13] is widely used. Typical search ranges of $(-16, 15)$ horizontally and vertically are used. The N-step search reduces computational complexity by a factor about 30 to 40, but still provides good

coding efficiency when compared with that of a full search. However, the reference block size used in N-step search is still a function of search range, and N-step search is also not suited for sequences with large motion or large picture sizes.

Other categories of fast motion estimation algorithms that overcome the shortcomings of the N-step search algorithm are described in [14, 15]. These use two steps to estimate the motion vectors. First, an initial vector is determined from candidate vectors of its neighboring temporal and spatial blocks, the candidate with the smallest prediction error is chosen as the initial vector. In the second step, a vector refinement with a small search range is carried out around the initial vector to obtain the final vector for the block. This type of fast motion estimation algorithm [15] can provide a factor about 100:1 speedup, usually with a quality loss of about 0.3 to 0.6 dB. Since the algorithm uses a fixed number of candidate vectors and refinement search range is of fixed size, the memory requirements and the computational complexity of the algorithm are constant. In addition, due to its self-adjustable nature to motion transitions, the algorithm adapts to the motion in the video sequence and hence is equally efficient for sequences with slow or fast motion, which makes the algorithm work well for a wide range of sequences.

In developing an embedded video encoder, the mode decision process needs to be simplified as well. Instead of supporting all the features allowed by the video standard, the encoder may drop some less critical features up front. Doing so will sacrifice some video quality but can dramatically reduce the coding complexity. To minimize the quality loss for a given complexity, careful investigation should be conducted to test the quality impact and estimate the complexity of individual modes so that the least important modes can be dropped in the encoder design. For example, in H.264 baseline profile [6] up to 16 motion vectors per macroblock and 16 reference frames can be used. Searching through all these vectors and then selecting a motion mode with least prediction error would require huge amount of processing power. To avoid this, one may first consider dropping all the vectors of block size below 8×8 and limiting the maximum number of reference frames to two, because study [16] shows that the quality impact of such a simplification is marginal but complexity reduction is substantial. Another consideration is to identify the less useful features on the basis of the characteristics of the applications. Dropping those features may not affect the coding efficiency but can ease the computational burden quite a bit. For example, one may skip H.264 features like the slice group map, reference frame reordering, and reference frame marking process from the encoder side in the mobile digital video broadcasting with H.264 [6].

Rate control is another tool that impacts video quality significantly. Rate-control algorithms are needed to do frame-level bit allocation according to the target bit rate and frame rate and to compute the quantization scale for macroblocks. The macroblock-level quantization scale computation is expensive because it can require 32-bit multiply and integer division. For constant bit rate (CBR) applications, additional control needs to be added to the algorithm to guarantee that there is no buffer underflow or overflow at anytime [2, 3]. Special rate-control strategies are usually applied to enhance the coding quality at certain special video situations such as scene cuts, fades, and dissolves. For a wireless video application in which there is normally only one I-frame at the beginning in an entire transmission session, the rate-control algorithm must provide intramacroblock refreshment to ensure error recovery. More critically, for two-way video applications like video-phony, the rate control may need to support the functionality such as fast I-frame update (if there are burst errors in wireless transmission [20]) or run-time modification of bit rate, frame rate, and picture size (fading channel [21]).

Because of the limited MIPS in the embedded systems, some computationally intensive featureslike the rate-control strategy dealing with scene changes or local complexity measurement [22] to utilize the properties of human visual systems improving visual qualitymust be dropped from the rate-control algorithm. Instead of macroblock-based quantization scale computation, sometimes the quantization scale adjustment may be performed at slice level or even at picture level to save on computation. Nevertheless, the rate control of an embedded video encoder still needs to hit the target bit rate with satisfactory accuracy and support some advanced features such as intrarefreshment and run-time modification of bit rate, frame rate, and picture size for wireless video applications.

R-D optimization [23] has proven to be the most efficient encoding optimization tool that significantly improves the video quality [16]. However, it requires multiple passes of encoding to calculate the R-D cost of different modes. Due to its extremely high complexity, it is very hard to apply the R-D optimization to an embedded video encoder. However, some lightweight R-D optimization concepts, such as joint motion text coding, might be implemented in an embedded system to achieve some additional quality improvements.

Unlike the encoder implementation, which uses algorithms such as motion estimation, mode decision, rate control, and R-D optimization that need careful tradeoffs in design, the decoder implementation is more or less fixed. A decoder compliant to a certain profile of standard must support all the features that the profile specifies. A standard compliant decoder has no freedom to drop any normative feature from implementation. However, for error-prone applications like wireless video, error detection and error concealment must be implemented on the decoder side. There is no standardized way to perform error detection and concealment: The quality and complexity tradeoff should be made in the design.

Error detection [25] is to ensure no decoder crash in any circumstances, even if audio/speech bit streams are fed into the video decoder. Errors in the bit stream can be detected by checking events such as decoded syntactic element has illegal value, the motion vectors are outside the predefined range, and the number of DCT coefficients decoded exceeds the block size [26]. The more conditional checks that are put into the decoder, the more robust the decoder will be. However, the conditional checks are expensive instructions on programmable devices; thus, too many checks will surely decrease the decoder speed. A good trade-off is to implement only the essential conditional error checks into the decoder to minimize the overhead for error detection. Such a trade-off requires careful study and extensive testing of the decoder.

The video coding standards provide error-resilience tools to ensure error recovery once the error is detected. For example, in MPEG-4 [24] error-resilience tools like resynchronization marks, data partitioning (DP), and reversible variable-length coding (RVLC) are specified. These tools allow the decoder to resume decoding at the next resynchronization point and perform decent error concealment for impaired slices by using the preserved motion vectors (DP) or recover partial text data by decoding the packets backward (RVLC).

Error concealment is used to repair the lost and corrupted macroblocks of a decoded picture after the errors are detected. In the past decade, several error-concealment techniques have been investigated and developed. The error concealment can be done in the DCT coefficient domain [27], in the spatiotemporal domain [29–31], or by using the recursive block matching on the decoder side [28].

However, due to the high complexity of those algorithms, most of them have not found practical use in the handheld video products. Instead, the most common way to do error concealment is by memory copy, which replaces the erroneous macroblocks of the current decoded picture with the colocated macroblocks of previously decoded pictures. If there are more MIPS left for the decoder, more advanced error concealment algorithms can be used. For example, instead of simple memory copy, an erroneous macroblock can be concealed by motion compensation using the motion vectors of its neighboring macroblocks. Just like error detection, the designer must decide what kind of error concealment algorithm to apply, by jointly considering the computational and memory requirements of the algorithm and the bit error rate of the application.

4 Embedded Video Codec Design Flow

Typically, the following steps are involved in a good embedded video codec development:

- Understanding the chip architecture
- Understanding the codec algorithms
- Modularity and application programming interface (API) definition
- Reference software development in Golden C
- Platform-specific development and porting
- Kernel optimization and integration
- Concurrent processing
- Overall optimization
- Stress and conformance testing

4.1 Understanding the Chip Architecture

To achieve the maximum performance, it is very important to first understand the chip architecture. The main factors that should be taken into consideration are processor architecture, memory map, memory access types, instruction set, and any coprocessors supporting the main processor. The chip architecture determines the functional partitioning, memory allocation and data flow of the codec.

Texas Instruments TMS320C6X [9] and TMS320C5X [10] DSPs are two typical examples of embedded multimedia processors that are finding many applications in consumer electronic equipment. As shown in Fig. 3, the TMS320C6X devices are based on VelociTI, an advanced very long instruction word (VLIW) architecture. The C6000 DSP core has eight highly orthogonal functional units, including two multipliers and six arithmetic units, and 32 general-purpose registers of 32-bit length, providing the compiler and assembly optimizer with many execution resources. Eight 32-bit RISC-like instructions are fetched by the computer-processing unit (CPU) each cycle. VelociTI's instruction packing features allow these eight instructions to be executed in parallel, in serial, or in parallel/serial combinations. All instructions operate on registers and can execute conditionally. Its memory maps consist of internal memory, internal peripherals, and external memory. The internal program memory can be mapped into the CPU address space or operate as a cache. The internal and external data memory can be accessed by the CPU or via DMA, reading or writing in 8-bit bytes, 16-bit half-words or 32-bit words.

The TMS320C54X DSP architecture is different from that of TM320C6X. C54X was primarily designed for speech applications. It includes hardware acceleration for common speech functions such as Viterbi accelerator and single-cycle instructions for FIR filter. Unlike TMS320C6X in which the architecture is highly pipelined, C54X architecture mainly executes instructions in a sequential manner. The C54X instruction sets enable extremely small code size for DSP functions, allowing the DSPs to take maximum advantage of on-chip RAM. The internal memory can be accessed by CPU in 16-bit or in 32-bit, but not in 8-bit byte. There are no conditional instructions on the C54X. As shown in Fig. 4 the other key features of the architecture include

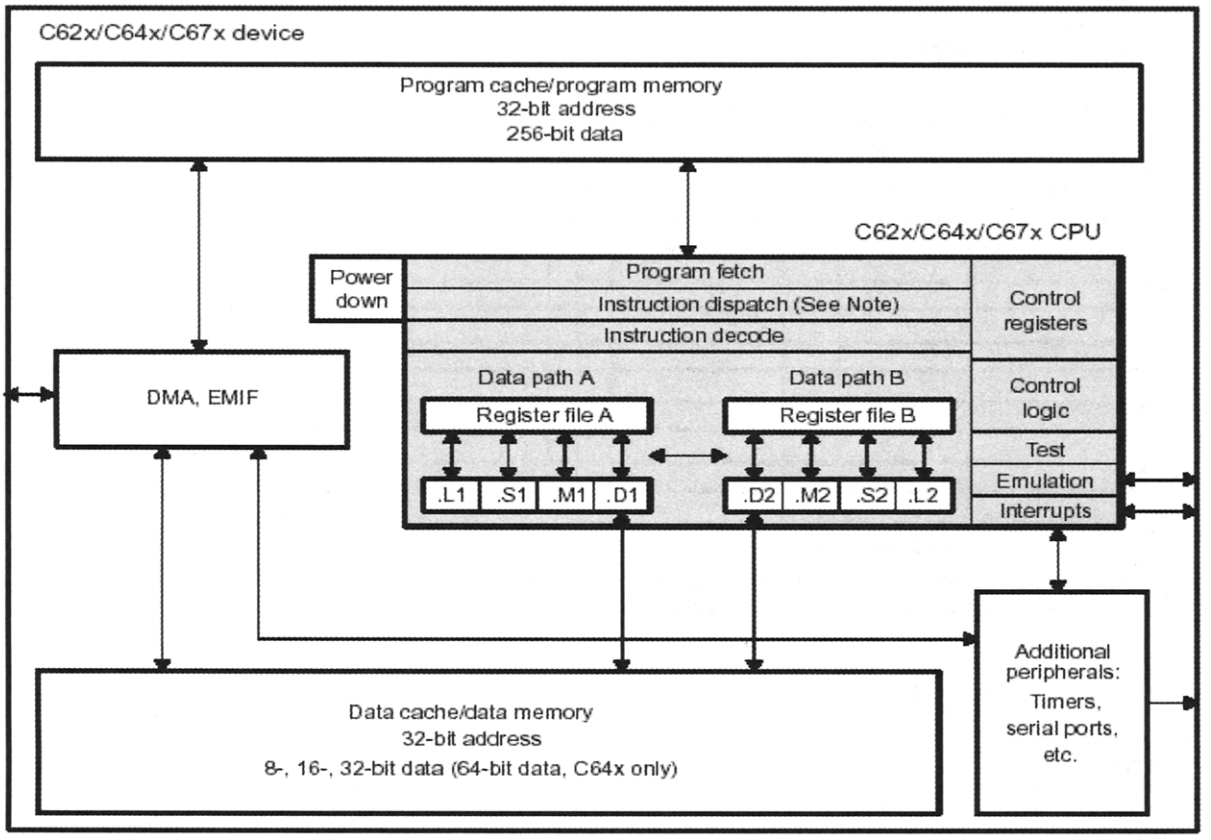

FIGURE 3 Block diagram of TMS320C6X.

the following:

- A 40-bit adder and two 40-bit accumulators support crucial parallel instructions that execute in only one instruction cycle.
- A second 40-bit adder available at output of the multiplier allows nonpipelined multiply accumulator (MAC) operation as well as dual addition and multiplication in parallel.
- Single-cycle normalization and exponential encoding support floating-point arithmetic that is useful in voice coding.
- A 17×17 multiplier allows 16-bit signed or unsigned multiplication, with rounding and saturation control — all in one instruction cycle.
- Eight auxiliary registers and a software stack enable an advanced fixed-point DSP C compiler.

Different chip architectures lead to different programming styles. To achieve the desired performance, the code should be developed in a way that fits the architecture of target device. On TM320C6X, where the CPU can execute up to eight instructions per cycle, the code should be written in a parallel fashion with least amount of memory dependency, so that as many instructions as possible can be scheduled in parallel by the compiler. For iteration loops that contain small amount of instructions inside, it is necessary to unroll the loop. The loop unrolling expands the small loops so that enough number of instructions available to execute in parallel thus utilizes full resources of the C6X architecture.

Since the TM320C54X CPU normally executes only one instruction per cycle, the code can be written in sequential fashion. But attention should be paid to the conditional operations and memory accesses. Because there is no conditional instruction on C54X, the local branches are expensive and thus should be minimized. Best effort should be made to avoid having local branches in inner iteration loops. Moreover, video coding mostly uses 8-bit byte per pixel storage format—for example in motion compensation the reference blocks are stored in 8-bit per pixel. In this case, two reference pixels (two bytes) have to be combined and loaded into C54X accumulator as one 16-bit word, then unpacked as two bytes at run-time for processing. The codec designer should carefully consider such restrictions of the chip architecture when developing the code to get the most performance out of the chip.

4.2 Understanding the Codec Algorithms

Developing an efficient embedded video codec requires not only the knowledge of the chip architecture, but also

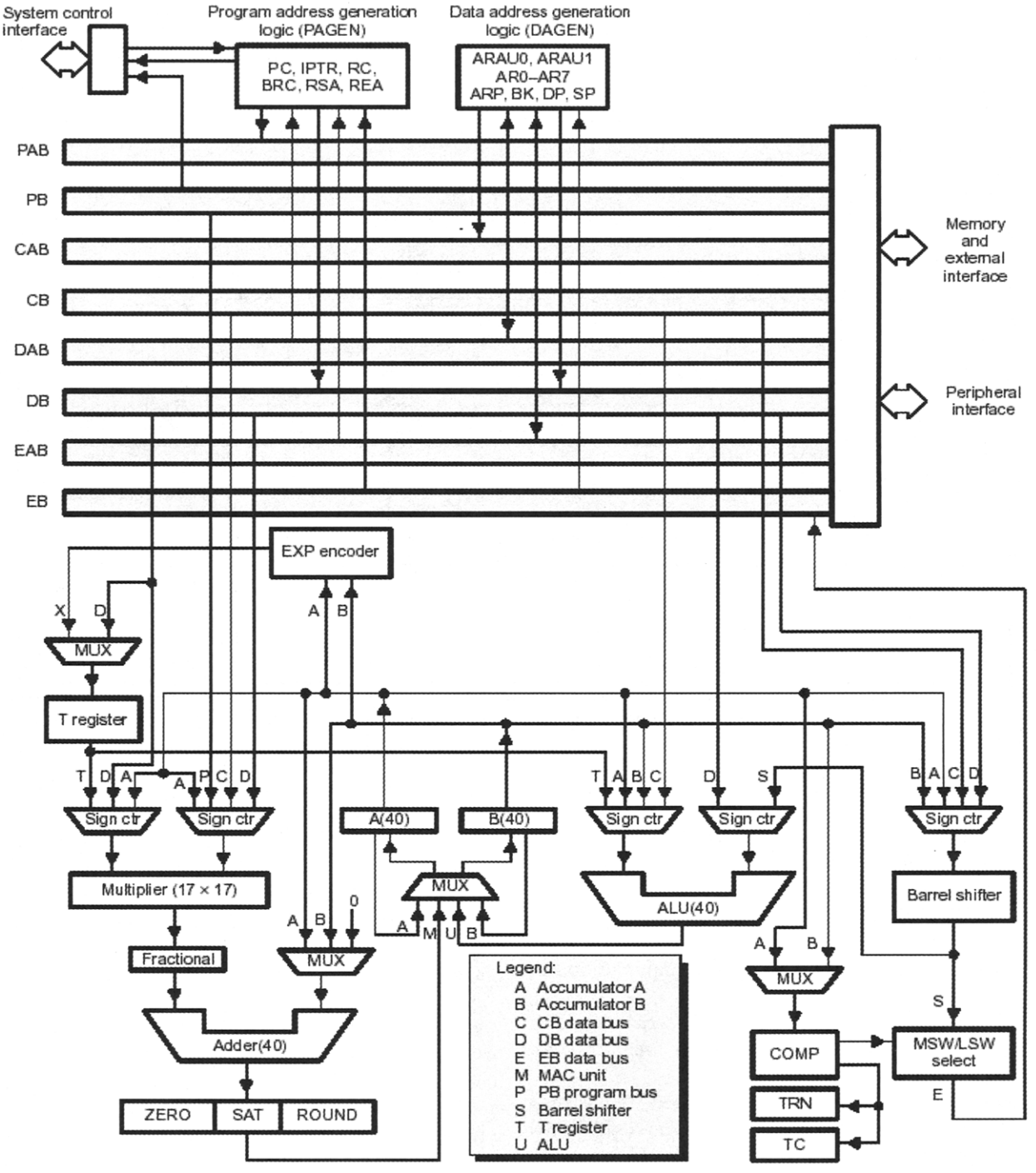

FIGURE 4 Block diagram of TMS320C54X.

deep understanding of the codec algorithms. Although the major coding tools used in the video standards are essentially very similar at high level, they could be significantly different at a more detailed level. Inside each a video coding algorithm, it is important to understand the dynamic range and semantics of each syntactic element to define the appropriate data type for the element and implement appropriate error detection on the decoder side. For each of the coding tools such as transform, quantization, VLC/VLD, and motion compensation, one needs to understand not only the algorithmic detail, but also the implementation details such as input/output data accuracy, memory requirement, memory dependency, and any potential parallelism in implementation.

Different chip architectures may lead to different implementation for a same coding tool. For example, the most efficient realization of 8×8 DCT transform on TM320C6X is to implement it as fast "butterfly" structure [33], whereas on TM320C54X the fastest execution of the same DCT transform is to implement it as "direct matrix multiply." This is due to the fact, that on C6X there are only two multiply and two memory access units among the eight parallel function units, and hence these become the bottleneck for "direct matrix multiply" type implementation of DCT. A butterfly structure on the other hand, reduces the number of multiplication and memory accesses and hence makes it fully suitable for implementation on C6X. On TMS320C54X, there are instructions that can execute two memory accesses, one multiplication, and one addition in a single cycle. These accelerated instructions make the DCT implementation with direct matrix multiply most efficient on C54X.

4.3 Modularity and Applied Programming Interfaces Definitions

Another key concern in developing embedded video codecs is to make the implementation reusable and portable across multiple platforms. One way to accomplish this is to make the codec modular and structured. To enable this, we need to develop APIs for the various components of the software. Figure 5 shows two kinds of APIs—high-level APIs and low-level kernels that are defined in an example video codec implementation.

Different applications may have different requirements in the actual instantiation of the video codec based on the use scenario. For example, a video encoder might be required to support run-time modification of bit rate, frame rate, and picture size in the rate-control algorithm. It may also be required to support fast I-picture update whenever it is requested by decoder or to support the conditional intramacroblock refreshment for error-prone applications. Similarly, a decoder might be required to return the decoded vector field, which could be used for implementing different error-concealment or video stabilization strategies based on the application. In most applications, a video codec is normally integrated with audio/speech codec to realize audio/visual application. Hence, it is important the make the video codec modular so that the video system interface can be tailored to integrate easily with the audio encoding and playback. The high-level APIs should be defined to enable such required functionalities easily. We can broadly classify the low-level kernels, into three categories: encoder specific, decoder specific, and encoder/decoder shared common kernels. It is important to identify the common set of the encoder and decoder functional blocks to minimize the code size, enable reuse, and reduce the development efforts.

We describe below an example embedded video codec implementation and the associated high-level APIs and low-level kernels. There are four high-level APIs in this example.

1. VideoEncodeInit(EncodeInitInputPars, EncodeInitOutputPars)
   Video encoder initialization, including encoder memory allocation, sequence-, picture-level coding parameters initialization, and coprocessor (if any) parameters initialization.
2. VideoEncodePic(EncodePicInputPars, EncodePicOutputPars)
   Encode one picture. The input parameters may include the initial quantization scale, target number of bits for the picture, memory location of reference pictures, original picture and reconstructed picture, and memory location of output bit stream. The output might include coding status, number of bits generated by picture, bit stream, and final quantization scale of the picture.
3. VideoDecodeInit(DecodeInitInputPars, DecodeInitOutputPars)
   Video decoder initialization, including decoder memory location, and coprocessor (if any) parameters initialization.

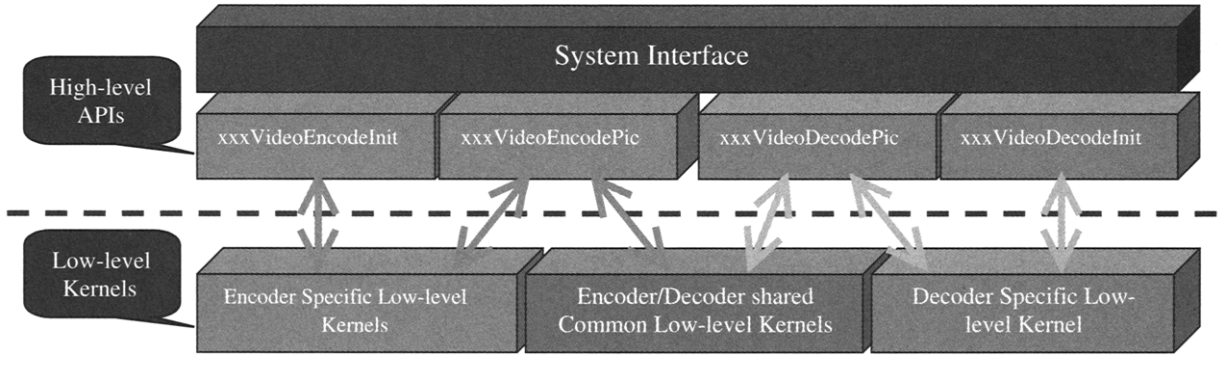

FIGURE 5 The relationship between the high-level APIs and low-level kernels in embedded video codec.

4. VideoDecodePic(DecodePicInputPars, DecodePicOutputPars)
   Decode one picture. The input parameters may include memory location of reference pictures and reconstructed picture, memory location of input bit stream, and output picture format. The output may include the decoding status, number of bits used by the decoded picture, motion vector field, and so forth.

The high-level APIs call the low-level kernels to realize the encoding/decoding functionality. The encoder specific low-level kernels include:

1. Encoder memory allocation: Centralized memory allocation on the encoder side.
2. Encoder high-level parameters initialization: Initialization of parameters such as picture size, target bit-rate, and so forth.
3. Encoder coprocessor initialization: Initialization of the coprocessors.
4. Rate control: Bit allocation to each macroblock according to bit stream buffer status and macroblock content.
5. Motion estimation: Estimation of the motion vectors for a macroblock.
6. Mode decision: Selecting the best coding mode (e.g., intra/inter, 1/4MV) for the current macroblock.
7. Forward transform: for example, 8×8 DCT transform.
8. Forward quantization of DCT coefficients.
9. Sequence level header encoding.
10. Picture level header encoding.
11. Slice level header encoding.
12. Macroblock overhead encoding: Encoding macroblock and block type such as intra, inter, codec, skipped, and so forth.
13. Motion vector encoding.
14. VLC of DCT coefficients.
15. Intra (AC/DC) prediction.
16. Encoder utility routines for bit stream handling.

The decoder-specific, low-level kernels are defined as below (error detection is normally implemented in kernels 1 to 6)

1. Sequence level header decoding
2. Picture level header decoding
3. Slice level header decoding
4. Macroblock overhead decoding: Decode MTYPE (macroblock type), CBP (coded block pattern), and so forth
5. Motion vector decoding
6. Variable length decoding of DCT coefficients
7. Inverse intra (AC/DC) prediction
8. Decoder utility routines for bit stream handling
9. Error concealment: To conceal corrupted or lost macroblocks by using certain error concealment scheme such as the memory copy

The common low-level kernels shared by encoders and decoders are as follows:

1. Inverse quantization of DCT coefficients
2. Inverse transform
3. Motion compensation
4. Macroblocks reconstruction: Adding prediction blocks and IDCT output together and clipping results to [0:225] range
5. Loop-filter (for H.264 only)
6. Boundary macroblock padding: Prepad the reference picture boundaries for unrestricted motion compensation (for MPEG4, H.263, H.264 only)
7. Data reorganization (e.g., YUYV interleaved to YUV separated)

To share the common low-level kernels, the encoder and decoder must have common picture storage format.

4.4 Reference Codec Software Development in Golden C

One of the key recommended steps in developing an embedded video codec is to first develop a platform-independent video codec implementation in what is called as *Golden C*. The C programming language is usually the language of choice for embedded video codec development. The goal of this implementation is to be platform independent but serve as an efficient reference for the final platform-specific implementation. One of the key uses of this is that we can always refer back to this implementation to ensure compliance of the final implementation to the standard. Other advantages include the ability to quickly be able to target this Golden C implementation to various platforms and various applications as needed.

For most of the popular video coding standards, the trend is for the standards bodies to provide a public domain reference software implementation of the standard. However, this reference code is usually meant to be instructional about the standard and is not written to be an efficient implementation of the standard, particularly from an embedded system implementation point of view. Therefore, in creating an efficient and optimal implementation of a standard, it is recommended to implement the entire codec in Golden C from scratch, taking the standards based public domain codec as the functional reference code.

If both the encoder and decoder need to be developed, it is recommended to first start with the decoder implementation. For the decoder, there is a conformance bit stream test suite available for exercising all the features required by standard. The decoding of all the conformance bit streams will guarantee a standard-compliant implementation of decoder. Once the compliant decoder is in place, the encoder can then be built up by reusing the common low-level kernels, developing the encoder counterparts of the

decoder-specific kernels, and finally adding the encoder specific motion estimation, mode decision, and rate-control algorithms. In this way, the most parts of code are similar for both the encoder and decoder, making the debugging of the codec much easier.

Some of the recommended programming rules for the Golden C implementation are as follows:

1. No usage of dynamic memory allocation such as "malloc()." All the memory allocation should the centralized and performed in the codec initialization API. Certain memory overlapping may be used to minimize the "on-chip" memory requirement.
2. Picture buffers should always be located in the on "off-chip" memory external memory. No direct memory access to the picture buffer should be assumed. Dedicated block writing/reading routines should be implemented to perform data transfers between the picture buffer and "on-chip" data buffer. The typical operations are writing or reading bit stream to and from off-chip memory, loading the original macroblocks and reference blocks from the picture buffer, and writing out reconstructed macroblocks to the picture buffer. The dedicated data transfer routines will be replaced by the DMA routines when the code is ported on hardware board.
3. Except for static tables such as VLC/VLD Huffman tables, no global variables should be used. Using global variables makes code migration from one codec to other difficult.
4. No big arrays (larger than 128 bytes) should be opened inside the local functions. Big local data arrays could cause stack overflow problem.

4.5 Platform-Specific Development and Porting

Starting from the Golden C implementation, the platform-specific implementation is then developed by taking the hardware architecture of target platform into account. The platform-specific implementation is designed to be able to compile and run on the target device and to produce the bit-exact results as the Golden C implementation. Certain modifications need to be introduced to the Golden C code to form the platform-specific implementation. The major changes are as follows:

1. Memory map: The codec initialization API to do memory allocation for the on-chip and off-chip data buffers need to be modified according to the actual memory map of the chip.
2. Memory addressing: Some platforms such as TM320C5x may not support 8-bit byte memory accesses. Hence, the related routines should be rewritten to avoid unsupported memory access type.
3. Data transfer between on-chip and off-chip memory: Data transfer routines that are used for data transfer between on-chip and off-chip memory should be replaced by the actual DMA utility routines.
4. Run-time support library: Functions such as "printf" should be removed because of the large code size of these functions.
5. System support routines: The hardware system initialization routines and other system support routines should be integrated into the project when compiling the code on the hardware device.

The platform-specific C-code should compile and run on both the device simulation platforms like PC/UNIX and the target device hardware system. It is suggested to keep Golden-C part of code for items 1, 3, and 4, and to turn on item 5 only when code is complied for hardware device. Compiler directives should be used to switch between the Golden C implementation and the platform-specific implementation for related routines. Keeping the platform-specific software executable on PC/UNIX simulators makes it easy to introduce future changes to the code since these changes can be first debugged using the more advanced development tools on PC/UNIX device simulators and then later ported on the device hardware platform.

4.6 Kernel Optimization and Integration

At this point in the flow, the platform-specific implementation running on the device is still not optimized in terms of performance. It mainly serves as a functional reference for next stage of kernel-level optimization on the target device. The final optimized version of codec is normally a hybrid version, with high-level control logic in C and low-level kernels in assembly or other coprocessor languages. All the encoder/decoder-specific and common low-level kernels are candidates to be optimized. The kernel optimization is usually the most time-consuming stage in embedded video codec development.

The chip architecture determines the specific kernels to be optimized. For devices such as TMS320C6X, in which a highly efficient C-compiler is available, C-level kernel optimization with loop unrolling and intrinsic operators can already provide about 90% of performance optimization for highly pipelined kernels such as sum of absolute differences computation in motion estimation and block interpolation in motion compensation. Only highly inter-dependent kernels such as VLC/VLD need to be carefully scheduled and optimized in linear or hand assembly. On the platforms like TMS320C54X, however, the C-compiler can achieve only about 50% of performance optimization on the well-designed C-code. Thus, assembly optimization for all kernels is the only way to achieve the best possible performance for the codec.

It is more challenging to carry out kernel optimization on platforms with multiple processors. The multimedia chips that support video functionality often consist of several processors, with hard-wired blocks for some of the computationally intensive function.

Memory access can also consume significant amount of cycles. Sometimes the memory access is very expensive if the clock rate is high. Therefore, it is important to minimize the number of memory accesses. For a device like TM320C6X with single-cycle, 32-bit, 16-bit, and 8-bit, memory access, it would be more efficient to modify the kernels to combine 16-bit or 8-bit memory access into 32-bit memory access. Sometimes, it is advantageous to restructure the kernels to reduce the number memory accesses. For example, combining VLD and inverse quantization kernel into one on the decoder side can cut the memory access cycles by half of the number that would be needed if those two kernels were implemented separately. Avoid or minimize the memory bank conflicts in these memory accesses.

Another optimization step is to minimize the DMA transfer overhead. For some platforms, there is significant overhead caused by issuing DMA requests. Hence, the number of DMA requests per macroblock should be kept as low as possible. If there is enough on-chip data memory on the device, the common strategy is to combine data of several macroblocks into one group and use DMA engine to move the data piece by piece instead of macroblock by macroblock to minimize the number of DMA transfers.

Once the kernel is optimized, integrate the kernels into the codec one by one. Because there is a standalone C reference function available for each kernel, this methodology is advantageous for debugging the optimized kernels and localizing the bugs if there are problems in the integration phase.

4.7 Concurrent Processing

Once the kernel optimization and integration is done, the next performance optimization step is concurrent processing. If there are multiple processors on the chip that can run independently, then some of the tasks can be run concurrently to further improve the codec performance.

Figure 6 illustrates one case of concurrent processing on the video decoder side. In this case, there are two processing units on the chip, the CPU (e.g., TMS320C6X core) and the DMA. DMA can run without intervention of CPU once it is set up and kicked off by the CPU. As shown in Fig. 6, after the motion vectors are decoded for the current macroblock, the location of reference blocks in SDRAM is known. The CPU can set up the DMA and issue the DMA request to load the reference blocks for the motion compensation. While the CPU is performing the VLD, inverse quantization, and IDCT, the DMA is loading the reference blocks from SDRAM to on-chip data memory in the background (see block B of Fig. 6). After the motion compensation and the reconstruction of the current macroblock, another DMA request can be issued to write the reconstructed macroblock from on-chip memory to SDRAM. Here the CPU is doing the macroblock overhead and vector decoding for the next macroblock, while the DMA is writing out the current reconstructed macroblock (see block a of Fig. 6). Hence, with the concurrent processing between the CPU and DMA, the decoder performance is improved because the data transfer between on-chip and off-chip memory is hidden behind the processor tasks.

The more processing units a chip has, the more complicated the concurrent processing will be. If a device has several processing units (e.g., on TMS320DM270 [8] there are C54X, iMX, VLCD/QIQ, DMA and ARM7—five processing units), the concurrent scheduling becomes complex because it needs to resolve the resource conflicts between tasks running on the different processing units. The concurrent scheduling normally involves the following steps:

1. Benchmarking the entire codec; getting accurate cycle count measurement for all the kernels.
2. Partitioning the encoding and decoding processes into tasks—one task may contain several steps of processing (kernels).
3. Drawing the dependency (processing unit and memory buffer usage, timing dependency) diagram of tasks

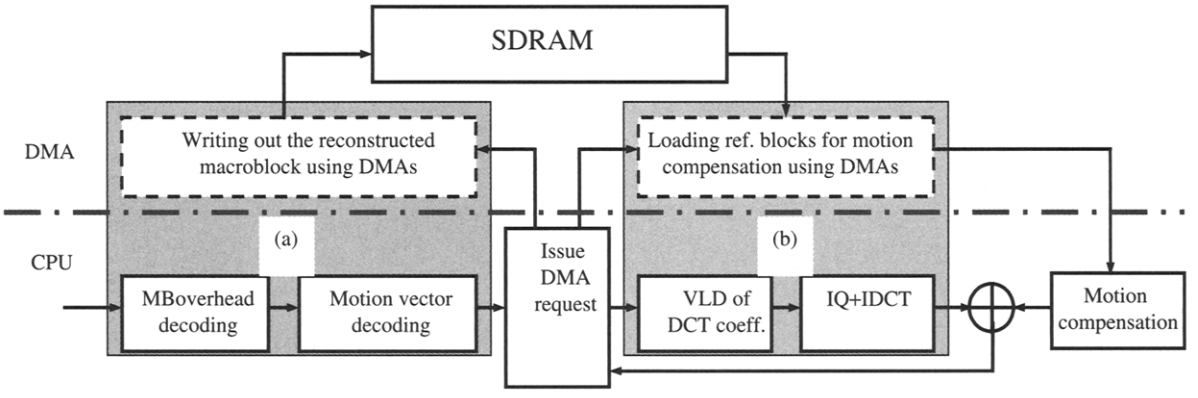

FIGURE 6 Example of concurrent processing between direct memory access and computer processing unit for video decoding.

and repeating step 2 to resolve the resource conflicts. Computing the cycle counts for each task according to kernel benchmark.

4. Drawing the concurrent scheduling diagram according the task cycle count, timing, and resource usage.
5. Modifying the code according to the scheduling diagram to realize the concurrent processing.

4.8 Overall Optimization

After the concurrent processing is done, the final tuning of the code can further increase the codec performance. The overall optimization is mainly for the embedded video codec running on a device with multiple processors. The main idea is to measure the loading of each processor, identify which processor is the bottleneck, and then consciously move some kernels from one processor to another to balance the loading among different processors. If a kernel is must be moved to a different processor to balance processor loading, optimization for the particular kernel will need to be redone according to the architecture of the new processing unit.

A typical example of such a multiple processor chip is TMS320DM270. The TMS320DM270 programmable DSP-based solution from Texas Instruments is a highly integrated video and imaging engine offering excellent performance, leading edge process technology, and flexibility for next generation portable media products. The DM270 (see Fig. 7), which contains the TMS320C54XTM DSP, the ARM7TDMI® RISC processor, imaging peripherals, and video and imaging coprocessors, is a highly integrated, programmable platform for the DSC and other multimedia applications including portable multimedia jukeboxes, camera phones, DVD players, televisions, and digital video recorders. DM270's programmability comes from a DSP-based imaging coprocessor that enables manufacturers to implement their own proprietary image processing algorithms in software. The interface is flexible enough to support various types of CCD and CMOS sensors, signal-conditioning circuits, power management, SDRAM, shutter, and iris and autofocus motor controls.

The programmable DM270 supports a variety of video, imaging, audio, and voice compression standards, including, but not limited to JPEG, motion-JPEG, MPEG-1, MPEG-2, MPEG-4, H.263, H.264, DivX, Windows Media Video (WMV), as well as audio standards such as MP3, Advanced Audio Coding (AAC) and Windows Media Audio (WMA). Supported voice standards include G.711, G.723.1, and G.726. The DM270 system-on-a-chip (SoC) has the ability to run various operating systems, including Nucleus, Linux, ulTRON, VxWorks.

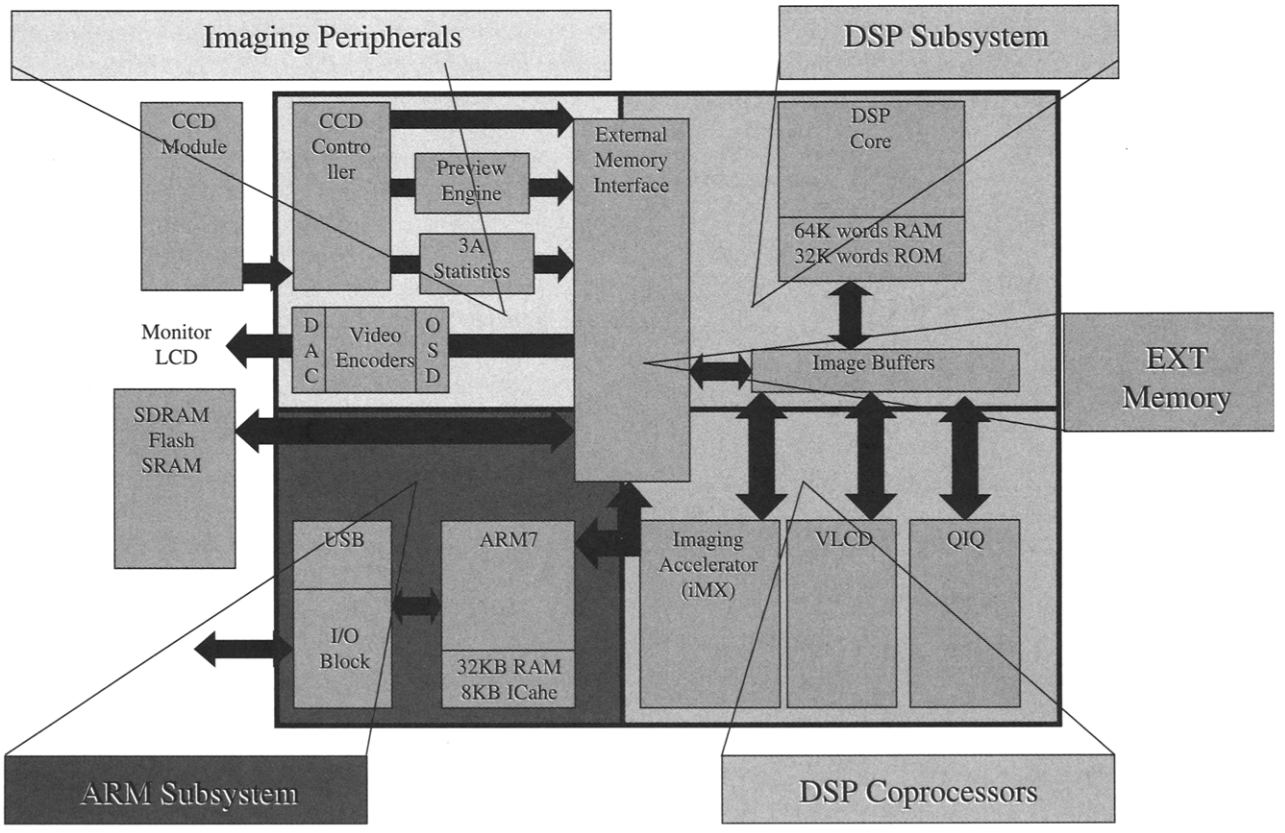

FIGURE 7 Diagram of TMS320DM270.

The DM270 architecture represents the cutting-edge trend for the programmable multimedia chip design, with the most critical parts of the imaging/video function blocks in hardware and the less critical parts in software. This architecture provides customers with both the high performance and flexibility. The DM270 DSP-subsystem, on which imaging/video function is realized, is made of the C54X DSP, IMX accelerator, VLCD/QIQ coprocessor, DMA controller plus the dedicated on-chip memory buffers. The VLCD/QIQ coprocessor supports VLC/VLD/Q/IQ of DCT-coefficients for MPEG1, MPEG2, MPEG4, H.261, and H.263. The imaging processing accelerator iMX is designed for speedup of high-pipelined image/video processing functions, such as motion estimation, motion compensation, and DCT transform.

For the MPEG4 simple profile [3] decoder implementation on DM270, for example, kernels are running on different processors. The iMX is used for all the parallel operations like motion compensation, inverse DCT transform, macroblock reconstruction, data reorganization, and so forth. The VLCD/QIQ coprocessor performs the MPEG4 VLD and inverse quantization, while the C54X DSP is responsible for the VOL, VOP, slice and macroblock header decoding, and other high-level control code.

In implementing an MPEG-4 codec on DM270, no assembly optimization will be needed for quantization, inverse quantization, or VLC/ VLD of the DCT coefficients. Simple register setup for the QIQ/VLCD engine will do the job as there are dedicated coprocessors for these functions, although for kernels like motion estimation, motion compensation with high-level of parallelism, iMX coprocessor should be the used. The rest of the kernels are optimization on C54X in assembly language.

With the overall optimization done, the DM270 is able to provide simultaneous playback of 30 frames per second for MPEG-4 simple profile at VGA (640×480) resolution simultaneously with AAC-LC audio decoder.

4.9 Stress and Conformance Testing

After the codec optimization is done, extensive testing should be carried out before the code is released to market. Code testing under unusual conditions may uncover some deep system—level bugs rarely seen under normal circumstance. The code testing may prove to be a very time-consuming process for embedded video codec development.

For an embedded video encoder, the testing is to make sure that the encoder generates the standard compliant bit streams with market-acceptable quality for target applications. It would be helpful to test the encoder with field trial sequences to verify the coding quality, effectiveness of the designed encoder functionality, and the robustness of the code. The encoder also needs to run stressing tests to make sure that it does not crash after a long time of continuous encoding. The encoder also needs to be tested with video sequences of varying levels of scene complexity from low-motion to high motion and low spatial detail to high spatial detail. The standards bodies also usually provide a set of standard video sequences to test the video codec. It is also important to test the codec under varying lighting conditions.

The testing of an embedded video decoder is more complicated. For the functional tests, the conformance bit stream test suite can be used to verify whether the decoder output on the device is identical to the anchor sequences corresponding to these bit streams. For robustness tests, the decoder can be tested with long bit streams. However, for the error-protection part of tests, a significant amount of time will have to be spent. The decoder is required to be functional under all circumstances. The decoder is not supposed to crash even if the decoder is fed with "garbage" bit streams. A common way to develop the robust error-protection scheme is to take a compliant bit stream that excises all major coding features, corrupt the data byte by byte so that error type and position are known in advance, and input the corrupted data into the decoder to see if it crashes. If it does crash, knowing the error type and position is helpful for finding the crash reason and fixing the problem.

5 Summary

Digital video capture and playback has become a standard feature for many handheld consumer electronic products such cas digital still cameras, camera phones, camcorders, and PDAs. Video compression is used in these products for accommodating the limited storage and transmission requirements. Video compression standards (MPEG/ITU-T) are popular as they guarantee interoperability among products. The block-based nature of these standards makes them implementation friendly to embedded video solutions on programmable multimedia chips. The programmable nature of embedded solutions provides a cost-effective way to support multiple audio/visual formats or add upgrades and new applications with rapid time-to-market. Due to the limited memory size and processing power of programmable multimedia chips used in consumer appliances, the embedded video codec design needs to make good trade-off between quality and complexity. Particularly, efforts should be made to develop lightweight algorithms for motion estimation, mode decision, rate control, error detection, and error concealment. Understanding of the chip architecture and the codec algorithm is the key to designing the most efficient embedded video codec. Characteristics of chip architecture and codec algorithm determine the code partitioning, data flow, and memory allocation of the codec. A good trade-off between code size and code efficiency, minimization of memory access, memory bank conflicts, cache miss rate and DMA transfer overhead, and balanced processor

loading in concurrent mode are some of the key factors that contribute to an efficient implementation of embedded video codec.

6 References

[1] ISO/IEC 11172-2 (MPEG1), "Information technology: coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s: part 2 video," (August 1993).

[2] ISO/IEC 13818-2, ITU-T "Rec. H.262: Information technology: generic coding of moving pictures and associated audio information—part 2: video," (1995).

[3] ISO/IEC 14496-2, International Organization for Standardization, *Final Draft of Internal Standard: Information technology Coding of Audio-Visual objects: Visual* (1998).

[4] International Telecommunications Union Telecommunications Standardization Sector, "Recommendation H.261: video codec for audiovisual services at px64 kbits," (Helsinki, March 1993).

[5] International Telecommunications Union Telecommunications Standardization Sector, "Recommendation H.263: video coding for low bit rate communication," (Geneva, 1996).

[6] ISO/IEC 13818-10, ITU-T *Rec. H.264: Information technology: Coding of Audio-Visual objects: Part 10: Advanced Video Coding* (2003).

[7] M. Zhou and R. Talluri, "DSP-based real-time video decoding," *IEEE Digest of ICCE'99* 296–297 (1999).

[8] TMS320DM270 product fact sheet, from Texas Instruments, http://focus.ti.com/pdfs/vf/vidimg/dm270fs.pdf.

[9] TMS320C6X User Guides, from Texas Instruments, http://dspvillage.ti.com/docs/catalog/resources/techdocs.jhtml?familyId=132&navSection=user_guides.

[10] TMS320C5X User Guides, from Texas Instruments, http://dspvillage.ti.com/docs/catalog/resources/techdocs.jhtml?familyId=114&navSection=user_guides.

[11] R. Li, B. Zeng, and M. L. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.* 4, 438–442 (1994).

[12] L. M. Po and W. C. Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.* 6, 313–317 (1996).

[13] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast block matching motion estimation," *IEEE Trans. on Image Processing,* 9, 287–290 (2000).

[14] L. K. Liu and E. Feig, "A block-based gradient descent search algorithm for block motion estimation in video coding," *IEEE Trans. Circuits Syst. Video Technol.* 6, 419–423 (1996).

[15] M. Zhou, "A fast motion estimation algorithm for MPEG-2 video encoding," *SPIE Proc. VCIP'99,* 3653, 1487–1495 (1999).

[16] M. Zhou, "Simplification of H.26l baseline coding tools." 1TV-TQ. 6/16, Doc. JVT-B030 (2002).

[17] H. S. Kong, A. Vetro, and H. Sun, "Combined rate control and mode decision optimization for MPEG-2 transcoding with spatial resolution reduction," *IEEE Int. Conf. Image Process.* 1, 161–164 (2003).

[18] D. Turaga and T. Chen, "Classification based mode decisions for video over network," *IEEE Trans. Multimed.* 3, 41–52 (2001).

[19] M. Hamdi, J. W. Roberts, and P. Rolin. "Rate control for VBR video coders in broadband networks," *IEEE JSAC—Journal on Selected Areas in Communications,* August 1997.

[20] C. Y. Hsu, A. Ortega, and M. Khansari, "Rate control for robust video transmission over burst-error wireless channels," *IEEE J. Sel. Areas Commun.* 17, 756–773 (1999).

[21] J. Razavilar, K. J. Ray Liu, and S. I. Marcus, "Jointly optimized bit-rate/delay control policy for wireless packet networks with fading channels," *IEEE Trans. Commun.* 3, 484–494 (2002).

[22] ISO/IEC JTC1/SC2/WG11/N0400, "Test Model 5, Draft Revision 2," (April 1993).

[23] G. J. Sullivan and T. Wiegand, "Rate-distortion optimization for video compression," *IEEE Sign. Process.* 5, 74–90 (1998).

[24] R. Talluri, "Error-resilient video coding in the ISO MPEG-4 standard," *IEEE Commun.* 36, 112–119 (1998).

[25] K. Ekram, G. Hiroshi, S. Lehmann, et al., "Error detection and correction in H.263 coded video over wireless network," *Proc. 12th International Packetvideo Workshop (PV 2002)* (2002).

[26] K. Bhattacharyya and H. S. Jamadagni, "DCT coefficient-based error detection technique for compressed video stream," *IEEE Int. Conf. Multimed. Expo.* 3, 1483–1486 (2000).

[27] Y. S. Lee, K.-K. Ong, and C.-Y. Lee, "Error-resilient image coding (ERIC) with smart-IDCT error concealment technique for wireless multimedia transmission," *IEEE Trans. Circ. Syst. Video Technol.* 13, 176–181 (2003).

[28] M.-J. Chen, C.-S. Chen, and M.-C. Chi, "Recursive block-matching principle for error concealment algorithm," *Proc. 2003 Int. Symp. Circ. Syst. (ISCAS2003)* 2, 528–531 (2003).

[29] W. Zeng, "Spatial-temporal error concealment with side information for standard video codecs," *Proc. Int. Conf. Multimed. Expo. (ICME)* 2, 113–116 (2003).

[30] S. Belfiore, M. Grangetto, E. Magli, et al., "Spatial-temporal video error concealment with perceptually optimized mode selection," *Proc. IEEE Int. Conf. Acoustics Speech Sign. Process.* 5, 748–751 (2003).

[31] W. Y. Kung, C. S. Kim, and C.-C.J. Kao, "A spatial-domain error concealment method with edge recovery and selective directional interpolation," *Proc. IEEE Int. Conf. Acoust. Speech Sign. Process.* 5, 700–703 (2003).

[32] T. P. Chen and T. Chen, "Second-generation error concealment for video transport over error prone channels," *IEEE ICIP* (2002).

[33] E. Feig and S. T. Winograd, "Fast algorithms for discrete cosine transform," *IEEE Trans. Signal Proc.* 40, 2174–2193 (1992).

[34] A. N. Netravali and B. G. Haskell, *Digital Pictures: Representation, Compression, and Standards,* 2nd ed. (Plenum Press, New York, 1995).

[35] D. Talla, C. Y. Hung, R. Talluri, et al., "Anatomy of a portable digital mediaprocessor," *IEEE Micro* 24, 32–39 (2004).

[36] I. E. G. Richardson, *H.264 and MPEG-4 Video Compression* (John Wiley & Sons, New York, 2003).

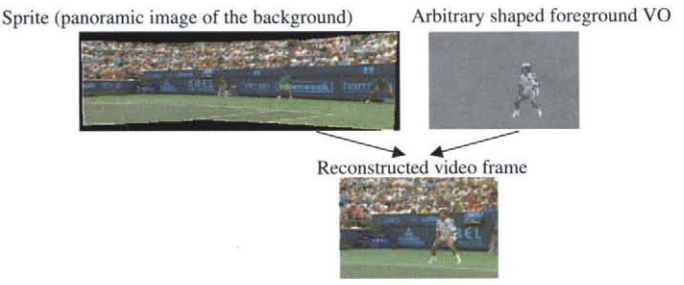

FIGURE 6.5.7 Sprite coding of a video sequence (courtesy of Dr. Thomas Sikora, Technical University of Berlin, [6]). VO, video object.

FIGURE 6.5.17 A decoded frame of the sequence *Foreman* (a) without the in-loop deblocking filtering applied and (b) with the in-loop deblocking filtering (the original sequence *Foreman* is courtesy of Siemens AG).

FIGURE 6.6.1 Embedded video codec aplication in handheld and portable multimedia products.

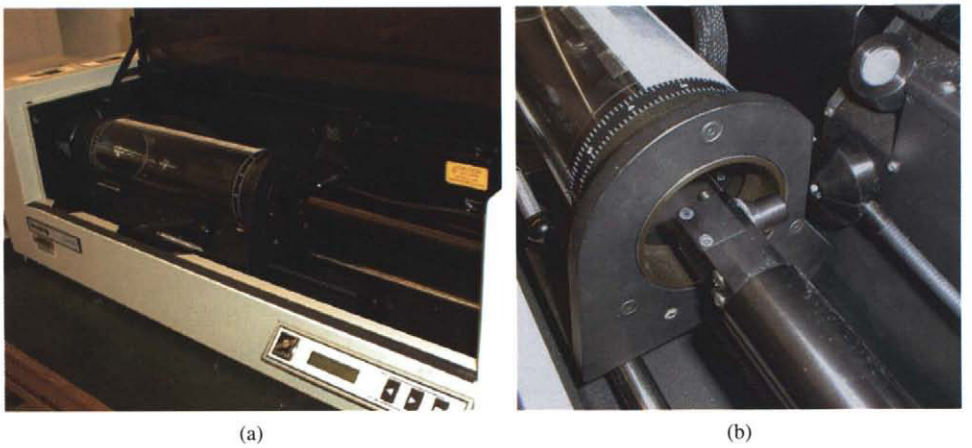

FIGURE 7.1.1 High resolution drum scanner: (a) scanner with cover open, and (b) closeup view showing screw-mounted "C" carriage with light source on inside arm, and detector optics on outside arm.