Combining Efficiency and Error Robustness

Wireless MPEG-4 Video Communication on DSP Chips

Madhukar Budagavi, Wendi Rabiner Heinzelman,
Jennifer Webb, and Raj Talluri

Wireless video communication has long been a technological fantasy dating back before Dick Tracy and the Jetsons. One of the earliest science-fiction novels, *Ralph 124C 41+* ("one to foresee"), by Hugo Gernsback, had a cover depicting a space-age courtship via videophone, shown in Fig. 1 [14]. (In fact, it was Gernsback who designed and manufactured the first mass-produced two-way home radio, the Telimco Wireless, in 1905 [25].) While wireless speech communication has become a reality, video communication still poses a number of technological challenges.

When compared to speech communication, the main challenge for video communication has always been the enormous bandwidth requirement. For instance, broadcast (one-way) analog television channels in the US are spaced by 6 MHz, compared to FM radio channel spacing, which is only 200 kHz. For digital high-definition television (HDTV), 19.3 Mbps can be transmitted in a 6 MHz channel [11]. Even for short-range wireless communication, e.g., Bluetooth [3], the high bandwidth needed for TV quality video is not practical. For long-range, point-to-point, two-way wireless video communication, it is even less feasible to dedicate TV-like bandwidth to masses of individual consumers. Current wireless communicators support voice coders (vocoders) operating at much lower bit rates, e.g., 8 to 13 kbps.

▲ *Fig. 1. This Frank R. Paul illustration, circa 1911, depicts video communication in 2660.*

As wireless telephony becomes commonplace, more and more features are being supported on wireless networks [30]. By the year 2001 [31], [8], third-generation wireless standards will enable multimedia communication, supporting outdoor bit rates ranging from 144 to 384 kbps and indoor bit rates up to 2 Mbps. The specification for multimedia communication over third-generation mobile networks is being finalized under the auspices of the "Third-generation partnership project" (3GPP) [1]. 3GPP is a global initiative of several organizations, including the European Telecommunications Standards Institute (ETSI), the American National Standards Institute (ANSI) committee T1, the Japanese Telecommunication Technology Committee

1053-5888/00/$10.00©2000IEEE

(TTC), and Association of Radio Industries and Businesses (ARIB). Multimedia standards such as MPEG-4 (Motion Pictures Expert Group) [29] and H.324 [24] have also been recently completed just as the required processing power is becoming affordable. These standards are compatible with the 3GPP standard. A few wireless multimedia communicators are already available in some markets.

As more network bandwidth becomes available, advances in digital compression technology and use of lower-resolution video formats reduce the bandwidth requirements. The small size of wireless communicators typically limits the display size available for video, but even a small display is adequate for many applications. For instance, a sub-quarter common intermediate format (SQCIF) display (128 × 96 pixels) can be useful for videoconferencing, surveillance, news, or entertainment. A small format also requires less bandwidth and memory (18 kbytes per frame for 4:2:0 SQCIF), and coding artifacts may be less objectionable. Even for small-resolution video, bandwidth requirements for transmission of *raw* video data are prohibitive (about 4 Mb/s for SQCIF video at 30 fps). Hence the video data has to be compressed before it can be transmitted over wireless channels. The standard video compression algorithms [33] (e.g., H.263, MPEG) use motion vectors between frames to encode temporal redundancy, and the energy-compacting discrete cosine transform (DCT) to encode spatial redundancy, followed by entropy encoding. Predictive coding and variable-length code words are used to obtain a large amount of compression. However, a number of issues arise as a result of the digital compression and decompression.

Processing digital video requires a significant amount of memory, computation, and internal data transfer, all of which impact the price and the battery life of a wireless communicator. Furthermore, highly compressed video may contain visible artifacts. Because compression removes redundancy from the video data, compressed data is more sensitive to channel interference. Predictive coding causes errors in the reconstructed video to propagate in time to future frames of video, and the variable-length code words cause the decoder to easily lose synchronization with the encoder in the presence of bit errors. Rapid progress is being made in all of these areas simultaneously.

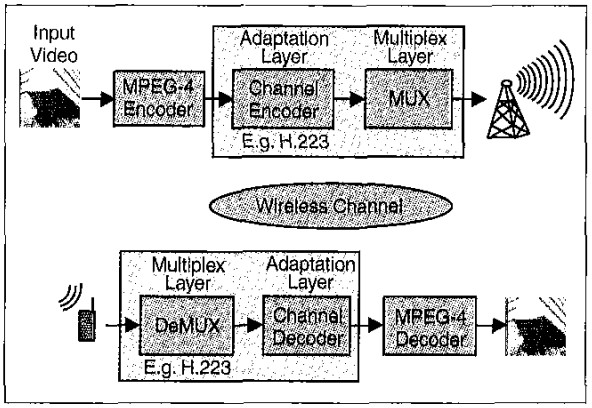
To make the compressed bitstream more robust to channel errors, the MPEG-4 video compression standard incorporated several error resilience tools in its *simple* profile to enable detection, containment, and concealment of errors. These are powerful source-coding techniques for combating bit errors when they occur at rates less than 10^{-3}; however, present-day wireless channels can have much higher bit error rates (BERs). The harsh conditions on mobile wireless channels result from multipath fading due to motion between the transmitter and the receiver, and changes in the surrounding terrain. Multipath fading manifests itself in the form of long bursts of errors. Hence, some form of interleaving and channel coding is required to improve the channel conditions. Using a combination of source and channel coding, it is possible to achieve acceptable visual quality over error-prone wireless channels with MPEG-4 simple-profile video compression. The structure of an MPEG-4 compressed bitstream also lends itself to using unequal error protection, a form of joint source-channel coding, to ensure fewer errors in the important portions of the bitstream.

Fig. 2 depicts wireless video communication between two MPEG-4 terminals using an error-resilient H.223 multiplexer [22]. Such a system can be cost-effectively implemented using digital signal processing (DSP) chips, which are microprocessors that are tailored to implement signal processing tasks efficiently [10], [26], [27].

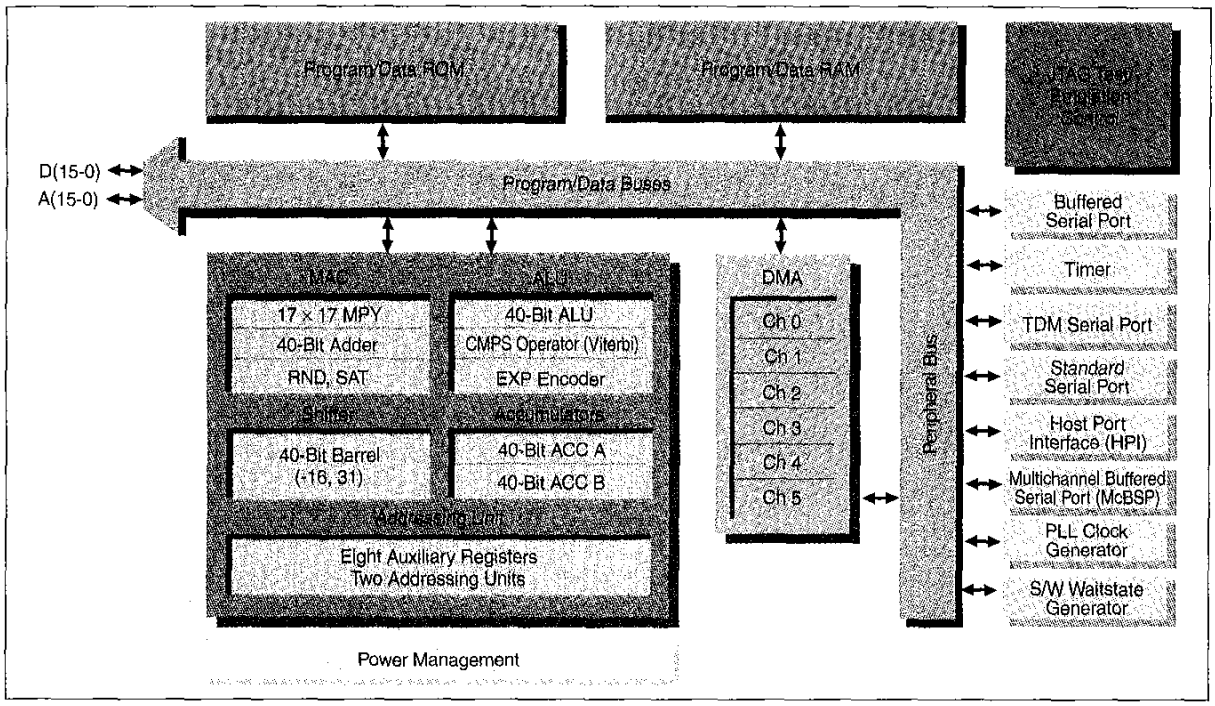
Features that make DSP-based platforms ideally suited for implementing wireless video communicators include:

▲ Low power consumption
▲ Viterbi [13] accelerators for channel coding
▲ Single-cycle multiply and multiply-accumulate for fast calculation of transforms (DCT and inverse DCT) and quantization
▲ Barrel shifters and bit-manipulation support for efficient coding/decoding of variable-length codewords
▲ Various memory access modes for efficient motion estimation/compensation and data transfer

Whether alone, or in combination with specialized coprocessors, the programmable nature of DSPs makes it economical to add upgrades and new applications with rapid time-to-market. With a programmable approach, various algorithmic tradeoffs can be made, based on processing needs and capability. Software can often be reused as faster DSPs become available. Applications that require significant computation, yet must be affordable to



▲ *Fig. 2. Wireless video terminals can implement source coding and channel coding on a DSP chip. The MPEG-4 video encoder is used to compress the input video data so that it can be transmitted over bandwidth-constrained mobile channels. The error conditions of the mobile channel may mandate the use of some form of channel coding to improve the channel conditions. The adaptation layer of H.223 provides a standard channel-coding tool. The multiplex layer of H.223 is used to multiplex video, audio, and data so that they can be played out synchronously at the receiver.*



▲ *Fig. 3. DSP architectures feature multiple functional units, multiple buses, and multiple memory banks, to support specialized single-cycle instructions with simultaneous multiple memory accesses (TMS32054x example).*

consumers, are good candidates for implementation on a DSP chip.

In this article, we discuss the design and implementation of wireless video communication systems on DSPs. We will cover both the video coding and the channel coding aspects of the problem. The emphasis of the article will be on highlighting the issues involved both from an algorithmic standpoint as well as from a DSP standpoint. We discuss the first DSP implementation, to the authors' knowledge, of the MPEG-4 simple profile video standard. In the next section, we give an overview of DSPs and highlight the salient features that make them especially well-suited for wireless applications. In "MPEG-4 Simple Profile Video Compression," we describe MPEG-4 simple profile video coding. In order to facilitate interoperability, it is important that wireless devices use standardized compression algorithms. In "Implementation Tradeoffs," we discuss our implementation of the MPEG-4 simple profile video codec on Texas Instruments' TMS320C54x DSP and discuss the implementation issues involved. We highlight some general and application-specific DSP instructions on the TMS320C54x DSP that enable us to implement the core operations in the video coder efficiently, often in a single cycle, with minimal control overhead. In "Performance," we describe the performance of MPEG-4 video compression for a variety of content and formats. In "Channel Coding and H.223" and "Unequal Error Protection," we give a brief description of channel coding with the H.223 standard. Then, we describe several channel coding experiments using both unequal and equal error protection on MPEG-4 video sent through a simulated GSM channel. Unequal error protection, which ensures fewer errors in the important portions of the MPEG-4 video bitstream, provides improved quality when compared to equal error protection under harsh error conditions.

DSP Processors

DSP processors differ significantly from general purpose processors, in architecture, instruction sets, power-efficiency, and cost [10], [26], [27]. Because DSPs are used for compute-intensive signal processing tasks, they have special instruction sets, with circuitry for single-cycle execution of common signal processing primitives, such as a multiply-accumulate (MAC). A Harvard memory architecture [10], [27] or variant [27], with multiple memory banks and buses, is used to access instructions and data fast enough to supply the processing units (see Fig. 3). While DSP implementations offer the flexibility of programmability, the capabilities should be matched to the needs of the consumer product, to avoid paying for unneeded features. Therefore, manufacturers offer *families* of DSP chips with various configurations of memory, various power requirements, and various clock speeds.

These real-time processors make up the fastest growing segment of the semiconductor market, with revenues expected to reach $10 billion by 2001 [37]. In this highly competitive market, new DSP chips are announced almost every month. It is beyond the scope of this article to catalog all available DSP processors. Undoubtedly, new DSP chips will be announced for third-generation wire-

less communication, offering low power dissipation, high performance, and cost effectiveness to meet the needs of a variety of terminal and infrastructure communications equipment. Examples using the TMS320C54x, from Texas Instruments' low-power 'C5000 family of DSPs, are familiar to the authors and are used to illustrate various DSP features. However, the reader is advised to contact manufacturers for current, complete information on available DSP processors [10], [27].

Architecture

DSP architectures can be characterized broadly in terms of data path, program control, and instruction set. For higher memory bandwidth, architectures may use multiple memory banks or multiported memories. Register-indirect addressing modes are used to avoid including memory addresses in the instruction, which would increase the code size and the number of cycles to fetch instructions from program memory. DSPs also support memory-to-memory data transfers using direct memory access (DMA). Support for zero-overhead loops and conditional execution is common for pipelined architectures, to avoid wasting clock cycles flushing data from the processing pipeline. Typically, DSP instruction sets support a single-cycle multiply-accumulate (MAC) instruction, or even multiple simultaneous MACs, to implement FIR filtering and correlation efficiently. Other specialized instructions may be included in the instruction set. DSP architectures are innovatively designed to maximize throughput when processing data for a variety of target applications.

For example, the TMS320C54x architecture reduces Viterbi "butterfly update" [13] operations down to only four instruction cycles for GSM channel decoding. Its other key features include:

▲ Multibus architecture with one program bus, three separate data buses, and four address buses, for efficient data access

▲ A 40-Bit Arithmetic Logic Unit (ALU) including a 40-bit barrel shifter and two independent 40-bit accumulators, for single-cycle instructions with shifting

▲ A 17- × 17-bit parallel multiplier coupled to a 40-bit dedicated adder for non-pipelined single-cycle MAC operation

▲ An exponent encoder to compute the exponent of a 40-bit accumulator value in a single cycle for data normalization and bit manipulation

▲ Two address generators with eight auxiliary registers and two auxiliary register arithmetic units which facilitate multiple data operand operations

For wireless applications, in addition to having specialized instructions, DSP architectures are further constrained to be size-, cost-, and power-efficient. Although the cost of programming a fixed-point processor is greater than a floating-point processor, fixed-point processors are exclusively used in wireless networks because of the reduced cost, size, and power. Fixed-point software development costs become negligible when economy of scale is considered. 16-bit fixed-point is sufficient for video processing on byte data, which has little need for fractional arithmetic.

Low Power

A key limitation to wireless communications is power consumption. In every DSP cycle, power is dissipated charging or discharging capacitors in the circuitry. Thus, it is important for low-power chip designers to reduce the number of transistors (gate count) wherever possible. Because capacitance is proportional to the dielectric constant K, DSP manufacturers are also working on finding a lower-K material to replace silicon dioxide as the interlayer dielectric for on-chip interconnect [19]. For CMOS technology, power is proportional to the product of the chip's effective capacitance, clock frequency, and the square of its supply voltage [$P \alpha C \times f \times V_{dd}^2$]. Thus, it is a challenge to keep DSP power low while adding features and increasing clock frequency for more processing performance.

Low-power DSPs include IDLE, or power-down, modes. Because the design of the instruction set determines how many cycles are required to perform a task, it also impacts the power consumption. When a DSP task is done more efficiently, the DSP chip can spend more time in IDLE mode. In the IDLE modes, the DSP chip enters a dormant state and dissipates considerably less power than in normal operation.

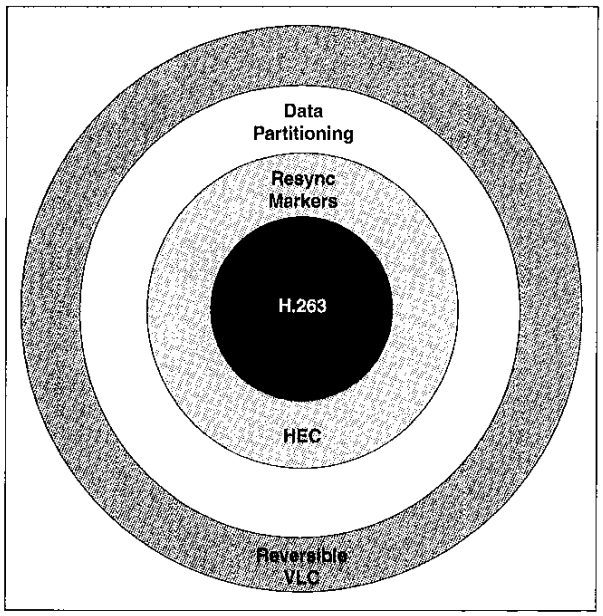
At the system level, power is reduced by integrating more functions onto a DSP chip, implementing tasks normally handled by off-chip ASICs or microcontrollers. The programmable nature of the DSP makes it possible to reload the processor code as needed, which becomes increasingly important for multimedia applications and multi-functional devices. This integration not only yields power savings, but space savings as well.

Processing Performance

Processing performance depends on clock rate, instruction set/architecture, and memory management. On-chip memory provides faster access, compared to off-chip memory, but costs more. For video applications, frames of data are typically stored off-chip. DSP chips with DMA support transferring blocks of data between off-chip and on-chip memory without interfering with processing.

The clock rate of a DSP is limited by the time constant to charge and discharge capacitance, and the propagation time for data to ripple through a series of logic gates. The time constant (along with the power dissipation) can be reduced by using lower-K materials, as mentioned earlier. Propagation delay is reduced by having fewer gates closer together. Closer spacing (and smaller chip size) has been achieved in the past by shrinking the interconnect line-width, but there can be problems with electromigration if the width becomes

too small. Some DSP manufacturers are investigating different materials for interconnect lines to further narrow the line-width while avoiding electromigration, to produce higher clock rates [19].

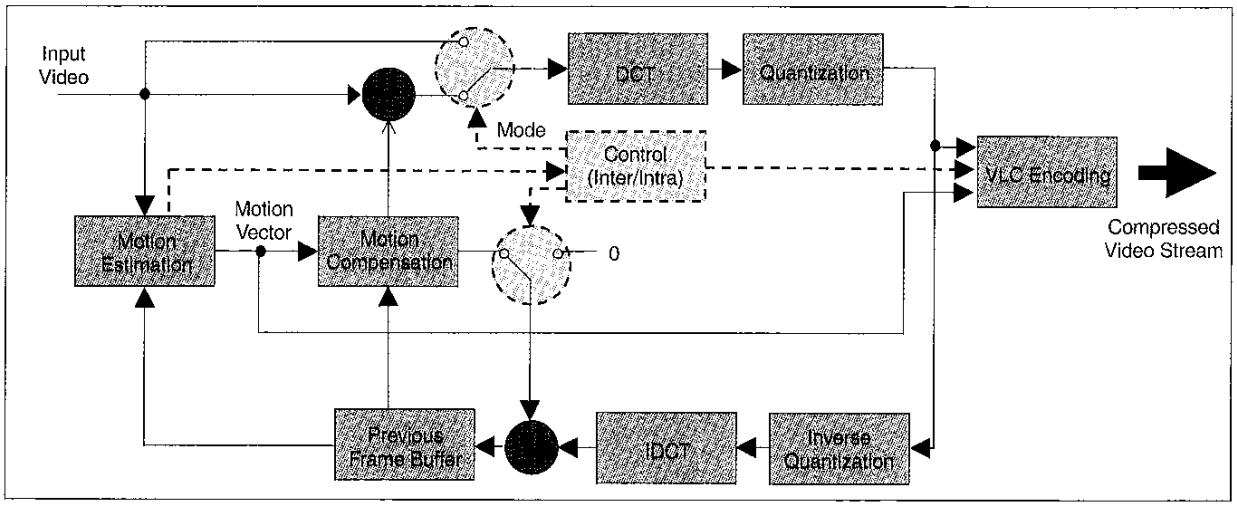


▲ *Fig. 4. The MPEG-4 simple profile includes error-resilience tools for wireless applications. The core of the MPEG-4 simple profile is the H.263 coder. Resynchronization markers and header extension code (HEC) provide the first level of error resilience support. Data partitioning builds on top of resynchronization markers to provide added error resilience through containment and localization of errors. Reversible VLCs, which can be used only if data partitioning is used, provides the final level of error resilience support, adding further error localization.*

The most cost-effective solution is obtained by using a DSP chip that best matches the needs of the system. To support the processing and memory requirements of a variety of wireless multimedia terminals, manufacturers offer multiple memory, DMA, power, and MIPS options within a given DSP family. With a DSP implementation, devices can be easily upgraded, when needed, as a result of the programmable nature of the DSP processors and the code compatibility among DSPs in the same family.

MPEG-4 Simple Profile Video Compression

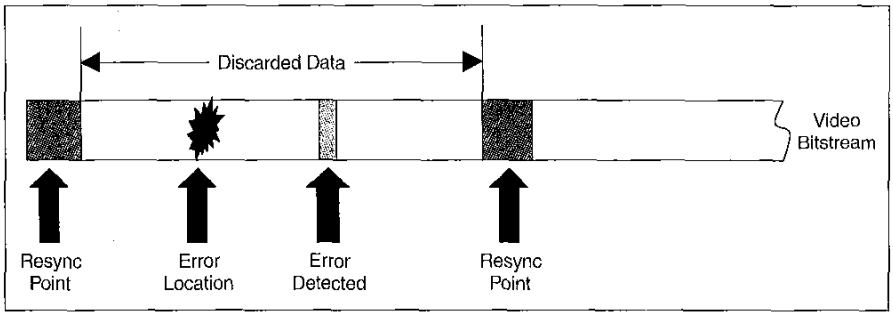
The MPEG-4 simple profile standard [29] uses compression techniques similar to H.263 [23], with some additional tools for error detection and recovery. The scope of the MPEG-4 simple profile is schematically shown in Fig. 4. Like H.263, video is encoded using a hybrid block motion compensation (BMC)/discrete cosine transform (DCT) technique. Fig. 5 illustrates a standard hybrid BMC/DCT video coder configuration. Pictures are coded in either *intraframe* (INTRA) or *interframe* (INTER) mode, and are called *I-frames* or *P-frames*, respectively. For intracoded I-frames, the video image is encoded without any relation to the previous image, whereas for intercoded P-frames, the current image is predicted from the previous reconstructed image using BMC, and the difference between the current image and the predicted image (referred to as the residual image) is encoded. The basic unit of information that is operated on is called a macroblock, and is the data (both luminance and chrominance) corresponding to a block of 16×16 pixels. Unlike previous MPEG video standards, there is no required pattern of I- and P-frame coding. Individual macroblocks within a P-frame can be coded in INTRA



▲ *Fig. 5. A standard video coder with block motion compensation, discrete cosine transform, and variable-length coding achieves high compression, leaving little redundancy in the bitstream. Input video macroblocks are coded in one of the two modes—inter or intra. Intercoding is typically used when there is sufficient correlation between adjacent frames and intracoding is used when there is not much correlation. The motion-estimation block feeds the correlation information to the control block, which then decides on the type of coding to be used. Though not shown in the diagram, the decision to use intracoding on error-prone channels also depends on the channel conditions because intracoding can stop error propagation.*

mode. All macroblocks must be INTRA-refreshed periodically to avoid the accumulation of numerical errors, but the INTRA refresh can be implemented asynchronously among macroblocks.

Motion information, in the form of motion vectors, is calculated for each macroblock in a P-frame. Note that MPEG-4 allows the motion vectors to have half-pixel resolution and also allows for four motion vectors per macroblock. Depending on the mode of coding used, the macroblocks of either the image or the residual image are split into 8×8 blocks, which are then transformed using the DCT. The resulting DCT coefficients are quantized, run-length encoded, and finally variable-length coded (VLC) before transmission. Since residual image blocks often have very few nonzero quantized DCT coefficients, this method of coding achieves efficient compression. Motion information is also transmitted for the intercoded macroblocks. Since a significant amount of correlation exists between neighboring macroblocks' motion vectors, the motion vectors are themselves predicted from already transmitted motion vectors, and the motion vector prediction error is encoded. The motion vector prediction error and the mode information are also variable-length coded before transmission to achieve efficient compression. In the decoder, the process described above is reversed to reconstruct the video signal. Each video frame is also reconstructed in the encoder, to mimic the decoder, and to use for motion estimation of the next frame.



▲ *Fig. 6. At the decoder, it is usually not possible to detect the error at the actual error occurrence location. Errors typically occur in bursts on wireless channels, corrupting many bits when the channel fades; hence, all the data between the two resynchronization points may need to be discarded.*

Due to the use of VLC, compressed video bitstreams are particularly sensitive to channel errors. In VLC, the boundary between code words is implicit. Transmission errors typically lead to an incorrect number of bits being used in VLC decoding, causing loss of synchronization with the encoder. Also, due to VLC, the location in the bitstream where the decoder detects an error is not the same as the location where the error has actually occurred. This is illustrated in Fig. 6. Once an error is detected, all the data between the resynchronization points are typically discarded. The error resilience tools in the MPEG-4 simple profile basically help in minimizing the amount of data that has to be discarded whenever errors are detected.

The error-resilience tools included in the simple profile to increase the error robustness [34], [5] are:

- ▲ Resynchronization markers
- ▲ Data partitioning
- ▲ Header extension codes (HEC)
- ▲ Reversible variable length codes (RVLC)

In addition to these tools, error concealment [38] should be implemented in the decoder. Also, the encoder can be implemented to limit error propagation using an adaptive INTRA refresh technique.

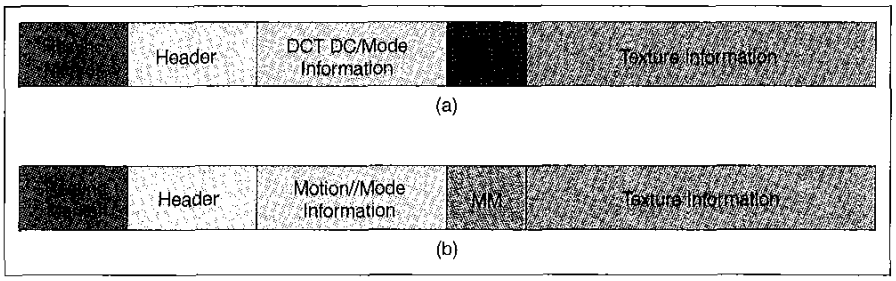
▲ *Fig. 7. Resynchronization markers help in localizing the effect of errors to an MPEG-4 video packet. The header of each video packet contains all the necessary information to decode the macroblock data in the packet.*

Resynchronization Markers

As mentioned earlier, a video decoder that is decoding a corrupted bitstream typically loses synchronization with the encoder due to the use of variable-length codes. MPEG-4 adopted a resynchronization strategy, proposed by Motorola, referred to as the "video-packet" approach. Packetization allows the receiver to resynchronize with the transmitter when a burst of errors (caused by fading due to movement and changes in topology) corrupts too much data in an individual packet.

A video packet consists of a resynchronization marker, a video-packet header, and macroblock data, as shown in Fig. 7. The resynchronization marker is a unique code, consisting of a sequence of zero-bits followed by a 1-bit, which cannot be emulated by the variable length codes used in MPEG-4. Whenever an error is detected in the bitstream, the video decoder jumps to the next resynchronization marker to establish synchronization with the encoder. The video-packet header contains information that helps restart the decoding process, such as the absolute macroblock number of the first macroblock in the video packet and the initial quantization parameter used to quantize the DCT coefficients in the packet. A third field, labeled HEC, is discussed in a later subsection. The macroblock data part of the video packet consists of the motion vectors, DCT coefficients, and mode information for the macroblocks contained in the video packet.

The predictive encoding methods are modified so that there is no data dependency between the video packets of a frame. Each video packet can be independently decoded irrespective of whether the other video packets of the frame are received correctly. A video packet always starts at a



▲ *Fig. 8. An MPEG-4 data partitioned video packet for (a) I-frames and (b) P-frames. Data partitioning uses additional markers (DCM and MM), and puts the most important information in the first partition of video packet, for better error concealment.*

macroblock boundary. The exact size of a video packet is not fixed by the MPEG-4 standard (the standard does specify the maximum size that a video packet can be); however it is recommended that the size of the video packets (and hence, the spacing between resynchronization markers) be approximately equal.

Data Partitioning

The data partitioning mode of MPEG-4, originally proposed by Texas Instruments, partitions the macroblock data within a video packet as shown in Fig. 8. For I-frames, the first part contains the coding mode and six DC DCT coefficients for each macroblock (four for luminance and two for chrominance) in the video packet, followed by a DC marker (DCM) to denote the end of the first part, as shown in Fig. 8(a). The second part contains the AC coefficients. The DCM is a 19-bit marker whose value is **110 1011 0000 0000 0001**. If only the AC coefficients are lost, the DC values can be used to partially reconstruct the blocks. For P-frames, the macroblock data is partitioned into a motion part and a texture part (DCT coefficients) separated by a unique motion marker (MM), as shown in Fig. 8(b). All the syntactic elements of the video packet that are required to decode motion related information are placed in the motion partition and all the remaining syntactic elements that relate to the DCT data are placed in the texture partition. The MM indicates to the decoder the end of the motion information and the beginning of the DCT information. The MM is a 17-bit marker whose value is **1 1111 0000 0000 0001**. If only the texture information is lost, data partitioning allows the use of motion information to conceal errors in a more effective manner. Thus, data partitioning provides a mechanism to recover more data from a corrupted video packet.

The DCM and MM were designed by Texas Instruments such that the marker bit patterns cannot be emulated by any combination of code words in the first partition. For instance, the MM was computed from the motion VLC tables using a search program so that it is Hamming distance 1 from any possible valid combination of the motion VLC table entries [35]. Note that the DCM and MM were only computed once based on the VLC tables, and are fixed in the standard.

Reversible Variable Length Codes (RVLCs)

Reversible VLCs, proposed by Toshiba, can be used with data partitioning to recover more DCT data from a corrupted texture partition. Reversible VLCs are designed such that they can be decoded both in the forward and the backward direction. MPEG-4 RVLC tables are designed from constant hamming-weight VLCs. Table 1 illustrates the idea behind the process. The first step in the process is the creation of a constant hamming-weight VLC. For a hamming-weight of one, we get a constant hamming-weight VLC as shown in column two of Table 1. Note that column two alone produces an ambiguous code book (101 could be 1,01 or 10,1). By adding a fixed

Table 1. Creation of RVLCs from Constant Hamming Weight VLCs.

Code Alphabet	*Constant Hamming-Weight VLC*	*RVLC*
1	1	111
2	01	1011
3	10	1101
4	001	10011
5	010	10101
6	100	11001

A fixed prefix (1 in this case) and a fixed suffix (1 in this case) are added to constant-weight VLCs to obtain a Reversible VLC. Decoding in both forward and backward directions can be achieved by searching for the third 1 in the RVLC codeword.

prefix and suffix of one, we get the RVLC as given in column three of Table 1. RVLC decoding can be achieved in both forward and backward directions by searching for the hamming-weight number of ones in the RVLC code word (three in the case of Table 1). Note that complements of such RVLC code words are also RVLC code words. MPEG-4 RVLCs make use of this technique, and also append an additional 2-bit suffix to finally form the RVLC code. The additional suffix contains the sign bit, and another bit that effectively doubles the size of the code book, yet limits the maximum code word size to 16 bits (which is good for DSP implementation). The fixed-length suffix can also be reverse decoded. Thus, the data in the texture partition can be parsed in either the forward or backward direction.

Fig. 9 illustrates the steps involved in two-way RVLC decoding in the presence of errors. While decoding the bitstream in the forward direction, if the decoder detects an error it can jump to the next resynchronization marker and start decoding the bitstream in the backward direction until it encounters an error. Based on the two error locations, the decoder can recover some of the data that would have otherwise been discarded. Because the error may not be detected as soon as it occurs, the decoder may conservatively discard additional bits around the corrupted region. Note that if RVLCs were not used, more data in the texture part of the video packet would have to be discarded. Thus, RVLCs enable the decoder to better isolate the error location in the bitstream.

Header Extension Code (HEC)

Important information that remains constant over a video frame, such as the spatial dimensions of the video data, the time stamps associated with the decoding and the presentation of this video data, and the type of the current frame (INTER-coded/INTRA-coded), are transmitted in the header at the beginning of the video frame data. If some of this information is corrupted due to channel errors, the decoder has no other recourse but to discard all the information belonging to the current video frame. In order to reduce the sensitivity of this data, a 1-bit field called HEC was proposed by Matsushita and introduced in the video packet header. When the HEC is set, the important header information that describes the video frame is repeated in the bits following the HEC. This duplicate information can be used to verify and correct the header information of the video frame. The use of HEC significantly reduces the number of discarded video frames and helps achieve a higher overall decoded video quality.

Error Concealment

The MPEG-4 standard does not specify what action the decoder should take when an error is detected. Several error-concealment techniques have been developed based on temporal, spatial, or frequency-domain prediction of the lost data [38]. The simplest form of temporal error concealment is to copy the lost data from the previous frame. Sometimes the missing motion vector can be predicted from neighboring macroblocks, or the motion vector may not have been lost if data-partitioning tools are used. However, temporal concealment cannot be used for the first frame, and may yield poor results for intracoded macroblocks or areas of high motion. Concealment in the spatial domain involves more computation for interpolation. In some cases, frequency-domain interpolation may be more convenient, by estimating the DC value and possibly some low-order AC DCT coefficients.

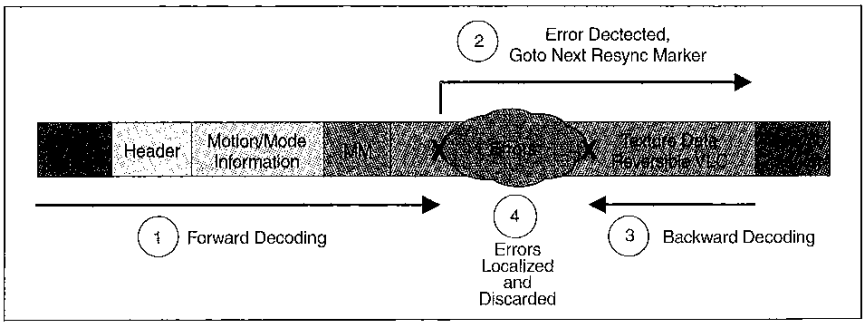
Adaptive Intra Refresh (AIR)

AIR is a standard-compatible encoder technique for limiting error propagation by using non-predictive INTRA coding. INTRA refresh forcefully encodes some macroblocks in INTRA mode to flush out possible errors. INTRA refresh is very effective in stopping the propagation of errors, but it comes at the cost of a large overhead; coding a macroblock in INTRA mode typically requires many more bits than coding in INTER mode. Hence, the INTRA refresh technique has to be used judiciously.

AIR adaptively performs INTRA refresh based on the motion in the scene. For areas with low motion, simple temporal error concealment works quite effectively. Since the high-motion areas can propagate errors to many macroblocks, any persistent error in the high-motion area becomes very noticeable. The AIR technique of MPEG-4 [21] INTRA refreshes the motion areas more frequently, thereby allowing the possibly corrupted high-motion areas to recover quickly from errors.

Implementation Tradeoffs

Careful attention to DSP implementation details is required to fully achieve the potential gains in performance, flexibility, and cost [15]. For best performance, memory allocation, data transfer, and ordering of instructions must be matched to the DSP architecture. Some general



▲ *Fig. 9. Reversible VLCs can be parsed in both the forward and backward direction, making it possible to recover more DCT data from a corrupted texture partition.*

Table 2. Some Single-Cycle (^) DSP Instructions Are Useful for Video Kernels ('C54x Examples).		
Syntax	*Expression*	*Useful for:*
ABDST *Xmem*, *Ymem*	B = B+|A(32-16)| A = (Xmem-Ymem) << 16	Motion estimation
ADD *Xmem*, *Ymem*, *dst*	dst = Xmem<<16 + Ymem << 16	Motion comp, DCT,IDCT
SUB *Xmem*, *Ymem*, *dst*	dst = Xmem<<16 - Ymem << 16	DCT/IDCT
MPY *Xmem*, *Ymem*, *dst*	dst = Xmem*Ymem, T=Xmem	DCT/IDCT
MAC *Xmem*, *Ymem*, *src* [,*dst*]	dst = src + Xmem*Ymem, T=Xmem	DCT/IDCT
MAS *Xmem*, *Ymem*, *src* [,*dst*]	dst = src - Xmem*Ymem, T=Xmem	DCT/IDCT
STH *src* [, *SHFT*], *Xmem*	Xmem = src << (SHFT-16)	Half-pixel interpolation, DCT/IDCT
EXP *src*	T = number of sign bits (src) - 8	VLD
(^) Single cycle counts assumes the use of dual-access RAM for data.		

issues involved in the implementation of video coding on DSPs are:

▲ Memory allocation: On most DSPs, on-chip memory is limited due to cost and power constraints. The mapping of variables to on-chip and off-chip memory has a significant impact on performance. On-chip memory provides faster access rates when compared to off-chip memory. The on-chip memory is used to store variables that are accessed frequently and repeatedly, e.g., the VLD tables, the motion vectors, the bitstream buffers, buffers for storing the inverse quantized DCT coefficients, buffers for storing the block of data to be used in motion compensation, and other intermediate variables. The reference frame and the current decoded frame are stored off-chip. To reduce the amount of on-chip memory required, decoding is typically carried out on a macroblock basis.

▲ Data transfer: Management of data transfer from off-chip to on-chip memory is another major issue in implementing video coding on DSPs. Most of the low-power DSPs are 16-bit DSPs and rarely provide support for byte access. Since accesses to memory outside the chip are slower and also consume more power, it might make sense to have the image data packed in external memory with two pixels per 16-bit memory location. However, this impacts the performance, as the pixels have to be unpacked first before they can be used.

▲ "DSP-friendly" algorithms: Video processing blocks such as IDCT can be implemented in a variety of ways. Each implementation varies in the number of operations required, the data flow involved, and the precision requirements on the intermediate variables. Often algorithms that have a regular data flow and whose precision requirements on the intermediate variables match the word-length of the DSPs, execute faster on the DSPs. For example, Chen's algorithm [7] for IDCT, which requires 16 multiplications and 26 additions, requires more DSP cycles to execute when compared to the even/odd decomposition algorithm presented in [32] that requires 20 multiplications and 28 additions. This is because Chen's algorithm requires a double-precision 16- × 32-bit multiplication, whereas the simpler even/odd decomposition algorithm requires only 16- × 16-bit multiplications to meet the IDCT precision requirements [20]. A 16- × 16-bit multiplication can be efficiently implemented on a 16-bit DSP in one cycle, whereas a 16- × 32-bit multiplication would have to be emulated using multiple cycles on the DSP.

▲ Development tools: The software tools available for fixed-point DSP code development have improved markedly from just a few years ago. C compilers are becoming more and more efficient; however, the current compiler technology is not intelligent enough to make use of the DSP resources in the most efficient manner. Thus, core kernels very often use optimized assembly library routines or must be hand-coded in assembly. Good development tools and libraries significantly reduce the time-to-market.

The following sections discuss in detail some implementation issues for implementing an MPEG-4 simple profile video codec. The standard does not in any way specify how the encoder should be implemented, other than that it must create a decodable bitstream. There is also significant latitude in some aspects of the decoder implementation. We start off with a discussion on the implementation of some core kernels of the video codec in the DSP assembly language to give an idea of the kind of support provided by the DSPs for such tasks.

Implementation of Video Kernels

Each of the standard video coding blocks (see Fig. 5) consists of a compute- and data-intensive kernel. In this section, we highlight some DSP instructions that help in implementing the kernels efficiently.

Motion Estimation

Motion estimation is carried out using block-matching techniques. The motion vector for a macroblock is the displacement between the macroblock being coded and the macroblock in the previous frame that best matches it. The matching criterion used is the sum of absolute differences (SAD) between all the pixels of the macroblock being coded and all the pixels of the macroblock in the previous frame. On the TMS320C54x DSP, the calculation of SAD, which is the core kernel in motion estimation, can be efficiently carried out using the ABDST instruction (see Table 2). An example usage of the instruction is

```
ABDST   *AR3+,  *AR4+
```

This instruction directs the TMS320C54x to:

▲ Add the absolute value of the Accumulator A to Accumulator B

▲ Subtract the contents of the memory location pointed to by AR3 from the contents of the memory location pointed to by AR4 and store the result in A

▲ Postincrement AR3 by 1 and postincrement AR4 by 1

By pointing AR3 to the macroblock being coded and AR4 to a macroblock of the previous frame, we can loop the ABDST instruction over all the pixels in the macroblock and obtain the SAD in Accumulator B at the end of the loop. By combining the absolute operation, differencing between two pixels, and calculating a running sum into a single instruction, the DSP basically executes the core operation in the calculation of SAD in one cycle instead of the three-to-four cycles it might take on a general purpose processor (GPP). Also note that the address postincrement operation, which is done in parallel while the instruction executes automatically, readies AR3 and AR4 to point to the next set of pixels to be differenced.

DCT/IDCT

DCT and IDCT are carried out on blocks of 8×8 pixels. DCT is a 2D transform, but since it is a separable transform, the 2D transform is typically implemented using 1D transforms on rows and columns. Many fast algorithms exist in the literature for implementing the DCT/IDCT, but care must be taken while implementing the algorithm to verify that they meet the precision requirements of the IEEE 1180 [20] standard. Traditionally, research in IDCT implementation has concentrated on reducing the number of multiplications involved often at the expense of a corresponding increase in the number of additions required. Since DSPs have a dedicated multiplier, reducing the number of multiplications involved in the algorithm is not the only consideration. Regularity of the data accesses also becomes an important consideration so that address pointer manipulations (which occur in parallel to the execution of instructions) supported by the DSP can be used. All the implementation algorithms basically consist of the traditional signal processing blocks of multiply, multiply-accumulate/subtract, add/subtract, and shifts, all of which can be efficiently implemented on the DSP, each in a single cycle. Table 2 lists the DSP instructions of TMS320C54x that perform these operations.

Traditionally, research in IDCT implementation has concentrated on reducing the number of multiplications involved often at the expense of a corresponding increase in the number of additions required.

Also involved with the IDCT is a saturation/clipping operation at the IDCT output, which clips the output to be in the range [-256,255]. By properly aligning the data, saturation can be done automatically whenever the IDCT output is stored. Since saturation operations have to be carried out on a per-pixel basis, automatic saturation becomes an important feature for the DSPs, as it usually takes up to eight instructions to do the same on a GPP.

Motion Compensation/Half-Pixel Interpolation

Motion compensation operations for a macroblock consist of adding the reconstructed residual error (output of IDCT) for the macroblock with its motion-shifted counterpart in the previous frame. Motion compensation is efficiently carried out using the ADD instruction that allows for parallel moves of two data operands.

Another kernel that is efficiently implemented using the ADD instruction is half-pixel interpolation. The basic operation involved in half-pixel interpolation is of the forms $(a+b+1)/2$ and $(a+b+c+d+2)/4$. Both of the division operations are integer division operations and can be efficiently implemented using right shifts. Shifting is another operation that can be done in parallel with store instructions on the TMS320C54x (see the STH instruction in Table 2), thus saving an additional instruction when compared to a GPP.

Variable-Length Decoding (VLD)

DCT run-level-last coded coefficients, motion vectors, and mode information are all variable-length coded. There are a number of ways to implement variable-length decoding in software [12]. Let d denote the length of the

RVLC code words, due to their structure, comprise a very sparse code book that cannot be decoded efficiently using the same table lookup techniques that are used for regular VLC codes.

longest code word in the discussions below. Since the length of the code word is not known in advance, the fastest way to decode a code word is by using a 2^d-element lookup table. d bits are read from the bitstream and are used directly to index into the lookup table. The output of the lookup table consists of the decoded symbol and the length of the code word. The bitstream pointer is advanced by the code word length, and the process is repeated on the remaining bits.

When d is on the order of 12 bits, as in the normal DCT VLC tables, the space required for the VLC lookup table becomes prohibitive. Also, such tables are wasteful because shorter codes have many repeated entries, e.g., with $d=12$, a code of length 4 bits will have 2^8 repeated entries. For most of the VLC codes used in the standards, there is a correlation between the length of the code word and the number of leading zeros in the code word. We make use of this structure in the code to reduce the memory requirements for the table [32], [36]. For example, the widely used Telenor R&D H.263 software [36] splits the DCT VLD tables into three different VLD tables based on the number of leading zeros, 0-2, 3-4, or 5-6 leading zeroes respectively.

By splitting into three classes, we require three lookup tables of sizes 2^7 each, instead of the 2^{12}-element lookup table required in the direct lookup scheme, leading to a savings in memory. More memory savings are obtained by splitting into more tables. However, the complexity involved in the decoding goes up with each additional table, since we now must first find out the class to which the code word belongs (a process which involves multiple if-then-else compare statements to find the number of leading zeros) before doing a table lookup. On the TMS320C54x, however, the calculation of the number of leading zeros in the code word is done in a single cycle using the EXP instruction (see Table 2), leading to a very efficient implementation. Since we get the number of leading zeros in the code word in a single cycle, we can afford to split the table into the maximum number possible. By doing so, we save about 50% on the memory required for the normal DCT, motion vector, and mode VLD tables when compared to the Telenor R&D H.263 software.

Implementation of the Decoder

The MPEG-4 simple profile requires that the decoder decode bitstreams with any of the error-resilience tools. The error detection logic is not specified in the standard, therefore leaving the choice to the implementer. However, Annex E of the MPEG-4 standard [29] provides a set of guidelines. The video coder can detect errors whenever illegal VLC code words are encountered in the bitstream or when decoding VLC code words leads to illegally decoded information (e.g., occurrence of more than 64 DCT coefficients for an 8×8 DCT block). Error concealment is also left to the implementer. Below are some issues involved in implementing the error-resilience features in an MPEG-4 simple profile decoder, with discussion of the tradeoffs involved.

▲ *Resynchronization markers (RS).* As previously discussed, the location in the bitstream where an error is detected is generally not the location where the error actually occurred due to the VLC. As a result, if an error occurs near the end of a video packet, it might not be detected until some bits from the next video packet are processed. If this occurs, then the RS at the start of the subsequent video packet is not detected, and the corresponding video packet is discarded even though it might have been received without errors. This has an adverse effect on the video quality, and may occur frequently, especially when the packet sizes are small. To prevent this from happening, the decoder could check for the RS before every bit is read. An alternate approach is to parse the bitstream twice, with the first parse delineating the video packet boundaries and the second parse doing the actual decoding. To support the two-parse approach, a bitstream buffer is required to store at least one video packet. The specification of the maximum packet size in the profile/level definition in MPEG-4 assists in bounding the memory requirement of this bitstream buffer, which must be stored in on-chip memory for faster memory access.

▲ *Data partitioning (DP).* With DP, an additional resynchronization point is added to the packet. The additional overhead required to implement data partitioning, when compared to the case when only RS is present, is that the decoder has to check for the motion marker after every motion vector has been decoded. Also, since the data for a macroblock is split across the motion and texture part of the video packet, the decoder can no longer do decoding on a macroblock basis. Since a video packet can contain data corresponding to an entire frame, enough memory should be reserved for storing the motion and mode information for the entire frame. Code size and complexity also increase for supporting data partitioning.

▲ *Reversible variable-length codes (RVLC).* With RVLC, the decoder has the option of two-way decoding DCT data, as shown in Fig. 9, if an error occurs in that partition of the packet. RVLC tables increase data memory requirements. RVLC code words, due to their structure, comprise a very sparse code book that cannot be decoded efficiently using the same table lookup techniques that are used for regular VLC codes. The MoMuSys [2] decoder implements RVLC decoding using a gigantic case statement, resulting in large code size and slow execution for

the worst case. The degree to which the decoder exploits RVLC data is left up to the implementer; decoders need not perform reverse decoding at all. Reverse decoding significantly increases the program complexity and the computational complexity in the presence of errors. Annex E of MPEG-4 [29] gives more details on the logic that can be used for forward and backward decoding.

▲ *Header extension code (HEC).* The header extension code is the one-bit flag in the video packet header, which when set to 1 indicates that additional information from the video frame header is repeated in the video packet following the HEC bit. The best performance in terms of error robustness is obtained when the HEC information is put in each video packet, but this increases the bits overhead. In practice, cross-checking the HEC information from all the video packets increases complexity since the actual decoding has to be delayed to find out which of the video packets do not contain errors, based on a majority vote. The way in which the HEC information is used in the decoder is left to the implementer. A simple scheme is to use the HEC information from the second video packet when the video frame header is corrupted.

Implementation of the Encoder

The standard provides many encoding options, but does not dictate which should be supported in the encoder, leaving implementers free to make tradeoffs between cost and performance. Unlike the decoder, not all features are required to be implemented, and error detection and correction are not needed. Since video is coded with respect to the previously reconstructed frame, implementation of the encoder includes many functions of the decoder, such as IDCT and inverse quantization. Therefore, the encoder generally requires both more memory and more computation than the decoder.

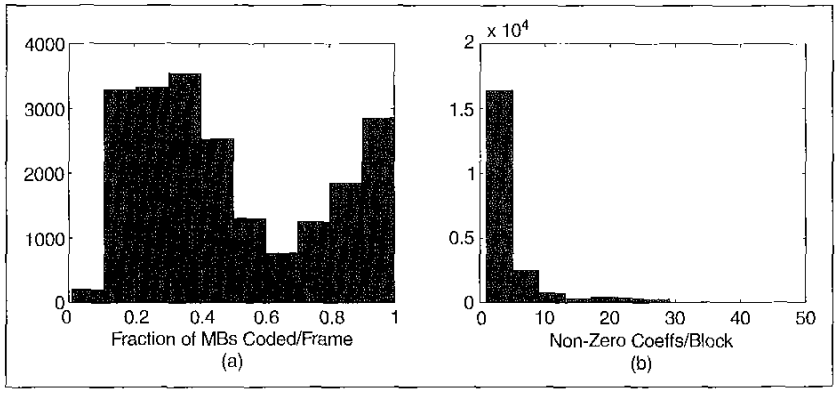
Motion estimation is particularly expensive, in terms of memory, data transfer, and computation. The public-domain tmn-2.0 encoder [36] reduces computation for half-pixel motion search by storing the entire reference frame interpolated horizontally and vertically, but this requires as much memory as four frames. This is unacceptable for a DSP implementation, because it more than doubles the memory requirement. While half-pixel refinement is not required for the encoder to be standard-compliant, it is necessary to obtain sufficient quality to be competitive at low bit rates. A possible alternative is to interpolate smaller portions of the reference frame at a time, though some values would have to be computed multiple times where the reference "windows" overlap. For minimal storage, another alternative is to store no interpolated data, but instead always recompute interpolated values when needed. Thus, there is a tradeoff between the amount of storage and computation required for half-pixel refinement.

Besides the memory requirement, motion estimation requires significant computation and data transfers. Unlike previous standards, which typically restrict the search window, the MPEG-4 standard supports unrestricted motion vectors (UMV) and includes an *fcode* for scaling motion vectors, such that a motion vector may reference any part of the previous frame. Performing block matching against the entire previous frame significantly increases the amount of computation and memory accesses for the encoder, although the encoder implementation is free to search a smaller region. Extensive research has been performed investigating computationally efficient motion-estimation algorithms [9].

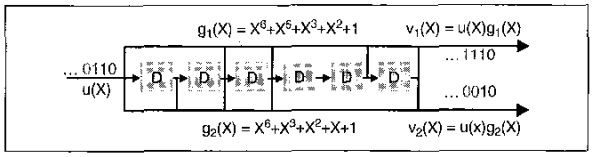
Rate control is another function implemented in the encoder, but not in the decoder. Though the computational and memory requirements may be minimal, these algorithms may be sensitive to reduced precision with 16-bit integers.

Among the error-resilience tools, data partitioning is the most cumbersome to implement in the encoder, because it essentially requires writing the bitstream twice. Rather than putting all bits representing a macroblock together in the bitstream, multiple substreams are created, which are copied into the final bitstream only after enough macroblocks have been coded to achieve the minimum packet size. This requires buffering of the substreams. Also, the byte alignment is not the same for the substreams and the final bitstream, so copying must be performed bit by bit.

Other error-resilience tools are fairly simple to implement in the encoder. Although resynchronization markers require keeping a count of the bits in a packet, this information is already being collected for the purposes of rate control; the only extra computation required is checking the bit count after coding each macroblock. For the en-



▲ *Fig. 10. Encoding over 160 bitstreams resulted in a wide range of coding complexity, depending on content, as shown by these histograms of (a) the fraction of MBs coded per frame and (b) the number of non-zero DCT coefficients per coded 8 × 8 block. The Y-axis in both the histograms denotes the number of macroblocks.*



▲ *Fig. 11. To achieve a rate-$\frac{1}{n}$ convolutional code, the source bits, u(X), are convolved with n generator functions, g1(X),..., gn(X). This figure shows a six-memory (64 state) rate-$\frac{1}{2}$ encoder. An RCPC code is obtained by puncturing the output of a rate-$\frac{1}{n}$ code. For example, to get a rate-$\frac{2}{3}$ code from the above rate-$\frac{1}{2}$ convolutional coder, the output is punctured as follows: ...0110 → ...0100110 → ...010110 where the crossed-out bits denote the punctured or discarded bits.*

coder, support for RVLC does not affect the amount of computation, but increases the memory requirements for the RVLC tables. Of course, an encoder implementation may omit any error-resilience tools, at the expense of degraded quality in error-prone environments.

Performance

The computational requirements for decoding MPEG-4 simple profile video varies from 10s of 'C54x MIPS to 100s of 'C54x MIPS, depending on the format, frame rate, bit rate, and content. Encoder computational complexity is roughly twice that of the decoder for low-complexity motion estimation with minimal overhead for data transfers. We encoded over 160 bitstreams, with a variety of formats, frame rates, bit rates, content, and using some, all, or none of the error-resilience tools. Fig. 10 shows histograms of the number of coded MBs per frame, and the number of nonzero coefficients per 8×8 block. The shape of the distribution varies depending on the choice of test sequences and the rate control algorithm of the encoder. The wide range of values in the histograms corresponds to the range of MIPS required.

As a proof of concept for wireless video, we ported a simple-profile SQCIF encoder and decoder to a 40 MHz TMS320C541. About half of all cell phones currently have a 'C54x inside. To the authors' knowledge, this is the first DSP implementation of the MPEG-4 simple profile standard. All error resilience tools have been implemented. With only compiled C code, the encoder can encode about one SQCIF frame per second, including overhead for data transfers. Encoding time with resynchronization markers is similar to H.263, and encoding with RVLC is about the same complexity as encoding with data partitioning. While data partitioning does increase encoder complexity, it is difficult to quantify based on C code results. By replacing critical sections of code with assembly routines, it is reasonable to expect the performance to improve. The decoder can decode about 20 SQCIF frames per second for a talking-head sequence without reverse decoding of RVLC codes. Reverse decoding for error recovery will slow decoder execution, but is not required. More powerful DSPs are available and are required for implementing both the encoder and decoder, or to support the common intermediate format (CIF) (352×288 pixels) or quarter CIF (QCIF) (176×144 pixels) and higher frame rates. Additional performance gains can be achieved with coprocessors, more efficient data transfer (e.g., DMA), and hand-coded assembly.

In addition to porting the code to a DSP, ETSI library routines were used to measure the decoding complexity for the 160+ bitstreams. SQCIF sequences were created by cropping QCIF sequences. Because this usually eliminated static background blocks for the most part, the same bit rate was used for encoding SQCIF and QCIF versions. The peak signal-to-noise ratio (PSNR) statistics were actually higher for low-motion QCIF sequences, compared to the SQCIF equivalent, but this is misleading because the higher PSNR was due to more background blocks. In high-motion sequences, the PSNR statistics were higher for the SQCIF sequences, as would be expected, given the same bit rate for a smaller format. Decoding SQCIF requires fewer MIPS than QCIF (between 15% and 50% less), and about half as much memory. The SQCIF format was found to be almost as useful as QCIF for low-motion sequences.

Higher bit rates were required to maintain acceptable quality for sequences in CIF format. QCIF sequences were created by downsampling CIF data. This resulted in more spatial detail per QCIF macroblock, and more nonzero DCT coefficients. Also, because the QCIF format has fewer motion vectors per frame than CIF, motion compensation is not as effective in representing frame differences. Thus, complexity does not scale linearly with the number of macroblocks.

QCIF and CIF sequences were encoded at both 10 and 15 frames per second. Most sequences could be coded at the same bit rate for either frame rate, with PSNR dropping less than 1 dB at the higher frame rate. However, for one sequence with sign language, the PSNR was actually higher at 15 fps, presumably because motion compensation was more effective at a higher frame rate. The MIPS requirement for 15 fps increased 15% to 50% over the MIPS required for 10 fps.

The bitstreams with error-resilience options incurred only a slight PSNR penalty. The MIPS requirement for parsing increased somewhat (up to 30%), but cycles for IDCT actually decreased, possibly reflecting that fewer bits were used for DCT coefficients as more bits were used for markers.

Channel Coding and H.223

Although the wireless network internally provides some level of channel coding, this may not be adequate to ensure good quality of video bitstreams. Because encoded video bitstreams are particularly sensitive to errors, and video decoders cannot tolerate the delay for retransmission of corrupted data, it may be necessary to increase the redundancy in the bitstream using channel coding. Er-

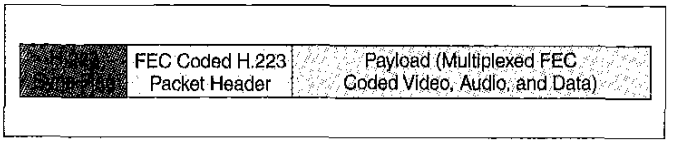
ror-correction techniques, such as forward error correction (FEC) [28], can be used to reduce the number of errors in the bitstream at the cost of increased overhead. One method of channel coding is convolutional encoding, where, in general, the input bit sequence is convolved with n generator functions to produce a rate-$\frac{k}{n}$ code, i.e., for every k source bits, there are n output bits with $k \le n$. Fig. 11 shows a rate-$\frac{1}{2}$ coder. The added redundancy is used at the decoder to detect and correct a certain number of errors. Viterbi decoding, a maximum a-posteriori decoding method, can be used to decode convolutional codes. A rate-$\frac{a}{b}$ coder can be achieved by puncturing, or discarding, the output bits from a rate-$\frac{1}{n}$ code. For every a input bits to the rate-$\frac{1}{n}$ coder, $(n \times a - b)$ of these bits are discarded. The remaining b bits are sent as the channel coded signal. Using punctured coding, many different rates can be achieved using the same generator functions, so the program complexity of the channel coder does not increase as the rates are changed. Rate-compatible punctured convolutional (RCPC) [16] encoding is a special type of puncturing where higher rate codes are subsets of lower rate codes. Various optimal puncturing tables have been created for different rate-$\frac{1}{n}$ codes [16].

As part of the H.223 multiplex standard [22], an adaptation layer may be used to provide additional protection from channel errors, beyond the level of service available from the network provider. The adaptation layer of the H.223 standard provides support for FEC using RCPC encoding of the data (H.223 uses a different kind of convolutional code than the one shown in Fig. 11). The amount of protection can be set based on the channel conditions and the amount of allowed overhead to bring the aggregate bit error rate down to a level at which the MPEG-4 error resilience tools can be effective and provide acceptable quality at the decoder.

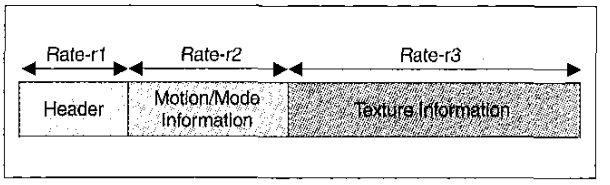
The FEC coded video data from the adaptation layer are sent to the multiplex layer, as shown in Fig. 2. The multiplex layer performs multiplexing of the video, audio, and data streams. In addition, the multiplex layer adds a resynchronization flag and a header to the multiplexed data (the payload). This flag is chosen so that it has good auto-correlation properties and has low cross-correlation with the data in the payload. Detection of the resynchronization flag is done at the H.223 decoder using correlation and thresholding. This allows a high degree of detection and a low degree of false detection in the presence of channel errors. The header added by the H.223 multiplex layer contains the length of the payload and a code into a multiplex table, which tells the decoder how to demultiplex the video, audio, and data. This header is protected using an extended Golay error correction code. Fig. 12 shows the structure of an H.223 packet.

The H.223 packets are sent over a wireless channel, such as a GSM or DECT (Digital European Cordless Telecommunications) channel. These are bandwidth-constrained, error-prone channels. At the receiver, the (possibly corrupted) packets are demultiplexed and FEC decoded using the multiplex and adaptation layers of H.223, respectively. The FEC decoding is performed using a maximum a-posteriori Viterbi decoder. The 'C54x family of chips has an integrated "Viterbi accelerator," a dedicated instruction set which can efficiently perform this computationally intensive operation [18].

The error-corrected video bitstream is sent to the source decoder. Since the bitstream may contain some residual errors, the video decoder must be error-robust. The video decoder implementation must detect errors, recover as quickly as possible, and perform error concealment.



▲ *Fig. 12. H.223, a multiplexing protocol for low-bit-rate multimedia communication, supports channel coding in the adaptation layer, before video, audio, and data streams are multiplexed to form the payload of an H.223 packet. A synchronization flag and packet header further protect the packet against channel errors.*



▲ *Fig. 13. Unequal error protection applied to an MPEG-4 packet. Rate-r1 is less than rate-r2, which is less than rate-r3 in order to provide the most protection to the header, the next level of protection to the motion information, and the least protection to the texture information. This assures that fewer errors will corrupt the important sections of the video packet.*

Unequal Error Protection

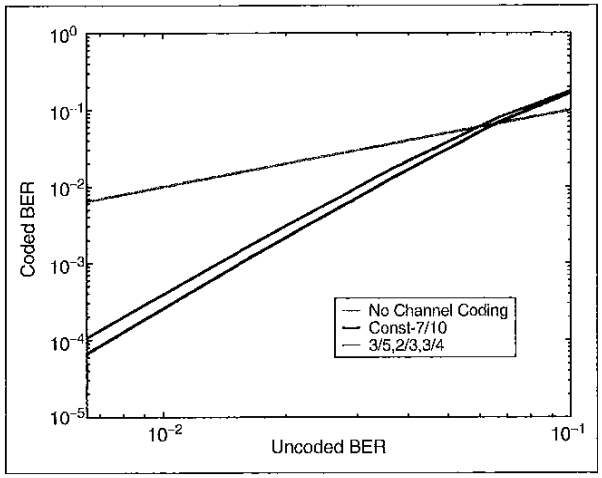
The amount of channel coding added to the data depends on the bandwidth requirements of the channel and the expected amount of distortion. If too much error protection is added to the data, it is a waste of bandwidth that could be spent on representing the data itself. If too little error protection is added to the data, the corruption might render the data completely unusable. Thus, there is a tradeoff between the amount of bandwidth that is allocated to the data and the amount that is allocated to the error protection. When data is compressed, it is very sensitive to errors; since each bit contains a large amount of information, it is especially important to protect highly compressed data. The structure of MPEG-4 compressed video can be exploited using unequal error protection to achieve the highest quality reconstructed video for a fixed channel-coding overhead cost. When using unequal error protection, the header gets the most protection since it contains the most important bits of the video packet. The

Multimedia communication requires significantly more bandwidth than speech alone, but third-generation wireless standards, combined with new video compression standards such as MPEG-4, will provide sufficient bandwidth to support many types of video applications.

motion bits would get the next highest level of protection, and the texture bits would receive the lowest level of protection, since without the texture information, the decoder can still perform motion-compensated concealment without too much degradation of the reconstructed picture. Using this system, the errors are less likely to occur in the important sections of the video packet. Fig. 13 shows an example of unequal error protection with an MPEG-4 video packet.

Experimental Setup

To test the use of unequal error protection, we ran several experiments using the sequences "Akiyo" and "Mother & Daughter" at both CIF and QCIF resolution. The quantization parameter was chosen so that the source coding output was approximately 48 kbps at 7.5 fps for the CIF images and 24 kbps at 10 fps for the QCIF images. Each reconstructed sequence contained 10 seconds of video.



▲ *Fig. 14. The total number of errors remaining in the bitstreams after channel coding versus the number of errors on the raw channel for unequal error protection with rates-$\frac{3}{5}$, $\frac{2}{3}$, and $\frac{3}{4}$ applied to header, motion, and texture, respectively, and equal error protection with rate-$\frac{7}{10}$ applied to each section of the video packet. These channel coding rates are effective in reducing the number of errors when the raw channel BER is less than 6%. For higher channel BERs, more powerful codes would be required.*

The sequences were coded using all the MPEG-4 error-resilience tools. The compressed bitstreams were then channel-coded using convolutional encoding of the data with either equal error protection (EEP) using a fixed rate-$\frac{7}{10}$ code or unequal error protection (UEP) using a rate-$\frac{3}{5}$ code for the header segment, a rate-$\frac{2}{3}$ code for the motion segment, and a rate-$\frac{3}{4}$ code for the texture segment. These EEP and UEP rates, chosen because they both give approximately the same amount of FEC overhead, were obtained by puncturing the output of the rate-$\frac{1}{2}$ coder shown in Fig. 11. The FEC-coded sequences were sent through a multiplexer, and the output packets from the multiplexer were sent through a GSM channel simulator. This simulator is based on a complex model of a GSM channel that has been fitted with data taken from a real GSM channel to get an accurate account of the errors found on this channel. The GSM channel simulator simulates a bursty channel caused by multipath fading when the transmitter is travelling at 3 km/hour. The received signal is quantized to eight bits; it is, therefore, in the range [-127,127], where the sign of the received signal represents the received bit and the magnitude represents the reliability. The reliability information is utilized by performing soft-input Viterbi decoding [13] (where the values range between [-127,127]), which improves the error-correction capabilities of the channel decoder compared with hard input decoding (where the values are quantized to a single bit, either 0 or 1, without regard to context).

Each FEC-coded bitstream was subjected to six different GSM channel conditions ranging from 0.3% to 12% BER (corresponding to a carrier-to-interference ratio of between 19 dB and 4 dB) in 50 different trials per channel condition. The corrupted received bitstreams were channel-decoded and the error-corrected bitstreams were source-decoded to find the quality (average PSNR) of the reconstructed video. For each of these trials, the first frame was transmitted without corruption. Since video coding heavily utilizes temporal compression, it is important that an initial frame be received error-free so that subsequent frames can be properly decoded. It is reasonable to assume that the first frame can tolerate some delay for retransmissions in order to ensure it is received error-free, as this will substantially improve the quality of the remaining session.

Results

In order to compare the different methods of adding channel coding to the compressed video, the results from the 50 trials at a given GSM channel error rate were averaged for both sequences. Fig. 14 shows the average BER that remains after channel decoding for each of the GSM channel BER conditions. For unequal error protection, this plot shows the total number of errors in the FEC decoded bitstream divided by the total number of bits. Since the different sections of the video packet were protected

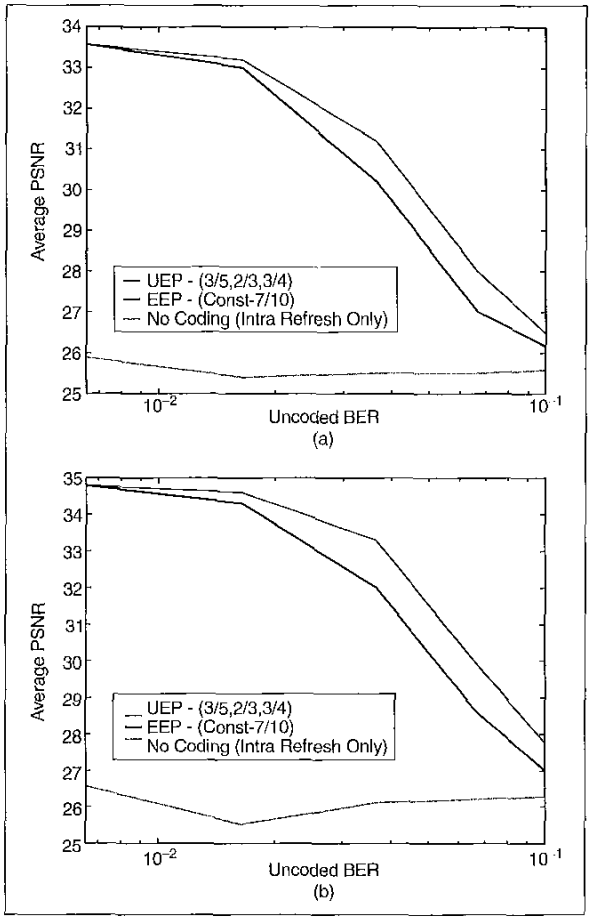
using different rate channel coders, the distribution of errors will not be uniform throughout the packet. In particular, the header will have fewer errors than what is depicted in this graph, while the texture information will have more errors than what is shown here.

Channel coding reduces the effective BER seen by the video decoder by over an order of magnitude for most of the raw channel conditions. However, the convolutional codes break down when the channel error rate is too high. Thus, for the GSM channels with a BER around 10%, the channel coding actually increases the effective BER seen by the decoder. Under such harsh conditions, the channel coder would need to use more powerful codes to reduce the BER. However, for the remainder of the GSM channel conditions, the FEC codes reduce the effective BER. This brings the number of bit errors remaining in the bitstream that is sent to the MPEG-4 decoder to a level at which the error resilience tools can work.

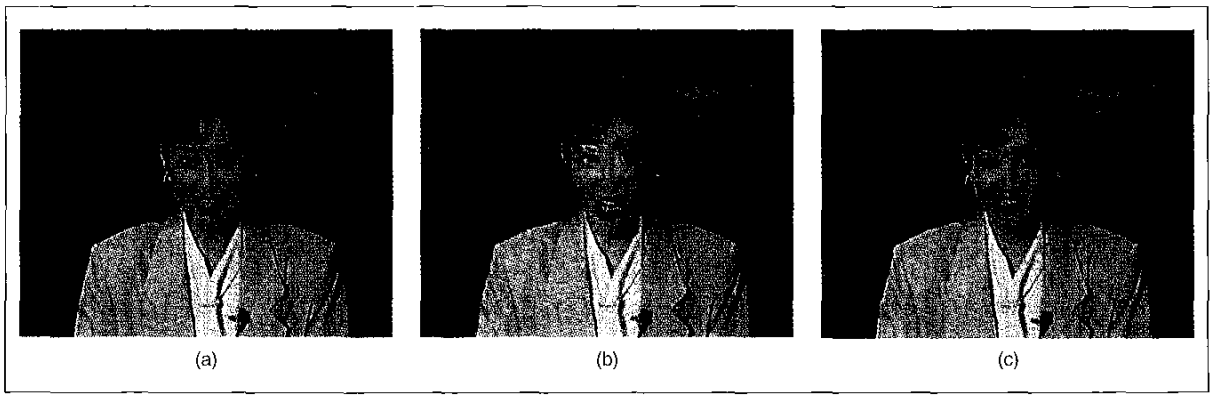
Fig. 15 shows a comparison of the average PSNR values obtained for fixed coding and unequal error protection. These plots show that unequal error protection produces the highest average PSNR for the reconstructed video for both CIF and QCIF images at high channel error rates. Since both coding methods require the same amount of FEC overhead, this improvement (as much as 1 dB) does not require additional bandwidth. In addition, for the error conditions shown here, the fixed rate-$\frac{7}{10}$ coder actually produces fewer errors in the channel decoded bitstream than the UEP coder (as shown in Fig. 14), yet it still produces lower quality reconstructed video. This is because the errors are spread evenly throughout the different portions of the video packet. Conversely, the unequal error protection coder may leave more errors in the channel decoded bitstream, but these errors are in less important portions of the video packet.

Fig. 16 shows a reconstructed frame of "Akiyo" when there are no channel errors and when the GSM channel error rate is 4%, and the video is protected using EEP with a rate-$\frac{7}{10}$ coder and UEP with a rate-$(\frac{3}{5}, \frac{2}{3}, \frac{3}{4})$ coder. These images also show the advantage of using unequal error protection when the channel error rate is high.

Rather than using the extra bandwidth for channel coding, it might be beneficial to spend these bits on forced intra-MB updates. These intra-MBs would stop error propagation and hence, improve reconstructed video quality. In order to test the effectiveness of using



▲ *Fig. 15. UEP improves average PSNR by as much as 1 dB, compared to EEP, when channel conditions are harsh. Results are for MPEG-4 video compressed with all the MPEG-4 error resilience tools. (a) CIF images. (b) QCIF images. These plots also show that it is much better to use a fixed overhead for channel coding rather than forced intra-MB updates at these channel error rates.*



▲ *Fig. 16. A frame of "Akiyo" visually shows the benefit of UEP, compared to EEP, when transmitting video over a simulated GSM channel with 4% BER. This figure shows the reconstructed frame with (a) no channel errors, (b) EEP coding and (c) UEP coding.*

intra-MBs, the video sequences were compressed with enough forced intra-MBs each frame to increase the source-coded bit rate to equal that of the FEC-coded bitstream when no extra intra-MBs are used. The results of this experiment are shown in Fig. 15, labeled "No Coding (Intra Refresh Only)." These plots show that it is much better to use the overhead for channel coding than forced intra-MBs at these high channel error rates. Using the overhead for intra-MB refresh increases the number of source bits that are corrupted due to channel errors, causing the reconstructed quality to be poor. As the channel error rates decrease below the levels tested here, it would probably be advantageous to reduce the number of bits spent on channel coding and increase the number of forced intra-MBs per frame to get the optimal reconstructed video quality.

A Total Solution

This article describes the error-resilience tools, channel coding, and processor capabilities for wireless video communication. Multimedia communication requires significantly more bandwidth than speech alone, but third-generation wireless standards, combined with new video compression standards such as MPEG-4, will provide sufficient bandwidth to support many types of video applications. Processing digital video requires a significant amount of memory, computation, and internal data transfer, yet even a 40-MIPS low-power DSP is capable of decoding reduced-resolution video. While faster DSPs will enable higher-resolution video applications, consumers should not expect TV quality. Compressed video is particularly sensitive to bitstream errors, and requires use of additional source- and/or channel-coding tools for robust transmission over wireless channels. The MPEG-4 video standard includes several source-coding tools that enable faster recovery and better localization of bitstream errors. The MPEG-4 video syntax also lends itself to using unequal error protection. Significant progress in the areas of antenna, receiver, modulator, and power-supply design is also being made, but that is beyond the scope of this article. At last, wireless video communication is becoming technologically feasible.

Some questions still remain. What are the quality-price combinations that consumers will buy? Will consumers be willing to pay for the bandwidth for video? What video applications will be in the highest demand (e.g., videophone, web browsing)? How profitable will video capability be for network providers and manufacturers of wireless communicators? The answers to these questions are unknown. What is certain is that whenever and wherever the market conditions are right, multimedia capability will enable a host of new wireless products.

Acknowledgment

We would like to thank Bruce Fette and the anonymous reviewers for their insightful comments on the article. Thanks to David Bartley, who introduced us to Hugo Gernsback's *Ralph 124C 41+*.

Madhukar Budagavi received his B.E. in electronics and communications engineering from the Regional Engineering College, Trichy, India, in 1991, and his M.Sc (Eng) in electrical engineering from Indian Institute of Science, Bangalore, India, in 1993. He received his Ph.D. in electrical engineering from Texas A&M University, in 1998. From 1993-95, he was first a Software Engineer and then a Senior Software Engineer at Motorola India Electronics Ltd., primarily developing DSP software and algorithms for the Motorola DSP chips. Since 1998, he has been a Member of Technical Staff in the Texas Instruments DSP Solutions R&D Center, working on MPEG-4 and wireless video communications. His current research interests include video coding, speech coding, and wireless and Internet multimedia communications.

Wendi Rabiner Heinzelman received her B.S. in electrical engineering from Cornell University, Ithaca, NY in 1995, and her M.S. in electrical engineering from the Massachusetts Institute of Technology (MIT) in 1997. She is currently working toward her Ph.D. in wireless sensor networks at MIT. Her research interests include energy-efficient network protocols, channel coding for wireless networks, image processing and video coding. She is a member of Eta Kappa Nu, Tau Beta Pi, and Sigma Xi, and is a Kodak Fellow.

Jennifer Webb received her B.S. in mathematics from Baylor University in 1982, her M.S. in computing science from Texas A&M University in 1983, and her Ph.D. in electrical engineering from the University of Illinois at Urbana-Champaign in 1995. She was a Lechner fellow at Texas A&M, and a Joint Services Electronics Program Fellow at UIUC. From 1983 to 1990, Webb worked as a Radar-Systems Analyst in the Defense Systems and Electronics Group at Texas Instruments, and was granted an educational leave-of-absence to pursue the Ph.D. degree. Her thesis research was in the areas of digital filter design and synthetic aperture radar imaging. She returned to Texas Instruments as a Member of the Technical Staff in the DSP Solutions R&D Center. Her most recent work is in the area of robust, low-bit-rate video processing for wireless applications. General research interests include digital signal and image processing and communications.

Raj Talluri received his Ph.D. in electrical engineering from the University of Texas at Austin in 1993. Since then he has been with Texas Instruments. He is currently the Chief Technology Officer for the Digital Still Camera business unit. He has been an active participant at the MPEG meetings and has represented Texas Instruments at MPEG for four years. He made a number of technical contributions to the MPEG-4 standard and chaired a

number of subgroups at MPEG-4. Talluri was the Co-Program Chair of the IEEE Workshop on Computer Vision applications, WACV '96. He is the Associate Editor of *IEEE Signal Processing Letters* and *IEEE Transactions on Multimedia*. He has authored a number of journal papers and conference articles in the area of video coding and computer vision. His current research interests include signal and image processing, wireless video coding, digital still cameras and computer vision. He is a member of IEEE Signal Processing Society.

References

[1] 3GPP web site, http://www.3gpp.org/.

[2] ACTS-AC098 MoMuSys (Mobile Multimedia Systems) website, http://www.uni-hannover.de/project/eu/momusys/overview.html.

[3] Bluetooth web site: http://www.bluetooth.com/.

[4] M. Budagavi, W. Rabiner, J. Webb, and R. Talluri, "Wireless MPEG-4 video on Texas Instruments DSP chips," *Proc. IEEE International Conf. Acoust. Speech and Signal Processing*, Phoenix, AZ, vol. 4, pp. 2223-2226, Mar 15-19, 1999.

[5] M. Budagavi and R. Talluri, "Wireless video communications," chapter in *Mobile Communications Handbook*, second edition, J. Gibson, editor, CRC Press, 1999.

[6] M. Budagavi, J. Webb, M. Zhou, J. Liang, and R. Talluri, "MPEG-4 video and image coding on digital signal processors," *J. VLSI Signal Processing—Systems for Signal Image and Video Technology*, Kluwer, to appear.

[7] W.-H. Chen, C.H. Smith, and S.C. Fralick, "A fast computational algorithm for discrete cosine transform," *IEEE Trans. Commun.*, vol. 25, no. 9, pp. 1004-1009, Sep. 1977.

[8] D. Clark, "Preparing for a new generation of wireless data," *IEEE Computer*, pp. 8-11, vol. 32, no. 6, Aug. 1999.

[9] F. Dufaux and F. Moscheni, "Motion estimation techniques for digital TV: a review and a new contribution," *Proc. IEEE*, vol. 83, no. 6, June 1995, pp. 858-876.

[10] J. Eyre and J. Bier, "DSP processors hit the mainstream," *IEEE Computer*, vol. 31, no. 8, Aug. 1998, pp. 51-59.

[11] FCC web site, DTV FAQ: http://www.fcc.gov/oet/faqs/dtvfaqs.html.

[12] C. Fogg, "Survey of software and hardware VLC architectures," *SPIE Image and Video Compression*, vol. 2186, 1994.

[13] G.D. Forney, "The Viterbi algorithm," Proc. IEEE, vol. 61, no. 3, pp. 268-278, Mar 1973.

[14] H. Gernsback, *Ralph 124C 41+, A Romance of the Year 2660*, 1911, originally published by Gernsback, now published by Buccaneer Books.

[15] E. De Greef, F. Catthoor, and H. De Man, "Program transformation strategies for memory size and power reduction of pseudoregular multimedia subsystems," *IEEE Trans. Circuits and Syst. Video Tech.*, vol. 8, no. 6, Oct. 1998, pp. 719-733.

[16] J. Hagenauer, "Rate-compatible punctured convolutional codes (RCPC codes) and their applications," *IEEE Trans. Commun.*, vol. 36, no. 4, pp. 389-400, Apr. 1988.

[17] W.R. Heinzelman, M. Budagavi, and R. Talluri, "Unequal error protection of MPEG-4 compressed video," *Proc. IEEE International Conf. Image Process.*, Kobe, Japan, Oct. 25-28, 1999.

[18] H. Hendrix, "Viterbi decoding techniques in the TMS320C54x family," Application note SPRA071, June 1996, Texas Instruments, Inc.

[19] D. Herrell, "Power to the package," *IEEE Spectrum*, vol. 36, no. 7, July 1999, pp. 46-53.

[20] *IEEE Standard Specification for the Implementation of 8 by 8 Inverse Discrete Cosine Transform*, IEEE Standard 1180-1990, 1990.

[21] K. Imura and Y. Machida, "Error resilient video coding schemes for real-time and low-bitrate mobile communications," *Signal Processing: Image Communication*, vol. 14, pp. 519-530, May 1999.

[22] ITU-T Recommendation H.223, "Multiplexing protocol for low bitrate multimedia communication," Annex C, Sep 1997.

[23] ITU-T Recommendation H.263, "Video coding for low bit rate communication."

[24] ITU-T Recommendation H.324, "Terminal for low bitrate multimedia communication."

[25] D. Kyle, *A Pictorial History of Science Fiction*, The Hamlyn Publishing Group Limited, Holland, 1976.

[26] P. Lapsley and G. Blalock, "How to estimate DSP processor performance," *IEEE Spectrum*, vol. 33, no. 7, July 1996, pp. 74-78.

[27] E. Lee, "Programmable DSP Architectures: Part I", *IEEE ASSP Magazine*, vol. 5, no. 4, Oct. 1988, pp. 4-19.

[28] Lin and Costello, *Error Control Coding: Fundamentals and Applications*, Prentice-Hall: New Jersey, 1983.

[29] MPEG-4 Video Group, "Overview of the MPEG-4 Standard," ISO/IEC JTC1/SC29/WG11 N2323, Dublin, Ireland, July 1998. http://drogo.cselt.stet.it/mpeg/standards/mpeg-4/mpeg-4.htm.

[30] M. W. Oliphant, "The mobile phone meets the Internet," *IEEE Spectrum*, vol. 36, no. 8, pp. 20-28, Aug. 1999.

[31] L. Robinson, "Japan's new mobile broadcast company: multimedia for cars, trains, and hand-helds," *Advanced Imaging*, Jul. 1998, pp. 18-22.

[32] S. Sriram and C.-Y. Hung, "MPEG-2 video decoding on the TMS320C6X DSP architecture," *Proc. 32 Asilomar Conf. on Signals, Systems, and Computers*, Pacific Grove, CA, Nov. 1998.

[33] G. Sullivan and T. Wiegand, "Rate-distortion optimization for video compression," *IEEE Signal Processing Magazine*, vol. 15, no. 6, Nov 1998, pp. 74-90.

[34] R. Talluri, "Error-resilient video coding in the ISO MPEG-4 standard," *IEEE Communication Magazine*, vol. 36, no. 6, June 1998.

[35] R. Talluri, I. Moccagatta, Y. Nag, and G. Cheung, "Error concealment by data partitioning," *Signal Processing: Image Commun.*, vol. 14, pp. 505-518, 1999.

[36] Telenor R&D H.263 software, http://www.fou.telenor.no/brukere/dvc/h263_software/.

[37] Texas Instruments official DSP www site: http://www.ti.com/sc/docs/dsps/products/.

[38] Y. Wang, Q.-F. Zhu, "Error control and concealment for video communications: A review," *Proc. IEEE*, vol. 86, no. 5, May 1998, pp. 974-997.