ARCHITECTURE ANALYSIS FOR LOW-DELAY VIDEO CODING

Ralf M. Schreier, A. M. Tushar Iqbal Rahman, Ganesh Krishnamurthy, Albrecht Rothermel

University of Ulm, Microelectronics Department, 89081 Ulm, Germany

email: ralf.schreier@uni-ulm.de

ABSTRACT

Low-delay video coding is a key technology for video conferencing as well as upcoming remote-monitoring and automotive video applications like rear-view cameras or night vision systems. As the ongoing progress in programmable DSP and ASIC technology allows cost effective and flexible implementations of the necessary hardware, compressed video transmission systems over multimedia busses will soon replace the current uncompressed systems even in latency critical applications.

In this paper, fundamentals and theoretic limits of low-delay video coding are discussed with respect to architectural consequences of real-time implementations. A general latency analysis for a compressed video transmission systems is presented considering algorithmic, architectural and transmission related delays.

1. INTRODUCTION

Current video compression standards can be grouped into intra-frame codecs (Motion JPEG, JPEG2000) and codecs with temporal prediction (MPEG-1/2/4, H.264). As successive frames of a video sequence have high correlation, temporal prediction can reduce the data rate significantly at the cost of higher computational complexity.

According to the system presented in Fig. 1, the latency of a compressed video transmission system can be subdivided into encoder latency, transmission latency and decoder latency. On the encoder side, the frame-reordering needed for the bi-directional prediction (B-frames) would introduce unacceptable latencies of several frame periods. Therefore, only the forward temporal prediction (I and P-frames) can be used for low-latency video transmissions.

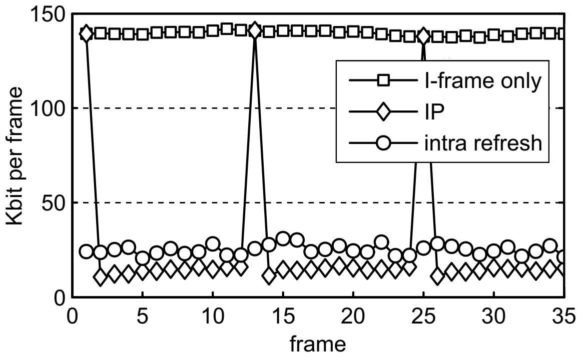


Figure 2: *Frame data rate for test sequence "mobile & calendar" (H.264, q=35, 1 reference frame).*

Figure 2 shows the rate variations for a typical video sequence using three different coding methods. To achieve short buffering delays it is generally desirable to have a constant bit rate (CBR) in short time intervals with minimum intervention of the rate control.

The standard IP coding scheme shows significant peak rates at the I-frames which consume up to 10 times more data rate than P-frames. As the encoder averages the rate peaks over the following frames, the IP-coding results in buffering delays of several frame periods for a CBR transmission. A reduction of buffering delays can be achieved by either accepting high peak data rates or by selecting appropriate coding modes. Using the "I-frame only" coding method is one solution for this problem because the rate variations in an intra-coded video sequence are fairly small and can be controlled easily.

For achieving higher compression ratios using temporal prediction, it is possible to embed intra coded macroblocks

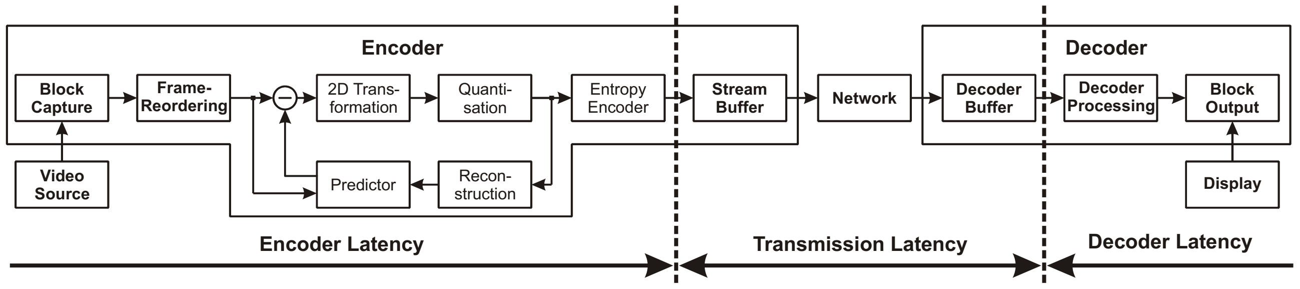


Figure 1: *Video transmission latency model.*

(MB) into predicted frames. The forced intra refresh method [1] enables low-latency transmissions at a relatively constant bit rate per frame as indicated in Fig. 2. Furthermore, a defined refresh time can be guaranteed with protection mechanisms for the refreshed areas [2]. At the beginning of the video sequence a single I-frame with coarse quantization (q=51) is transmitted. Both, the I-frame only and the Intra refresh method allow short buffering delays of one frame period to average out rate variations.

2. SYSTEM LATENCIES

Figure 3 shows a process scheduling diagram for a 25 Hz progressive-scan transmission with CBR on frame level. The analysis is based on the assumption that processing in any stage can start immediately when all input data is available. Real-time constraints allow 40 ms time for each processing stage.

The block capture stage models the effects of continuous video sampling. Before the encoder processing starts, the block capture device collects the incoming uncompressed video stream until the complete video frame is buffered. During the processing of the frame, the encoder output buffer is filled with the compressed frame data which is then transmitted during the next frame period. A standard-conformant video decoder generally does not start processing a frame until it is completely available in its input buffer. Displaying the frame can start right after the processing is finished, therefore the block output does not account for the latency. The straight forward implementation of a frame based encoder (Fig. 3 (a)) introduces a significant delay of 160 ms between capturing and displaying a frame.

A reduction of the overall latency can be achieved by processing sections of frames rather than whole frames. The lower bound for such a system is determined by the size of the basic processing block, the 16 by 16 pixel MBs. For standard resolution D1 video, 16 lines of video (one slice) introduce 16 * 64 µs = 1 ms latency in the block capture device. As illustrated in Fig. 3 (b) the theoretical lower limit for a system working at CBR on slice level is 4 ms latency. If the encoder ensures the just in time availability of data, a slightly modified decoder can start the decoding process before the frame is transmitted completely.

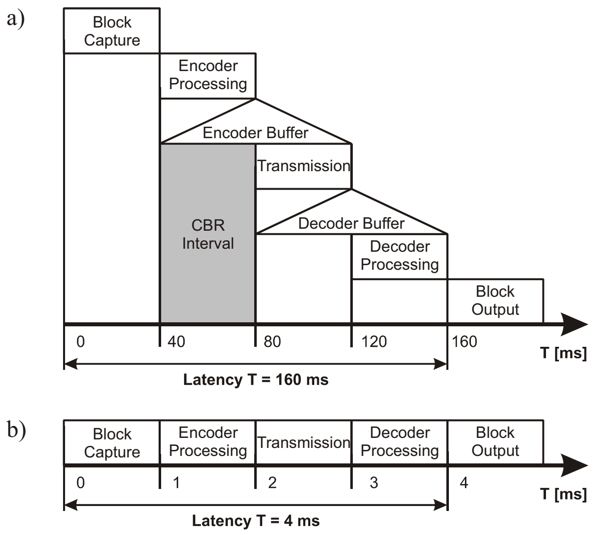


Figure 3: *Process scheduling for 25 Hz progressive scan compressed video transmission with (a) frame based and (b) slice based processing.*

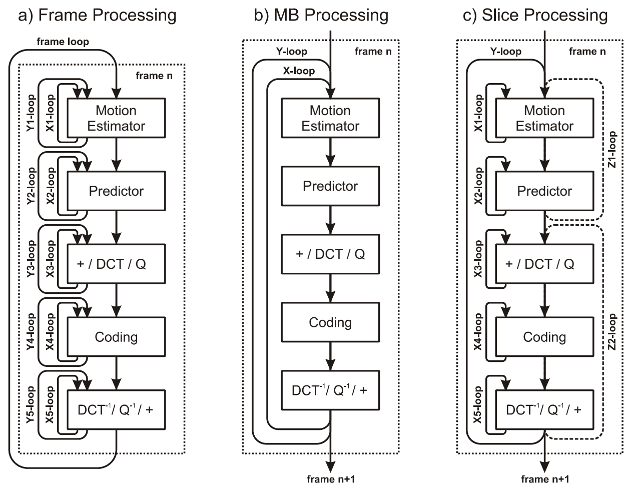


Figure 4: *Program flow options for video encoders. For combined MB / slice processing (c), the x loops are replaced by the z loops.*

3. ENCODER PROGRAM FLOW OPTIMIZATIONS

The basic video coding algorithm described in Fig. 1 leaves the system designer certain options for organizing the sequence of operations on a macroblock level. Depending on the flows illustrated in Figure 4, different minimum latencies and memory requirements will result for a real-time video encoder. In the standard configuration of current video standards, the macroblocks of a frame are coded in a strictly increasing order in lines of macroblocks starting at the top left corner of the frame. In this document, a complete line of macroblocks is called a slice, although most video standards use a more flexible definition for this term.

In a frame-based processing scheme (Figure 4 (a)), the x and y MB loops are located inside the major processing functions. The sequence of processing is to perform the motion estimation for all MBs of a frame in a first step, then the prediction for all MBs and so on.

Moving the x and y loops up to the frame level loop as shown in Figure 4 (b) results in pipelining of MBs and minimizes the latency and memory requirements [3]. A similar processing scheme is used by H.264 codec [4], where several iterations per MB are performed to determine the best coding mode. If a non-iterative program flow is assumed, pipelining stages can be inserted after each major processing stage in the codec. However, a MB processing scheme is disadvantageous for supporting parallel processing architectures like processors with hardware support or multi-core processors, as the frequent synchronization of parallel processing units increases the complexity and can lead to unbalanced work loads over time.

A good trade-off between latency and complexity can be achieved by slice-based processing schemes. As illustrated in Figure 4 (c), the x-loop is on the same level as in the frame processing scheme, whereas the y-loop is moved up to the frame loop level. In this approach, pipelining stages can be inserted after each x-loop operation, which eases the implementation on parallel architectures.

In the slice-based implementation, processing units must only be synchronized at the end of a slice, whereas the MB pipelining results in a significantly higher synchronization frequency (D1 resolution, 720*576 pixels):

$$f_{sync, slice} = 25\frac{frames}{s}\cdot 36\frac{syncs}{frame} = 900\frac{syncs}{s}$$

$$f_{sync, MB} = 25\frac{frames}{s}\cdot 1620\frac{syncs}{frame} = 40500\frac{syncs}{s}$$

Reducing the frequency of synchronizations also allows more flexible distribution of work loads on processing units which is especially critical in the entropy coding unit where the work load increases with the data rate. A rough analysis of the bit rate per MB reveals, that large MBs require two to three times the average bit rate in an intracoded frame. As the average MB data rate is far below the peak MB rate, it makes sense to average out these work load variations using a slice-based processing scheme. Longer synchronization intervals generally allow better averaging of work load fluctuations but they also increase the system latency. For an I-frame only video encoder, a fairly constant work load in the entropy coding unit can be achieved if the CBR averaging interval is equal to the processing interval, e.g. slice based processing with a constant slice bit rate.

Depending on the architecture and work loads, the slice-level pipelining stages can be rearranged. The combined MB/slice processing illustrated with the Z loops in Figure 4 (c) was developed according to the profiling results of the DSP-optimized MPEG-2 encoder. For the dual-core DSP target architecture [5], the system is split into two pipelining stages between the Z1 and Z2-loop. Each loop is mapped to one DSP core which results in a balanced work load for the desired coding parameters.

4. FRAME MEMORY REQUIREMENTS

High-performance signal- and general purpose processors use on-chip SRAM memories to reduce the latencies caused by accesses to external DRAMs. The size of on-chip memories is typically limited to several hundred kBytes, which is not enough to store a complete D1 video frame, requiring 720*576*1.5 = 622 kBytes for 4:2:0 color sub-sampling.

A detailed memory organization diagram of the MPEG-2 TM5 encoder [6] in IP coding mode is shown in Figure 5. The encoder allocates three pixel data memories for the current frame (1) and the reconstructed frames (4), (5). Furthermore, intermediate results of the prediction (2) and the DCT (3) are saved in frame memories because they are needed by more than one following processing blocks. The frame memories 1 to 3 can be re-used instantaneously, whereas the reconstructed frame memories 4 and 5 can be reused at half frame frequency.

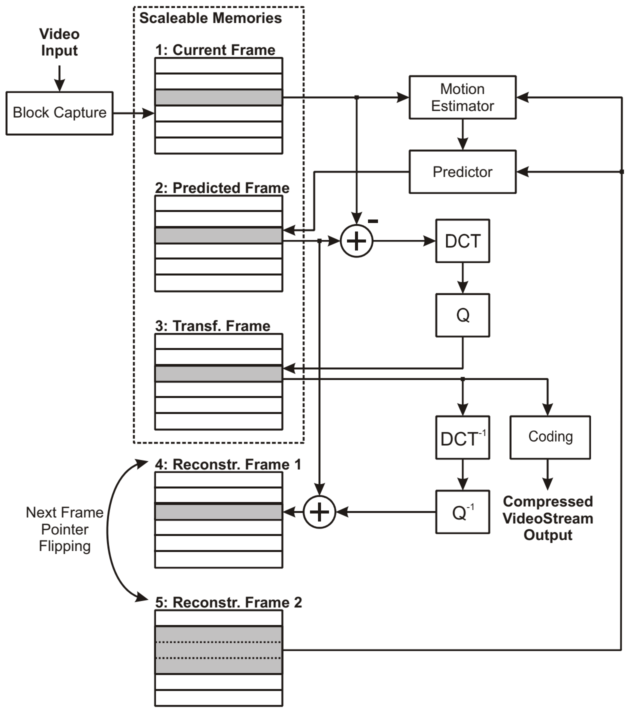


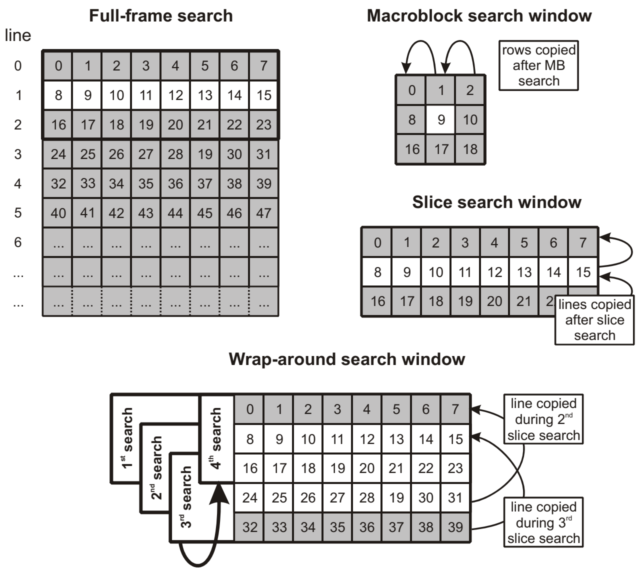
Figure 5: *IP-mode video encoder frame memory structure*

In a frame-based processing scheme, the whole image must be accessed in each processing stage from external memory, which results in very high memory bandwidth requirements and bad cache usage. Localizing memory accesses improves the cache usage and can be achieved by processing sections of a frame sequentially. The lower-bound for localization is the MB based processing on 16 by 16 pixel image regions. This allows reducing the cached parts of frame memories 1, 2 and 3 to 16*16 pixels. From the old reference frames, only the pixels of the search window around the current MB are accessed, e.g. a 48 by 48 pixel image segment for a +/- 16 pixel motion search range.

On the implementation side, the MB based processing requires an adaptation of the frame memory address calculations in low-level functions because of the modified picture geometry in the cache. The slice based processing overcomes this drawback by caching complete rows of MBs but it requires significantly larger cache memories.

5. MOTION ESTIMATION MEMORY

Optimizing the memory requirements of the motion estimation and prediction functions of the video encoder is especially critical due to the frequent accesses and large size of the referenced image section. As illustrated in Figure 5, accesses to the scaleable memories can be restricted to a small, 16 line region of a frame. For good motion estimation it is however necessary to cover a sufficient large vertical search range which



Full-frame search
line
Macroblock search window
rows copied after MB search
Slice search window
lines copied after slice search
Wrap-around search window
1st search
2nd search
3rd search
4th search
line copied during 2nd slice search
line copied during 3rd slice search

Figure 6: *Motion estimation memory organization.*

is practically at least +/- 16 lines around the current slice. Hence, even for a relatively small vertical search range it is necessary to provide 3 rows of macroblocks from every reference frame in the cache.

Figure 6 shows four memory organisations scenarios for the motion estimation which result in different cache in copy operation counts summarized in Table 1. Border effects which result in a slight reduction of memory reshuffling operations, are not considered. Considering MB 9 in line one of the current frame and a search range of +/- 16 pixels, pixel data from MBs 0, 1, 2, 8, 9, 10, 16, 17 and 18 is needed from the reference frame. The MB search window results in the smallest memory usage but it requires adaptation of the motion estimation and prediction functions.

After the motion estimation for the current MB is performed, the pixels of the search window are copied one column to the left, resulting in 2*3*N cache reshuffling operations for a whole frame of N pixels. Furthermore, the new content of the right most column is loaded from external memory, resulting in 3*N load operations. Using a wrap-around addressing scheme instead of memory reshuffling would increase the computational complexity of the motion estimation algorithms, whereas memory reshuffling can be performed by the DMA (direct memory access) controller at the cost of high memory access bandwidth requirements.

In the case of a slice search window, slices 0, 1 and 2 must be buffered, requiring 48 * 720 * 1.5 bytes = 51.8 kBytes memory at D1 resolution. Using this approach, it is possible to shift the memory content in the frame segment after the motion estimation was performed for a complete slice. Therefore, the reference frame must only be loaded once from the external memory. It is however necessary to reshuffle 2 * N pixels for the complete frame estimate in the cache.

With the wrap-around search window, the reshuffling operations can be further reduced to 2/3 * N and can be executed in parallel with the motion estimation. Therefore, depending on the size of the available cache memories it is possible to exchange memory requirements and reshuffling operations according to the needs of the architecture.

	cache memory	load operations	reshuffle operations
frame	1 N	1 N	0
macroblock	9 * 256	3 N	6 N
slice	3 N/M	2 N	2 N
wrap-around	5 N/M	1 N	2/3 N

Table 1: *Motion estimation memory usage.*
N = number of pixels per frame, M = slices per frame

6. SUMMARY

In this document we presented an analysis for implementing low-latency video transmission systems. According to our model, three major sources of latency must be considered, namely the encoder and decoder processing latency and the transmission latency. A lower bound for the encoder processing latency is given by the block capture latency which models the continuous time video sampling process.

Choosing the optimal coding mode is a major step to reduce the transmission latencies. For achieving a constant bit rate per frame which allows low buffering latencies, either an intra coding mode or an improved intra refresh coding mode with significantly higher coding efficiency can be used. Both coding modes ensure error-free video display within a short time interval.

The effect of different processing schemes was discussed with respect to latencies, memory requirements, parallel processing options and work load averaging. For the system implementation we recommend using a slice-based processing scheme which enables compressed video transmission with a minimum latency of 4 ms. Finally, four memory organisation concepts for the motion estimation reference picture are presented which allowing a trade-off between memory requirements and cache load operations.

7. REFERENCES

[1] Tri D. Tran, Lurng-Kuo Liu, Peter H. Westering: *"Low-delay MPEG-2 video coding"*, Proc. SPIE - Int. Soc. Opt. Eng. (USA), USA: SPIE-Int. Soc. Opt. Eng, vol.3309, 1997, pp. 510-16.
[2] Ralf M. Schreier and Albrecht Rothermel: „Motion Adaptive Intra Refresh for the H.264 Video Coding Standard", IEEE Trans. on Consumer Electronics, Vol. 52, No. 1, Feb. 2006, pp. 249-253
[3] Gabby Yi, Ke Ning, Marc Hoffman: "Memory-organization challenges with real-time video encoding on embedded signal processors", EDN, 11/27/ 2003, available at www.edn.com
[4] H.264 JM ref. software, http://iphome.hhi.de/suehring/tml/download/
[5] "ADSP-BF561: Blackfin Embedded Symmetric Multi-Processor Data Sheet", available at http://www.analog.com
[6] MPEG-2 TM5 ref. software, http://www.mpeg.org/MSSG/#source