Parallelization of a H.263 Encoder for the TMS320C80 MVP

Authors: H. Mooshofer, A. Hutter, W. Stechele

ESIEE, Paris

September 1996
SPRA339

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI™) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

Contents

Figures

Tables

Parallelization of a H.263 Encoder for the TMS320C80 MVP

Abstract

The coding of digital video sequences has been paid increasing attention over the past few years. Algorithms and standards such as MPEG1, MPEG2 or H.261 have been developed allowing more and more compression. A new standard in this field is H.263 which has been recently adopted by the ITU. It is intended for videoconferencing, videophoning, surveillance and other low bit rate applications (below 28,8 kbit/s). Compared to its predecessor H.261 it has additional modes and improved motion estimation resulting in an improved coding efficiency and allowing to transmit video sequences over analog telephone lines. H.263 is computational intensive and needs a powerful hardware for its implementation.

This paper describes a real-time implementation of a H.263 encoder on a Texas Instruments(TI™) TMS320C80 (MVP) multiprocessor system. In order to exploit the four DSPs and the RISC processor of the MVP the encoding algorithm must be parallelized. The paper discusses the main issues of parallelization for the MVP: Load balance and appropriate scheduling taking into account the bandwidth of the memory interface and the size of internal memory. For real-time applications of video coders there is another important criterion: Minimal coding delay.

The key to parallelisation are the data dependencies, which are analyzed and described. There are two types of data dependencies: (i) between different operations and (ii) between adjoining parts of the picture. It is shown how the data dependencies lead to a partitioning of the algorithm into tasks. The data dependencies between adjoining macroblocks further reduce the number of parallelization alternatives and influence the scheduling.

With these results four basic ways of parallelization are compared and row-wise parallelization is found to be the best solution. The resulting concept of the encoder program is described and its scheduling and memory usage is shown. Experimental results are given.

The current state of the implementation is an encoder in compliance with the TMN5 test model. Since there are time consuming parts of the program written in C there is still potential for enhancement. It is described what can be done to improve speed and it is shown that it is necessary to use a faster algorithm for motion estimation instead of full search.

This document was part of the first European DSP Education and Research Conference that took place September 26 and 27, 1996 in Paris. For information on how TI encourages students from around the world to find innovative ways to use DSPs, see TI's World Wide Web site at www.ti.com.

Product Support on the World Wide Web

Our World Wide Web site at www.ti.com contains the most up to date product information, revisions, and additions. Users registering with TI&ME can build custom information pages and receive new product updates automatically via email.

Introduction

This paper describes an implementation of a H.263 encoder, which was done at the Technical University of Munich, Chair of Integrated Circuits. The focus of the research is on the encoder because it is more critical than the decoder and it also contains the essential parts a decoder consists of. The encoder is running on a MVP contains four signal processors and one RISC-processor and has the computational power to perform encoding in real-time. A main aspect of the implementation focused by this paper is parallelization. The paper discusses this aspect and the resulting implementation in detail.

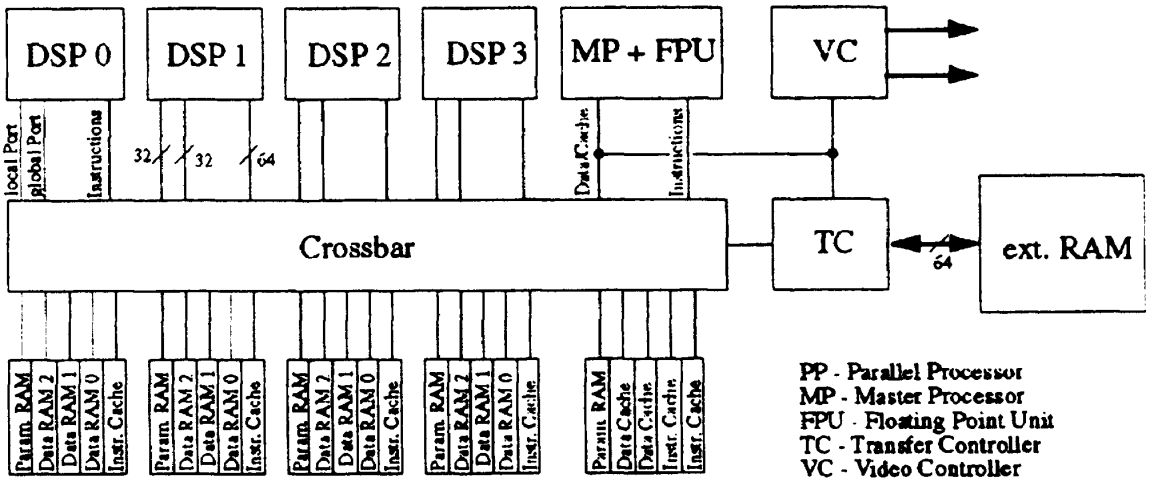
In the section, *Overview Over the MVP and H.263*, the structure of the MVP is shown and a short overview over H.263 is given. The section on *Parallelization* starts with a discussion of parallelization, next the algorithm is divided into tasks, data dependency and duration are examined. In the section, *The Implementation*, the alternatives for parallelization are discussed and the concept of the implementation is described. The *Results* section gives the results of the current state of the implementation. Further development and improvements are discussed in the *Conclusion*.

Overview Over the MVP and H.263

The MVP

The MVP is a MIMD multiprocessor system consisting of five independently operating processors. Four of the processors are signal processors, one is a RISC processor. The MVP also contains on-chip RAM, a crossbar and a transfer controller. Figure 1 shows the structure of the MVP. All blocks shown, except the one called ”ext. RAM“ are on-chip.

Figure 1. Block Diagram of the MVP



The MVP contains limited amount of on-chip memory and on-chip cache. The processors can access only on-chip memory in a direct way, while all accesses to the external memory are managed by the transfer controller. Between the RAMs and the processors the crossbar connects the processors to the addressed RAMs or caches and is capable of doing simultaneous connections.

The master processor has a data cache allowing to access external memory transparent from the programs view. By contrast the DSPs have no caches, which requires their programs to work on data loaded in the internal memories for efficiency reasons.

An more detailed overview over the MVP is given in [7].

H.263

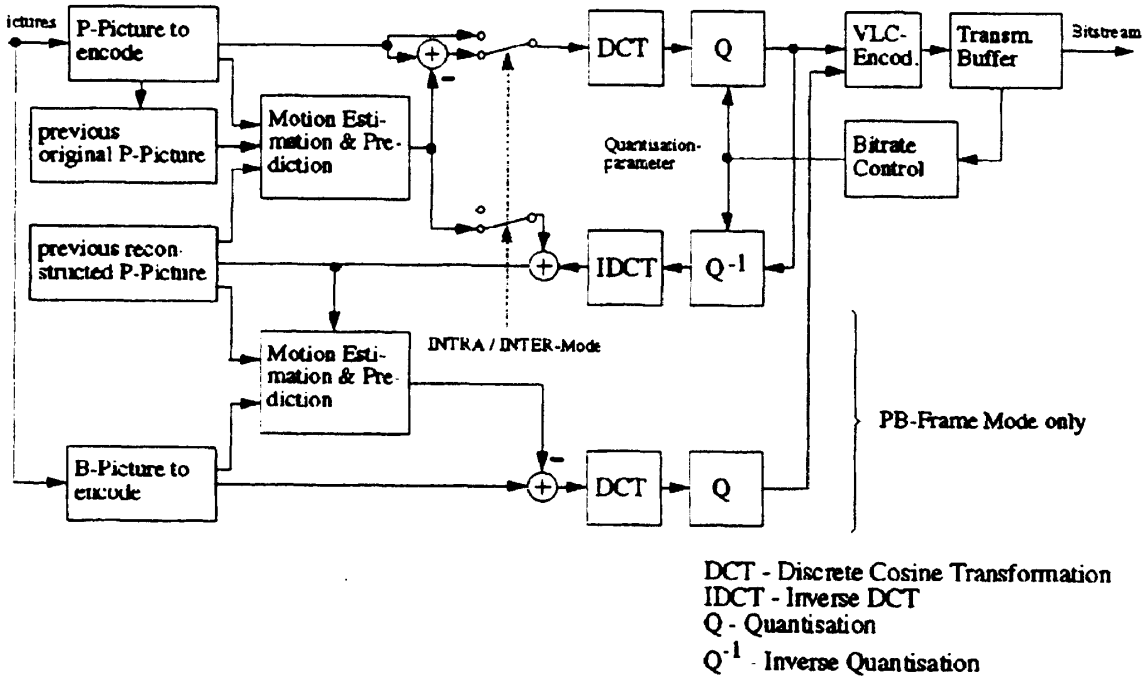
H.263 is a blockbased hybrid code. Each picture is partitioned into macroblocks, rectangles of fixed size containing luminance and chrominance information for this rectangle. It employs motion compensated prediction followed by DCT and quantization of the error signal.

H.263 has optional modes that can be used for coding, if switched on. There are:

- Syntax based arithmetic coding mode
- Advanced prediction mode
- Unrestricted motion vector mode
- PB-frame-mode

Figure 2 shows a block diagram of a H.263 encoder. The lower part of the block diagram is active only in the PB-frame-mode. Incoming pictures are divided into macroblocks. For each macroblock motion estimation and prediction is done on the basis of the reconstructed previous picture. The result is a motion vector on the one hand and the predicted macroblock on the other. Depending on how good the predicted macroblock matches the original one either INTRA or INTER mode is used for encoding. In INTER mode, which is chosen in most cases, the difference between the block to code and the predicted macroblock is calculated. This difference is DCT transformed and quantized using a variable quantization parameter. The resulting quantized coefficients are entropy coded and transmitted together with the motion vector and the other parameters. Additionally the result is fed into an inverse quantifier, is inverse DCT-transformed and the predicted macroblock is added. This is the same as in the decoder, since the encoder must use the same data for prediction as the decoder does. In the INTRA mode no prediction is done, instead the macroblock to code is DCT-transformed directly. The remaining is the same as in the INTER mode.

Figure 2. Block Diagram of a H.263 Encoder

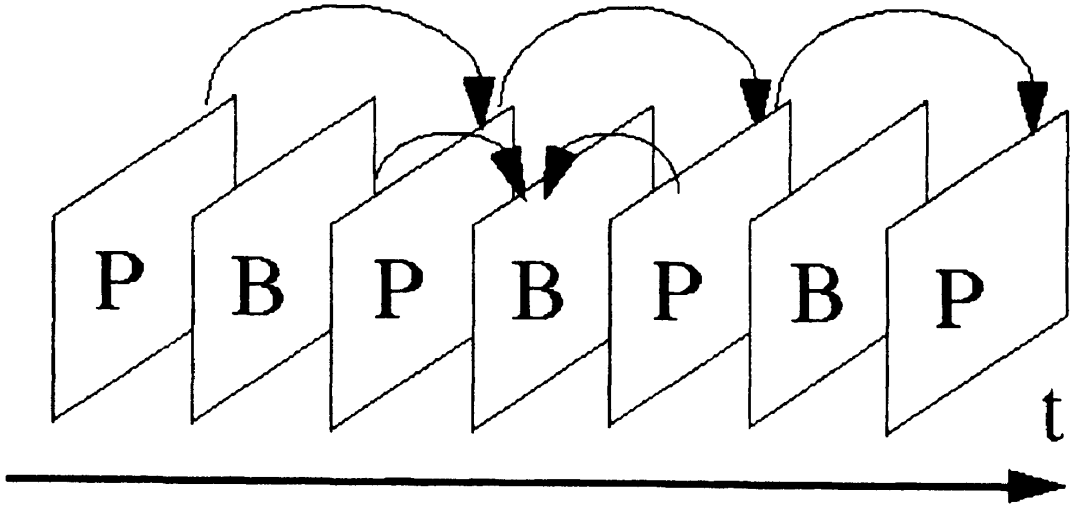


If the PB-frame mode is used, two pictures are coded as one unit called PB-frame. Figure 3 shows the sequence of pictures: Every second picture (marked with P) is coded in the usual manner (as described above), but predicted not from its direct predecessor but from the second previous one. The pictures between (marked with B) are predicted bidirectional - from their predecessor and their successor.

The recommendation H.263 does not describe the whole coding process. It is intended as a standard open for competition and thus only specifies the bitstream syntax, semantics and the decoding process. The operation of the encoder is not specified. The encoder described here is in accordance to the test model TMN5 [4].

In this section only a broad overview was given, more detailed information can be found in the references [3].

Figure 3. Sequence of Pictures in the PB-frame Mode



Parallelization

General Aspects

The MVP is a multiprocessor system. This requires to split the algorithm into separate pieces which can be executed at the same time.

Generally all operations can be executed in parallel, except when there is a data dependency between them. A data dependency in this case means that one operation needs the result of another one. It requires the operations to be executed consecutively therefore limiting the ways of parallelization. Other types of data dependencies, like two operations storing data to the same variable, can be easily broken up by placing the data into different variables.

The aim of parallelization is to break the algorithm into appropriate parts, to ensure that all processors are kept busy all time. This is not the case, if one processor has to wait for another producing some data it needs, and therefore calculation power is wasted. Thus it is the aim to achieve load balance by scheduling the operations in an appropriate way.

There are two ways of scheduling: the order of operations can be chosen when developing the program, which is called static scheduling, or it can be chosen while running the program, called dynamic scheduling. Generally it is easier to implement and debug static scheduling, because a running program will always be executed in the same way. Of course there are situations where it is more appropriate to use dynamic scheduling. This is the case, when the duration of the operations is not known when developing the program or when it may change, for example because of optimizations. Sometimes the duration of operations depends also on the data processed. An example is motion estimation: When full search is done, motion estimation can be done by calculating the sum of absolute distance (SAD) for all motion vectors within a specified area. To speed up motion estimation SAD calculation for a certain motion vector can be terminated when the previously found minimum (for another motion vector) has been reached. The result of this is the same as without modification, but in the second case the point, when SAD calculation can be stopped, varies within a wide range depending on the video sequence examined i.e. the duration of the operation ,,motion estimation“ is data dependent.

The MVP consists of five independent processors with independent instruction streams. This makes it to be a MIMD system, which always needs synchronization for scheduling an operation. To avoid a large synchronization overhead operations should not be of fine granularity.

As mentioned before the PPs require data to process to be in internal RAM. Since internal RAM is very small compared to the amount of data processed by H.263 data has to be loaded into internal RAM before processing and the results of an operation have to be written back afterwards. The bandwidth of the external interface limits the amount of data transfers. When an algorithm is parallelized it has to be paid attention to the amount of data transfers. It does not make sense to waste bandwidth by transferring data to external RAM and immediately back. Avoiding this can be done by grouping operations working on the same data as described below.

As mentioned before the size of the internal memory is limited. The MVP has 6 kbyte of data RAM per parallel processor. The parameter RAM usually can not taken into account for checking if there is enough memory for a certain implementation, since the parameter RAM has to keep the stack, internal parameters, the command tables for the TC and the global variables of small size.

As a result of real-time requirements the coding delay should be minimal. This influences the parallelization as well, since a picture is captured not at once, but as a sequence of pixels.

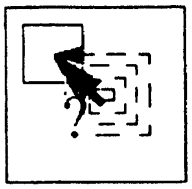
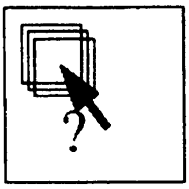
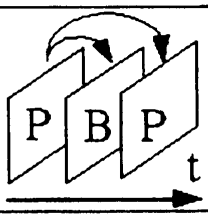
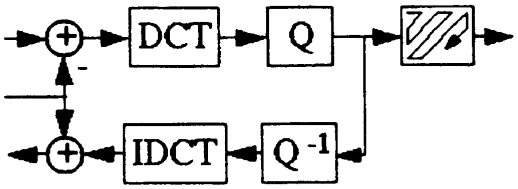
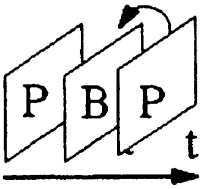
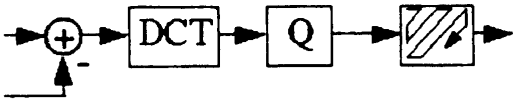
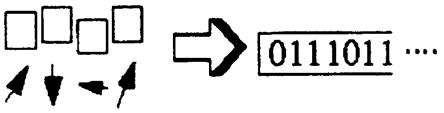
Grouping of Operations into Tasks

H.263 consists of many operations starting with the raw search of motion vectors ending with encoding of the coefficients. It is difficult to keep the overview over all the different possibilities of parallelization, when having too many operations. Additionally the MVP is a MIMD system, which implies that the result of parallelization should not be of too fine granularity. And, of course, it should be avoided to transfer data without any necessity. Hence the operations are grouped according to their data dependency. Operations working on the same data are grouped within one task while operations working on different data or having only a small amount of common data are separated into different tasks.

The data dependencies vary also depending on the coding mode. Decision was made, that the program should be capable of coding in advanced prediction mode and in PB-frame mode. As a result the program structure must take the additional data dependencies into account.

Our implementation of the encoder is in accordance with the TMN5 test model. In this algorithm motion estimation is done in two steps: First a full search is done for all integer motion vectors within the range of +15 to -15 for each component of the motion vector, the motion vector with the smallest SAD is chosen. In the second step all half-pixel vectors surrounding the previously found integer pixel vector are examined. For grouping of the operations it is important to note, that during raw search the block to code is compared with the previous original picture and during fine search with the previous reconstructed picture. Thus both are put into different tasks. The fine search for p- and b-macroblocks work on the same part of the previous reconstructed picture, thus they are put together, but the forward prediction has to be kept separate, because it needs the motion vectors of the right neighbour p-macroblock and can not be done sequentially after fine search. A lot of operations, namely the subtraction of the predicted macroblock, DCT, quantization, zig-zag-scanning, inverse quantization, IDCT and the addition of predicted macroblock all work on just one single block and all on the same. This is ideal for grouping them within the fourth task. The next two tasks are the counterparts to the previous two, when coding a b-macroblock in the PB-frame mode. They work on the data of the b-macroblock and are only executed in the PB-frame mode. Finally, all the operations translating the coefficients, motion vectors and coding information into a bitstream are put into task 7.

Table 1. Division into Tasks

Task 1: Motion Estimation (Raw search, 1 pel accuracy) • Raw search of p-vectors (±15 pel search area) • Calculation of SAD to average • Decision if INTRA or INTER-Mode	
Task 2: Motion Estimation (Fine Search, 0.5 pel accuracy) • Half-pixel search of p-vectors (±0.5 pel around previous min.) • Half-pixel search of delta-b-vector (±2 pel search area) • Decision if 1 or 4 vectors are used	
Task 3: Forward Prediction • Prediction of P-MBs • Forward Prediction of B-MBs	
Task 4: P-MB Processing • Subtraction of the predicted P-MBs • DCT • Quantization • Zig-zag scanning • Inverse Quantization • IDCT • Addition of the predicted P-MBs	
Task 5: Backward Prediction • Backward Prediction of B-MBs • Combination of the result of forward and backward prediction	
Task 6: B-MB Processing • Subtraction of the predicted B-MBs • DCT • Quantization • Zig zag scanning	
Task 7: Encoding • Generation of the „Picture Header“ • Generation of the „GOB Header“ • Generation of the „MB Header“ • Prediction of the vectors • Encoding of the vectors • Encoding of the coefficients	

Data Dependencies Between Adjoining Macroblocks

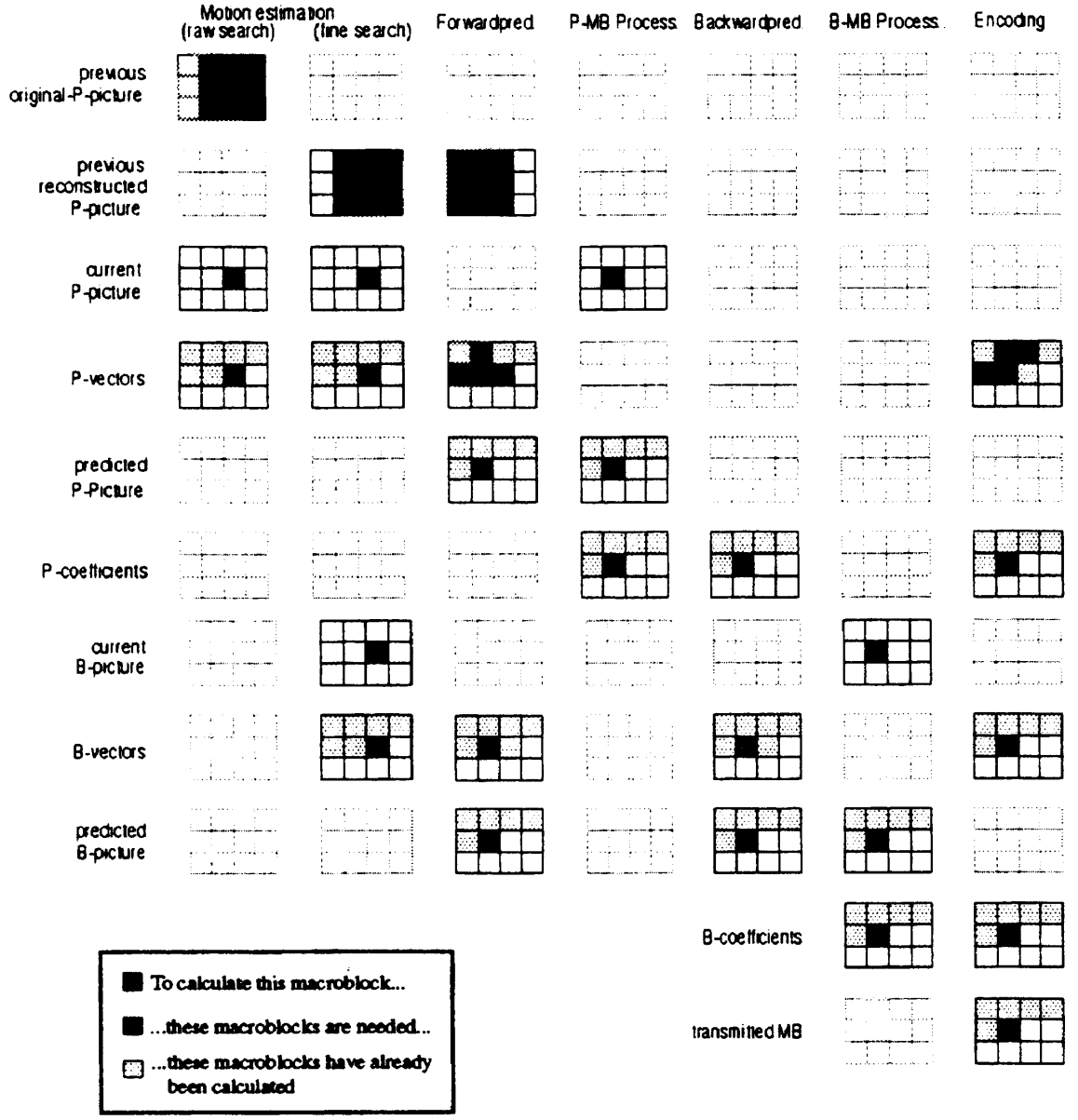
As discussed in the previous section, data dependencies are the key to parallelization. To evaluate and display data dependencies between operations a data flow plan is usually sufficient. In the field of image processing there is another type of data dependencies, which has to be taken into account as well: The data dependencies between different regions of the image, in this case between adjoining macroblocks.

These data dependencies are shown in Figure 4. One box (which is divided into 4*3 smaller boxes) represents a part of a picture, where the small boxes represent one macroblock. The boxes in a row stand for a certain kind of information, for example the contents of the p-picture or the motion vectors for it. A column stands for a certain task. Figure 4 shows the data needed for performing a certain task. The data needed for the task is marked grey, the result is marked black.

For the operation ,,fine search“/,,forward prediction“of course not the complete 3*3 macroblock wide part is needed. A smaller part is needed, which can lie anywhere within this area.

As a consequence of the data dependencies the forward prediction can not immediately follow the motion estimation, because the motion vectors of the left, upper and right macroblock are needed in advanced prediction mode.

Figure 4. Data Dependencies Between Adjoining Macroblocks



Durations of the Tasks

As mentioned before, for parallelization it is necessary to know the duration of the tasks. An exact evaluation of the duration would make it necessary to write the program for the MVP and evaluate it there. Since it was necessary to have this evaluation before writing the MVP program, the H.263 encoder running on a sparc workstation was examined. For each task the percentage of time used is shown depending on the encoder mode. Since the time needed for data transfers is not taken into account the sum of a column in Table 2 is less than 100%.

The syntax based arithmetic coding mode has almost no influence on the duration, hence it is not listed in the table.

Table 2. Duration of the Tasks

	P	P+A	PB	PB+A
Task 1	78,39%	78,90%	69,60%	70,39%
Task 2	7,29%	6,99%	13,93%	13,68%
Task 3	0,47%	1,28%	0,79%	1,43%
Task 4	5,86%	5,03%	4,75%	4,14%
Task 5	0,00%	0,00%	0,58%	0,49%
Task 6	0,00%	0,00%	2,22%	1,94%
Task 7	0,27%	0,23%	0,44%	0,37%

P = PB - frame mode off
PB = PB – frame mode on
A = Advanced prediction mode on

The Implementation

Comparison of the Parallelization Strategies

It is evident, that parallelization must be within one picture, because the coding delay would be excessive otherwise, but apart from that there are several different ways. The four basic ways of parallelization shall now be compared.

The first method is picture-wise parallelization: A task is performed on the macroblocks of the whole picture. The next task is performed, when completed with the previous one, and so one. Picture-wise parallelization is easy to implement, because scheduling is very simple. Since none of the tasks is data dependent to itself applied on a neighbour block any scheduling can be used for a certain task. On the other hand it has two main disadvantages: The coding delay increases by the time needed to code one picture, since the coded data is produced by the last task. The coding delay is an important factor for video communication applications, because it may disturb communication if it is too excessive. Additionally, because of the small size of internal memory, which can hold at most 16 macroblocks, data has to be loaded from external to internal memory before each task and the results have to be written back afterwards. This results in an increased number of transfers.

The second way is pipelining of the tasks. This means that all processors are working in parallel, one processor performing the first task, the second processor performing the second one, and so on. One macroblock is handled in sequence by processor 1, processor 2,… This has the advantage that the coding delay is small, since the picture can be processed in the same sequence the outgoing data stream is transmitted - exactly like a sequential program would do. The scheduling is simple, motion estimation is at least one block ahead of the following task because of the pipeline. The main disadvantage is that the duration of the operations is very different. Raw search takes about 3/4 of the computing time, while all other tasks take only 1/4. To avoid a poor speedup the processors must be distributed in this ratio to the seven tasks. This means that at least 3 processors must do raw search, which would break up the pipelining concept and would make a more difficult scheduling necessary. But there is another reason, why this concept is not ideal: The duration of the operations is not exactly known when developing the program and it may change due to optimizations. This would make it necessary to change the scheduling, when the program has been developed or after optimizations, which is not useful.

The third alternative is macroblock-wise parallelization: Each processor operates on one single macroblock at a time and does all tasks for this macroblock. All processors work in parallel on different macroblocks. When one has completed with a macroblock it starts processing another macroblock. When having a closer look on the data dependencies discussed in *Data Dependencies Between Adjoining Macroblocks* one will notice, that this is not possible because of the data dependency of the prediction: For prediction of a macroblock the motion vectors of the left, upper and right neighbour macroblock are needed. When modifying the scheduling in the way of separating motion estimation (task 1 and 2) from the other tasks, this results in a possible solution, but is also leads to difficult dynamic scheduling. Furthermore additional transfers are necessary because of the separation.

Row-wise parallelization, the fourth alternative, is a compromise between Picture-wise and macroblock-wise scheduling. All operations are performed for one row in parallel by the processors. When completed with the first operation, the next operation is performed on this row. When one row is completely processed it is continued with the next row. The properties reflect this. By contrast to picture-wise parallelization no scheduling modification is needed, since in both cases no data dependencies are between operations processed at the same time. By contrast, the coding delay is much lower, because data output is produced at the end of each row. This is more than the coding delay when using macroblock-wise parallelization, but still relatively small. Only 1 row of macroblocks (plus 1 additional row for the motion vectors) of external buffer is needed for the currently processed row. The speedup is not maximal because of the synchronization needed after each task performed on a row. Because the tasks are not processed as a sequence additional transfers are needed.

Table 3 shows a summary of the properties of the four different strategies.

Table 3. Advantages and Disadvantages of the Four Basic Strategies

Criterion	Picture view	Pipelining	Macroblock wise	Row-wise
Processor load	balanced	balancing not possible	balanced	good, but not optimal
Speedup	good	poor	good	good, but not maximal
Dynamic scheduling	possible	impossible	difficult	possible
Scheduling modification	no	necessary if prog. changed	necessary	no
Rate control	picture wise	macroblock wise	depending on scheduling	row wise
Transmission buffer size	big	small	small	relative small
Coding delay	big	small	small	relative small
External buffer	1 picture	no	no	2 rows
Additional transfers	yes	no	yes	yes
Extendability of the algorithm	good	difficult	difficult	difficult

Description of the Implementation

As discussed in the previous chapter operation-wise parallelization and pipelining have strong disadvantages. Macroblock-wise parallelization by contrast can be an alternative. However, for this implementation row-wise parallelization was chosen, because it can be done with static scheduling and because it can be modified in that way, that the number of transfers to external memory can be reduced significantly.

One row is processed by all four DSPs at a time. Task 2 to 6 are performed in a sequence, since this reduces the number of transfers. Internal memory is only capable of containing 6kByte which is equivalent to 16 macroblocks. Thus it is necessary to execute task 1 separately from the others. Encoding is done by the master processor, because it can be easily splitted off and the master processor shall be utilized as well.

The result is the scheduling strategy displayed in Figure 5.

Figure 5. Row-wise Scheduling

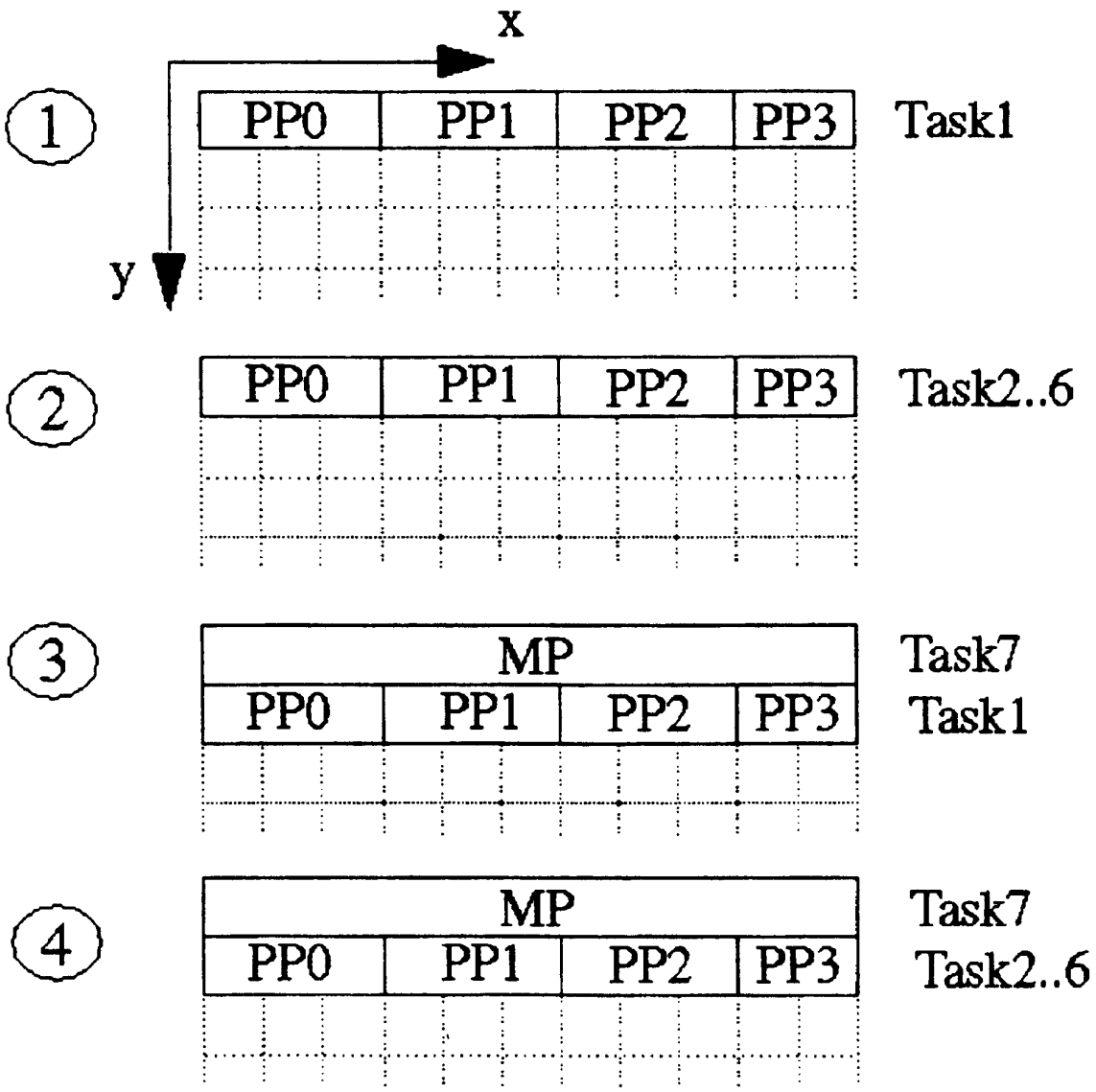
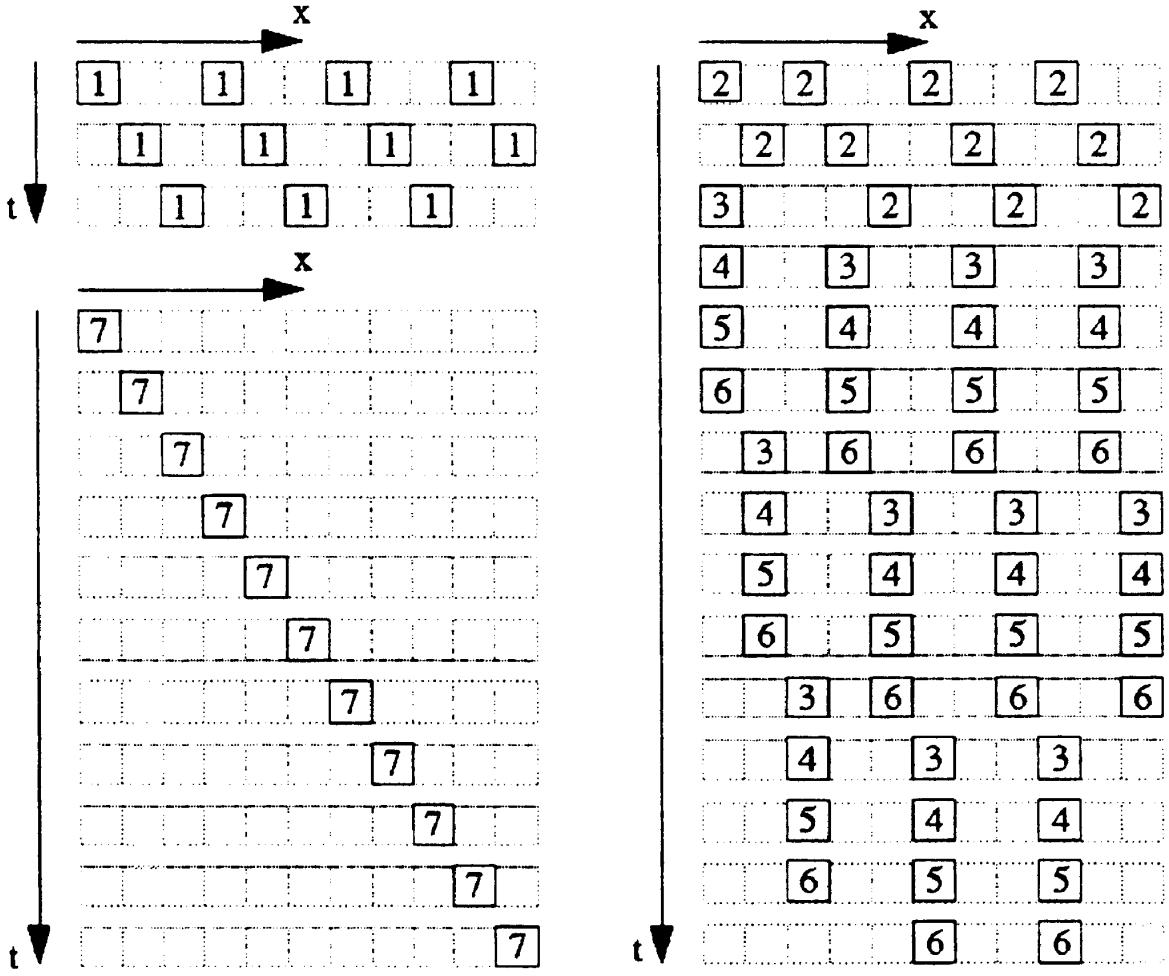


Figure 6 shows the scheduling within a row. The scheduling for task 1 and task 7 is simple: The macroblocks are processed sequentially. For task 2 to 6 scheduling is more complicated. At the beginning fine search is executed for the left, the current and the right macroblock. Having done this task 3 to 6 follow for the current macroblock. Afterwards the current position is shifted right by one macroblock. Then fine search is done for the right macroblock. For the left macroblock and for the one at the current position fine search has already be done in the previous step. Afterwards task 3 to 6 are performed for the macroblock at the current position. This continues until the right end of the area to process is reached. During this process task 2 is not executed for blocks, where it has been executed before, instead the previously calculated results are used.

Figure 6. Scheduling Within a Row



The usage of internal memory is shown in Table 4. The amount of internal memory needed of each task is shown in number of macroblocks. The buffer for the original P- and B-macroblock, for the P- and B-coefficients and for the reconstructed P-macroblock are not needed at the same time and therefore can be at the same place. They must not be counted twice, which is symbolized by brackets. The buffer for the currently processed macroblock must be capable of holding 16 bit values and thus its size must be 2. After prediction has been done the left column of the 4*3 part of the reconstructed picture is no longer needed and thus it can be used for another purpose.

When adding the memory required it turns out, that in the worst case an amount of 14 macroblocks is needed at the same time. This fits into the internal RAM.

Table 4. Usage of Internal Memory During the Execution of Task 2 to 6

	Task 2		Task 3		Task 4		T.5	T.6
	N	PB	N	PB	N	PB	PB	PB
original P-MB	1	1			1	1		
original B-MB		(1)						1
4*3 MB part of reconstructed picture	12	12	12	12	9	9	9	9
predicted P-MB			1	1	1	1		
predicted B-MB				1		1	1	1
currently processed MB					2	2		2
coefficients of P-MB					(1)	(1)		
coefficients of B-MB								(1)
reconstructed P-MB					(1)	(1)	1	
Total:	13	14	13	14	13	14	11	13

Results

The current implementation of the encoder is basically written in ,,C“. Motion estimation and some time consuming parts of task 4 have been assembler optimized. A sequential and a parallel version have been implemented to allow comparing the execution time. For encoding the pictures 0 to 123 of the QCIF sequence ,,Susie“ have been used. The sequence was encoded on a C80 running with 40MHz clock without using optional coding modes. Table 5 shows the duration of the tasks for processing one row of macroblocks, Table 6 shows the total CPU time (which is the sum of the CPU-time of the processors) and the encoding time (which is the time it actually took to run the program) per picture. The encoding time of the parallel version is 29095ms. This equivalent to a frame rate of 4.26 frames per second.

Table 5. Duration of the Tasks

	sequential version		parallel version	
	INTRA	INTER	INTRA	INTER
Task 1	0.34 ms	55.24 ms	1.16 ms	55.96 ms
Task 2..6	20.43 ms	25.32 ms	24.92 ms	30.80 ms
Task 7	6.13 ms	4.99 ms	7.32 ms	5.04 ms

Table 6. Duration per QCIF Picture

	sequential	parallel
Total CPU-Time	94951 ms	101908 ms
Encoding time	94951 ms	29095 ms

The resulting speedup is 3.26. There are two reasons, why the speedup is smaller then the number of processors: On the one hand the CPU-time is prolonged, since loading of the instruction caches and data transfers are delayed because of collisions, on the other hand the processors are not utilized all the time.

Conclusion

The coding speed of the current implementation is below real-time. Since there are time consuming parts of the program written in ,,C“ there is still room for assembler optimizations.

To answer the question, how fast an encoder according to TMN5 can be, the maximal picture rate was estimated. The part which needs the most processor time, the raw search was coded in assembler and optimized. Different assembler versions have been compared. On the assumption, that the whole program could be optimized to that degree and that full speedup is reached the maximum picture rate is displayed in Table 7.

Table 7. Estimation of Maximal Reachable Speed for a TMN5 H.263 Encoder Running on a MVP

Advanced Prediction Mode	PB-frame mode off		PB-frame mode on	
	Ceasing of SAD-calculation	Multiple arithmetic	Ceasing of SAD-calculation	Multiple arithmetic
Off	7,93	5,17	8,83	5,76
On	3,99	5,17	4,446	5,76

For each combination of modes two different assembler versions turned out to give best results. Both are displayed in the table above. The version ,,ceasing of SAD-calculation“ operates like mentioned in the *Parallelization* chapter, the version ,,multiple arithmetic“ exploits special features of the DSPs within the MVP, namely the possibility to split the 32bit wide ALU into four 8bit wide parallel operating ALUs. A combination of both would be less efficient. The details of this estimation can be found in [8] .

Though this estimation was done at a time, when the encoder was completely written in C and compared to the results given in Table 7 it turned out to be too pessimistic, the table shows clearly that real-time requirements can not be reached when TMN5 is implemented. The consequence must be algorithmic changes. As shown in the section, *Durations of the Tasks*, the raw search needs most of the computing time, it offers the biggest potential for time saving. Thus the further work is to substitute full search and use hierarchical algorithms for motion estimation [5] or other enhanced algorithms e.g. [6]. A slightly modified ,,C-version“ of the algorithm proposed in [5] has been implemented and it has shown to be about 6 times faster than the assembler version of full search.

Literature

[1] T. Bräunl: ,,Parallele Programmierung“ (in German), Vieweg Verlag, 1993

[2] J. Pitas: ,,Parallel Algorithms for Digital Image Processing and Neuronal Networks“, John Wiley & Sons, 1993

[3] ITU-T Recommendation H.263: ,,Video coding for low bitrate communication“, International Telecommunication Union, 1996

[4] ,,Video codec test model, TMN5“, http://www.nta.no/brukere/DVC/tmn5/, Telenor Research, 1995

[5] K. M. Nam, 3-S. Kim, R-H. Park, Y. S. Shim: ,,A Fast Hierarchical Motion Vector Estimation Algorithm Using Mean Pyramid“, IEEE Transactions on Circuits and Systems for Video Technology, Vol.5, No.4, August 1995

[6] B. Liu, A. Zaccarin: ,,New Fast Algorithms for the Estimation of Block Motion Vectors“, IEEE Transactions on Circuits and Systems for Video Technology, Vol.3, No.2, April 1993

[7] ,,TMS320C80 Multimedia Video Processor (MVP) - Technical Brief“, SPRU106, Texas Instruments

[8] H. Mooshofer: ,,Untersuchung und Implementation des H.263 Videokommunikationsstadards auf dem MVP“, TU München, Lehrstuhl für Integrierte Schaltungen, 1996