VERIFYING EXTERNAL DATA MEMORY INTERFACE FOR H.263 VIDEO DSP WITH MEMORY SIMULATOR

J. Alakarhu, J. Niittylahti, T. Sihvo and J. Tanskanen

Digital and Computer Systems, Tampere University of Technology
P. O. Box 553, Tampere, FINLAND
{juha.alakarhu, jarkko.niittylahti, tero.sihvo, jarno.tanskanen}@tut.fi

ABSTRACT

In this paper, we present the simulator-based method to estimate the time required by the external data memory accesses in the H.263 video encoding. Different frame rates and picture resolutions are considered. The Video DSP structure considered here consists of several parallel on-chip DSP units and it is optimized for the H.263 video encoding. Execution time is coarsely divided between control/non-sequential processing, parallel processing, and external data memory traffic. To evaluate the performance in early design phase, one must find out the time required by each part. The memory simulator method described here gives an estimate of the time required by the external memory accesses. With this estimate, one can also make sure that the proposed partitioning between internal and external data memories is correct and the required memory bandwidth for the external data memory is not too high.

1. INTRODUCTION

Since video processing requires huge processing power and evolving video coding standards have a need for more flexible architecture, programmable on-chip parallel processor architectures [1] are a very attractive solution. Fortunately, many parts of the video coding algorithms can be easily processed in parallel. E.g., the operations of the hybrid coding scheme (like H.263) can be divided into low level tasks, medium level tasks, and control operations [2]. The low-level tasks are computationally intensive, regular operations with data-independent function [3], thus lending themselves to the single instruction multiple data (SIMD) type of parallel processing. The low-level tasks have completely regular data and control flow, but they require more than 85% of the overall computational rate of the hybrid coding scheme [4]. Since the data flow is predictable in video processing, low memory hierarchy can be used [5]. To provide the high memory bandwidth required by the parallel processors, parallel memory modules are usually used [6-10]. A single frame to be processed may not fit into the fast internal memory. Thus, a larger and slower external, frame buffer is needed. The division between internal and external data memories should be done properly, so that the required bandwidth between them is realizable.

By using a high-level software model of the system we can achieve a proposal of the division. After that, using a memory simulator, we can verify that this division is fast enough. The simulation-based method presented in this paper makes it possible to do the division in early design phase before actual implementation. The method is very quick, no clock cycle accurate VHDL or Verilog models are needed. The simulator also helps to select an optimal DRAM device for the external memory.

2. SYSTEM ARCHITECTURE

We use a small, low-power, fixed point DSP core with 16-bit data path as a building block of the system. Programmable processor cores and the parallel memory establish the scalable macroblock coding engine. Programmability makes it possible to execute also other video coding applications than just H.263 encoding considered here. In the design phase, the number of the cores (here 1, 2, 4 or 8) used for the SIMD processing can be selected according to the required processing power, making the architecture scalable. Using simple similar cores (instead of dedicated HW-blocks) according to predefined system concept may speed up the designing for different kind of video coding applications.

A block diagram of the proposed video DSP architecture is shown in Figure 1. The architecture consists of n SIMD type parallel processors, a control processor, a direct memory access (DMA) controller, internal parallel data memory with m memory modules, and external data memory or frame buffer. In the following, only data memory is considered. The DMA controller operates between the external memory and the internal parallel memory. The DMA also loads the video sequence to be encoded to the external memory. The input sequence is read to the external memory macroblock by macroblock basis. The encoded bitstream is not stored in the external memory. The internal address space of the parallel memory is divided into three areas for different coding purposes [10]. It is assumed that

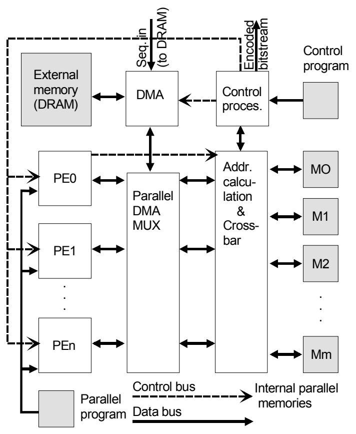

Figure 1: The proposed video DSP architecture.

two of the parallel memory areas can be accessed simultaneously to increase the internal memory bandwidth. The control processor controls SIMD parallel processors and the DMA controller. In addition, the control processor is assumed to be capable performing bit-stream parsing and variable length coding (VLC). The tasks considered for the parallel processing are the sum of absolute difference (SAD) calculation for 16×16 blocks, the calculations for choosing the coding mode for the macroblock, interpolation of the macroblock, SAD for 16×16 blocks in half-pixel motion estimation (where the interpolated search-area is used), macroblock subtraction and addition, discrete cosine transform (DCT) and its inverse (IDCT), and clipping function (for the macroblock after motion compensation or IDCT).

2.1 Modeling the system architecture

The encoder model used here is based on the H.263 [11] encoder source code version 2.0 from Telenor. For simplicity, all the optional coding modes were discarded. The code was modified so that the tasks to be processed in parallel by the SIMD processors were capsulated to their own functions. These functions process only data in defined two-dimensional tables. The tables represent internal memory areas. External memory is modeled with a file. In our source code, DMA operations were modeled by DMA functions, which take care of the references between the two-dimensional tables and the reserved frame storage file. The referred addresses from these parts of the code were stored in to a file, which acts as the input file for the memory simulator.

Internal parallel memory has 7168 16-bit memory locations. The search area [-15, 15] for the motion estimation and some macroblocks fit into the parallel memory at the same time. The parallel memory consists of dual-port memory blocks, allowing simultaneous DMA and parallel processing operations. The external memory contains four frames. The frames are stored in a row by row order starting from the upper left corner of the picture and ending to the lower right corner. In each frame, the luminance picture comes first and the two corresponding chrominance pictures after that. Space is allocated for the frame being currently processed, for the next frame to be processed, for the previously processed frame, and for the frame being currently reconstructed.

There were at most five different locations in encoding chain where the transfers between the two-dimensional tables and the frame storage file were needed:

A. storing the reconstructed macroblock to the frame storage (384 bytes),
B. storing the macroblock to be coded to the frame storage (384 bytes),
C. loading the chrominance blocks corresponding to the best matching luminance block from the frame storage (we decided to interpolate also the chrominance blocks and needed $10 \times 10 + 10 \times 10 = 200$ bytes instead of just $8 \times 8 + 8 \times 8 = 128$ bytes),
D. loading the search-area from the frame storage for the motion estimation (the number of bytes depends on the size and the location of the search area),
E. loading the macroblock from the frame storage for the encoding (384 bytes).

Memory transfers are all block transfers. Items expressed with capital letters in the list above will be referred in the rest of this paper. In Figure 2, scheduling and overlapping according to different dependencies of considered functions is shown. Parallel processing is assumed to run all the time. Parallel functions are executed sequentially and it is beneficial, if there is not much delay between them. Thus, e.g., the listed DMA operations should take at most the same time as scheduled for the parallel functions.

3. MEMORY SIMULATOR

A simulator capable to evaluate the performance of dynamic RAMs has been created. The problem with this memory type is that its performance may vary dramatically depending on the memory references being accessed. Therefore, simulations are required to find out the achievable bandwidth in the application. This kind of executable simulator does not require any external simulator environment, needed when using, e.g., VHDL-models. It makes it easy to test various design alternatives.

The memory simulator yields the time and the clock cycles spent by the external memory traffic. This information

- helps to make sure in early phase of design that memory traffic between internal and external memory is tolerable,
- helps in scheduling the external memory transfers with processing, when also the time spent in control/sequential processing and parallel processing is known.

There are three memory architecture families supported by the simulator. The first one is the asynchronous DRAMs (FPM, EDO) and the others for synchronous DRAMs (SDRAM, SGRAM, DDR-SDRAM) and Rambus DRAMs (Concurrent, Base, Direct). The simulator can be used to select the most suitable architecture and device for the given application. Dynamic RAMs are modeled using 15-18 parameters depending on the architecture. The parameters assign information about the address mapping, the cell-size, the bus, and the timing of the DRAM device being simulated.

The simulation is based on an address trace. It is given in a file that contains the address and direction (read or write) of each access. Using the address trace as input for the simulation makes the simulator very general purpose in nature for various kinds of applications. The simulator is able to calculate the average bandwidth, the average write and read latencies, and the total time required to execute all the addresses.

The VHDL and Verilog models provided by DRAM manufacturers were used to verify the accuracy of the simulator.

	Available time for coding one inter macroblock										
DMA	**A, B**	**C**		**D, E**							
VLC								ZigZag, VLC			
Control	ME		FindHalfPel		MB_Pred					MB_Reconstruct	
Parallel	ME	Mode	Interpolation	HalfPel ME	MB -	DCT	Q	IQ	IDCT	MB +	Clip

	Available time for coding one intra macroblock				
DMA	**A, B, E**				
VLC			ZigZag, VLC		
Control					
Parallel	DCT	Q	IQ	IDCT	Clip

Figure 2: The scheduling of functions for intra / inter MB coding. Capital letters in DMA row refer to the list mentioned.

We found that the error is a few percents at maximum. This is satisfactory for this purpose.

4. SIMULATION RESULTS

Two test sequences were encoded to estimate the time consumed by the external memory. We used the memory simulator to evaluate the performance with Micron MT48LC8M8A2-8E SDRAM [15] and Micron MT4LC8M8E1_8M-6 FPM DRAM [14], which have an eight bits wide external data bus. Moreover, a memory interface with a 16 bits wide interface has been examined. MT4LC1M16E5_1Mx16-7 SDRAM [13] and MT4LC1M16C3_1Megx16-7 FPM [12] from the same manufacturer were used for this purpose. SDRAMs use 100 MHz bus and FPMs 66 MHz bus. FPM DRAM is one of the oldest and slowest DRAM architectures available at present. Its results present the worst case in this sense. On the other hand, SDRAM presents a modern mass-production DRAM.

The first test sequence (Dancer) contains 100 subsequent 352x288 common intermediate format (CIF) frames. Encoding this, using the full-search [-15,15] motion estimation algorithm, produces 137783032 external eight bits wide memory accesses. 69410016 memory references were generated when using 16 bits wide words. The other sequence (Claire) contains 100 subsequent 176x144 quarter CIF (QCIF) frames. This sequence lead to 33006600 references with eight bits wide words and 16679920 references with 16 bits wide words (full-search [-15,15] motion estimation algorithm). Also the three-step [-7,7] motion estimation algorithm was tested. In this case, the Dancer sequence lead to 91491320 (8-bit) and 46366580 (16-bit) external memory references. On the other hand, encoding Claire lead to 22412808 (8-bit) and 11383644 (16-bit) DRAM references. A new search-area was loaded for each macroblock. This causes loading a lot of the same data. Optimizing this could lead to significant savings in memory references.

Figures 3 and 4 coarsely show the results obtained by simulating the DRAM accesses. If the desired frame rate is 30 fps, the maximum time for the accesses is 3.33 s, because there are 100 frames to be encoded. This is provided that the DMA is able to operate totally concurrently with the other tasks. According to our study, this is a relevant presumption. An example of simulations verifying this is shown in Figure 5. First ten frames of the Claire sequence were encoded using the three-step algorithm. In this case, the number of the SIMD processors was assumed to be four. With a 100 MHz clock frequency, they seem to be able to code more than 30 QCIF frames per second. Clock cycles for the parallel functions were obtained with a DSP's instruction set simulator. Assembler code was compiled from our C-model and partly hand-optimized. Since the simulator does not support simulating of parallel cores, we verified the functionality of the code for a single DSP core and scaled the cycle counts of parallel functions for different number of processors. The scaling factors were obtained from the different simulations of the parametrizable (according to the number of processors) hand written assembler codes.

As one can see, the performance of SDRAM is adequate in all of the simulated cases. It can be used with an eight bits wide data bus. Then, in the worst case, the DRAM accesses take about half of the maximum allowed time. Also a lower frequency can be considered to raise the bus utilization and decrease the power consumption. Instead, FPM DRAM seems to be too slow in some cases. The time consumed for the DRAM accesses by this architecture using full search [-15,15] is about double to what is allowed. This architecture could be suitable with a 16 bits wide data bus, especially with the three-step [-7,7] algorithm. The larger search area used with the full-search algorithm causes more memory traffic. Therefore, the memory bandwidth requirement with it is higher.

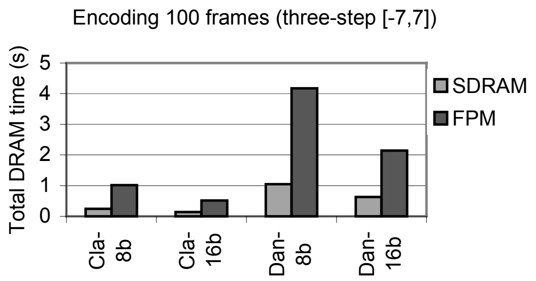

Figure 3: Total DRAM time when encoding 100 frames with three-step motion estimation algorithm

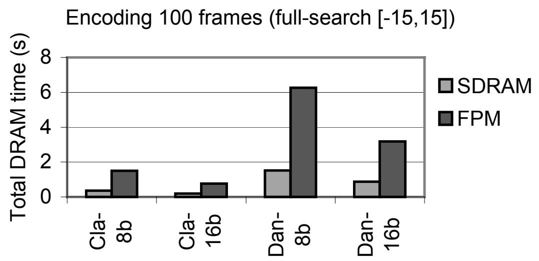

Figure 4: Total DRAM time when encoding 100 frames with full-search motion estimation algorithm

The SDRAM architecture has much better performance than FPM in this application. Modern DRAMs indeed achieve much better bandwidth than their older counterparts with this kind of application (a lot of regular accesses). The performance of the external memory depends mostly on the bandwidth. The somewhat poor latency of the new architectures is not a problem. One should also notice how well the wider data bus could be exploited. The total DRAM time almost halved with the 16 bits wide data bus. Again, this happens due to the regular accesses to large areas in the memory space.

5. CONCLUSIONS

The performance of modern DRAMs is sensitive to the memory trace. Therefore, it is important to test their performance in the application before the implementation. The memory simulator method described here proved to be very useful. We could make sure that the required performance for the external memory devices is not too high in general. Moreover, it makes it easy to select the most suitable DRAM architecture and configuration. However, although the simulator can be considered very fast, the simulation time is an order of an hour with traces containing over hundred million accesses. This limits experiments slightly. It

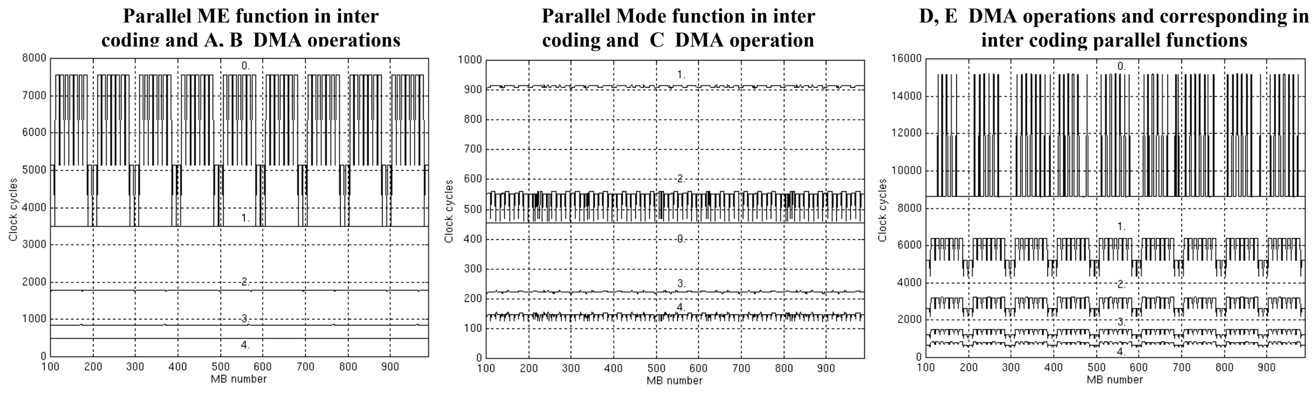

Figure 5. An example of simulations results for parallel functions and DMA operations. Ten frames from the beginning of sequence Claire were coded (results for first intra are not shown). In figures curve 0. corresponds parallel function(s) and other curves DMA operations with different DRAM memory types ; 1. FPM - 8 b , 2. FPM - 16 b, 3. SDRAM - 8 b, and 4. SDRAM - 16 b.

may be a good idea to make a lot of trials with shorter traces. Then, select the most suitable ones and simulate them with longer traces.

We discovered that modern DRAMs perform very well in this kind of application that contains a lot of regular accesses to somewhat large memory area. Here the relatively long time to start a burst does not have very notable effect on the total performance of the memory system.

We have presented a video DSP architecture and verified that its performance is adequate for H.263 video encoding with 30 QCIF frames per second. The verification is based on models of parallel processing, sequential processing, and DRAMs. The models allow performance evaluation in an early design phase. This way it is possible to make sure that the architecture is suitable before the implementation.

ACKNOWLEDGEMENTS

We would like to thank VLSI-Solution for their co-operation and support.

REFERENCES

[1] K. K. Parhi and T. Nishitani, editors, *Digital Signal Processing for Multimedia Systems, Signal Processing Series*. Marcel Dekker, Inc., New York, Basel, U.S.A., 1999.

[2] K. Gaedke, H. Jeschke, and P. Pirsch, “A VLSI based MIMD architecture of a multiprocessor system for real-time video processing applications,” *Journal of VLSI Signal Processing*, vol. 5, nos. 2/3, pp. 159-169, April 1993.

[3] S. Y. Kung and Y.-K. Chen, “On architectural styles for multimedia signal processors,” in *Proc. 1st IEEE Workshop on Multimedia Signal Processing*, Princeton, NJ, U.S.A., June 1997, pp. 427-432.

[4] P. Pirsch and T. Wehberg, “VLSI architecture of a programmable real-time video signal processor,” in *Proc. of SPIE Conf. Digital Image Processing and Visual Communication Technologies in the Earth and Atmospheric Sciences, SPIE* vol. 1301, 1990, pp. 2-12.

[5] S. Dutta, W. Wolf, and A. Wolfe, “A Methodology to Evaluate Memory Architecture Design Tradeoffs for Video Signal Processors,” *IEEE Trans. Circuits Syst. Video Technol.,* vol. 8, no. 1, pp. 36-53, Feb. 1998.

[6] K. Rönner and J. Kneip, “Architecture and applications of the HiPAR video signal processor,” *IEEE Trans. on Circuits Syst. Video Technol.,* vol. 6, no. 1, pp. 56-66, Feb. 1996.

[7] J. Kneip, K. Rönner, and P. Pirsch, “A data path array with shared memory as core of a high performance DSP,” in *Proc. Int. Conf. Applicat. Specific Array Processors,* San Francisco, U.S.A., Aug. 1994, pp. 271-282.

[8] H. Yamauchi, Y. Tashiro, T. Minami, and Y. Suzuki, “Architecture and implementation of a highly parallel single-chip video DSP,” *IEEE Trans. on Circuits Syst. Video Technol.,* vol. 2, no. 2, pp. 207-220, June 1992.

[9] M. Gössel, B. Rebel, and R. Creutzburg, *Memory Architecture & Parallel Access,* Elsevier, Amsterdam, 1994.

[10] J. Tanskanen and J. Niittylahti, "Parallel Memories in Video Encoding," in *Proc. 1999 IEEE Data Compression Conference,* Snowbird, Utah, U.S.A., pp. 552, March, 1999.

[11] Draft ITU-T Recommendation H.263. Line transmissions of non-telephone signals. Video coding for low bitrate communication, May 1996.

[12] Micron, Inc., *1 MEG x 16 FPM DRAM,* Datasheet, 1999.

[13] Micron, Inc., *1 MEG x 16 SDRAM,* Datasheet, 1999.

[14] Micron, Inc., *8 MEG x 8 FPM DRAM,* Datasheet, 1999.

[15] Micron, Inc., *64 Mb: x4, x8, x16 SDRAM, Datasheet,* 1999.