Co-design and Implementation of the H.264/AVC Motion Estimation Algorithm Using Co-simulation

R.R. Colenbrander, A.S. Damstra, C.W. Korevaar, C.A. Verhaar, A. Molderink
Faculty of Electrical Engineering, Mathematics and Computer Science
University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands
e-mail: a.molderink@utwente.nl

Abstract

This paper proposes an efficient implementation of the H.264/AVC motion estimation algorithm in hardware and software. Furthermore, a complete co-design trajectory from the HW/SW partitioning to the actual implementation on two different targets is shown. A Leon 3 + FPGA and an ARM + Montium implementation have been successfully realized. The FPGA implementation shows a speed-up of 43.6× whereas the Montium implementation shows a speed-up of 21.5×, both compared to a software-only implementation. Power consumption is 42.0 mW for the FPGA and 60.2 mW for the Montium. A co-simulation tool, CosiMate, is used to achieve both on target implementations in just five weeks.

Keywords: H.264/AVC, motion estimation, HW/SW co-design, co-simulation, multiple target implementation

1. Introduction

The H.264/Advanced Video Coding codec is widely used for video applications, for example in HDTV, but also in mobile video and internet applications. The increasing video resolutions and the increasing demand for real-time encoding require the use of faster processors. However, power consumption should be kept to a minimum. Using profiling, we found that the motion estimation algorithm is the most computationally intensive part of the encoder. We implement this part efficiently on two different targets. In doing this we show a complete co-design trajectory from the partitioning to the actual implementation. Besides an implementation on a FPGA, the Montium seems an interesting target to compare to, based on the results in [3]. Co-simulation was used to perform continuous functional testing of the HW/SW co-design.

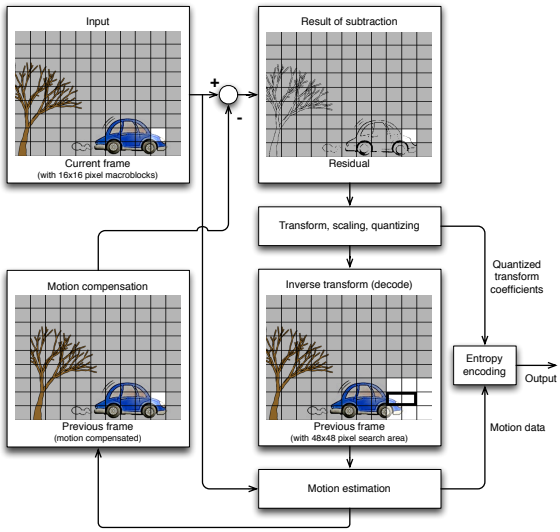


Figure 1. Illustration of the H.264/AVC encoder

1.1. The H.264/AVC codec

H.264/AVC is an industry standard for compressing video [1]. The H.264/AVC standard was first published in 2003. It offers better compression efficiency and greater flexibility in compressing, transmitting and storing video. Compared with standards such as MPEG-2 and MPEG-4 Visual, H.264/AVC can deliver better image quality at the same compressed bitrate, or deliver the same image quality at a lower bitrate. The improved compression performance comes at the price of increased computational cost [2].

The H.264/AVC encoder contains three steps: prediction, transformation/quantization and entropy encoding. Motion estimation is part of the prediction step.

Figure 1 illustrates the H.264/AVC encoder. In the top left corner the current frame for a video is shown. A frame is divided in 16×16 pixel macroblocks.

Although the entire frame is shown, we illustrate the encoder only for the white macroblock. The motion estimator has two inputs: a macroblock (MB) from the current frame and a 48×48 pixel search area (SA) from the previous frame (shown in the bottom right of figure 1). The motion estimator finds the best matching block in the search area. The H.264/AVC standard does not specify how this should be implemented. A possible implementation is shown in the following pseudocode:

```
for(MB=0; MB < 1089; MB++) {
  for(pixel=0, SAD=0; pixel < 256; pixel++) {
    SAD += abs(MB[pixel] − SA[pixel]);
  }
  if(SAD < bestSAD)
    bestSAD = SAD;
}
```

Two nested loops are used to iterate over all possible comparisons. The comparisons are evaluated by using the sum of absolute differences (SAD). Encoding one second of a 176×144 pixel QCIF video (99 macroblocks per frame) at 25 frames per second requires $1089 \times 256 \times 99 \times 25 = 689,990,400$ SAD operations. The vector between the position of the macroblock in the current frame and the position of the best matching block in the previous frame is called the motion vector. The current macroblock and the best matching block are subtracted, shown in the top right of figure 1. The result is a residual, which is transformed, scaled and quantized. The quantized transform coefficients and the motion data are then stored or transmitted.

The H.264/AVC standard allows for seven modes with variable block sizes: a 16×16 pixel macroblock can be divided in smaller blocks to yield a better compression efficiency.

1.2. Target platforms

Some algorithms perform best on fine-grain reconfigurable architectures whereas others perform better on coarse-grain reconfigurable or general purpose processing (GPP) tiles. Our two target platforms are Leon 3 + FPGA and ARM + Montium [3]. Both incorporate a GPP and a reconfigurable architecture. The general purpose processors we use are an ARM and a Leon 3. The Leon 3 is an open-source synthesizable VHDL model of a 32-bit processor compliant with the SPARC V8 architecture. Two types of reconfigurable architectures are used: fine-grain (FPGA) and coarse-grain (Montium).

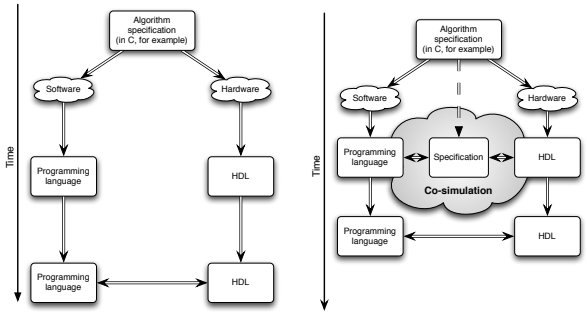
The Montium Tile Processor (TP) is a 16-bit word level reconfigurable architecture that obtains significant lower energy consumption than DSPs for fixed-point digital signal processing algorithms. The Montium TP targets computationally intensive algorithm kernels that are dominant in both power consumption and execution time. In contrast to a conventional DSP, the Montium TP does not have a fixed instruction set, but is configured with the functionality required by the algorithm at hand. The Montiums performance and energy efficiency are comparable to ASIC. The Montium TP has a low silicon cost, as the core is very small. For instance, the silicon area of a single Montium TP with 10 KB of embedded SRAM is 2.4 mm^2 in 0.13 μm CMOS technology. A Montium TP consists of five identical ALUs to exploit spatial concurrency in order to enhance performance [3].

The rest of this paper is organized as follows. Section 2 gives a brief overview of the related work on motion estimation and co-simulation. Analysis results are presented in section 3. Our proposed solution is divided in two parts, a virtual prototype and multiple target implementations, and can be found in section 4. Results are presented in section 5. In section 6 we present the conclusions and finally section 7 gives suggestions for future work.

2. Related work

According to [4] and [5] motion estimation is the most computationally intensive part in a typical video encoder. In recent years many architectures have been proposed for more efficient motion estimation in H.264/AVC, where HW/SW co-design is used to move the motion estimation algorithm to parallel hardware. Song et al. [6] focus on the algorithm itself, whereas other papers implement their proposed solution in hardware. Different hardware architectures have been proposed, such as ASIC, GPU and FPGA. The focus is on achieving the highest data reuse and minimal operation redundancy to achieve highly efficient motion estimation. While the objective in [5] and [7] is to outperform a software-only implementation, Ou et al. [4] aims at meeting the real time requirements of new video applications at the lowest possible clock speed. In [8] the main goal is to achieve low power consumption. The characteristics of these works are summarized in table 2. In this table the values for 'Max. fps QCIF' represent the maximum number of QCIF frames that can be processed at the specified clock frequency. The 'MHz 30 QCIF' values give the minimum clock frequency needed for processing QCIF at 30 fps.

In figure 2 two co-design approaches are identified. The first co-design approach is to develop both the software and hardware separately. Verification does not



(a) Traditional approach (b) Co-simulation approach

Figure 2. Co-design approaches

Table 1. Profiling results

	Foreman		Husky	
Function	**Samples**	**%**	**Samples**	**%**
FPBlockMotionSearch	203,685	32	464,592	45
dct_luma	69,365	11	73,475	7
biari_encode_symbol	45,455	8	93,972	9

(a) Fixed Block Size Motion Estimation (Mode 1)

	Foreman		Husky	
Function	**Samples**	**%**	**Samples**	**%**
FPBlockMotionSearch	1,364,340	64	3,044,848	73
SATD	111,033	5	121,831	3
UMVLineX	86,373	4	233,444	6

(b) Variable Block Size Motion Estimation (Modes 1-7)

take place until the design is deployed to a specific hardware platform. This can lead to late detection of mistakes in the HW/SW partitioning and implementation. This traditional approach is visualized in figure 2a.

In the second approach all subsystems are verified in one environment. However, this is a difficult task [9]. There are currently three methods to verify heterogeneous systems [9]. One is to represent all systems in one HDL, which can involve model degradation. The second method is to use a simulator that supports all different HDLs used and the third method is to use different simulators for each system and verify the integrated system using co-simulation. According to Vicente et al. [10], co-simulation is useful in HW/SW co-design. The co-simulation design approach is depicted in figure 2b.

Co-simulation can be done by either connecting two simulators, known as direct coupling, or by the use of a co-simulation backplane. Furukawa et al. [11] propose a backplane tool and perform a case study in which an MPEG encoder/decoder using multiple processors is designed. However considerable drawbacks such as a performance bottleneck caused by centralized communication have been identified [9].

Groothuis et al. [12] show that the issues encountered in HW/SW co-design, such as late integration and testing, apply as well for mechatronic system design. Besides a hardware and software view, here also the mechanics view and the control view need to be integrated. A generic tool-independent co-simulation framework is needed to allow for testing and verification between views.

3. Analysis

We use the reference C code for the H.264/AVC encoder (JM 8.6) from ISO [13] as a basis for our own C code. To find a computationally intensive function, profiling is performed using OProfile [14]. The results are summarized in table 1 and are in accordance with [5]. The profiling results indicate that the motion estimation function takes up to 45% of the CPU power when the first mode, a fixed block size of 16×16, is used.

The reference code contains optimizations to break out of the loops shown in the pseudo code, which reduces the computational intensity.

To analyze the performance of the implementation two short videos, ‘Foreman’ and ‘Husky’, are used. Four characteristic macroblocks are selected from these videos: ‘Foreman low’, ‘Foreman avg’, ‘Foreman high’ and ‘Husky avg’. The macroblocks represent low, average and high computational intensity.

4. Proposed solution

Our proposed solution is to first develop a fully functional virtual prototype which is verified by means of co-simulation. Using this virtual prototype two target implementations can be implemented and tested.

4.1. Virtual prototype

In the reference code a video file is read from disk, encoding is performed and the result is written to disk. Instead of executing the motion estimation function on the host CPU we want to offload this function to dedicated hardware.

For the virtual prototype VHDL is used to describe this hardware. A co-simulation with our C code is used to perform functional verification of the integrated design. The co-simulation environment which is used for this verification is shown in figure 3. The VHDL code is developed in such a way that it has the same behavior as the reference code, while exploiting the parallel architecture available in the target platforms.

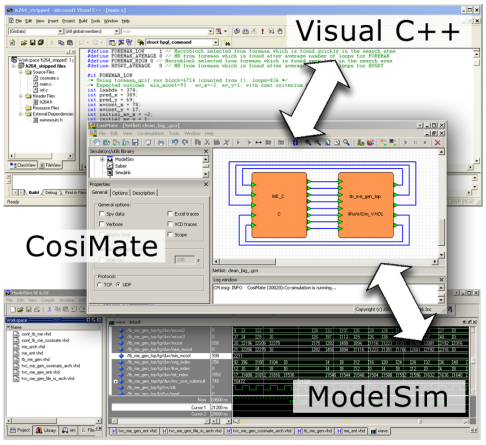


Figure 3. The co-simulation environment

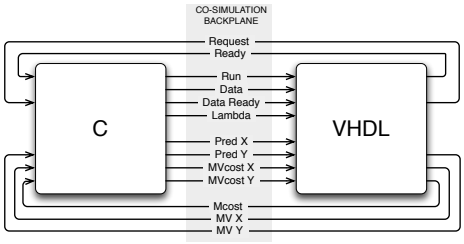


Figure 4. The co-simulation connection diagram

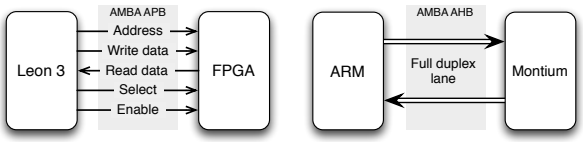
A state machine is designed that keeps track of the various states of the algorithm: initializing data (reading macroblock and reference frame), performing the SAD calculations (four SADs are calculated in parallel) and returning the result. For every macroblock and search area provided to the VHDL code the motion data is returned. The two optimizations to reduce the number of loop iterations are also included.

The co-simulation connection diagram is shown in figure 4. The connection diagram provides insight in the data that is transferred between C and VHDL. This is useful to map the connection diagram on any bus on any target.

4.2. Multiple target implementations

Using the virtual prototype a stripped version of the C code is created, while leaving the VHDL unchanged. The stripped C code only contains the part relevant for performance analysis. A parallel effort is started to implement the stripped version on two targets: the Leon 3 + FPGA and the ARM + Montium platforms.

For the Leon 3 + FPGA implementation the VHDL motion estimation code from the virtual prototype can again be used without alteration, while the interface with the co-simulation backplane has to be mapped



(a) Leon 3 - FPGA (b) ARM - Montium

Figure 5. Multiple target implementations

to the actual AMBA APB bus. Figure 5a gives an overview of this target platform.

For the ARM + Montium implementation the translation from VHDL to Montium code is made by hand. The algorithm on the Montium simultaneously calculates four SADs on four ALUs. A fifth ALU keeps track of the motion data, for which it uses the intermediate outputs of the first four ALUs. An exhaustive motion search algorithm is implemented in which all possible motion vectors are evaluated one by one. The optimizations implemented in VHDL, and thus on the FPGA, are designed for, but not implemented on the Montium.

The mapping from the co-simulation backplane to the AMBA AHB bus between the ARM and the Montium is simplified by using the features available in an operating system running on the ARM. Figure 5b gives an overview of this target platform.

5. Results and discussion

We use co-simulation to gain insight in the operation of the reference code. Bit by bit parts of the motion estimation algorithm are transferred from C to VHDL. The co-simulation allows for designing in many short iterations while verifying functional behavior. The result is a fully functional virtual prototype.

This virtual prototype is given to a Montium expert while we worked on the Leon 3 + FPGA implementation in parallel. Knowing that the virtual prototype has been verified allows for easier localization of errors introduced by the implementation process. The connection diagram in figure 4, which is part of the virtual prototype, is used to provide insight in the dataflow.

5.1. Implementation results

The FPGA target platform is an Altera Cyclone III EP3C25 FPGA. The FPGA clock frequency is 100 MHz, both for the Leon 3 softcore and our synchronous hardware. The number of QCIF frames processed per second depends on the amount of motion in the source video. In the analysis we found

Table 2. Overview of related work and this work

	[4]	[8]	[7]	[5]	[16]	This work	
Type	ASIC	ASIC	ASIC	GPU	FPGA	FPGA	Montium
Tech. [μm]	0.18	0.25	0.18	0.11	0.22	0.065	0.13
Search range	16x16	16x16	16x16	16x16	16x16	16x16	16x16
Block sizes	All	16x16	All	16x16	All	16x16	16x16
Gate count	597k	35k	88k	-	7381[1)]	339[1)]	392k
Freq. [MHz]	200	290	290	400	120	100	100
Max. fps QCIF	7,895	481	706	54	296	20.5	11.6
MHz 30 QCIF	0.76	18.07	12.32	222.55	12.17	146.3	258.4

[1)] For FPGA this number is the number of Logic Elements

Table 3. Number of clock cycles (x1000) on different target platforms

	Foreman			Husky
Platform	**Low**	**Avg**	**High**	**Avg**
Leon 3	910	1100	2515	2150
Leon 3 + FPGA	5.0	22	63	49
ARM	100	840	2390	1870
ARM + Montium	85	85	85	85
ARM + Montium[2)]	*6.1*	*27*	*77*	*60*

[2)] Estimation for non-exhaustive search

four characteristic macroblocks. Of these characteristic maroblocks, two were simulated on the Leon 3 (without offloading) in ModelSim: 'Foreman low' and 'Husky avg'. From this simulation the number of clockcycles needed to process these macroblocks is found. Based on these numbers the number of clockcycles for 'Foreman avg' and 'Foreman high' were estimated. All four macroblocks were also simulated on Leon 3 + FPGA in ModelSim. The results, which do not include the number of clockcycles required for data transfer, can be found in table 3. From these numbers the speed-up gained by implementing a HW/SW partitioning can be determined. The total (static and dynamic) energy consumption for executing the motion estimation algorithm on the FPGA has been estimated by using the Altera Cyclone PowerPlay Early Power Estimator worksheet [15]. The speed-up and power consumption figures can be found in table 4. With regard to the bus usage sending all data to the FPGA takes 11,543 clock cycles while receiving the motion data takes 18 clock cycles. Table 2 shows the characteristics of related work and this work. The maximum number of QCIF frames processed per second and the minimum clock frequency needed to process QCIF frames at 30 fps have been determined based on 'Husky avg'.

The Montium target platform consists of an ARM946E-S and a Xilinx Virtex XC2V8000 FPGA containing the Montium TP. The clock frequency is 100 MHz, both for the ARM and the Montium. The ARM uses the same optimizations present in the Leon 3 and Leon 3 + VHDL implementations. The

Table 4. Speed-up factors and power consumption

		Foreman			Husky
Platform	**Metric**	**Low**	**Avg**	**High**	**Avg**
FPGA	Speed-up	181	50.3	39.8	43.6
	Power [mW]	42.0	42.0	42.0	42.0
Montium	Speed-up	1.17	9.88	28.1	22.0
	Power [mW]	60.2	60.2	60.2	60.2
Montium[2)]	*Speed-up*	*16.4*	*31.1*	*31.0*	*31.2*

ARM + Montium implementation uses an exhaustive search instead. Therefore the number of clockcycles needed to process a macroblock is always the same. For the case that the ARM + Montium implementation includes these optimizations we have estimated the number of clockcycles. The results can be found in table 3. From these numbers the speed-up gained by implementing a HW/SW partitioning can be determined. The energy consumption for executing the motion estimation algorithm on the Montium has been determined by using the energy numbers found in [17]. In [17] a FFT-512 algorithm has been implemented on the Montium which resembles our algorithm. The static and dynamic power consumption add up to 60.2 mW at 100 MHz. The speed-up and energy consumption figures can be found in table 4. With regard to the bus usage sending all data to the Montium takes 18,945 clock cycles whereas receiving the motion data takes 16,384 clock cycles. The high bus overhead is mainly caused by the operating system running on the ARM.

6. Conclusions

The Leon 3 + FPGA implementation shows a speed-up of 43.6× and a power consumption of 42.0 mW. The ARM + Montium implementation shows a speed-up of 21.5× and a power consumption of 60.2 mW. When we compare both implementations we see that the Leon 3 + FPGA outperforms the ARM + Montium, with respect to power consumption as well as performance.

We accomplished a full HW/SW co-design trajectory, including two target implementations, in five weeks. Co-simulation, using CosiMate, proved to be essential in achieving this. The two main advantages of co-simulation are: enabling short design iterations and providing insight in the dataflow through the connection diagram.

Translation of the virtual prototype to two target platforms was successful. Considering the short development time, our implementations show significant speed-ups.

7. Future work

The resulting architectures are outperformed by the related work, but there is room for improvements. For the current partitioning loop unrolling can be employed to increase the encoding speed. For further improvements a different HW/SW partitioning should be investigated, which is facilitated by our virtual prototyping approach. Also the design can be extended to include all seven modes of blocksizes to further improve encoder efficiency and quality.

Regarding the data transfer between hardware and software two optimizations are possible: using a better bus and re-using data. In the current approach the complete search area is transmitted for each macroblock while a high correlation between consecutive search areas may be expected.

Our current implementation is not yet a fully functional encoder. For a real-life application the full encoder needs to be implemented. The virtual prototype can be used as a starting point in this effort.

Finally, regarding this virtual prototype, an effort can be made to incorporate a more realistic bus into the co-simulation in order to facilitate the mapping from the co-simulation connection diagram to the actual bus.

References

[1] Joint Video Team (JVT), "Draft ITU-T recommendation and final draft international standard of joint video specification," ISO/IEC MPEG and ITU-T VCEG, ITU-T Rec. H.264/ISO/IEC 14 496-10 AVC, May 2003.

[2] G. J. Sullivan, P. Topiwala, and A. Luthra, "The H.264/AVC Advanced Video Coding Standard: overview and introduction to the fidelity range extensions," in *Applications of Digital Image Processing XXVII. Proceedings of the SPIE*, A. G. Tescher, Ed., vol. 5558, August 2004, pp. 454–474.

[3] L. Smit, G. Rauwerda, A. Molderink, P. Wolkotte, and G. Smit, "Implementation of a 2-D 8x8 IDCT on the reconfigurable Montium core," in *Field Programmable Logic and Applications, 2007 (FPL 2007). International Conference on*, 2007, pp. 562–566.

[4] C.-M. Ou, C.-F. Le, and W.-J. Hwang, "An efficient VLSI architecture for H.264 Variable Block Size Motion Estimation," in *IEEE Trans. Consumer Electronics*, vol. 51, no. 4, November 2005, pp. 1291–1299.

[5] C.-Y. Lee, Y.-C. Lin, C.-L. Wu, C.-H. Chang, Y.-M. Tsao, and S.-Y. Chien, "Multi-Pass and Frame Parallel algorithms of Motion Estimation in H.264/AVC for generic GPU," in *Multimedia and Expo, 2007 IEEE International Conference on*, July 2007, pp. 1603–1606.

[6] Y. Song, Z. Liu, T. Ikenaga, and S. Goto, "Ultra low-complexity fast Variable Block Size Motion Estimation algorithm in H.264/AVC," in *Multimedia and Expo, 2007 IEEE International Conference on*, July 2007, pp. 376–379.

[7] C.-L. Hsu and M.-H. Ho, "High-efficiency VLSI architecture design for Motion Estimation in H.264/AVC," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E90-A, no. 12, pp. 2818–2825, December 2007.

[8] S. Saponara and L. Fanucci, "Data-adaptive motion estimation algorithms and VLSI architecture design for low-power video systems," in *IEE Proc. Computer and Digital Techniques*, vol. 151, no. 1, January 2004, pp. 51–59.

[9] S. Kajtazovic, C. Steger, A. Schuhai, and M. Pistauer, *Automatic generation of a coverification platform*. Springer, 2005, ch. 11, pp. 187–203.

[10] C. Vicente-Chicote, A. Toledo, and P. Sánchez-Palma, *Image processing application development: From rapid prototyping to SW/HW co-simulation and automated code generation*, ser. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2005, pp. 695–666.

[11] T. Furukawa, S. Honda, H. Tomiyama, and H. Takada, *A hardware/software co-simulator with RTOS supports for multiprocessor Embedded Systems*, ser. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2007, pp. 283–294.

[12] M. Groothuis and J. F. Broenink, "Multi-view methodology for the design of embedded mechatronic control systems," in *Proceedings IEEE International Symposium on Computer Aided Control Systems Conference, Munich*. IEEE Control Systems Society, October 2006, pp. 416–421.

[13] (2007, December). [Online]. Available: http://standards.iso.org/ittf/PubliclyAvailableStandards/c039593_ISO_IEC_14496-5_2001_Amd_6_2005(E)_Reference_Software.zip

[14] (2007, December). [Online]. Available: http://oprofile.sourceforge.net

[15] (2007, July). [Online]. Available: http://www.altera.com/support/devices/estimator/pow-powerplay.jsp

[16] C. Wei and M. Z. Gang, "A novel SAD computing hardware architecture for Variable-Size Block Motion Estimation and its implementation with FPGA," in *ASIC, 2003. Proceedings of the 5th International Conference on*, vol. 2, October 2003, pp. 950–953.

[17] G. J. Smit, A. B. Kokkeler, P. T. Wolkotte, and M. D. van de Burgwal, "Multi-core architectures and streaming applications," in *System Level Interconnect Prediction, 10th international workshop on*, to be published in April 2008.