AN AMBA-COMPLIANT MOTION ESTIMATOR FOR H.264 ADVANCED VIDEO CODING[1]

Chao-Yung Kao
Department of Computer Science
National Tsing Hua University
Hsin-chu, Taiwan 300
g924305@oz.nthu.edu.tw

Youn-Long Lin
Department of Computer Science
National Tsing Hua University
Hsin-chu, Taiwan 300
ylin@cs.nthu.edu.tw

Abstract –*We propose a 2D VLSI architecture with 256 processing elements and a computation result reuse methodology for full search variable block size motion estimation (FSVBSME) in the next generation video coding standard H.264/AVC. Our pipelined engine can complete matching a candidate macroblock in every clock cycle. We implement a prototype on an SOC platform with a 32-bit RISC CPU core and field programmable gate array (FPGA) module. We equip the hardware accelerator with an AMBA-AHB on-chip-bus interface. Experimental results show that our proposed hardware accelerator delivers 480X speed-up when compared with software running at the same clock rate.*

Keywords: Motion Estimation, Video Coding, AMBA, Hardware Accelerator, Fast Prototyping, H.264.

1 Introduction

H.264/AVC is the latest video coding standard of the ITU-T Video Coding Experts Group (VCEG) and ISO/IEC Moving Picture Experts Group (MPEG) [1]. Its new features include variable block sizes motion estimation with multiple reference frames, integer 4x4 discrete cosine transform, in-loop deblocking filter and context-adaptive binary arithmetic coding (CABAC). H.264/AVC can save up to 50% bit-rate compared to MPEG-4 simple profile at the same video quality level. However, large amount of computation is required. Profiling report shows that motion estimation consumes about 60% of the total encoding time. Therefore, hardware accelerator for motion estimation is necessary in real-time applications.

Many motion estimation architectures have been proposed for MPEG-4 and H.264/AVC [2][3]. However, most of them are targeted towards ASICs. Verification is difficult and time-to-market is long. Reusability and flexibility are low, and it takes a lot of time and effort to integrate the hardware accelerator with software.

In this paper, we present a variable block size motion estimation engine with AMBA-AHB on-chip-bus interface and its prototype on an SOC platform consisting of embedded 32-bit RISC CPU core, memory modules, on-chip-bus, FPGA module, peripherals and development environment.

The rest of this paper is organized as following. In Section 2, we introduce the motion estimation algorithm used in H.264/AVC. In Section 3, we present our motion estimation architecture and the data reuse methodology. Our experimental results are presented in Section 4. Finally, we draw conclusions and point to possible directions for future research in Section 5.

2 Motion Estimation Algorithm

Motion estimation algorithms exploit the temporal redundancy of a video sequence. Among all the motion estimation algorithms, the full-search block-matching algorithm, as shown in Figure 1, has been proven to find the best block match, which causes the smallest sum of absolute differences (SAD). The minimum SAD is computed as formula (1) and (2).

$$SAD(i,j) = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1}\left|CB(m,n) - RB(m+i,n+j)\right| \quad (1)$$

$$SAD_{\min}(i,j) = \min(SAD(i,j)) \quad (2)$$

CB represents current block, and RB represents reference block. N is the block size and (i, j) is the motion vector. In H.264/AVC, each picture of a video is partitioned into macroblocks of 16X16 pixels and each macroblock can be subdivided into seven kinds of variable size sub-blocks (one 16x16 sub-block, two 16x8 sub-blocks, two 8x16 sub-blocks, four 8x8 sub-blocks, eight 8x4 sub-blocks, eight 4x8 sub-blocks, or sixteen 4x4 sub-blocks). Therefore, we have to find the motion vector and

[1] Supported in part by the National Science Council, R.O.C., under Grants No. NSC 92-2218-E-007-023, NSC 92-2220-E-007-009, and NSC 92-2220-E-007-017, by the Ministry of Economics Affairs, R.O.C., under Grant No. 92-EC-17-A-03-S1-0002, and by Taiwan Semiconductor Manufacturing Corp. (TSMC) under Grant No. 93A0002EA.

calculate the associated minimum SAD for each and every of 41 sub-blocks.

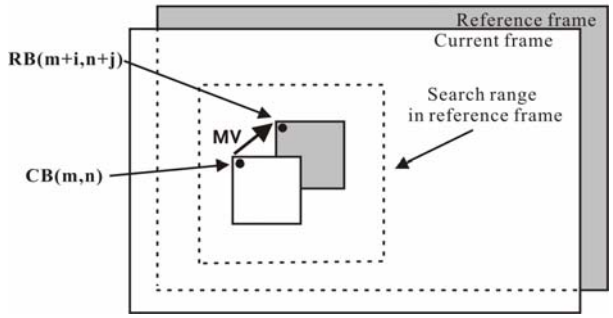

Figure 1. Block Matching

3 Proposed Motion Estimator

3.1 Hardware Architecture

We adopt the full search algorithm because its regularity is more suitable for hardware implementation. We reuse the computation results of SADs for 4x4 sub-blocks to calculate the SADs of 16x16, 16x8, 8x16, 8x8, 8x4 and 4x8 sub-blocks.

Our architecture is shown in Figure 2. There are total 256 processing elements (PE). PEs are grouped into 16 4X4 arrays. The detail of a PE array is depicted in Figure 3.

Let CRB(i, j) be the 16X16 candidate reference block upper-left anchored at coordinate (i, j) of the reference frame relative to the current macroblock. For example, CRB(-16, -16) denotes the most upper-left candidate reference block while CRB (-16, +16) denotes the most lower-left one. Totally, we have to match the current block against 33X33 = 1089 CRBs. Our proposed pipelined architecture delivers a throughput level of one CRB matching per clock cycle.

At the beginning, each PE holds a pixel of the current macroblock. Then, we load pixel data within the search range in the reference frame into the motion estimator according to the sequence illustrated in Table 1. One or two 1x16 search area pixels are input in each clock cycle and broadcasted to all PEs. Each PE computes the difference between the current macroblock pixel and the candidate reference block pixel. The results of 4x1 PEs are summed up and passed to the right. Afterward the SADs of 4x4 blocks will be stored in hardware queues for SAD computations of other types of sub-blocks. We can obtain sixteen 4x4 SADs in every clock cycle. Along the pipeline downwards, we also obtain 8 8x4 SADs, 8 4x8 SADs, 4 8x8 SADs, 2 16x8 SADs, 2 8x16 SADs and 1 16x16 SADS every cycle. The comparators will record the minimum SADs and the best motion vectors of 41 sub-blocks and output the results.

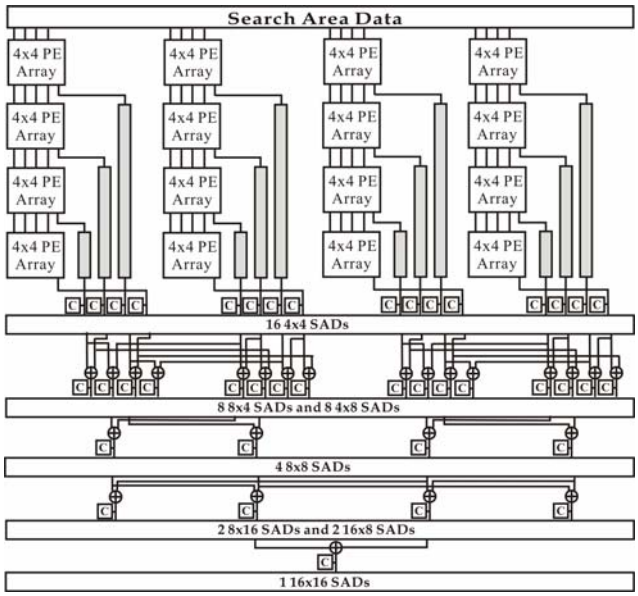

Figure 2. Proposed Motion Estimator Architecture

Table 1 shows that we complete loading CRB(-16, -16) between clock cycles 0 and 15. After clock cycle 47 all pixels of 33 leftmost CRBs (i.e., CRB(-16, -16), CRB(-16, -15), … CRB(-16, +16)) have been loaded. The loading of second column (i.e., CRB(-15, -16), CRB(-15, -15), … CRB(-15, +16)) is started at clock cycle 33 and partially overlapped with that of column 1. This allows us to eliminate bubble in the pipeline and, hence, achieve one CRB per clock cycle. So, the whole computation will take 15 + 33x33 + 4 = 1,108 cycles to complete, where 15 is the number of cycles to fill up the array and generate the first 16 4x4 SADs, 33X33 is the total number of CRBs, and 4 is the number of pipeline stages after the PE array.

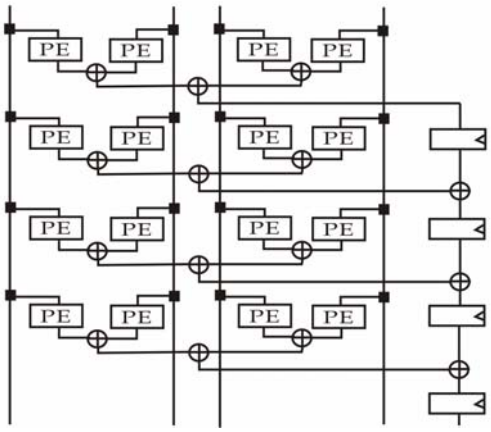

Figure 3. Architecture of the 4X4 PE Array

Table 1. Data Supply to the PE Array

Clk	Data Flow
0	Pixel[0~15] of row 0 to row 0~15 of PE array
1	Pixel[0~15] of row 1 to row 0~15 of PE array
:	:
32	Pixel[0~15] of row 32 to row 0~15 of PE array
33	Pixel[0~15] of row 33 to row 1~15 of PE array Pixel[1~16] of row 0 to row 0 of PE array
34	Pixel[0~15] of row 34 to row 2~15 of PE array Pixel[1~16] of row 1 to row 0~1 of PE array
:	:
47	Pixel[0~15] of row 47 to row 15 of PE array Pixel[1~16] of row 14 to row 0~14 of PE array
48	Pixel[1~16] of row 15 to row 0~15 of PE array
:	:
65	Pixel[1~16] of row 32 to row 0~15 of PE array
66	Pixel[1~16] of row 47 to row 1~15 of PE array Pixel[2~17] of row 14 to row 0 of PE array
:	:

3.2 Data-Reuse Methodology

Video signal processing generally requires a lot of memory bandwidth. In H.264/AVC, for applications of 30fps, 352x288 (CIF) pixels per frame with 5 reference frames and [+16,-16] search range, a straightforward implementation will consume 33GB/s of memory bandwidth. Therefore, exploiting data reuse is necessary.

The search ranges of two consecutive current macroblocks overlap with each other as depicted in Figure 4. We take advantage of this kind of global locality within a search area stripe [4]. Therefore, our local memory size is 42x32x8 = 12,288 bits. Each pixel of the search range will be used by matching for three current macroblocks, resulting in 66% reduction in memory traffic.

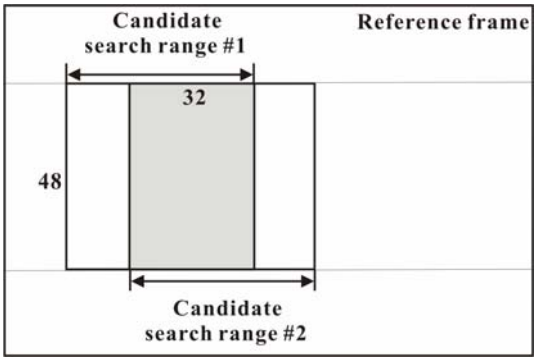

Figure 4. Global Locality within Search Area Stripe

3.3 Prototyping on Platform

We adopt the ALTERA Excalibur EPXA10DDR and its associated system-level development kits as our SOC design platform. The Excalibur EPXA10DDR as shown in Figure 5 is based on an enhanced FPGA device called EPXA10F1020C1 System-On-a-Programable-Chip(SOPC). It includes a 32-bit ARM922T RISC processor in hard-core form and embedded FPGA. The Excalibur has several additional hardware peripherals as listed below :

- ARM922T 32-bit RISC processor core
- 1000K-gate FPGA module
- 10/100 Mb Ethernet port
- Two 32-bit PCI slots.
- 256 KB Single Port SRAM
- 128 KB Dual Port SRAM
- 128 MB DDR SDRAM
- 32 MB Flash Memory

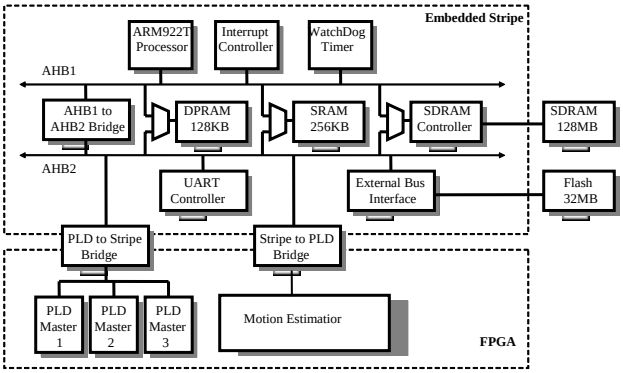

Figure 5. SOC Platform

We integrate the developed motion estimation IP into the SOC platform in five steps:

- Get familiar with the hardware interface architecture [5]
- Analyze input and output protocol of the hardware accelerator
- Verify the functionality of the hardware design
- Design a finite state machine for on-chip bus communication
- Write device driver software

Figure 6 depicts all interface-related components. The software component runs on the ARM922T CPU core. We have two hardware modules. The hardware accelerator carries out the main processing of motion estimation. The communication module has two sides: one communicates with the processor core via AMBA-AHB and the other communicates with the hardware accelerator. A finite state machine is included in the communication module for handling the bus protocol. The

communication module also contains a register file to synchronize the hardware accelerator and software driver.

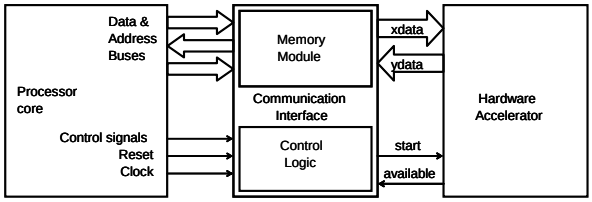

Figure 6. CPU-Accelerator Interface Architecture

We use the Excalibur Stripe Simulator (ESS) to verify our hardware/software co-design. By using hardware/software co-simulation, we can uncover functional errors early in the design process. The debugging effort is greatly reduced during FPGA prototyping. Furthermore, we can capture the bus behavior by observing the signal transfer on the AMBA-AHB bus between the CPU core and the hardware accelerator.

4 Experimental Results

Our design is implemented in Verilog and synthesized using the Synopsys Design Compiler with TSMC 0.13 μm standard cell library. The results are shown in Table 2.

Table 2. Synthesis Results

Target Processing Capability	720x480 30fps	1920x1080 30fps
Gate Count	95,934	197,225
frequency	50 MHz	285 MHz
Power consumption	113 mW	878 mW

Moreover, we verified our motion estimator using the FPGA module of the SOPC platform and list the compilation result reported by QuartusII in Table 3.

Table 3. FPGA Compilation Report

Target device for compilation	EPXA10F1020C1
ARM922T working frequency	12.5MHz
FPGA working frequency	12.5MHz
Total logic elements	23,057
Total memory bits	12,288
Total pins	68

Table 4 compares the motion estimation time between pure software and our hardware accelerator for matching a 16x16 current block against a 48x48 search range. The speed up is 480 times.

Table 4. Computation Time Comparison

	Pure Software	SW Driver co-work with Hardware Accelerator
Total time (s)	3.394	0.007

5 Conclusions

We have presented an AMBA-compliant motion estimator for H.264/AVC. We describes its architecture, data reuse methodology and a FPGA prototype. The experimental results show that the coding process with our motion estimator speeds up greatly.

In the future, we plan to integrate the design with other components of H.264/AVC, such as motion compensation, transform, deblocking filter and entropy coding.

References

[1] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, “Overview of the H.264/AVC video coding standard”, IEEE Transactions on Circuits and Systems for Video Technology, pp. 560-576, July 2003.

[2] S. Y. Yap, and J. V. McCanny, “A VLSI Architecture for Advanced Video Coding Motion Estimation”,Proceeding of IEEE International Conference on Application-Specific Systems, Architectures, and Processors, pp. 293-301, June 2003.

[3] Yu-Wen Huang, Tu-Chih Wang, Bing-Yu Hsieh, and Liang-Gee Chen, “Hardware Architecture Design for Variable Block Size Motion Estimation in MPEG-4 AVC/JVT/ITU-T H.264”, Proceedings of the 2003 International Symposium on Circuits and Systems, Vol 2, pp. 796-799, May 2003.

[4] Jen-Chieh Tuan, Tian-Sheuan Chang, and Chein-Wei Jen, “On the Data Reuse and Memory Bandwidth Analysis for Full-Search Block-Matching VLSI Architecture”, IEEE Transactions on Circuits and Systems for Video Technology, Vol 12, pp. 61-72, Jan. 2002.

[5] A. Baganne, J. L. Philippe, and E. Martin, “A Formal Technique for Hardware Interface Design”, IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, Vol 45, pp. 584-591, May 1998.