Research Article

A High-Throughput Hardware Architecture for the H.264/AVC Half-Pixel Motion Estimation Targeting High-Definition Videos

Marcel M. Corrêa, Mateus T. Schoenknecht, Robson S. Dornelles, and Luciano V. Agostini

Federal University of Pelotas (UFPEL), Group of Architectures and Integrated Circuits (GACI), 96010-900, Pelotas, RS, Brazil

Correspondence should be addressed to Marcel M. Corrêa, mmcorrea@inf.ufpel.edu.br

Received 2 June 2010; Revised 21 September 2010; Accepted 27 October 2010

Academic Editor: Gustavo Sutter

Copyright © 2011 Marcel M. Corrêa et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This paper presents a high-performance hardware architecture for the H.264/AVC Half-Pixel Motion Estimation that targets high-definition videos. This design can process very high-definition videos like QHDTV (3840×2048) in real time (30 frames per second). It also presents an optimized arrangement of interpolated samples, which is the main key to achieve an efficient search. The interpolation process is interleaved with the SAD calculation and comparison, allowing the high throughput. The architecture was fully described in VHDL, synthesized for two different Xilinx FPGA devices, and it achieved very good results when compared to related works.

1. Introduction

Video coding is an important research area due to the increasing demand for high-definition digital video for applications like video streaming over the internet, digital television broadcasting, video storage, and many others.

There are many video coding standards. These standards primarily define two things: a coded representation (or syntax) which describes the visual data in a compressed form and a method to decode the syntax to reconstruct the visual information [1]. The most recent standard is the H.264/AVC (Advanced Video Coding), designed to achieve the highest compression rates when compared to older standards [2]. However, this new standard has a very high computational complexity, which makes it difficult for software implementations to encode high-definition videos in real time when using the H.264/AVC complex features. For this reason, dedicated hardware architectures are a good solution for fast and efficient high-definition video coding. A hardware implementation is also required when the video encoder or decoder is inserted in an embedded system like a cell phone or a digital camera, and in this case, a high-throughput and low-power solution is essential.

A raw digital video has a high amount of redundant information that can be explored for compression purposes. There are three kinds of redundancy: the spatial redundancy is the similarity in homogeneous areas within a frame, the temporal redundancy is the similarity between sequential frames, and finally, the entropic redundancy is the similarity in the bit stream representation [1].

Figure 1 presents a block diagram of the H.264/AVC encoder, with its main operations: Inter-Frames Prediction, composed by the Motion Estimation (ME) and the Motion Compensation (MC) modules, Intra-Frame Prediction, Forward Transforms (T) and Quantization (Q), Entropy Coding, Inverse Quantization (IQ) and Transformations (IT), and Deblocking Filter [2].

This work proposes a high-performance architecture for the Half-Pixel Motion Estimation Refinement, designed to be integrated to a fast Motion Estimation architecture. The designed solution presented in this paper is fully compliant with the H.264/AVC standard, but it focuses on simplifications and optimizations to reach high processing rates, avoiding the use of the expensive RDO decision mode [1] in the interframe prediction of this standard, as will be explained in the next sections.

FIGURE 1: H.264/AVC encoder block diagram.

This paper is structured as follows. Section 2 introduces the Motion Estimation process, Section 3 introduces the half-pixel interpolation and search processes, Section 4 presents software evaluations, Section 5 shows the designed Half-Pixel ME Refinement architecture, Section 6 presents the synthesis results, Section 7 presents a comparison among related works, and finally, Section 8 concludes this paper.

2. Motion Estimation

The Motion Estimation is a module that explores and reduces the temporal redundancy of a video. As shown in Figure 2, it works by splitting the current frame into several macroblocks (16×16 pixels) and searching in the previous coded frames (reference frames) for the block that is most similar to the current one. When the most similar block is found, a motion vector (MV) is generated to indicate the position of this block in the reference frame.

The H.264/AVC brings some new features for the Motion Estimation, like the use of variable block sizes (VBSME), multiple reference frames, biprediction, and a more efficient subpixel accuracy [1].

The use of subpixel accuracy increases significantly the efficiency of the ME because the most similar block can be found in a fractional position, indicating a movement smaller than one pixel [2]. The subpixel accuracy is the focus of this work. This feature is the most important coding tool of the H.264/AVC ME because it generates the highest gains in terms of compression rates and it also increases the visual quality of the compressed video [1].

2.1. Decision Mode in the H.264/AVC Inter-Frame Prediction. The variable block size feature of the H.264/AVC ME makes the search for better matches by splitting a macroblock into smaller blocks possible. A macroblock can be divided in four partition options: 16×16 , 16×8 , 8×16 , or 8×8 . Each 8×8 partition (called sub-macroblock) can be further partitioned in three options: 8×4 , 4×8 or 4×4 .

As video coding is a lossy encoding process, the rate-distortion optimization (RDO) technique was proposed by [3] in order to define a metric for selection, among all possible ways to divide a macroblock, which is the most efficient one in terms for compression rate and video quality.

Equation (1) shows the Lagrangian rate-distortion formula, in which D denotes the distortion (measured in PSNR), R denotes the bit-rate, λ denotes the Lagrangian multiplier, and J is the final cost. The coding mode with the lowest cost J is chosen as the best option

$$J = D + \lambda \cdot R. \quad (1)$$

However, it is only possible to know the distortion (D) and bit-rate (R) of a macroblock partition or subpartition after reconstructing it, which means that every single partition and subpartition of a macroblock must be processed by all coding steps (ME search, residue generation, forward transforms and quantization, entropy coding, inverse transforms and quantization, and reconstruction) in order to choose the best partition and discard the others.

This way, the use of RDO generates a very good decision considering the tradeoff between compression rate and video quality, but RDO is a very complex and very expensive technique for video encoders targeting real-time processing. Its cost is especially prohibitive when hardware solutions are being considered. One of the points that this work focuses on is to decrease the complexity of the ME avoiding the use of the RDO decision mode, maintaining the compliance of the generated results with the H.264/AVC standard, as explained in Section 4.

3. Half-Pixel Motion Estimation Refinement

A characteristic that contributes to the high compression rates achieved by the H.264/AVC Motion Estimation is the possibility to generate fractional motion vectors with half-pixel or quarter-pixel accuracy [2]. In other words, a movement that happens from a frame to another in a video running at 30 frames per second is not restricted only to integer pixel positions.

Figure 3(a) shows an integer motion vector pointing to a 4×4 block that is directly presented in the reference frame, and Figure 3(b) shows a fractional motion vector pointing to a 4×4 block composed by half-pixels that were not presented in the reference frame. The half-pixel samples that are used by the Half-Pixel Motion Estimation must be obtained through the interpolation of the integer position samples. This way, the Half-Pixel Motion Estimation Refinement has

FIGURE 2: Motion Estimation process.

FIGURE 3: Motion vectors pointing to 4×4 blocks.

two steps: the Half-Pixel Interpolation Process and the Half-Pixel ME Search.

3.1. Half-Pixel Interpolation Process. To make the search for a better match block composed by half-pixels possible, a new search area must be generated around the integer position samples that compose the current best match chosen by ME. The half-pixel interpolation unit gets the best integer match block (composed by integer position samples) from the ME and interpolates an area composed by half-pixels around these samples.

A single half-pixel y that has adjacent integer positions is derived by first calculating an intermediate value called y_1 by applying the 6-tap FIR filter presented in (2), where A to F represent the nearest six integer luminance samples (0–255) in the horizontal or vertical directions. Then, (3) is applied to y_1 [2]

$$y_1 = A - 5B + 20C + 20D - 5E + F, \quad (2)$$

$$y = \text{Clip}_{0-255}[(y_1 + 16) \gg 5]. \quad (3)$$

A single half-pixel y that has adjacent half-pixel positions instead of integer positions (because it is diagonally aligned

between integer positions) is derived by first calculating an intermediate value called y_1 by applying the same 6-tap FIR filter (2), using as C and D the y_1 values of the two adjacent half-pixels and using as A , B , E , and F the y values of the other nearest half-pixels, and finally, applying (4) to y_1 [2]

$$y = \text{Clip}_{0-255}[(y_1 + 512) \gg 10]. \quad (4)$$

It is important to notice that to calculate a half-pixel that is diagonally aligned between integer samples, either horizontal or vertical nearest half-pixels can be used because these pixels will produce an equivalent result [2]. In this work, we consider the horizontal closest half-pixels.

This way, there are three half-pixels' types: (1) H type, calculated using the closest horizontal integer position samples, (2) V type, calculated using the closest vertical integer position samples, and (3) D type, calculated using the closest horizontal half-pixels (which are V type half-pixels).

In Figure 4, the position g is a V-type half-pixel and it can be interpolated using the group of integer position pixels $\{U, S, C, H, I, J\}$. The position y is an H-type half-pixel, and it can be interpolated using the group of integer position pixels $\{A, B, C, D, E, F\}$. The position x is a D-type half-pixel, and it can be interpolated using the group of half-pixels $\{cc, dd, g_1, m_1, ee, ff\}$, where g_1 and m_1 are the intermediate values of g and m as defined by the standard [2].

An important characteristic of the interpolation process is that whatever block size is being processed by the ME, the half-pixel interpolation will always need an extra three-pixel border around this block in order to generate the interpolated search area.

Figure 5 shows a half-pixel interpolated area around a block composed by integer position samples (best ME match). Figure 6 shows that the interpolated search area has two possible matches composed by V-type half-pixels (vertical motion), two possible matches composed by H-type half-pixels (horizontal motion), and four possible matches composed by D-type half-pixels (diagonal motion).

FIGURE 4: Half-pixel positions (white squares) between Integer positions (gray squares).

FIGURE 5: Interpolated half-pixel search area around a 4×4 block.

In Figure 6, the black squares represent the matches for each possible fractional motion vector.

For instance, the match that would generate the fractional motion vector presented in Figure 3(b), with a FMV (0.5, -0.5), is composed by the following group of half-pixels: {d7, d8, d9, d10, d12, d13, d14, d15, d17, d18, d19, d20, d22, d23, d24, d25}.

3.2. Half-Pixel Motion Estimation Search. Using the interpolated search area, the Half-Pixel ME Search will test all the eight possible matches inside this new interpolated search area to check if there is a block composed by half-pixels more similar to the original block than the block found by the ME. This search is done using a block-matching algorithm, which uses a distortion criterion to determine the most similar block. This criterion can be a simple arithmetic difference between blocks or more complex calculations. Among the most used distortion criterion are the Mean Square Error (MSE), Sum of Squared Differences (SSD), and the Sum of Absolute Differences (SAD) [1].

The SAD is commonly used in motion adaptive deinterlacing algorithms and motion estimation algorithms [4]. SAD is defined in (5), and it is the most used distortion criterion because of its efficiency and low cost for a hardware implementation [1]. It works by taking the absolute value of the difference between each pixel in the original block and the corresponding pixel in the possible match block (also called candidate block) that is being used for comparison. These differences are added to create a simple metric of block similarity. A more complex analysis about SAD can be found in [4]

$$\text{SAD}(P, O) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |P_{i,j} - O_{i,j}|. \quad (5)$$

Once the SAD values for all the eight possible matches composed by half-pixels are calculated, the Half-Pixel ME Search will check if there is a better match composed by half-pixel comparing the SAD values. If there is a better match, the motion vector must be modified adding the corresponding fractional motion vector to it.

4. Software Evaluations

Software evaluations were done to better evaluate the impacts of simplifications on the proposed design. Several video sequences with different resolutions (QCIF, CIF, 4CIF, HD 1920 $\times$ 1080) were coded using the default configuration of the H.264/AVC reference software [5]. This first evaluation was done in order to check the utilization rate of each possible macroblock partition and subpartitions in the Motion Estimation process. As a result, we observed that 94.75% of the chosen blocks had a size greater or equal to 8×8 pixels. This result was already expected, since the higher is the video resolution, the lower is the probability of use of subpartitions. This fact is confirmed by the discussions related to the High-Efficiency Video Coding (HEVC) standard (in development) [6] that are considering the exclusion of lower block sizes as 4×4 , 4×8 , and 8×4 of this new standard, since they are rarely used for the currently used video resolutions.

Then, a second evaluation was done to check the impact of simplifying the motion estimation excluding the variable block size feature by using only the 8×8 partition size. This evaluation considered the use of a quarter-pixel accurate ME. Five QCIF video sequences were coded using different ME features, and the results were evaluated using two metrics: PSNR and bit-rate. Tables 1, 2, and 3 show the results for each video sequence. Table 1 considered the use of all possible partitions' and subpartitions' sizes. Table 2 considered only the block sizes 16×16 , 8×8 , and 4×4 . Table 3 presents the results when only the 8×8 block size is used. Tables 2 and 3 also present the losses in PSNR and bit-rate caused by the reduction in the number of block sizes if compared with the optimal results presented in Table 1.

The use of only 8×8 blocks reduced the PSNR by 0.32 dB on average when compared to the optimal case. The increase in bit-rate was of 3.84% on average, but in some cases the bit

FIGURE 6: All the eight possible matches composed by half-pixels and their respective FMVs.

TABLE 1: Results for the most complex variable block size configuration (16×16 , 16×8 , 8×16 , 8×8 , 8×4 , 4×8 , 4×4).

Video Sequence	PSNR (dB)	Bit-rate (Kbps)
Akiyo	40.82	114.5
Bridge Close	37.38	155.33
Coastguard	34.12	272.33
Foreman	36.47	207.87
Mobile	32.59	391.81
Average	36.28	228.37

FS algorithm, 32×32 search area, 1 past reference frame, quarter-pixel accuracy.

TABLE 2: Results for a less complex variable block size configuration (16×16 , 8×8 , 4×4).

Video Sequence	PSNR (dB)	PSNR loss (dB)	Bit-rate (Kbps)	Bit-rate loss (Kbps)
Akiyo	40.76	0.06	115.10	0.60
Bridge Close	37.30	0.08	154.62	-0.71
Coastguard	34.00	0.12	271.72	-0.61
Foreman	36.30	0.17	212.15	4.28
Mobile	32.53	0.06	395.64	3.83
Average	36.18	0.10	229.85	1.48

FS algorithm, 32×32 search area, 1 past reference frame, quarter-pixel accuracy.

rate was reduced with this restriction (the negative numbers in Table 3).

Considering the presented results, we decided to design an architecture that supports only the 8×8 block size, since this decision simplifies to a great extent the hardware design without expressive quality and compression rate losses. Also, the use of a unique block size reduces drastically

TABLE 3: Results for the 8×8 block size configuration.

Video Sequence	PSNR (dB)	PSNR loss (dB)	Bit-rate (Kbps)	Bit-rate loss (Kbps)
Akiyo	40.47	0.35	116.53	2.03
Bridge Close	37.16	0.22	154.56	-0.77
Coastguard	33.80	0.32	277.21	4.88
Foreman	36.01	0.46	225.24	17.37
Mobile	32.34	0.25	412.16	20.35
Average	35.96	0.32	237.14	8.77

FS algorithm, 32×32 search area, 1 past reference frame, quarter-pixel accuracy.

the complete ME complexity, since it avoids the necessity of a decision mode architecture to choose which is the best block size. As explained before, RDO-based decision mode is very expensive for a hardware implementation targeting real-time processing, and most of the works proposing VBSME architectures do not mention this problem.

We also did evaluations to check the impact of different features of the H.264/AVC ME. One at a time, the following features were combined: the use of different block-matching algorithms, different number of reference frames, and, finally, the subpixel refinement (half-pixel and quarter-pixel accurate motion vectors). The best results among all ME tools were really achieved when the subpixel refinement feature was activated, and this is an important motivation for this work.

This way, this paper presents a Half-Pixel Motion Estimation Refinement hardware design that gives support only to the 8×8 block size in order to reduce the complexity and cost of both the motion estimation module and the H.264/AVC encoder itself. Also, this design must be fast enough to not

degrade the performance of a ME architecture that uses a fast block-matching algorithm, like the Diamond Search [7].

5. Half-Pixel Motion Estimation Refinement Architecture

This paper presents an architecture that gives the half-pixel accuracy to the ME process. The architecture is divided into two main parts: the half-pixel interpolation and the fractional motion estimation search. These two processes are interleaved to reduce the use of clock cycles. The complete architecture is presented in Figure 7, where for a better visualization, the control signals are omitted.

5.1. Half-Pixel Interpolation Unit Architecture. Our design uses an optimized architecture for the half-pixel interpolation unit, which was initially presented in a previous work [8]. This architecture uses an efficient arrangement of interpolation samples, and it needs only 34 clock cycles to generate an interpolated area around an 8×8 block.

5.1.1. Processing Unit. One of the improvements of our Half-Pixel Interpolation Unit is the use of an optimized processing unit (PU) when compared with that presented in [8]. The new PU uses fewer adders, and it was validated to generate all half-pixel types.

Equation (6) is the first step of the PU, and it calculates the y_1 value of a half-pixel. This equation is equivalent to that presented in (2), but we applied some arithmetic manipulations to allow a calculation designed only with shift adds, avoiding the use of multiplications. For V-type half-pixels, the y_1 value must be stored to be used later in D-type interpolation

$$y_1 = (A + F) + 4[4(C + D) - (B + E)] + 4(C + D) - (B + E). \quad (6)$$

The second step of the processing unit calculates the half-pixel luminance value applying (3) to y_1 if the control unit indicates the interpolation of V- or H-type half-pixels, or (4) if it indicates the interpolation of D-type half-pixels.

To achieve a higher frequency, our PU uses a three-stage pipeline as shown in Figure 8. Deeper pipeline configurations are also possible because a high throughput is more important than a low latency for the interpolation process.

Nine PUs were used in a module called Filters Line which is presented in Figure 7. It is able to interpolate an entire line of H-type half-pixels, a column of V-type half-pixels, or a line of D-type half-pixels in a single step.

5.1.2. Buffers. The data flow of our Half-Pixel Interpolation Unit is very similar to that presented in [8]. Five buffers were used to store and shift the integer position samples, V-type samples, V_1 -type values, H-type samples and D-type samples. All buffers are connected to the Filter Line in order to provide its inputs and to store its outputs.

The buffer for integer position samples stores a 14×14 block (an 8×8 block plus the three-pixel border). This buffer

has two outputs: an entire line used for H-type interpolation and an entire column used for V-type interpolation. It is able to shift its lines and columns in order to change its outputs.

The buffer for V-type half-pixels stores a 14×9 area (two blocks in an 8×9 area plus a vertical three-pixel border). The buffer for V_1 -type values stores a 12×9 area composed by the y_1 values for each V-type half-pixel used in D-type interpolation. Both buffers have an entire line as output, and they are able to shift it for D-type interpolation.

The buffer for H-type half-pixels stores a 9×8 area (two 8×8 blocks). The buffer for D-type half-pixels stores a 9×9 area (four 8×8 blocks).

5.2. Half-Pixel ME Search Architecture. The half-pixel search process has two steps: the SAD calculation and the SAD comparison.

The SAD values are calculated in parallel with the interpolation process. The SAD buffer in Figure 7 stores a total of nine values, the best integer SAD and the SADs of the eight possible half-pixel matches.

The comparison uses only two extra clock cycles to check if there is a SAD value smaller than the best match SAD since the main part of the search process is done in parallel with the interpolation process. Another cycle is necessary to add the FMV to the MV. This sum does not happen if there is no fractional motion.

5.2.1. SAD Calculation. Since our interpolation unit can interpolate an entire line or column of a half-pixel type in a single step, the SAD calculation is done in parallel with the interpolation process without increasing the use of clock cycles. The SAD for 8×8 blocks is a 13-bit value, and there is a SAD value for each half-pixel possible match block.

Figure 9 shows a SAD Tree (ST). It is a module that calculates the SAD value for a possible match block line by line. Our design has a total of four STs connected to a buffer that stores the original block (which is currently being processed by the ME) and to the V-, H-, and D-type buffers. It works by calculating the SAD value of a single line or column and storing it in an accumulator register. Our STs use a 2-stage pipeline configuration, taking 9 clock cycles to calculate the SAD value for the entire 8×8 block.

5.2.2. SAD Comparison. Figure 10 shows the SAD Comparator (SC). It can compare four SAD values in parallel, and it uses a two-stage pipeline configuration.

In the first step, the comparator stores the smallest SAD value and its corresponding MV among the V- and H-type possible matches, and then it compares this SAD to the best integer match SAD. Extra clock cycles are not necessary since this comparison occurs in parallel with the D-type interpolation and SAD calculations.

In the second step, the comparator stores the smallest value among the D-type possible matches and compares it to the smallest SAD obtained in the first step. Two extra clock cycles are necessary to obtain the smallest SAD value and its corresponding FMV.

FIGURE 7: Half-pixel ME Refinement architecture.

FIGURE 8: Interpolation processing unit architecture.

FIGURE 9: SAD Tree architecture.

TABLE 4: Buffers cost results.

Buffer	8-bit registers	13-bit registers	16-bit registers	8-bit 4:1 MUX	8-bit 2:1 MUX	16-bit 2:1 MUX
Integer positions	196	—	—	196	—	—
V-type	126	—	—	126	—	—
V ₁ -type	—	—	108	—	—	108
H-type	72	—	—	—	72	—
D-type	81	—	—	—	81	—
Original block	64	—	—	64	—	—
SAD values	—	10	—	—	—	—

FIGURE 10: SAD Comparator architecture.

TABLE 5: Half-Pixel ME Refinement cost results.

Module	Slices	Slice Flip-Flops	4-input LUTs
Filter Line (Half-Pixel Interpolation Unit)	1606	470	3051
SAD Trees (4 STs)	480	196	996
SAD Comparator	64	34	105
Half-Pixel ME Search Unit	544	230	1001
Control Unit	18	8	29
Complete Half-Pixel ME Refinement Architecture	2168	708	4081

6. Synthesis Results

The proposed Half-Pixel ME Refinement architecture for 8×8 blocks was fully described in VHDL and synthesized to a Xilinx Virtex4 XC4VLX15 FPGA device using the Xilinx ISE 10.1 [11] synthesis tool.

Concerning the cost, for this first implementation all buffers were mapped as register banks. Table 4 shows the cost of these buffers in use of hardware elements. The registers are necessary to store luminance samples and SAD values, and the multiplexers are necessary to shift these data. The V₁ type and the integer position are the buffers with highest hardware consumption. V₁-type buffer consumes the highest amount of resources because it must store the intermediate values of the Filters Line before the clip operation; this way each position must store 16-bit samples.

TABLE 6: Performance comparison.

	Yalcin and Hamzaoglu [9]	Oktem and Hamzaoglu [10]	This Work
FPGA Device	Xilinx VirtexII	Xilinx VirtexII	Xilinx VirtexII
Frequency (MHz)	85	60	91
Cycles to process a macroblock (16×16)	720	768	148
Support to variable block size	Yes	Yes	No

Table 5 shows the cost of the different modules of the complete architecture in terms of use of FPGA resources. Most of the FPGA resources are consumed by the Filter Line, making the half-pixel interpolation unit the most expensive part of all the refinement process. Table 5 also shows that most of the resources consumed by the Half-Pixel ME Search Unit are used by the four SAD Trees.

Concerning the performance, our design uses 37 clock cycles to do the Half-Pixel ME Refinement for an 8×8 block, but since a macroblock is composed by four 8×8 blocks, the refinement for a macroblock takes 148 clock cycles. This way, when synthesized to a Xilinx Virtex4 FPGA, our design can run at 140 MHz and process 3,784 million of motion vectors or more precisely 262 HD (1280×720) frame/s, 116 Full HD (1920×1080) frame/s or 30 QHDTV (3840×2048) frame/s.

7. Related Works

There are few works in the literature focusing on a high-performance Half-Pixel ME Refinement for H.264/AVC standard targeting high-definition videos. However, we did not find any architecture giving support only to the 8×8 block size.

The architectures presented in [9, 10] support variable block size, but these works do not discuss the decision mode. However, our design uses a very optimized way to interpolate and organize samples of a unique block size in order to use less clock cycles.

As Table 6 shows, when synthesized to a Xilinx VirtexII XC2V40, our design achieves a frequency of 91 MHz, being able to process Full HD (1920×1080) videos in real time (30 frames/s or faster). The architecture presented in [9] is suitable for Full HD (1280×720) videos, and the architecture presented by [10] is suitable for VGA (640×480). Our solution needs the lowest number of cycles to process the Half-Pixel ME Refinement of one macroblock among the related works. Our solution also reaches the highest operation frequency among these works. Even when supporting only the 8×8 partition size, the losses in quality are of only 0.32 dB in PSNR as presented in the software evaluation section.

8. Conclusions

This paper presented a dedicated hardware architecture for the H.264/AVC Half-Pixel Motion Estimation Refinement, designed to be part of a complete Motion Estimation hardware able to process very high-definition videos in real time.

When synthesized to a Xilinx Virtex4 FPGA, this architecture achieved a maximum operation frequency of 140 MHz. At this frequency, our design is able to process QHDTV videos (3840×2048) in real time (30 frames/s). The high operation frequency of the Half-Pixel ME Refinement is important also to not degrade the performance of a complete ME architecture where the half-pixel architecture will be integrated.

Our solution uses the lower number of clock cycles to process one macroblock when compared to related works. Our solution also reaches the highest operation frequency when compared with those works. Joining these two results, it is possible to conclude that our solution is able to reach the highest processing rates among the related works.

References

- [1] I. G. Richardson, *H.264 and MPEG-4 Video Compression*, Wiley, New York, NY, USA, 2003.
- [2] Joint Video Team (JVT) of ITU-T VCEG and ISO/IEC MPEG, "Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification," ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC, May 2003.
- [3] G. J. Sullivan and T. Wiegand, "Rate-distortion optimization for: video compression," *IEEE Signal Processing Magazine*, vol. 15, no. 6, pp. 74–90, 1998.
- [4] A. Sudarsanam, A. Dasu, and K. Vaithianathan, "Analysis and design of a context adaptable SAD/MSE architecture," *International Journal of Reconfigurable Computing*, vol. 2009, Article ID 789592, 21 pages, 2009.
- [5] JM15.1, "H.264/AVC JM Reference Software," March 2010, <http://iphome.hhi.de/suehring/tml/>.
- [6] G. Sullivan and J. Ohm, "Meeting Report of the First Meeting of the Joint Collaborative Team on Video Coding (JCTVC-A200)," Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, April 2010.
- [7] P. M. Kuhn, *Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation*, Kluwer Academic Publishers, Norwell, Mass, USA, 1999.
- [8] M. Corrêa, M. Schoenknecht, R. Dornelles, and L. Agostini, "A high performance hardware architecture for the H.264/AVC half-pixel interpolation unit," in *Proceedings of the 4th Southern Programmable Logic Conference (SPL '10)*, pp. 81–86, Ipojuca, Brazil, March 2010.
- [9] S. Yalcin and I. Hamzaoglu, "A high performance hardware architecture for half-pixel accurate H.264 motion estimation," in *Proceedings of the 14th International Conference on Very Large Scale Integration and System-on-Chip (VLSI-SoIC '06)*, pp. 63–67, October 2006.
- [10] S. Oktem and I. Hamzaoglu, "An efficient hardware architecture for quarter-pixel accurate H.264 motion estimation," in *Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD '07)*, pp. 444–447, August 2007.
- [11] Xilinx, "FPGA and CPLD Solutions from Xilinx, Inc," March 2010, <http://www.xilinx.com/>.