Minimum Sum of Absolute Differences implementation in a single FPGA device

Joaquín Olivares[1], Javier Hormigo[2], Julio Villalba[2], and Ignacio Benavides[1]

[1] Dept. of Electrotechnics and Electronics, University of Córdoba, Spain
[2] Dept. of Computer Architecture, University of Málaga, Spain

Abstract. Block based motion estimation is one of the critical task in today video compression standards such as H.26x, MPEG-1, -2 and -4 standards. Most of the block based motion estimation algorithm are based on computing the sum of absolute differences (SAD) between corresponding elements in the candidate and reference block. In this paper a FPGA design for fast computing of the minimum SAD is proposed. The hardware unit proposed is intended to augment a general-purpose core. Thanks to the use of the on–line arithmetic (OLA) two goal are achieved: it is possible to implement a full 16×16 macroblock SAD in a single FPGA device and it permits us to speed up the computation by early truncation of the SAD calculation when the involved candidate is bigger than the current reference SAD. Reconfigurable devices allows us to change 8×8 or 16×16 pixels per block models quickly and easily. For a 16×16 SAD unit 1945 look–up tables (LUTs) are required at the frequency of 425 MHz. Comparison with other related works are provided.

Keywords: motion estimation, FPGA, sum of absolute difference, on–line arithmetic, reconfigurable hardware

1 Introduction

Motion estimation (ME) plays an important role in video coding and processing systems since motion vectors are critical information for temporal redundancy reduction. It has been widely employed in the H.26x, MPEG-1, -2 and -4 video compression standards. Motion estimation is defined as searching the best motion vector which is the displacement of the coordinate of the best similar block in previous frame for the block in current frame. The most commonly used metric to calculate the distortion is the Sum of Absolute Differences (SAD) [1], which adds up the absolute differences between corresponding elements in the candidate and reference block.

The heavy computational cost of the block matching algorithms (BMA) can be a significant problem in real–time coding applications. To reduce the computational complexity different VLSI architectures are designed to speed up the associated massive arithmetic calculation [2][3].

However, the necessity of specialized hardware goes against the demand of flexibility required by the current coding video systems. A viable solution to this problem is to use a programmable processor core along with a field–programmable gate–arrays devices (FPGAs) which is in charge of performing critical tasks. The reasons for and the benefits of using FPGAs are: increased flexibility and quick adaption of new developments; sufficient performance; and faster design times are achieved by re-using IP cores and high-level design languages (such as VHDL) speed up designs significantly. In this context, our design is intended to speed up the computation of the minimum SAD by its implementation on a FPGA (*SAD processor* in figure 1), while a core processor supplies the reference and candidate blocks to the FPGA device (*dispatcher* in figure 1). In this paper we propose a FPGA architecture to compute the minimum SAD. This design can be integrated with any BMA (full search or another efficient search strategy).

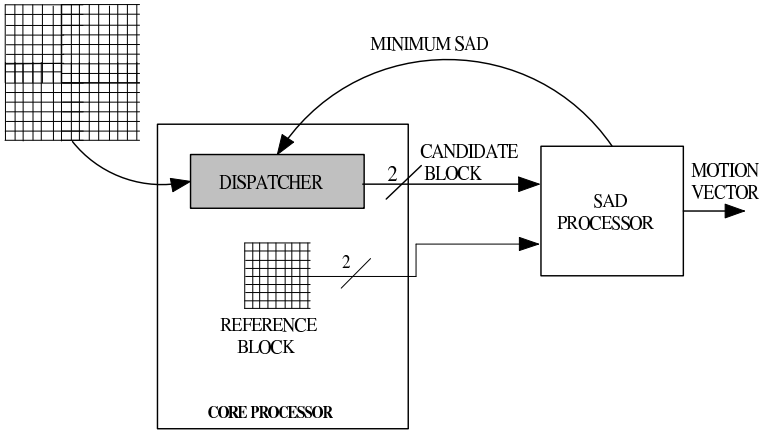

Fig. 1. Motion estimation system

In spite of inherent parallelism in SAD, the full parallel implementation requires a large amount of operands for the typical block size (16×16 pixel macroblock needs 512 8–bit operands). Due to the large amount of hardware, the computation of the SAD in only one row of a macroblock (16×1) is implemented on a FPGA device in [1], and they propose between replicate or pipeline the design to obtain the 16×16 computation. In [4] four FPGA chips with 1234 I/O pins each are used for a completely parallel design. On the other hand, the use of on–line arithmetic (OLA) for motion estimation is proposed in [5] to speed up the computation by early truncation of the SAD calculation. In [5], a serial architecture (pixel by pixel) for 4×4 blocks is proposed based on ASIC implementation.

We present a new parallel on–line architecture to carry out the minimum SAD computation for a 16×16 macroblocks. OLA works in a digit-serial mode which fits very well the FPGA architecture characteristic. Thus, the hardware requirements are drastically reduced which allows us to implement the complete SAD operation for the full macroblock in a single FPGA device.

This paper is organized as follows: in section 2 a brief description of the OLA techniques is provided, in section 3 we deal with the computation of the minimum SAD using OLA, section 4 shows the implementation of the proposed design in FPGA devices, a comparison with other proposed works is presented in section 5, and finally section 6 gives the main results.

2 On-line arithmetic

On-line arithmetic techniques have been considered to solve many signal processing problems, such us digital filtering, Fourier transform and others [6]→[9]. Recent works have presented the attractiveness of OLA for FPGAs designs [10].

The basic idea of OLA is to perform computations overlapped with the digit-by-digit communications of operands/results [6]. OLA algorithms operate in a digit-serial manner, beginning with the most significant digit. To generate the first digit of the result, $\delta+1$ digits of the input operands are needed. Thus, after δ digits of the operands are received, for each new digit of the operands, a new digit of the result is obtained. For this reason, δ is known as on-line delay. Due to the on-line delay, after the last digit of the inputs are introduced in the system, a number of zero digits equal to the on-line delay has to be introduced to ensure a correct result.

The MSD first mode of computation requires a flexibility in computing digits on the basis of partial information about inputs. This is achieved by the use of redundant representation system. In a redundant representation with radix r, each digit has more than r possible values. This allows several representation of a given value. Therefore, there is a flexibility in choosing an output digit at a given step so that, a compensation can be introduced if needed.

SD representation system is used in this paper. In radix-2 SD representation, the digit set is $\{-1, 0, 1\}$. Two bits are required to represent each digit, as shown in Table 1. The first bit is negative weighted and the second one is positive weighted. This number system eliminates the long carry propagation chains in addition operation, although requires the carry of the two previous digits.

In short, the advantages of using on-line arithmetic are: due to digit serial nature it reduces the number of signal lines connecting modules, the MSD first computation allows subsequent calculations to occur at much earlier stage, and it eliminates carry propagation chains since it uses a redundant number representation system.

3 On-line computation of the minimum SAD

The goal of our FPGA design is to find which of the candidate blocks (supplied by the dispatcher) best match the reference block. The most commonly used metric to determine the best match is the sum of absolute differences (SAD). Thus, our design compute the minimum SAD among all the candidate blocks. To do this, a search iteration is performed for each candidate block.

On each search iteration, the SAD corresponding of a candidate block is computed using all its pixels simultaneously. The value obtained is compared with the reference SAD (SADr) which is the minimum SAD computed before this iteration. If the current SAD (SADc) is less than SADr, it is stored as SADr for the remaining search iterations. Both the SAD computation and comparison operation is performed using on-line arithmetic. This allows us to begin the comparison when the first digit of the SAD is obtained and early stop the computation if the digits computed are enough to ensure that SADc is greater than SADr. The on–line comparator that we design for SAD comparison is described in [11].

The SAD adds up the absolute differences between corresponding elements in the candidate and reference block.

$$SAD = \sum_{i=1}^{N} \sum_{j=1}^{N} |c_{i,j} - r_{i,j}| \tag{1}$$

where $r_{i,j}$ are the elements of the reference block and $c_{i,j}$ the elements of the candidate block. Thus, the computation of the SAD is divided in three steps:

1 Computation of differences between corresponding elements $d_{i,j} = c_{i,j} - r_{i,j}$
2 Determine the absolute value of each differences $|d_{i,j}|$
3 Add all absolute values

Conversion to SD representation and difference computation: In radix-2 SD representation, each digit is composed by two bits, the first one negative weighted and the second one positive weighted. Thus, a SD number can be interpreted as the difference of two unsigned numbers, the one composed by bits positive weighted of each digit, minus the one composed by bits negative weighted. In fact, to convert SD number to non redundant representation, this difference is performed.

Table 1. Digit codification in radix-2 signed-digit representation

Digit value	Digit representation
+1	01
0	00
0	11
-1	10

This property is used to perform simultaneously the conversion of each pixel value to SD and the difference between pixels of reference block and current block with no computational cost. In that way, each digit of the value $d_{i,j} = c_{i,j} - r_{i,j}$ is obtained in SD representation by only taking the corresponding bit of $c_{i,j}$ as the positive weighted and the corresponding bit of $r_{i,j}$ as the negative weighted, since $c_{i,j}$ and $r_{i,j}$ are unsigned numbers.

Absolute value: To compute the absolute value of $d_{i,j}$, the sign of this value have to be changed if $d_{i,j}$ is negative. In SD the negation operation is performed by interchanging both bits of each digit. Since MSD-first mode of computation is used, the sign detection of $d_{i,j}$ is performed on-the-fly by checking if the first non zero digit of $d_{i,j}$ is positive (01) or negative (10). The digits of $d_{i,j}$ are received in MSD-first mode and go directly to the output while they are zero (00 or 11). If the first non zero digit received is positive (01), this and all the remaining digit correspond directly with the output. Nevertheless, if the first non zero digit received is negative (10), the bit of this and all the remaining digit are interchanged to obtain the output. The absolute value operation is performed with no on–line delay.

Sum of absolute differences: The absolute difference of all the pixels corresponding to the current and reference blocks are computed in parallel. Then N^2 absolute difference blocks are required. An on-line adder tree is used to obtain the sum of all $d_{i,j}$ values. FPGAs are reconfigurable devices that allows us to change easily from 8x8 to 16x16 pixels per block in spite of the MPEG4 stream required. A standard SD on-line adder is shown in figure 2.

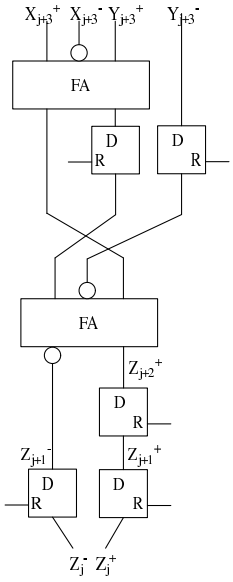

Fig. 2. On-line adder design.

The number of addition steps of the complete adder tree is $\log_2(N^2)$. In radix-2 SD representation the on–line delay of the addition is two. That is, the MSD of the result is obtained two cycles after the MSD of the inputs come to the adder. Nevertheless in our case, the carry bit is used as MSD of the results and this digit is obtained one cycle before. Therefore, the on-line delay of the complete adder-tree is $2\log_2(N^2)$, but the first digit of the results is obtained $\log_2(N^2)$ cycles earlier.

Once the first digit of the SAD corresponding to the current block is obtained, the comparison of the current SAD and the minimum SAD until this moment can begin. Thanks to the fact that MSD-first mode of computation is used, efficient comparison algorithm can be applied.

4 FPGA implementation of the SAD processor

Figure 3 presents the architecture of the design corresponding to the SAD processor. The absolute value of the differences is computed for each pair of pixels ($|c_{i,j} - r_{i,j}|$) and their summation is calculated on the N^2–Operand OLA adder. The result is stored digit by digit on SADc register, and it is simultaneously compared with the corresponding digit of SADr in the comparator (COMP) [11]. If at any cycle the condition SADc > SADr is detected, the computation is stopped and a new candidate block is required. Otherwise, if the condition SADc < SADr is verified, SADc is stored in SADr when less significant digit of the SAD is calculated.

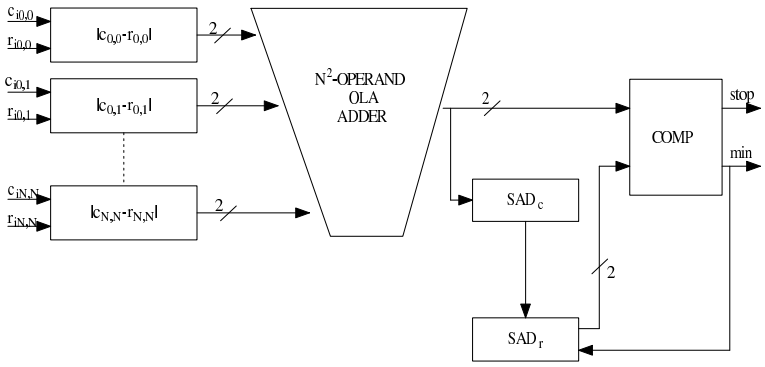

Fig. 3. SAD processor architecture

The timing of the computation for the 4×4 SAD processor is shown in figure 4. In each cycle, the outputs corresponding to the absolute value block ($|c_{i,j} - r_{i,j}|$, each of the four steps in the adder-tree ($\sum(i)$), and the comparator (COMP) are represented. In the comparator, it is not really the output but the last digit that is used for the comparison. The zeroes digits represents the zero

values which have to be introduced in the input due to the on-line delay of the system. Since each addition has an on-line delay of two, and absolute value block and comparator has no on-line delay [11], eight zeroes are required in this case.

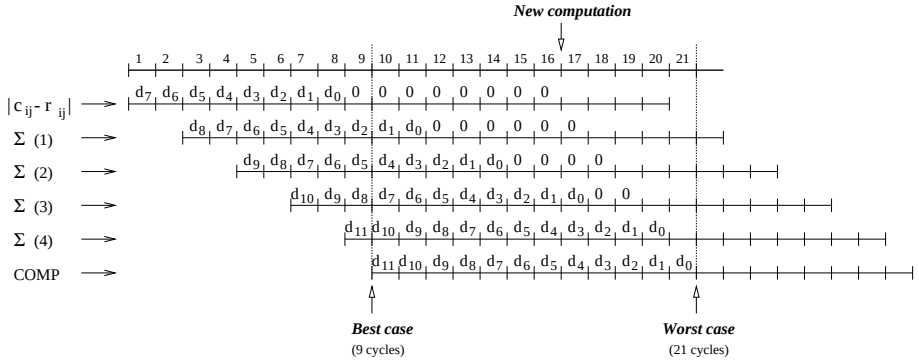

Fig. 4. Timing of the 4×4 SAD processor

The worst case takes place when a new minimum SAD is found, and then 21 cycles are required to the full process, where the last one is to store SADc in SADr. However, as figure 4 shows, a new SAD computation can start after 16 cycles (after the 8 digits and 8 zeroes are introduced) and then, this period of time is the maximum between two consecutive SAD computation. This period is reduced if the candidate SAD is rejected before. In the best case, it happens after the analysis of the MSD of the candidate SAD, that is after 9 cycles. Therefore, the number of cycles for a SAD computation and comparison is between 9 and 16 cycles for a 4×4 SAD processor. This period is in the range between 13 and 20 cycles for 8×8 block size and between 17 and 24 cycles for 16×16 block size.

A 16×16 block size requires 512 operands. A conventional representation suppose 512×8 bits, this is an unavailable number of inputs for any FPGA. With SD representation we reduce its to 512 - 2 bits operands; on the other hand, we propose to store the reference block into distributed FPGA memory, reducing to 256 - 2 bits. Also, we can implement an 8×8 block size solution with 64 - 2 bits inputs.

The design has been implemented on the Xilinx SPARTAN-II and VIRTEX-II FPGA families for three different block sizes. For compilation, simulation and implementation, we use the Xilinx ISE Series 5.2i. The main results of the implementation is shown in table 2. The ratio area/number of pixels is relatively low, due to the serial-digit nature of on-line computation. The maximum clock frequency is independent of the block size since when the number of operators increases, only the number of parallel operations and the number of steps in the adder-tree increase. Although this value strongly depends on the technology used (as we see in Table 2), our results are very promising.

Table 2. Area and clock frequency corresponding to different FPGA implementations.

	SPARTAN-II	VIRTEX-II
Block size	Area (4 inputs-LUTs)	
4x4 (16 pixels)	246	241
8x8 (64 pixels)	603	595
16x16 (256 pixels)	1982	1945
	Maximum Frequency(MHz)	
	231.24	424.99

Table 3 shows how the area and delay are distributed in the different parts of the design, for the 16×16 SAD processor. Note that the percentage given refers to the total number of LUTs of the SAD processor. The maximum clock frequency of the global system is determined by the delay of the adder, depending on the FPGA family, since the basic cells are slightly different. The area is mainly occupied by the absolute value blocks and the adder-tree, due to the large amount of operands for this block size.

Table 3. Distribution of LUTs and delay in the 16×16 SAD processor.

	Time Delay (ns)		Area	
Parts	SPARTAN-II	VIRTEX-II	LUTs	%
Absolute difference	3.675	1.839	1024	52.7%
Adder-tree	4.3246	2.353	768	39.5%
Comparator	3.783	1.907	7	0.3%
Control and Connectivity	-	-	146	7.5%

5 Comparison

In [1] the computation of the SAD for 16 pixels (SAD16), which is equivalent to a a row of a macroblock for MPEG, is implemented on a FPGA device. The design is based on carry–save adders which perform the computation in parallel over all the digit of the data. According to the authors, the design is synthesized using FPGA Express from Synopsis by targeting the FLEX20KE family from Altera, obtaining an area of 1699 LUTs and a maximum frequency of 197 MHz with a latency of 19 cycles (96ns). The results of our implementation using the VIRTEX-II family is used for comparison, since it has similar performance. The worst case for our equivalent design (4×4 or 16 pixels) takes place when a new minimum SAD is found, and then 21 cycles are required to the full process (see section 4), that is, to compute SAD16 plus compare with the previous minimum

and store it, which means 49ns at the frequency of 425 MHz. Besides, our design only requires 241 LUTs, that is seven times less area than in [1]. However, the current compression standard systems require 16×16 block sizes (and also 8×8 for MPEG-4).

The authors give some notes about how to extend the design to compute a 16×16 SAD in two way. The first one is based on using 16 SAD16 units (one for each row) and a final adder tree. They estimate that 27 clock cycles are required. Nevertheless, the number of LUTs for the design is close to 30000, which does not seem feasible for the current FPGA devices. Our 16×16 design requires only 1945 LUTs, which is easily implemented on a single FPGA device.

The second approach presented in [1] is based on reusing the SAD16 units to compute the SAD of all the 16 rows which are buffered to finally add up them. This involves 42 clock cycles with a larger area size due to buffering and the fact that longer binary data (16 bit instead of 12 bits) must be supported. Moreover, the intrinsic pipeline behavior of the SAD16 units is eliminated. For a similar area, our design computes a SAD each 24 cycles for the worst case (including the comparison, see section 4)).

On the other hand, the solution proposed on [4] involves the use of four Altera STRATIX EP1S80 devices with 1234 I/O pins. This design uses 7765 LCs and requires 29 cycles for a SAD computation at a frequency of 380 MHz. It means that our design obtains better time performance using much less hardware requirement.

We would like to emphasize that the previous comparisons involve our worst case (16 cycles for 4×4 SAD and 24 cycles for 16×16 SAD). However, the best case implies that after the analysis of the MSD of the candidate SAD we reject it, which involve only 9 cycles for 4×4 SAD and 17 cycles for 16×16 SAD (see section 5). On the other hand, our results include the comparison which, for the designs of [1] and [4] involve several clock cycles (due to carry propagation).

6 Conclusion

In this paper a FPGA implementation of the motion estimation core based on the computation of the minimum SAD has been stated. The proposed core can be integrated with a full–search algorithm or any more efficient search strategy. The computation is carried out by using on–line arithmetic. The different operations involved in the SAD computation have been efficiently adapted to on–line arithmetic, and a new comparator design with no on–line delay has been proposed. This allows us to implement the design in a single FPGA device. Besides, we speed up the computation by early truncation of the SAD calculation when the involved candidate is bigger than the current reference SAD. Furthermore, the FPGA implementation of the design makes possible to reconfigure the hardware to deal with 8×8 and 16×16 pixel blocks according to the MPEG-4 standard requirements.

We show the delay and area details of the implementation for 4×4, 8×8 and 16×16 pixel block sizes. We also provide comparison with other current related works, which shows the benefits of our design.

References

1. S. Wong, S. Vassiliadis, S. Cotofana "A Sum of Absolute Differences Implementation in FPGA Hardware", 28th Euromicro Conference, pp.183–188, 2002.
2. Y. Lai, L. Chen, "A data-interlacing architecture with two-dimensional data-reuse for full-search block-matching algorithm",IEEE Tran. on Circuits and Systems for Video Technology, Vol. 8 ,Is. 2, pp. 124–127, 1998.
3. S.b. Pan, S.S. Chae and R.H. Park, " VLSI Architecture for block matching algorithms using systolic arrays" IEEE Trans. Circuits Syst. Video Tech.,vol. 6, pp.67–73, Feb, 1996.
4. S. Wong, B. Stougie, S. Cotofana " Alternatives in FPGA-based SAD Implementations", Proc. IEEE International Conf. on Field-Programmable Technology, pp. 449–452, 2002.
5. C. Su and C. Jen, "Motion estimation using MSD-first processing", IEE Proc. Circuits Devices System, vol. 150, No. 2, pp. 124–133, 2003.
6. M. Ercegovac and T. Lang, "On-Line Arithmetic for DSP Applications", 32nd Midwest Symposium on Circuits and Systems, pp. 365–368, 1989.
7. Lau, D.; Schneider, A. Ercegovac, M.D.; Villasenor, J., "FPGA-based structures for on-line FFT and DCT" Proc.7th IEEE Symposium Field-Programmable Custom Computing Machines, pp. 310–311, 1999.
8. Brackert, R.H., Jr.; Willson, A.N., Jr.; Ercegovac, M.D."High-speed recursive digital filter using on-line arithmetic", IEEE Int. Symposium on Circuits and Systems, pp. 1552–1555, 1989.
9. Rajagopal, S.; Cavallaro, J.; "On-line arithmetic for detection in digital communication receivers",15th IEEE Symposium on Computer Arithmetic, pp. 257–265, 2001.
10. McIlhenny, R.; Ercegovac, M.D.;"On the design of an on-line FFT network for FPGA's", 33rd Asilomar Conference on Signals, Systems, and Computers, vol. 2, pp.1484–1488, 1999.
11. Hormigo, J.; Olivares, J.; Villalba, J.; Benavides, I.;"New on-line Comparator with no on-line delay", 8th World Multiconference on Systemics, Cybernetics and Informatics, 2004 (acepted).