Journal of Universal Computer Science, vol. 18, no. 2 (2012), 264-285
submitted: 3/1/09, accepted: 6/12/11, appeared: 28/1/12 © J.UCS

Reconfigurable VBSME Architecture Using RBSAD

Joaquín Olivares
(University of Córdoba, Córdoba, Spain
olivares@uco.es)

Abstract: This paper presents an architecture which is capable of processing variable block size motion estimation (VBSME) and which is able to apply pixel precision reduction techniques in a reconfigurable way. The design has been carried out by using online arithmetic, which allows to process all motion vectors of a block in just one iteration. The system has been implemented on FPGA and just requires 7724 slices, reaching a performance of 55 4CIF frames per second (fps) in full precision and of 72 with 4 bit precision. Results for different search areas 31×31, 32×32, and 46×46 are presented. Using 4bit precision real time processing for HDTVp is achieved. Thanks to the reduced cost and high performance, this architecture is perfect for mobile devices.

Key Words: Video, High-Speed Arithmetics, Parallel Architectures, Special-Purpose and Application-Based Systems
Category: H.5.1, B.2.4, C.1.4, C.3

1 Introduction

One of the applied techniques to save structural and computational costs regarding video compression consists in reducing the precision of pixels. In video compression in MPEG-4 AVC/H.264, one of the processes with a higher cost is the Variable Block Size Motion Estimation (VBSME)[Ruiz and Michell 2011]. The cost function which is usually evaluated in hardware processors is Sum of Absolute Differences (SAD), due to its simplicity. When applied on reduced precision pixels, the function is called Reduced Bit SAD (RBSAD). The application of RBSAD techniques in videos whose pixels are represented by 8 bits is justified due to the fact that the reduction of the precision provokes a very slight loss of quality: between 0.2 and 1% for a reduction of 2 bits, and between 1% and 2% for 4 bits [He et al. 2000][Agha et al. 2005]. Three popular videos were also studied in [Baek et al. 1996], Foreman had a loss of quality of 0.18dB for 4 bits precision, and, with the same precision, Miss America had a loss of quality of 0.77dB.

When an architecture working in RBSAD is implemented, this is usually non-reconfigurable, as the processing is based on the number of pixels and no computational saving is achieved by reducing the size of those pixels.

This paper presents an architecture which processes the motion estimation on the bit plane level, processing in each iteration the corresponding bit. If the RBSAD mode is activated, a processing saving will be automatically achieved, entailing an increase on the ratio of frames processed per second. This may also

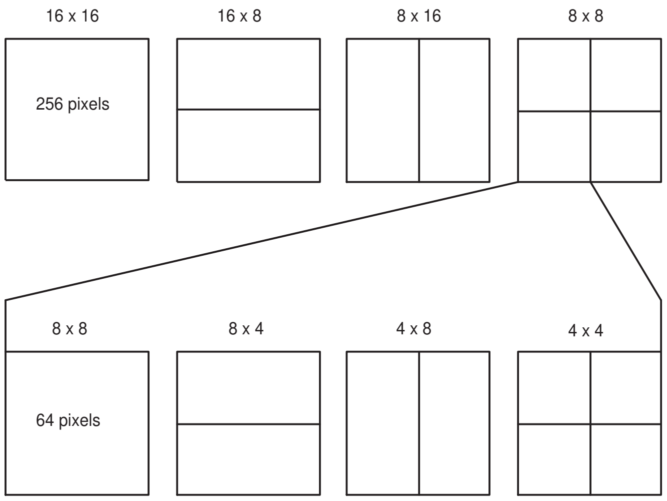

Figure 1: Block formats in MPEG-4 AVC/H.264.

reduce the necessary processing to reach real time processing, achieving at the same time a reduction in the energy consumption of the device, being this latter a desirable characteristic for processors to be used in mobile devices.

VBSME is characterized by analyzing several possibilities for a 16×16 block, considering 4×4 sub-blocks and all possible combinations for them between adjacent sub-blocks. Fig. 1 shows all possible sub-blocks. It will be necessary to process the SAD for each of them.

This paper is organized in the following way: after explaining the foundations of RBSAD, Section 2 depicts the basics on the search algorithm, on the used metrics, and on the chosen arithmetic. Section 3 includes a detailed description of all different components which have been used. Section 4 is aimed at showing the architecture, as well as explaining the pipeline and showing the timing, both for the full precision and for a 4 bit reduction. The most important results are shown in Section 5. Then, Section 6 presents other works related to this issue, comparing the most significant results in a table. In Section 7, the most important conclusions are presented.

2 Foundations

2.1 Algorithms for the calculation of the SAD

There are different algorithms for implementing the SAD processing. When developing a specific processor, Full Search Block Matching Algorithm (FSBMA) is commonly chosen for two reasons [Ghanbari 2003]: when processing all possible cases, it always obtains the optimal value, and the execution flow is regular. This latter reason allows to take advantage of the data stored in registers, as there are large overlapping data areas, and thus prevents complex data control structures which slow down the global processing. In this paper, the use of other search algorithms, which process a subset of the possible candidates [Chan and Siu 2001][Kim et al. 2002], has been discarded, as the irregular data flow provokes latencies in the processors to load the data every time a block is evaluated. The search algorithms on candidate subsets are usually employed in general purpose processors, which do not have any specific data managers for a given particular processing.

The most widely used metrics for establishing the best Motion Vector (MV) is SAD [Pan et al. 1996][Wong et al. 2002]. The block with the lowest SAD is searched in the adjustment among all possible candidate blocks. The SAD computes the absolute difference for each pixel of the candidate and reference blocks:

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

where $r_{j,k}$ and $c_{j,k}$ are the pixels of the reference block and candidate block, respectively.

2.2 Signed-Digit Representation

This work is based on Signed-Digit Radix-2 (SD) [Avizienis 1961] representation. Its properties allow to implement the difference operation with a simple intertwining of the data. It is also an arithmetic which may operate bit by bit with very basic structures. With this, a very fine grained parallelism is going to be achieved, and therefore the whole block is to be processed at the same time. The bit by bit processing, together with the processing of all pixels in parallel, allows to design a processor which is efficient and reconfigurable for the RBSAD. The SD consists of a trivalued base of elements: [-1, 0, 1], the system is redundant and the values are formed from two bits as shown in Table 1:

Online Arithmetic (OLA) [Ercegovac and Lang 2004] is used to carry out the serial sum. As can be seen later on, using OLA does not entail any additional cost in the conversion to SD, as it takes place by using the calculation of the differences. The final conversion when finishing a processing does not entail either a cost, as what is really interesting for the purposes of this paper is knowing the optimal vector. Then, the calculated SAD is discarded and it is not necessary to implement a final conversion from SD to a twos complement.

Table 1: SD representation and its binary correspondence.

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

3 Components of the architecture

When using the SD representation system and the OLA techniques, the design of the components is not usual, and it is worth being presented. The different parts of the SAD processing can be structured as follows:

– Obtaining the differences between the pixels: $d_{j,k} = c_{j,k} - r_{j,k}$

– Establishing the absolute value for each difference: $|d_{j,k}|$

– Summing all absolute differences : $\sum |d_{j,k}|$

Once a SAD is obtained, it needs to be compared to the lowest SAD value which has been obtained until that moment. For that purpose, it is necessary to design comparators for OLA and SD.

3.1 Absolute difference in SD using OLA techniques

A value represented in SD can be interpreted as the difference between two numbers represented without a sign, so that one of them provides a positive weight to the value whereas the other adds a negative weight. Thus, the bits of the pixels can be inserted without having to provide any further processing, for example: If we take two pixels, say c and r, from the candidate and reference frames, the absolute difference and the conversion to SD will be represented by the following equation:

$$\begin{array}{l} |c-r| = c_7r_7c_6r_6c_5r_5c_4r_4c_3r_3c_2r_2c_1r_1c_0r_0 \text{ when } c \leq r \\ |c-r| = r_7c_7r_6c_6r_5c_5r_4c_4r_3c_3r_2c_2r_1c_1r_0c_0 \text{ when } c \geq r \end{array} \qquad (2)$$

The Process Element (PE) which has been used implements the absolute difference in one only step. For such purpose, it examines the bits of each pixel in Most Significant Digit (MSD) mode, and directly sends them to the first stage of the adder tree. Fig. 2 shows the basic PE of the presented architecture.

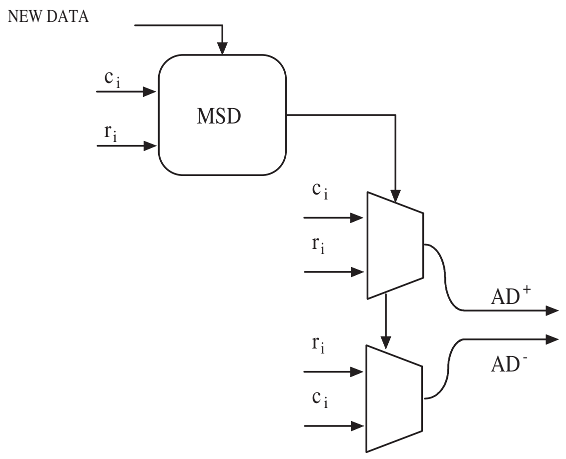

Figure 2: PE processing absolute differences.

As can be seen, it just consists of two 2:1 multiplexors, so that the output is selected depending on a comparator which determines that the generated value is in absolute value.

The comparator receives the digits of a bit coming from the pixels and closes a latch which determines the active multiplexor as the compared digits are different. If the compared digits are equal and the latch is not closed, it is irrelevant which of the two multiplexors is active.

Since the MSD-first mode of computation is being used, the sign detection of $d_{j,k}$ is performed on-the-fly by checking whether the first non-zero digit of $d_{j,k}$ is positive (01) or negative (10). The digits of $d_{j,k}$ are received in MSD-first mode and go directly to the output when they are zero (00 or 11). If the first non-zero digit which is received is positive (01), this and all the remaining digits correspond directly with the output. Nevertheless, if the first received non-zero digit is negative (10), the bits of this and all the remaining digits are interchanged to obtain the output. In Table 2, the values considered to design the MSD module shown in Fig. 2 are detailed. c_i and r_i columns represent possible combinations for bit i. The column *Operation* is $c < r$ or $c > r$ when the relation between c and r is known, otherwise, this field appears in blank. In these cases, no changes succeed. – is used in impossible combinations.

The signals to interpret it are:

- c: candidate macroblock pixel.
- c_i: bit of c.
- r: reference macroblock pixel.

Table 2: Combinations processing AD

Operation	c_i	r_i	L_n	G_n	AD	L_{n+1}	G_{n+1}
	0	0	0	0	00	0	0
	0	0	0	1	–	–	–
	0	0	1	0	00	1	0
	0	0	1	1	00	1	1
$c < r$	0	1	0	0	01	1	0
	0	1	0	1	–	–	–
$c < r$	0	1	1	0	01	1	0
$c > r$	0	1	1	1	10	1	1
$c > r$	1	0	0	0	01	1	1
	1	0	0	1	–	–	–
$c < r$	1	0	1	0	10	1	0
$c > r$	1	0	1	1	01	1	1
	1	1	0	0	11	0	0
	1	1	0	1	–	–	–
	1	1	1	0	11	1	0
	1	1	1	1	11	1	1

– r_i: bit of r.

– $NewData$: 1 when a new AD will be processed. Otherwise, 0.

– L: Latch. Close to 1 when both digits are different. 0 at the beginning.

– G: Greater. Close to 1 when $c > r$. 0 at the beginning.

Based on this analysis, the MSD block is designed with a simple state machine, as can be seen on Fig. 3

3.2 The adder and the adder tree using OLA techniques

When working with OLA techniques, the adder shall sum two numbers in SD representation, each of them consisting of one SD digit, and consisting each digit of two bits. Due to this, the OLA basic adder implements the adding in two stages[Ercegovac and Lang 2004]. In future it might be useful to study whether it is possible to improve the adder tree using heuristics proposed recently in [Parandeh-Afshar et al. 2011]. Furthermore, it is necessary to take into account that each SD digit has a negative weight and a positive one. The architecture of the OLA adder can be seen on Fig.. 4. Two types of OLA adders will be used, with or without output flip-flops.

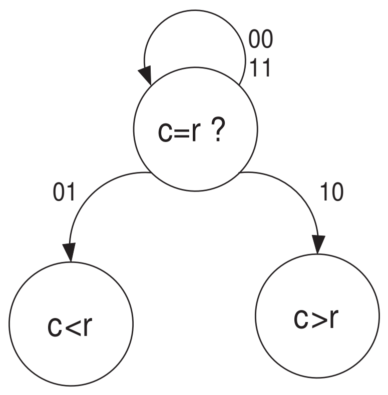

Figure 3: MSD block.

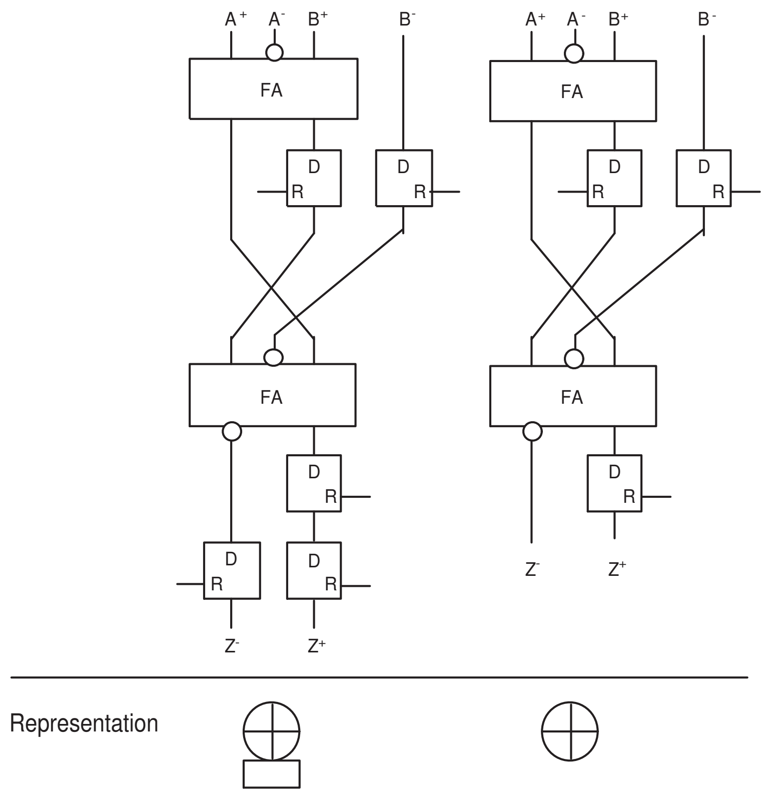

Figure 4: OLA adders.

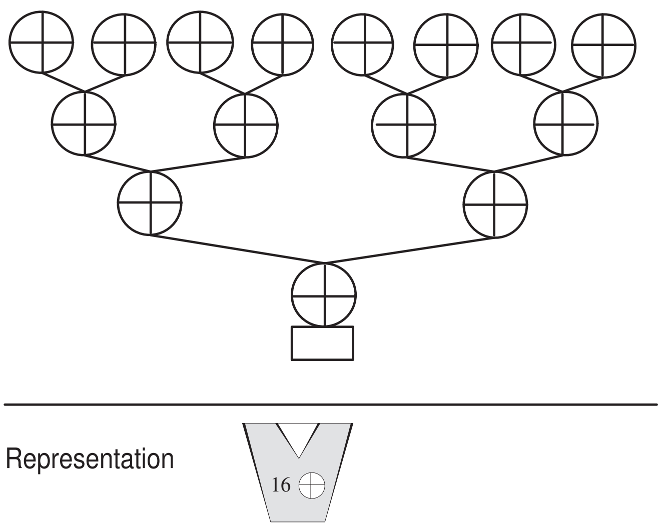

Figure 5: Adder tree: 16-OLA adder tree.

From it, an adder tree able to sum 16 absolute differences is developed. This adder tree is called *16-OLA Adder tree* and will be used afterwards, see Fig. 5.

When the processing on the SAD over VBSME is carried out, it is common in other architectures to need more than one round, as the block is not fully processed in each iteration [Chen et al. 2006]→[Wei et al. 2008]. This architecture works with a very fine grain, in such a way that parallel processing of N^2 pixels of the block may be made, being $N = 16$. As it is possible to process the whole block in parallel in just one iteration, it is very simple to calculate the vectors of all possible sub-blocks in parallel. For this purpose, a first adder called *VBSME-N8*, which is characterized by processing the SAD for a sub-block of 8×8 and of all its smaller size sub-blocks, has been designed, and 4 *16-OLA adder tree* adder trees are used in it. The *VBSME-N8* adder is presented on Fig. 6. As can be seen, it is similar to a common adder tree for of 8×8, adding two more adders in order to obtain the SAD of the 4×8 sub-blocks.

Using the adder *VBSME-N8* as the base, the global adder used in this architecture is built. This adder can be seen on Fig. 7 and is called *VBSME-N16*. It is built by adding two adders on a common adder tree, just as has happened with *VBSME-N8*, so as to obtain the SAD of the 8×16 sub-blocks. Each processed SAD needs to be compared. The design of the OLA comparator is shown in the following subsection.

The global hardware increment respect to a conventional 16×16 adder tree is only ten SD OLA adders. Also 41 registers are used to store the SAD for all

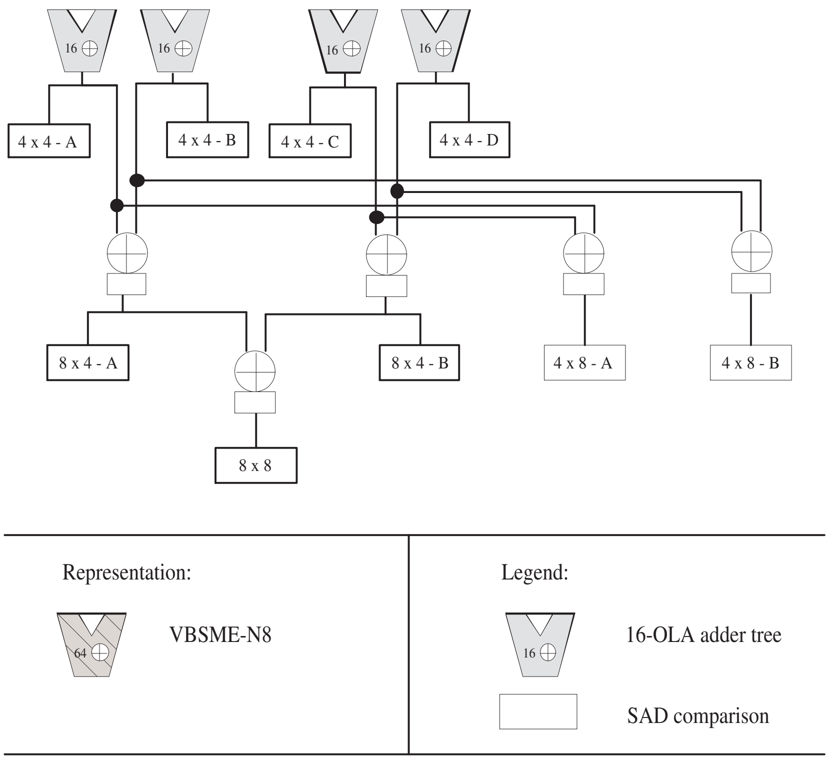

Figure 6: Adder tree: VBSME-N8.

possible sub-blocks.

3.3 The SD comparator for OLA technology

Differently to what happens in a comparator based on traditional arithmetic, where, as soon as two different digits working in MSD mode are found, the highest number of those to be compared is established, when comparing SD digits, it must be kept in mind that they can have a negative weight. This can mean that it is not possible to establish such dimension relationship when comparing two numbers and evaluating the first pair of different digits. The architecture of the comparator is obtained from the finite state machine shown on Fig. 8.

The mathematical justification which allows to obtain such state machine is thoroughly detailed in [Olivares et al. 2006]. Other work based on previous architecture [Song and Akoglu 2011], presents early termination for VBSME using SD arithmetic.

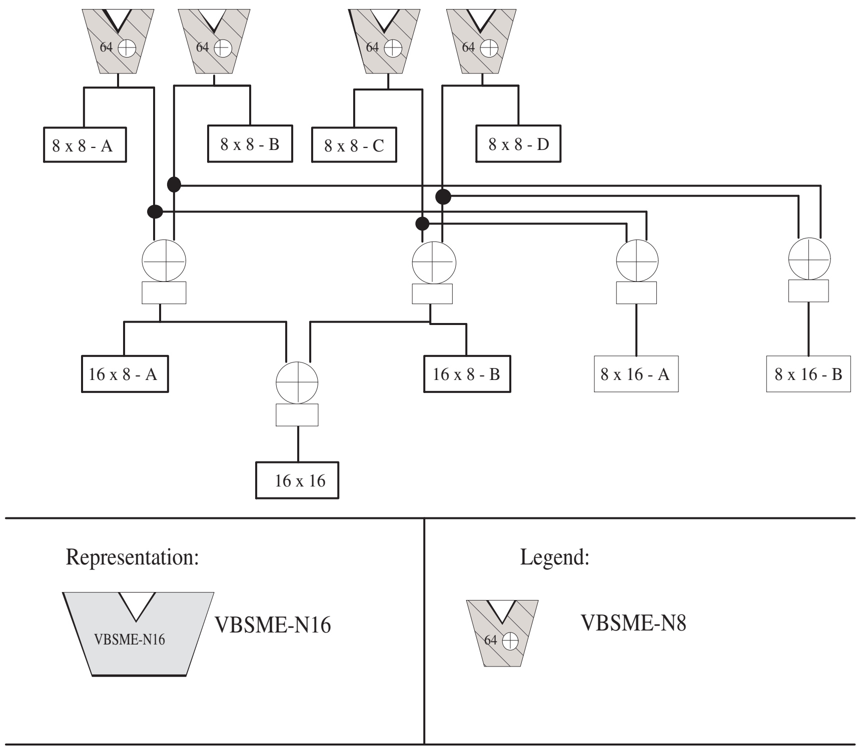

Figure 7: Adder tree: VBSME-N16.

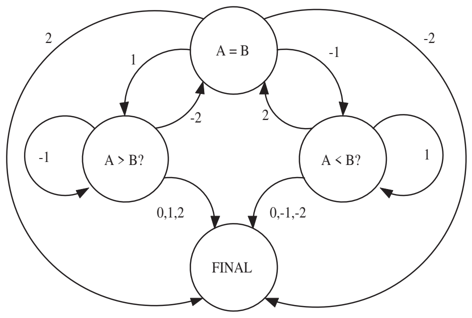

Figure 8: Finite state machine which is used as the base for the design of the OLA comparator.

4 RBSAD architecture for VBSME

Once the components have been presented, the architecture is formed from those components, beginning on a first level for the processing of all absolute differences

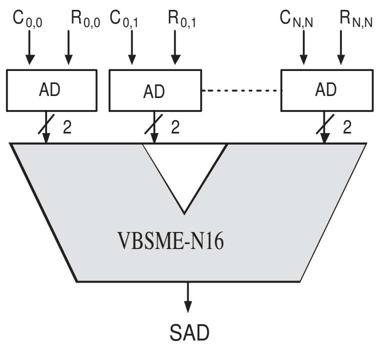

Figure 9: Global architecture of the full system.

of a block, that is to say, $N^2 = 256$ absolute differences in as many PE. On a second level, *VBSME-N16* structure can be found, which includes the adder tree and which will have the 41 comparators for all calculated vectors integrated. This is presented on Fig. 9.

4.1 Analysis of the segmentation

The analysis takes place according to the different segments of the architecture. The first segment is used for the processing of the absolute differences, being the following ones used for the adder tree. A last segment has to be added for the comparator.

Table 3 presents the timing for full precision, whereas Table 4 shows the timing for reduced precision to 4 bits per pixel. As can be seen, for full precision 28 clock cycles are necessary to compare all the bits of the 16×16 SAD, whereas for the 4 bit precision case, it is reduced to 24 clock cycles. However, this parameter is not very relevant, as the timing depends indeed on the number of bits per pixel and on a series of zeros which need to be inserted in the adder tree for a correct adding, corresponding the number of zeros with the number of levels on the adder tree: $log_2 N^2$. Tables 3 and 4 are to be interpreted in the following way: column AD shows each digit on which the AD has been processed, column S(4×4) is shown when every digit corresponding to the SAD for a 4×4 block is obtained. The following columns need to be interpreted in the same way, each of them specifying the clock cycle when the result for the SAD of the corresponding sub-block is obtained. The last column, Comp, shows in which clock cycle, the result of the comparison with the best SAD for each of the sub-blocks is obtained.

$log_2 N^2$ segments are used in the adder tree, represented on columns $S(i)$, and a final segment is needed for the last comparator. Sub-block comparisons

Table 3: Timing: Full precision

		Add						
Clock cycle	AD	4x4	4x8 8x4	8x8	8x16 16x8	16x16	Comp	
1	d7							
2	d6							
3	d5							
4	d4	d11						
5	d3	d10					C4x4	
6	d2	d9	d12					
7	d1	d8	d11				C4x8-8x4	
8	d0	d7	d10	d13				
9	0	d6	d9	d12			C8x8	
10	0	d5	d8	d11	d14			
11	0	d4	d7	d10	d13		C8x16-16x8	
12	0	d3	d6	d9	d12	d15		
13	0	d2	d5	d8	d11	d14	C16x16	
14	0	d1	d4	d7	d10	d13	d14	Next
15	0	d0	d3	d6	d9	d12	d13	MB
16	0	0	d2	d5	d8	d11	d12	begins
17	d7	d0	d1	d4	d7	d10	d11	
18	d6	0	d0	d3	d6	d9	d10	
19	...	0	0	d2	d5	d8	d9	
20		d11	0	d1	d4	d7	d8	
21		d10	0	d0	d3	d6	d7	
22		...	d12	0	d2	d5	d6	
23			d11	0	d1	d4	d5	
24			...	d13	d0	d3	d4	
25				d12	0	d2	d3	
26				...	d14	d1	d2	Current
27					d13	d0	d1	MB
28					...	d15	d0	ends
29						d14	d15	

are shown on column *Comp*. The first digit of all 4×4 SAD sub-blocks can be compared starting after clock cycle 4; all 4×8 and 8×4 SAD sub-blocks comparisons start after clock cycle 6; 8×8 sub-blocks comparisons start after 8; 16×8 and 8×16 start after 10; and, the final 16×16 comparison starts after 12. All comparisons are finished at clock cycle 28 but a new macroblock starts its

Table 4: Timing: 4bit precision

		Add						
Clock	AD	4x4	4x8	8x8	8x16	16x16	Comp	
cycle			8x4		16x8			
1	d7							
2	d6							
3	d5							
4	d4	d11						
5	0	d10					C4x4	
6	0	d9	d12					
7	0	d8	d11				C4x8-8x4	
8	0	d7	d10	d13				
9	0	d6	d9	d12			C8x8	
10	0	d5	d8	d11	d14			Next
11	0	d4	d7	d10	d13		C8x16-16x8	MB
12	0	0	d6	d9	d12	d15		begins
13	d7	0	d5	d8	d11	d14	C16x16	
14	d6	0	d4	d7	d10	d13	d14	
15	...	0	0	d6	d9	d12	d13	
16		d11	0	d5	d8	d11	d12	
17		d10	0	d4	d7	d10	d11	
18		...	d12	0	d6	d9	d10	
19			d11	0	d5	d8	d9	
20			...	d13	d4	d7	d8	
21				d12	0	d6	d7	
22				...	d14	d5	d6	Current
23					d13	d4	d5	MB
24					...	d15	d4	ends
25						d14	d15	

processing at clock cycle 17 for $N = 16$ thanks to the pipeline, which is shown in Table. 3.

In Table 4 the timing for 4 bit precision is shown. The first digit of all SAD sub-blocks can be compared starting at same clock cycle of full precision. 4 clock cycles are saved to start the next block.

The analysis of the pipeline is the following, $2log_2N + 2$ clock cycles are required to process the most significant bit of each pixel. In full 8 bit representation, another 7 clock cycles must be added for the rest of bits, which means a total of $2log_2N+9$ necessary clock cycles for fully completing a block. To process with reduced precision, it is necessary to subtract from such formula the number of bits which we wish to reduce, and which we will call RB, being then the final formula as follows:

$$2log_2N + 9 - RB \text{ clock cycles per block} \tag{3}$$

5 Results

5.1 Hardware implementation

FPGAs are useful in video coding tasks due to high-performance and parallel processing [Torabi and Vafaei 2012]. The architecture has been modeled in VHDL, has been compiled using ISE 9.2, and has been implemented in a Virtex5 series XC5VLX50 FPGA, included in a Xilinx Virtex-5 LXT PCI Express Development Kit. ISE frequency was 380.1 MHz, since this board includes a 100 MHz oscillator, 380 MHz clock signal was generated using Digital Frequency Synthesis (DFS), multiplying by 19/5.

The results are presented in Table 5 for the architecture and for the global system, featuring the data managers:

Table 5: Results

Architecture	Slices	2,899
	LUTs	2,345
Global System	Slices	7,724
	LUTs	3,556
Frequency		380.1 MHz

The performance of the developed system is presented depending on the number of frames per second (fps) which the architecture is able to process for a video format 4CIF, see Table 6. The architecture and the data managers have been developed to work with a search window of 32×32. The performance for a window of 31×31 and of 46×46 is also shown, so as to compare other works.

The maximum clock frequency is $380.1MHz$. To work on real-time processing of 4CIF video format at 30 frames per second (fps), a clock frequency of $233.47MHz$ is required. Using full precision a maximum throughput of 55.14 is obtained for a search window of 31×31, using 4bit precision this throughput increases up 72.10 fps.

Table 6: Throughput

Search window:	32 × 32	31 × 31	46 × 46
Precision	4CIF fps	4CIF fps	4CIF fps
8	48.85	55.14	14.69
7	51.90	58.58	15.60
6	55.36	62.49	16.65
5	59.32	66.95	17.84
4	63.88	72.10	19.21

Table 7: PSNR average drop using RBSAD

	Sub-block size		
Precision	16×16	8×8	4×4
NTB=2	0.03%	0.10%	0.19%
NTB=4	0.19%	1.54%	6.42%
NTB=6	3.84%	13.09%	29.64%

With HDTVp 1280 × 720 format, a throughput of 24.26 fps is obtained. Using 4bit precision this value is improved to 31.73 fps, real time processing for HDTVp is achieved.

Besides, the design for obtaining the equivalent cost of a VLSI circuit has been compiled, obtaining a cost of 54Kg for a technology of 0.13μm. This architecture can be useful for mobile devices.

5.2 PSNR evaluation

The PSNR drop in function of the NTB is presented in Table 7 for several sub-block sizes, these average values were obtained with Foreman, Miss America, and Table Tennis sequences. Foreman PSNR is also analyzed in [Chen et al. 2008]. For a 16 × 16 block, the PSNR drops less than 0.2% for 2 and 4 truncated bits, which means that the truncated pixel is more likely to have the same motion vector as the untruncated pixel. However, for the 4×4 block size, and NTB = 4, the PSNR drops 6.4%. This is because there are more matched candidates using a truncated pixel for the 4 × 4 sub-block, which could lead to incorrect motion vectors.

Using smaller block sizes, the number of pixels involved during motion prediction is reduced. Due to the truncation error, there is a tendency for smaller

Table 8: Comparison of results

	Ribeiro'07	Yap'03	Yap'04	Sayed'08	Song'06	This work full prec.	This work 4bit prec.
# PE	8	16	16	31	16	256	256
Search window	32×32	31×31	31×31	46×46	32×32	31×31	31×31
CC/MB	-	4496	4096	-	4096	4352	3328
4CIF (fps)	12	13	45	20	40.7 estimated	55.1	72.1

Table 9: VLSI Technological results

	Yap'03	Yap'04	Sayed'08	Song'06	This work
Gate count (Kgates)	108	61	175	67.7	54
Frequency (MHz)	100	294	187.7	266	-
Technology (μm)	0.13	0.13	0.13	0.18	0.13

Table 10: FPGA Technological results

	Ribeiro'07	Sayed'08	This work	
Slices	4730		7724	7724
LUTs		22010	3556	4125
Freq. (MHz)	323		380.1	266.4
FPGA	Virtex 4	Virtex 4	Virtex 5	Virtex 4

blocks to yield matched candidates, which could lead to the wrong motion vector. Thus, it could assert that truncating pixels using smaller blocks results in poor prediction [Bahari et al. 2009].

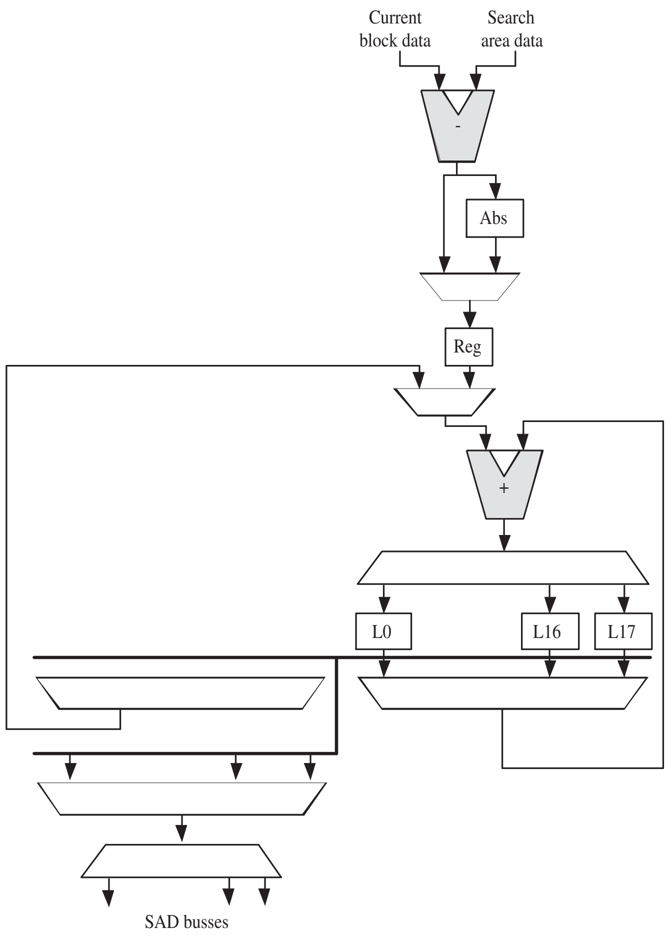

Figure 10: Yap'03: Process element.

6 Other works

This section presents other recent works in order to evaluate and compare the designed architecture. The results are shown in Table 8,9,10, so they can be easily compared.

Table 8 offers data on the architecture and performance, where CC/MB is Clock Cycles/Macroblock. Table 9 is devoted to technological data for the systems which have been implemented on VLSI. Finally, Table 10 shows the technological data for the architectures which have been implemented on FPGA.

6.1 Ribeiro'07

In [Ribeiro and Sousa 2007], a reconfigurable architecture for several search algorithms is presented. The low throughput is not relevant because this architecture is not optimized to improve the results for FSBMA. However, this architecture is interesting to show how other search algorithms work in hardware processors.

6.2 Yap'03

In [Yap and Mccanny 2003], a 1D architecture is presented, the key aspect is the shuffling, permutation, and combination of partial SAD values within each PE. The author claims that this architecture is suitable for devices requiring a small silicon area. The PE of this architecture is shown in Fig. 10.

As there is another improved architecture presented by the same author analyzed in subsection 6.3, the comparison will be based on this latter.

6.3 Yap'04

In [Yap and Mccanny 2004], Yap presents a VLSI architecture similar to the one presented before. The PE has been improved, see Fig. 11, and the architecture saves 20 latency cycles. It is a curious work, as the structure of the architecture is very similar to that of subsection 6.2, whereas the PE is more complex and has a higher hardware cost. However, comparing the results with his previous work, the author achieves a saving of more than 43% in the hardware cost and multiplies by 3 the highest work frequency.

If the work presented here in full precision mode is compared with the architecture of Yap'04, a better performance is achieved, with an improvement of more than 20%. In the same way, the cost is reduced in 10%. When RBSAD is applied, the improvement in performance represents more than 50%.

6.4 Sayed'08

The architecture presented in [Sayed et al. 2008] worked on a search window of 46×46, processing 31 vectors in parallel. Though the performance is just 20 fps, it is not directly comparable, since it develops a much higher search work than the other evaluated architectures. The cost of this architecture implemented on a Virtex4 FPGA is very high.

To compare with this author, column 46×46 of Table 6 has been filled in. For full precision and for 4bit precision the performance obtained multiplies by 2.75 and 3.6 in comparison with the architecture of Sayed, respectively. On the other hand, the architecture presented in this work saves more than 83% of the

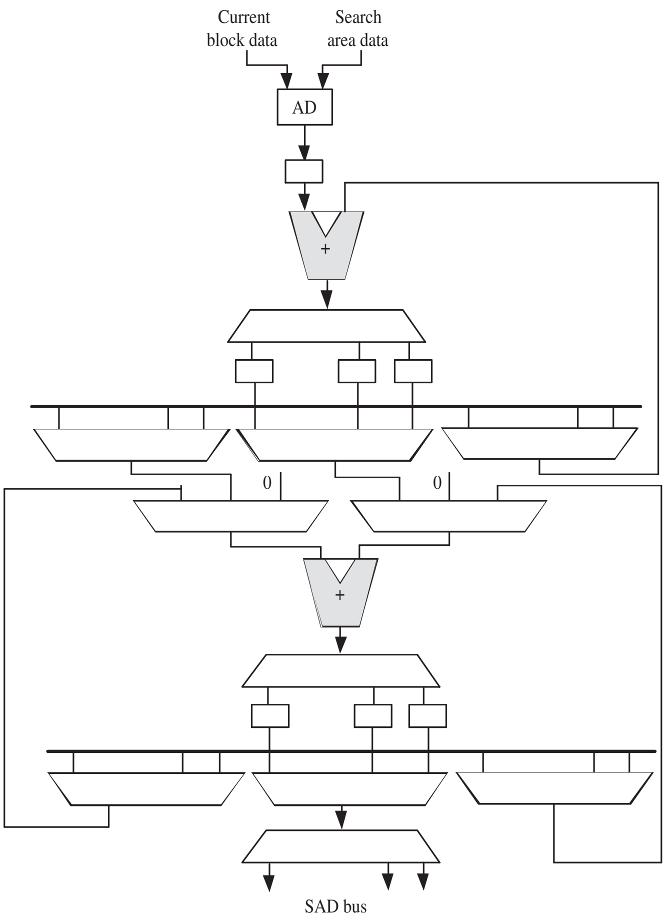

Figure 11: Yap'04: Process element.

hardware cost in the comparison in LUTs, whereas the saving in gates has been estimated in around 70%.

The architecture of Sayed has been shown because, if adapted to a search of 31 × 31, reducing from 31 to 16 PEs, the hardware cost of the SAD is also expected to be reduced in almost 50%, and the performance is to increase in a similar percentage as well, as it will pass from 31 iterations to 16. With these approximations, the architecture is estimated to reach 40 fps with a cost of around 90 Kg. Also with these estimations our architecture has better performance and hardware cost.

6.5 Song'06

Song presents a VLSI architecture which achieves a high performance with an adjusted hardware cost. It is also a 1D architecture. In full precision mode, the architecture presented in this paper achieves a higher performance with an improvement of over 37% for Song. The cost is also reduced in 20%. When RB-SAD is applied, the improvement in performance increases, with improvements of more than 75%. In this way, no hardware cost reduction has been estimated, as the system is reconfigurable and admits different precisions, and therefore the cost adjusts to the highest precision case.

7 Conclusions

This paper has presented a bidimensional architecture capable of processing the N^2 in parallel thanks to bit level processing instead of full pixel level processing. An architecture with a really small hardware cost and with a high performance is achieved, which makes it perfect for mobile devices.

The reduction of the precision of the pixels enables a significant increase of the performance, up to 31% for a 4 bit reduction. This performance improvement cannot be achieved in other architectures which are based on the processing on pixels, as this one does not depend on the number of bits per pixel.

The presented architecture can be compared to the others in full precision mode. This architecture achieves a higher performance than the rest, as well as a reduction in cost. These improvements are due to the fact that the architecture can work in 2D without increasing the cost, because of the work on bit plan level. The high processing frequency achieved is also due to the fine grain of the PE. When RBSAD is applied, the improvement in performance increases significantly over the others.

This architecture, once implemented on FPGA, can reconfigure the precision with which the system works, depending on a parameter which controls the counters managing the data refresh and the control system.

Acknowledgment

The author wishes to remark the great task carried out by the Xilinx University Program, XUP, which have donated software and materials which have been very useful to partially finance this work.

References

[Agha et al. 2005] Agha, S., Dwyer, V.M., Chouliaras, V.: "Motion estimation with low resolution distortion metric"; Electronics Letters, vol. 41, no. 12, (Jun 2005).

[Avizienis 1961] Avizienis, A.: "Signed Digit Number Representation for Fast Parallel Arithmetic"; IRE Trans. on Electronic Computers, Vol. EC-10, (1961), 389400.

[Baek et al. 1996] Baek, Y., Oh, H.S., and Lee, H.K.: "An efficient block-matching criterion for motion estimation and its VLSI implementation"; IEEE Trans. on Consumer Electronics, 42 (1996), 885–892.

[Bahari et al. 2009] Bahari, A., Arslan, T., Erdogan, A. T.: "Low-Power H.264 Video Compression Architectures for Mobile Communication", IEEE Trans. on Circuits and Systems for Video Technology, vol. 19, no. 9, pp. 1251–1261, Sept. 2009.

[Chan and Siu 2001] Chan, Y., Siu, W.: "An efficient Search Strategy for Block Motion Estimation Using Image Features"; IEEE Trans. on Image Processing, vol.10, (Aug 2001), 1223–1238.

[Chen et al. 2006] Chen, C.-Y., Chien, S.-Y., Huang, Y.-W., Chen, T.-C., Wang, T.-C., Chen, L.-G.: "Analysis and Architecture Design of Variable Block-Size Motion Estimation for H.264/AVC"; IEEE Trans. on Circuits and Systems-I: Regular Papers, vol. 53, no. 2, (Febr 2006), 578–593.

[Chen et al. 2008] Chen, Y-H., Chien, S-Y., Chen, C-Y., Huang, Y-W., Chen, L-G.: "Analysis and Hardware Architecture Design of Global Motion Estimation"; Journal of Signal Processing Systems, vol. 53, issue 3, (Dec 2008), 285–300.

[Ercegovac and Lang 2004] Ercegovac, M., Lang, T. : "Digital Arithmetic"; Morgan Kaufmann, (2004).

[Ghanbari 2003] Ghanbari, M.: "Standard Codecs: Image Compression to Advanced Video Coding"; The Institution of Electrical Engineer, (2003).

[He et al. 2000] He, Z-L. , Tsui, C-Y. , Chan, K-K, Liou, M.L.: "Low-power VLSI design for motion estimation using adaptive pixel truncation"; IEEE Trans. on Circuits and Systems for Video Technology, vol. 10, issue 5, (Aug 2000), 669–678.

[Kim et al. 2002] Kim, J., Byun, S., Kim, Y., Ahn, B.: "Fast Full Search Motion Estimation Algorithm Using Early Detection of Impossible Candidate Vectors"; IEEE Trans. on Signal Processing, vol. 50, (Sep 2002), 2355–2365.

[Liu et al. 2007] Liu, Z., Huang, Y., Song, Y., Goto, S., Ikenaga, T.: "Hardware-Efficient Propagate Partial Sad Architecture for Variable Block Size Motion Estimation in H.264/AVC"; In Proc. of ACM Great Lakes Symposium on VLSI'2007., (2007), 160–163.

[Olivares et al. 2006] Olivares, J., Hormigo, J., Villalba, J., Benavides, J.I., Zapata, E.: "SAD Computation Based on Online Arithmetic for Motion Estimation"; Microprocessors and Microsystems, vol. 30, issue 5, (2006), 250–258.

[Pan et al. 1996] Pan, S.B., Chae, S.S., Park, R.H.: "VLSI Architecture for Block matching Algorithms using Systolic Arrays"; IEEE Trans. on Circuits and Systems for Video Technology, vol. 6, (Feb 1996), 67–73.

[Parandeh-Afshar et al. 2011] Parandeh-Afshar, H., Neogy, A., Brisk, P., Ienne, P.: "Compressor tree synthesis on commercial high-performance FPGAs"; ACM Trans. Reconfigurable Technol. Syst., vol. 4, issue 4, (Dec 2011), 1–19.

[Ribeiro and Sousa 2007] Ribeiro, M., Sousa, L.: "A Run-Time Reconfigurable Processor for Video Motion Estimation"; In Proc. of Field Programmable Logic and Applications, IEEE International Conference on., (2007), 726–729.

[Ruiz and Michell 2011] Ruiz, G. A., Michell, J. A.: "An Efficient VLSI Architecture of Fractional Motion Estimation in H.264 for HDTV"; Journal of Signal Processing Systems, vol. 62, issue 3, (2011), 443–457.

[Sayed et al. 2008] Sayed, M., Badawy, W., Jullien, G.: "Towards an H.264/AVC HW/SW Integrated Solution: An Efficient VBSME Architecture"; IEEE Trans. on Circuits and Systems-II: Express Briefs, vol. 55, no. 9, (Sept 2008), 912–916.

[Song and Akoglu 2011] Song, Y., Akoglu, A.: "Bit-by-Bit Pipelined and Hybrid-Grained 2D Architecture for Motion Estimation of H.264/AVC"; Journal of Signal Processing Systems, (2011), 1–14.

[Song et al. 2006] Song, Y., Liu, Z., Ikenaga, T., Goto, S.: "VLSI Architecture for

Variable Block Size Motion Estimation in H.264/AVC with Low Cost Memory Organization"; IEICE Trans. Fundamentals, vol. E89, no. 12, (Dec 2006), 3594–3601.

[Torabi and Vafaei 2012] Torabi, M., Vafaei, A., "A Fast Architecture for H.264/AVC Deblocking Filter Using a Clock Cycles Saving Process"; Journal of Signal Processing Systems, (Jan 2012), 1–8;

[Wei et al. 2008] Wei, C., Hui, H., Jiarong, T., Jinmei, L., Hao, M.: "A High-performance Reconfigurable VLSI Architecture for VBSME in H.264"; IEEE Trans. on Consumer Electronics, vol. 54, no. 3, (Aug 2008), 1338–1345.

[Wong et al. 2002] Wong, S., Vassiliadis, S., Cotofana, S.: "A Sum of Absolute Differences Implementation in FPGA Hardware"; 28th Euromicro Conference (EUROMICRO'02), Dortmund, Germany, (2002), 183–188.

[Yap and Mccanny 2003] Yap, S.Y., Mccanny, J.V.: "A VLSI Architecture for Advanced Video Coding Motion Estimation"; In Proc. of Application-Specific Systems, Architectures, and Processors, 2003. IEEE International Conference on., (2003), 293–301.

[Yap and Mccanny 2004] Yap, S.-Y., McCanny, J.V.: "A VLSI Architecture for Variable Block Size Video Motion Estimation"; IEEE Trans. on Circuits and Systems II. vol. 51, no. 7, (Jul 2004), 384–389.