

# Accumulator Size Minimization for a Fast Cumulant-Based Motion Estimator

Jaime S. Cardoso, *Student Member, IEEE*, and Luís Corte-Real, *Member, IEEE*

**Abstract**—The implementation of fast dedicated processor for block matching motion estimation based on cumulants matching criteria implies the optimization of all of its components. Special care should be spent with the multiply-accumulate unit that is the core of many digital signal processing systems. Therefore, its optimization may be of outmost importance, specially if a significant number of such units are present in the platform. In this paper, the minimization of the size of one such unit is provided for a specific application, although the results have relevance in other scenarios.

**Index Terms**—Cumulants, higher order statistics, multiply-accumulate unit, optimization.

## I. INTRODUCTION

THE MOTION estimation is usually the most time consuming task in video processing systems, specially if complex tasks like cumulant-based block matching is used. It implies that for real time applications this critical operation must be implemented with a special care.

One way to achieve the high performance required in such situations is to use custom-computing machines, specifically tuned for the applications.

We developed a dedicated processor and implemented it in a field-programmable gate array (FPGA) based computing platform, [1]. This system was built as an expansion slot card for the PC bus. The custom processor executes the most computationally intensive task of motion estimation: the block-matching algorithm using a cumulant matching criteria.

In order to optimize the mapping of the dedicated processor into the FPGA platform, special care was taken in the optimization of all of its elements.

This paper presents the optimization effort carried on in the implementation of one of such elements, the multiply-accumulate (MAC) unit. Results can be adapted to other applications.

### A. Cumulant-Based Motion Estimation

The implemented algorithm, based on third-order cumulants (see [2]) is the block matching algorithm described in [3]. The estimated displacement vector  $d$ , belongs to the search region  $\mathcal{R}$ , assumed to contain the largest possible horizontal and vertical displacement and minimizes

$$J_3(d) \triangleq \sum_m \sum_n \left| \hat{C}_{f_{k-1}f_k f_{k-1}}(m; n) - \hat{C}_{f_{k-1}f_{k-1}f_{k-1}}(m - d; n) \right|$$

Manuscript received October 24, 2003; revised August 19, 2005. This paper was recommended by Associate Editor S. Chen.

The authors are with Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto/INESC Porto, Porto 4200-465, Portugal (e-mail: jaime.cardoso@inescporto.pt; lreal@inescporto.pt).

Digital Object Identifier 10.1109/TCSVT.2005.858614

where  $f_k$  is the current frame,  $f_{k-1}$  is the previous one,  $m, n$ , and  $d$  are 2-D variables,  $\sum_m = \sum_{m_1} \sum_{m_2}$ ,  $\sum_n = \sum_{n_1} \sum_{n_2}$ , and  $\hat{C}_{f_1 f_2 f_3}(m; n)$  is a estimation of the third-order cumulant

$$C_{f_1 f_2 f_3}(m; n) \triangleq E[f_1(x)f_2(x+m)f_3(x+n)].$$

Given  $N \times N$  blocks of the 2-D processes  $f_1(x)$ ,  $f_2(x)$ , and  $f_3(x)$ , the estimation of the third order cumulant is defined as

$$\hat{C}_{f_1 f_2 f_3}(m; n) \triangleq \frac{1}{N^2} \sum_{x=1}^N f_1(x)f_2(x+m)f_3(x+n)$$

where  $\sum_{x=1}^N = \sum_{x_1=1}^N \sum_{x_2=1}^N$ . Since the computational requirement for the third order cumulant estimation, as defined above, is too high for real-time implementation, we have implemented the following simplifications.

- 1) For each image line of a block,  $m_{f_k}(i)$ , subtract the average of that line from its elements, in order to get the normalized lines  $xo_{f_k}$ .
- 2) Estimate the third-order moments for each normalized line  $xo_{f_k}$  and for each pair of values  $(m; n)$  of interest, in the following way:

$$m_3^{xo_{f_k}}(m; n) = \frac{1}{N} \sum_{l=\min}^{l=\max} xo_{f_{k-1}}(l)xo_{f_k}(l+m)xo_{f_{k-1}}(l+n),$$

where  $\min$  is equal to the maximum value of  $(0, -m, -n)$  and  $\max$  is equal to the minimum of  $(N-1, N-1-m, N-1-n)$ —now  $l, m$ , and  $n$  are 1-D variables. The maximum used index of variation was 3 for  $m$  and  $n$ , as used in [3].

- 3) Estimate the third-order moment of the block as the average of the moments of each line.

By software simulation in a group of test sequences we have verified that these simplifications have a minor impact in the estimated motion vector, while reducing the computation load as much as fifty times.

### B. Custom Processor

**1) Reconfigurable Platform:** The reconfigurable platform is composed by two main boards (see Fig. 1). The RVC (Reconfigurable Vector Coprocessor) has 5 XILINX FPGAs (X0, X1, X2, X3, and X4), a main memory of 1 M  $\times$  32 bit and four separate 256 K  $\times$  8 bit SRAMs (see [4]).

A second board that hosts up to four processing nodes (PPK, after Polygon Positioning Kernel, from the first application this system was used to). Each one has a XC4085 XLA-08 XILINX FPGA and two 32 K  $\times$  16 bit SRAM. The PPK array communicates with the RVC via a 36-bit data bus and two more lines for handshake.

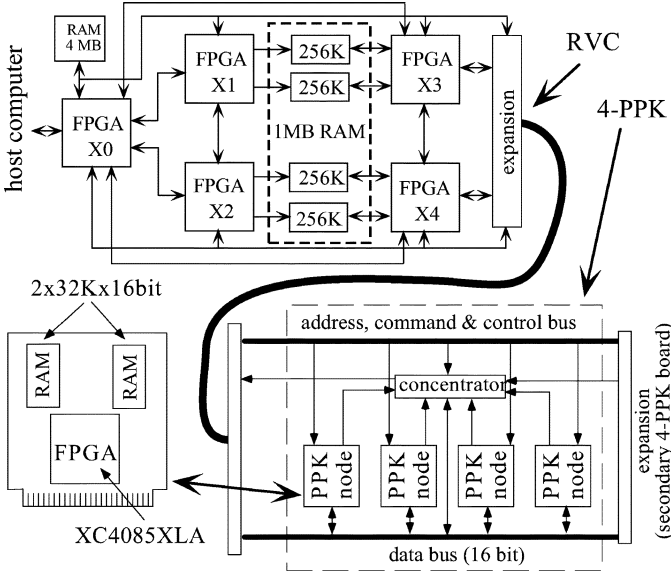


Fig. 1. Reconfigurable platform (from [4]).

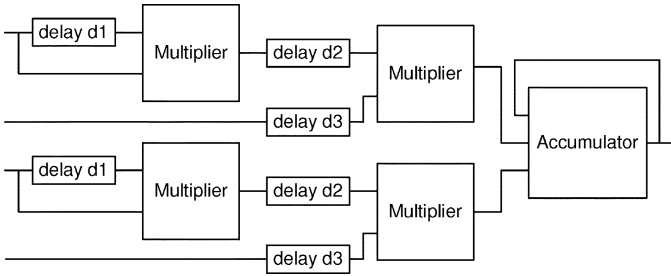


Fig. 2. Computation core architecture.

2) *Dedicated Processor Architecture:* Besides the interface with the host and with the main memory, mapped in X0, the custom processor is composed of four independently addressable RAMs, working as cache memories, that hold the current frame and the average of each segment of 16 consecutive pels, computed in X3 and X4 while transferring the frame from the main memory to the cache RAMs. The computation core is a multiply/accumulate unit of one pair of 2 cascaded multipliers and one accumulator, allowing the processing of two block rows simultaneously (Fig. 2) and is replicated in the four PPK nodes. The multiplier is a combinatorial integer one, generated by an application developed for the effect.

For a full description of the implemented architecture, see [1].

3) *Accumulator Specification:* The input to each multiplier is the difference between an element of a 16 element vector and the average of that vector. Each element of the original vector is an unsigned 8 bit integer.

A simplified view of the accumulator core is sketched in Fig. 3. The cascaded multipliers were collapsed into a three input multiplier, the second pair of multipliers was discarded because its only purpose is to duplicate the work done by clock cycle and the input to the multipliers was generalized to come from three different sources instead of two, with one of them fed delayed to one of the multipliers.

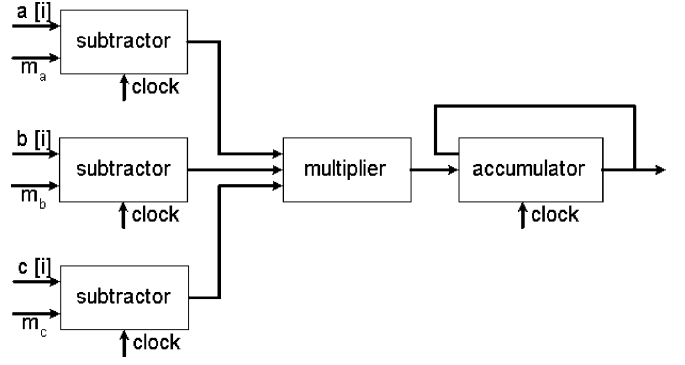


Fig. 3. Simplified MAC unit.

During sixteen clock cycles all the elements of the vectors are sequentially processed, each one being subtracted from the average, multiplied and accumulated.

In order to maximize the number of computing units per FPGA, there was the need to know the maximum value that could be accumulated in this MAC unit. Note that we want to know not only the maximum of the final accumulated value but also the maximum of the partial sums, because one of these may be bigger than the final accumulated sum.

A crude approach to the problem would lead to an accumulator size of 31 bits ( $9 + 9 + 9 + 4$ ).

4) *Paper Organization:* The remainder of this paper is organized as follows: Section II presents a definition of the problem to solve. In Section III the problem is addressed, with the steps of the mathematical demonstration clearly presented. Finally, the conclusions are drawn in Section IV.

## II. PROBLEM DEFINITION

We will focus on the maximization problem and later on achieve the minimum of the function under study.

Given 3 vectors  $a$ ,  $b$ , and  $c$  of 16 elements each, each element being an integer between 0 and 255, when is the function

$$f = \sum_{i=1}^x a_0[i]b_0[i]c_0[i], \quad 1 \leq x \leq 16$$

maximized, with  $a_0[i] = a[i] - m_a$ ,  $b_0[i] = b[i] - m_b$ , and  $c_0[i] = c[i] - m_c$ , where  $m_a$ ,  $m_b$  and  $m_c$  are the average of the 16 elements from  $a$ ,  $b$ , and  $c$ , respectively?

## III. PROBLEM RESOLUTION

We will walk toward the solution, presenting each intermediate result as a claim.

*Claim 3.1:* There is an absolute maximum of  $f$  with all  $a[i] = 255 \vee a[i] = 0$ ,  $1 \leq i \leq 16$  (the same applies to  $b[i]$  and  $c[i]$ ).

*Proof:* Let us suppose that we have found an absolute maximum, with some  $a[k] = p$ ,  $p \neq 0 \wedge p \neq 255$ ,  $0 \leq k \leq 16$ .

- $k > x$  ( $a[k]$  does not belong to the sum). If we consider another solution differing only in  $a[k]$ , with  $a[k] = 0$ , we have in the new solution  $m'_a = m_a - p/16$  and  $a'_0[i] = a_0[i] + p/16$ ,  $1 \leq i \leq 16$ ,  $i \neq k$ .

The sum takes the following value:

$$\begin{aligned} f &= \sum_{i=1}^x a'_0[i] b_0[i] c_0[i] \\ &= \sum_{i=1}^x a_0[i] b_0[i] c_0[i] + \frac{p}{16} \sum_{i=1}^x b_0[i] c_0[i]. \end{aligned}$$

On the other hand, if we consider the solution with  $a[k] = 255$ , it follows that  $m''_a = m_a + q/16$ , with  $q = 255 - p$  and  $a''_0 = a_0[i] - q/16$ ,  $1 \leq i \leq 16$ , and  $i \neq k$ . The new value taken by  $f$  is

$$\begin{aligned} f &= \sum_{i=1}^x a''_0[i] b_0[i] c_0[i] \\ &= \sum_{i=1}^x a_0[i] b_0[i] c_0[i] - \frac{q}{16} \sum_{i=1}^x b_0[i] c_0[i]. \end{aligned}$$

So we see that changing  $a[k] = p$  to  $a[k] = 0$  or to  $a[k] = 255$  will increase the value of the function, depending on the sign of  $\sum_{i=1}^x b_0[i] c_0[i]$ . Concluding, it is always possible to move to a solution with a greater value of  $f$  if we have  $0 < a[k] < 255$ .<sup>1</sup> The same applies to  $b[i]$  and  $c[i]$ .

- $k \leq x$  This time we have to consider also the change in the sum due to the change in the term  $a[k]$ . In the new solution with  $a[k] = 0$  the change due to that term will be

$$\begin{aligned} &(-m_a + p/16) b_0[k] c_0[k] - (p - m_a) b_0[k] c_0[k] \\ &= -\frac{15p}{16} b_0[k] c_0[k] \end{aligned}$$

and the total variation is given by

$$\begin{aligned} &\frac{p}{16} \sum_{i=1, i \neq k}^x b_0[i] c_0[i] - \frac{15p}{16} b_0[k] c_0[k] \\ &= p \left[ \frac{1}{16} \sum_{i=1, i \neq k}^x b_0[i] c_0[i] - \frac{15}{16} b_0[k] c_0[k] \right]. \end{aligned}$$

In the solution with  $a[k] = 255$  the same change takes the value

$$\begin{aligned} &(255 - m_a - q/16) b_0[k] c_0[k] - (p - m_a) b_0[k] c_0[k] \\ &= \frac{15q}{16} b_0[k] c_0[k] \end{aligned}$$

and the total variation is given by

$$\begin{aligned} &-\frac{q}{16} \sum_{i=1, i \neq k}^x b_0[i] c_0[i] + \frac{15q}{16} b_0[k] c_0[k] \\ &= q \left[ -\frac{1}{16} \sum_{i=1, i \neq k}^x b_0[i] c_0[i] + \frac{15}{16} b_0[k] c_0[k] \right]. \end{aligned}$$

Therefore, the changes are always of opposite signs, implying that is always possible to get a solution with a greater value of  $f$ , taking  $a[k] = 0$  or  $a[k] = 255$ .<sup>2</sup>

<sup>1</sup>if  $\sum_{i=1}^x b_0[i] c_0[i] = 0$  the new solution is not better but is also a maximum.

<sup>2</sup>if  $-(1/16) \sum_{i=1, i \neq k}^x b_0[i] c_0[i] + (15/16) b_0[k] c_0[k] = 0$  the new solution is not better but is also a maximum.

**Conclusion:** At least an optimal solution is among those with  $a[i] = 0 \vee a[i] = 255$ ,  $1 \leq i \leq 16$ . (The same applies to  $b[i]$  and  $c[i]$ .)

**Claim 3.2:** In an optimal solution the terms in the sum are all positive.

**Proof:** We are now only interested in solutions with  $a[i]$ ,  $b[i]$ , and  $c[i]$  equal to 0 or 255—binary variables. Hence, we shall consider that  $a[i] = 0 \vee a[i] = 1$ ,  $b[i] = 0 \vee b[i] = 1$ , and  $c[i] = 0 \vee c[i] = 1$ , and at the end we just multiply the maximum value attained with these new variables by  $255^3$  to get the maximum of the initial problem.

Notice that  $a[i] = 0$  implies  $a_0[i] < 0$  and  $a[i] = 1$  implies  $a_0[i] > 0$  (the same to  $b[i]$  and  $c[i]$ ). So of the eight possible distinct sequences to  $a[i]b[i]c[i]$ , namely 000, 001, 010, 011, 100, 101, 110, and 111, only half of them, 001, 010, 100, and 111 will give positive terms in the sum, while the others will contribute with negative terms.

Is it worth to look for solutions with negative terms in the sum? No, because a partial sum without that negative terms would have a greater value. Is it worth to look for solutions with positive terms in the  $16 - x$  terms out of the sum? No, because a partial sum with those positive terms included in the sum (increasing the value of  $x$ ) would have a greater value of  $f$ .

Summarizing, it is enough, for each partial sum, to look for an optimal solution among those with only positive terms in the sum, corresponding to terms  $a[i]b[i]c[i]$  equal to 001, 010, 100, or 111, and in the  $(16 - x)$  terms out of the sum with only negative terms, corresponding to 110, 101, 011, and 000.

It is important to stress that with these constraints, we do not guarantee to find the maximal value of each partial sum. However, we are sure that we will find the optimal solution, the maximum of the maximum of each partial sum.

**Claim 3.3:** Complementing one vector only modifies the sign of  $f$ . Complementing two vectors leaves the value attained by function unaltered.

**Proof:** If we complement the elements of a vector, e.g.,  $a, a' = 1 - a[i]$ ,  $1 \leq i \leq 16$  we will get a new solution with  $m'_a = 1 - m_a$  and  $a'_0[i] = a'[i] - m'_a = (1 - a[i]) - (1 - m_a) = -(a[i] - m_a) = -a_0[i]$ . Hence,  $a'_0[i]b'_0[i]c'_0[i] = a'_0[i]b_0[i]c_0[i] = -a_0[i]b_0[i]c_0[i]$ . We conclude that complementing a vector changes only the sign of the sum  $f$ ; if we complement 2 vectors the value of  $f$  does not change. Therefore, we can move from an optimal solution to another, complementing 2 vectors. This property will be used later.

**Claim 3.4:** In an optimal solution the terms out of the sum must all be equal.

**Proof:** It is enough to verify that if we have out of the sum some  $a[n] = 0$  and some  $a[m] = 1$ , we can improve the solution by changing  $a[n]$  to 1 or  $a[m]$  to 0

$$a[n] = 0 \rightarrow a[n] = 1, \quad n > x$$

$$m'_a = m_a + 1/16$$

$$a'_0[i] = a_0[i] - 1/16, \quad i \leq x$$

$$f = \sum_{i=1}^x a'_0[i] b_0[i] c_0[i]$$

$$= \sum_{i=1}^x a_0[i] b_0[i] c_0[i] - \frac{1}{16} \sum_{i=1}^x b_0[i] c_0[i]$$

and

$$\begin{aligned}
 a[m] &= 1 \rightarrow a[m] = 0, \quad m > x \\
 m'_a &= m_a - 1/16 \\
 a'_0[i] &= a_0[i] + 1/16, \quad i \leq x \\
 f &= \sum_{i=1}^x a'_0[i] b_0[i] c_0[i] \\
 &= \sum_{i=1}^x a_0[i] b_0[i] c_0[i] + \frac{1}{16} \sum_{i=1}^x b_0[i] c_0[i].
 \end{aligned}$$

We see that all terms  $a[i]$  out of the sum must be equal (the same applies to  $b[i]$  and  $c[i]$ ). So  $a[i]b[i]c[i]$  out of the sum must be one and only one of the terms 011, 101, 110, and 000. But we can go a little further. As complementing two vectors does not change the value of  $f$  we can make that all terms out of the sum be equal to 000.

Clarifying: we will look for an optimal solution among those with positive terms in the sum (001, 010, 100, and 111) and 000 out of it.

Let  $n_1, n_2, n_4 \in n_7$  the number of times that the terms  $a[i]b[i]c[i]$  corresponding to 001, 010, 100, and 111 appear in the sum, respectively.

*Claim 3.5:*  $\max(n_1, n_2, n_4) < \min(n_1, n_2, n_4) + 2$ .

*Proof:* With the constraints imposed to our search, in the solutions we are restricted to, we have  $m_a = (n_4 + n_7)/16$ ,  $m_b = (n_2 + n_7)/16$ , and  $m_c = (n_1 + n_7)/16$ .

The function  $f$  to maximize can be written as

$$\begin{aligned}
 f &= \sum_{i=1}^x a_0[i] b_0[i] c_0[i] \\
 &= \sum_{i=1}^x (a[i] - m_a)(b[i] - m_b)(c[i] - m_c)
 \end{aligned}$$

with

$$\begin{aligned}
 \sum_{i=1}^x a[i] b[i] c[i] &= n_7 \\
 \sum_{i=1}^x a[i] b[i] m_c &= n_7 m_c = n_7(n_7 + n_4)/16 \\
 \sum_{i=1}^x a[i] c[i] m_b &= n_7 m_b = n_7(n_7 + n_2)/16 \\
 \sum_{i=1}^x b[i] c[i] m_a &= n_7 m_a = n_7(n_7 + n_1)/16 \\
 \sum_{i=1}^x a[i] m_b m_c &= (n_4 + n_7) m_b m_c \\
 &= (n_4 + n_7)(n_2 + n_7)(n_1 + n_7)/256 \\
 \sum_{i=1}^x b[i] m_a m_c &= (n_2 + n_7) m_a m_c \\
 &= (n_4 + n_7)(n_2 + n_7)(n_1 + n_7)/256 \\
 \sum_{i=1}^x c[i] m_a m_b &= (n_1 + n_7) m_a m_b \\
 &= (n_4 + n_7)(n_2 + n_7)(n_1 + n_7)/256 \\
 \sum_{i=1}^x m_a m_b m_c &= x(n_4 + n_7)(n_2 + n_7)(n_1 + n_7)/16^3.
 \end{aligned}$$

the previous expression of  $f$  simplifies to

$$\begin{aligned}
 f &= n_7 - n_7 x + 2n_7/16 \\
 &\quad + (48 - x)/(16^3)(n_4 + n_7)(n_2 + n_7)(n_1 + n_7).
 \end{aligned}$$

Suppose, without loss of generality that  $n_1 = \max(n_1, n_2, n_4)$ , and  $n_4 = \min(n_1, n_2, n_4)$ . We intend to prove that we must have  $n_1 < n_4 + 2$  in an optimal solution. To do so, suppose that we have an optimal solution with  $n_1 \geq n_4 + 2$ . Lets make in a new solution  $n'_1 = n_1 - 1$ ,  $n'_2 = n_2$ ,  $n'_4 = n_4 + 1$ , and  $n'_7 = n_7$ . The change in  $f$  is of  $(4 - x)/(16^3)(n_2 + n_7)(n_1 - n_4 - 1)$ , which is positive because  $n_1 > n_4 + 1$ .

**Conclusion:** in an optimal solution the difference between  $n_1$  and  $n_4$  must be less than 2, implying that  $n_1 = n_4 + \delta_1$ ,  $\delta_1 = 0 \vee \delta_1 = 1$  and  $n_2 = n_4 + \delta_2$ ,  $\delta_2 = 0 \vee \delta_2 = 1$ . So  $n_1 + n_2 + n_4 = 3n_4 + j$ ,  $0 \leq j \leq 2$  and  $n_1 + n_2 + n_4 + n_7 = x$  is equivalent to  $n_7 + 3n_4 + j = x$ .

For a given  $x$  we just have to change  $n_7$  from 0 to  $x$  to find among those solutions the optimal one (setting  $n_7$  and  $x, n_4$  and  $j$  are univocally known and so  $n_2$  and  $n_1$ ). So we must calculate  $f$  2 + 3 + 4 + ... + 17 = 152 times—in the original problem we had  $256^{48}$  different solutions!

#### A. Results

Computing the function  $f$  in the identified 152 different solutions we conclude that the function attains its maximal value of 33 162 750 at, for instance

$$\begin{aligned}
 a[] &= \{0, 0, 0, 0, 0, 0, 0, 0, 255, 255, 255, \\
 &\quad 255, 255, 255, 255, 255\} \\
 b[] &= \{0, 0, 0, 0, 255, 255, 255, 255, 0, \\
 &\quad 0, 0, 0, 255, 255, 255, 255\} \\
 c[] &= \{255, 255, 255, 255, 0, 0, 0, 0, 0, \\
 &\quad 0, 0, 0, 255, 255, 255, 255\}.
 \end{aligned}$$

That value is representable with 26 bits in two's complement.

#### B. Minimum of the Problem

Of course we also have to address the minimization problem to know if the minimum is also representable with 26 bits in two's complement. We could follow a path similar to the one followed in the maximum calculation. However that is not necessary. Bearing in mind that complementing a vector changes only the sign of  $f$ , the minimum of  $f$  must be minus the maximum. So it is also representable with 26 bits.

#### IV. CONCLUSION

The performance of a system is often limited by one key task. That is especially true in video processing systems, where the motion estimation is usually the most time consuming task. This implies that for real time applications this critical operation should be implemented with a special care.

It has been presented how an effective optimization of a key component in a custom motion estimator processor can bring important gains in comparison with an equivalent, non optimized implementation. The optimal size for the accumulator in a multiply accumulative unit was mathematically reached for

this specific application. These results can be extended and/or adapted for similar applications.

#### ACKNOWLEDGMENT

The authors would like to thank Professor J. C. Alves of Faculdade de Engenharia da Universidade do Porto/INESC Porto, for his helpful contribution to the specification of the problem addressed in this paper.

#### REFERENCES

- [1] J. S. Cardoso, J. C. Alves, and L. Corte-Real, "FPGAs' based hardware applied to block-based motion estimation for real-time video processing," in *Proc. 11th Portuguese Conf. Pattern Recognition (RECPAD2000)*, Porto, Portugal, 2000, pp. 455–458.
- [2] C. L. Nikias and J. M. Mendel, "Signal processing with higher-order spectra," *IEEE Signal Process. Mag.*, vol. 10, no. 3, pp. 10–37, Jul. 1993.
- [3] J. M. M. Anderson and G. B. Giannakis, "Image motion estimation algorithm using cumulants," *IEEE Trans. Image Process.*, vol. 4, no. 3, pp. 346–357, Mar. 1995.
- [4] J. C. Alves and J. S. Matos, "RVC—A reconfigurable coprocessor for vector processing applications," in *Proc. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM'98)*, Napa Valley, CA, 1998, pp. 258–259.