A High-Performance Sum of Absolute Difference Implementation for Motion Estimation

Jarno Vanne, Eero Aho, Timo D. Hämäläinen, and Kimmo Kuusilinna

***Abstract*—This paper presents a high-performance sum of absolute difference (SAD) architecture for motion estimation, which is the most time-consuming and compute-intensive part of video coding. The proposed architecture contains novel and efficient optimizations to overcome bottlenecks discovered in existing approaches. In addition, designed sophisticated control logic with multiple early termination mechanisms further enhance execution speed and make the architecture suitable for general-purpose usage. Hence, the proposed architecture is not restricted to a single block-matching algorithm in motion estimation, but a wide range of algorithms is supported. The proposed SAD architecture outperforms contemporary architectures in terms of execution speed and area efficiency. The proposed architecture with three pipeline stages, synthesized to a 0.18-μm CMOS technology, can attain 770-MHz operating frequency at a cost of less than 5600 gates. Correspondingly, performance metrics for the proposed low-latency 2-stage architecture are 730 MHz and 7500 gates.**

***Index Terms*—Early termination mechanism, motion estimation, sum of absolute difference (SAD), SAD architecture.**

I. Introduction

BLOCK-BASED motion estimation searches for the best matching block between the current and reference macroblocks (MBs). For the operation, the sum of absolute differences (SAD) is one of the most frequently employed criteria [1], [2]. As a result, motion estimation produces a motion vector (MV), which represents the motion of the MB.

Well known full-search (FS) is the simplest, but the most computation-intensive block-matching algorithm (BMA). To decrease the computational complexity, numerous optimized search algorithms have been developed including three-step search (TSS) [3] and new diamond search (DS) [4]. However, motion estimation still clearly dominates the whole encoding process. With DS algorithm, it is shown to be near 50% of all the complexity in an MPEG-4 video encoder [5]. Since a major part of motion estimation is pure SAD computation, it is well motivated to focus on SAD architectures.

An optimal hardware realization for SAD computation varies for different BMAs. Systolic architectures are a good match for FS due to simplicity and regularity [2], [6], but not for irregular BMAs including DS. Architectures, which are better tailored for irregular BMAs, are considered in [6]–[11].

Manuscript received May 9, 2005; revised September 9, 2005 and December 22, 2005. This work was supported in part by the Academy of Finland under Grant 104487, Tampere Graduate School in Information Science and Engineering, Nokia Foundation, Emil Aaltonen Foundation, Heikki and Hilma Honkanen Foundation, and HPY Foundation. This paper was recommended by Associate Editor L.-G. Chen.

J. Vanne, E. Aho, and T. D. Hämäläinen are with the Institute of Digital and Computer Systems, Tampere University of Technology, FI-33101 Tampere, Finland (e-mail: jarno.vanne@tut.fi).

K. Kuusilinna is with Nokia Research Center, FI-33721 Tampere, Finland.

Digital Object Identifier 10.1109/TCSVT.2006.877150

Compared to referenced SAD architectures, our proposal includes modified elements, which process arithmetic operations in a novel and efficient way. In addition, the proposed architecture provides several early termination mechanisms and sophisticated SAD computation control.

The rest of the paper is organized as follows. Section II describes the related work with SAD architectures supporting irregular BMAs. Section III considers the theory related to the proposed SAD algorithm. Section IV proposes our novel SAD architecture unit by unit. Performance comparisons between contemporary SAD architectures and our architecture are shown in Section V. Section VI concludes the paper.

II. Previous Work

This paper merely concentrates on SAD implementations. At the current block location (x, y), the SAD criterion is defined as

$$\mathrm{SAD}(x,y,i,j) = \sum_{l=0}^{M-1}\sum_{k=0}^{N-1} |A_{(x+l,y+k)} - B_{(x+i+l,y+j+k)}| \tag{1}$$

where $A_{(x+l,y+k)}$ and $B_{(x+i+l,y+j+k)}$ indicate pixels of the current block and the reference frame, respectively. The size of the block is $M \times N$ and SAD computation is performed in the search area location (i, j) which is the displacement of the candidate block compared to the current block. The candidate block yielding the minimum SAD value determines $\mathrm{MV} = (i, j)$ for the location (x, y).

SAD architecture can functionally be divided into three stages: absolute difference calculation, accumulation of absolute differences, and minimum SAD determination.

A. Absolute Difference Calculation

The purpose of the first stage is to calculate absolute differences between the candidate and current MB pixels.

Vassiliadis *et al.* in [7] introduce a unit customized for detecting and inverting the smaller one of the pixel values. This paper refers to the unit as *Vassiliadis' smaller operand inverter*. The unit calculates inversion with one's complement arithmetic. Hence, to obtain finally a proper two's complement SAD value, the introduced inversion error has to be compensated afterwards by an additional *correction term*.

Another absolute difference unit is presented by Jehng *et al.* in [6]. The unit produces an absolute difference with a one's complement adder surrounded by a few logic gates. The implementation is called *Jehng's absolute difference unit*.

The third conventional absolute difference unit is presented by Chen *et al.* in [10]. It is an improved embodiment of Jehng's unit and it is referred to as *Chen's absolute difference unit*. In the unit, the low performance end-around carry chain in one's

complement adder is removed and the possible one-bit error is compensated later by a *correction bit*.

B. Accumulation of Absolute Differences

In the second stage, the produced absolute differences are accumulated. Parallelization that is exploited in the pixel accumulation can vary from purely serial to fully parallel.

Jehng *et al.* in [6] present an accumulation unit to be used with Jehng's absolute difference units. In this paper, the adder tree implementation is called *Jehng's adder tree unit*.

Despite the additional correction bits, Chen's absolute difference units are also well suited for the adder tree. The correction bit accumulation is implemented by substituting the correction bits to the carry-in inputs of the adders in the tree. The modified tree is referred to as *Chen's adder tree unit* [8].

The accumulation of absolute differences can also be implemented with *CSA tree unit* [12], [13]. The carry–save adder (CSA) tree compresses the incoming absolute differences to carry and sum vectors. The carry propagation is performed during the last stage with a fast adder. Designing the CSA tree structure with more complex calculation elements [14] than full- and half adders is out of the scope of this paper.

Chen *et al.* in [9] present a recursive CSA tree for SAD computation. It is hereafter called *Chen's compression array unit*. Besides absolute differences and correction bits, Chen's compression array is capable of compressing previously compressed carry and sum vectors which are fed back to the array. Contrary to the accumulation units presented above, Chen's compression array produces two output vectors which are added together in the minimum SAD determination stage.

Vassiliadis' smaller operand inverters could also be coupled to the presented four accumulation units. However, each Vassiliadis' unit produces two output values and a common correction term. Therefore, the used accumulation unit has to be extended for $2u+1$ input values in order to complete two's complement partial SAD value for u pixels in one stage.

C. Minimum SAD Determination

Typically, conventional architectures execute the minimum SAD determination stage in two successive phases. In the first phase, two partial SAD values are added together to compute the current SAD value, whereas the resulting SAD value is compared to the minimum SAD value in the second phase. To speed up the operation, Chen *et al.* [10] present an enhanced minimum SAD determination unit here called *Chen's MV determination unit*. It performs SAD value comparison without completing actual SAD values at all. SAD values are utilized for comparison purposes only and the MV is the output from the unit.

III. Proposed SAD Algorithm

Let us consider a single SAD operation performed between two $M \times N$ blocks. In this special case, locations (x, y) and (i, j) are fixed in (1). Therefore, (1) can be rewritten to

$$\mathrm{SAD} = \sum_{l=0}^{M-1}\sum_{k=0}^{N-1} |A_{l,k} - B_{l,k}| = \sum_{l=0}^{M-1}\sum_{k=0}^{N-1} \mathrm{ABS}_{l,k}. \quad (2)$$

Since all $A_{l,k}$ and $B_{l,k}$ are nonnegative n-bit numbers, all $\mathrm{ABS}_{l,k}$ can also be presented with n bits.

The proposed absolute difference calculation stage computes absolute differences according to Theorem 1.

Theorem 1: For a single operand pair $A, B \in [0, 2^n - 1]$, absolute difference $\mathrm{ABS} \in [0, 2^n - 1]$ can be expressed as

$$\mathrm{ABS} = |A - B| = \begin{cases} (A + B') + 1, & A > B \\ (A + B')' + 0, & A \le B \end{cases} \quad (3)$$

where $B' = 2^n - 1 - B$ and $(A + B')' = 2^n - 1 - (A + B')$ are one's complement representations of B and $(A + B')$, respectively.

Proof: For $A > B$,

$$A + B' = A + 2^n - 1 - B = |A - B| - 1 + 2^n. \quad (4)$$

Since $|A - B| \in [1, 2^n - 1]$, $(A + B') \in [2^n, 2^{(n+1)} - 2]$, i.e., $A + B'$ generates always a carry (2^n). Hence, operand 2^n can be ignored in (4) and 1 has to be added to $A + B'$ in order to yield the correct result $|A - B|$.

For $A \le B$, $(A + B') \in [0, 2^n - 1]$ and

$$A + B' = A + 2^n - 1 - B = 2^n - 1 - |A - B| \quad (5)$$

$$\Leftrightarrow (A + B')' = |A - B|. \quad (6)$$

As shown in (5) and (6), the correct result in this case is obtained by taking one's complement of $A + B'$. □

Let us assume that the proposed absolute difference calculation stage completes simultaneously M absolute differences, i.e., $\mathrm{ABS}_l = |A_l - B_l|$ for $l = 0, 1, \ldots, M-1$ are computed in parallel. Hence, M absolute differences can also be accumulated in parallel. Without loss of generality, let P out of M accumulated absolute differences be $(A > B)$ cases and $(M-P)$ absolute differences be $(A \le B)$ cases. The accumulation yields a SAD value $(\mathrm{SAD}M)$ for M absolute differences as

$$\begin{aligned} \mathrm{SAD}M &= \sum_{l=0}^{M-1} \mathrm{ABS}_l \\ &= \sum_{l=0}^{P-1} ((A_l + B_l') + 1) + \sum_{l=P}^{M-1} ((A_l + B_l')' + 0) \\ &= \sum_{l=0}^{P-1} 1 + \sum_{l=P}^{M-1} 0 + \sum_{l=0}^{P-1} (A_l + B_l') + \sum_{l=P}^{M-1} (A_l + B_l')'. \end{aligned} \quad (7)$$

Since addition of real numbers is an associative operation, the operands $((A_l + B_l')', (A_l + B_l'), 1, 0)$ in (7) can be in arbitrary order.

To complete a SAD value (2) for $M \times N$ blocks, N $\mathrm{SAD}M_k$ values (for $k = 0, \ldots, N-1$) have to be computed and added together. As in [9], parallelism and time-multiplexed hardware reuse of the proposed accumulation stage are increased by utilizing a recursive implementation. Via recursion, a single $\mathrm{SAD}M_k$ value computation and the addition

of it to the sum of the already-computed $\mathrm{SAD}M$ values $(=\mathrm{SAD}M_{k-1}+\cdots+\mathrm{SAD}M_1+\mathrm{SAD}M_0)$ can be performed in parallel.

In practice, the proposed accumulation stage produces two addition terms called sum $(\mathrm{SAD\_S}_k)$ and carry $(\mathrm{SAD\_C}_k)$ vectors, where $\mathrm{SAD\_S}_k+\mathrm{SAD\_C}_k=\mathrm{SAD}_k=\mathrm{SAD}M_k+\cdots+\mathrm{SAD}M_1+\mathrm{SAD}M_0$. The $\mathrm{SAD\_S}_k$ and $\mathrm{SAD\_C}_k$ values are recursively obtained as

$$\begin{aligned}&\mathrm{SAD\_S}_k+\mathrm{SAD\_C}_k\\&\quad=\mathrm{SAD\_S}_{k-1}+\mathrm{SAD\_C}_{k-1}+\sum_{l=0}^{P-1}1+\sum_{l=P}^{M-1}0\\&\qquad+\sum_{l=0}^{P-1}(A_l+B'_l)+\sum_{l=P}^{M-1}(A_l+B'_l)'\end{aligned}\tag{8}$$

for $k=0,1,\ldots,N-1$ with $\mathrm{SAD\_S}_{-1}=\mathrm{SAD\_C}_{-1}=0$. The complete SAD value equals $\mathrm{SAD}_{N-1}=\mathrm{SAD\_S}_{N-1}+\mathrm{SAD\_C}_{N-1}$.

In the proposed minimum SAD determination stage, $\mathrm{SAD\_S}_k$ and $\mathrm{SAD\_C}_k$ values are added together to complete SAD_k value. In parallel with the SAD_k computation, $\mathrm{SAD\_S}_k+\mathrm{SAD\_C}_k$ value is also compared against the current minimum SAD value (MIN_SAD). Theorem 2 determines a new minimum SAD value.

Theorem 2: A new minimum SAD value has been found, if

$$\mathrm{SAD\_S}_{N-1}+\mathrm{SAD\_C}_{N-1}+\mathrm{MIN\_SAD}''<2^{(n+q)}\tag{9}$$

i.e if the addition in (9) generates no carry. In (9), $\mathrm{MIN\_SAD}''=2^{(n+q)}-\mathrm{MIN\_SAD}$ is two's complement representation of MIN_SAD, $\mathrm{SAD\_S}_{N-1}+\mathrm{SAD\_C}_{N-1}\le 2^{(n+q)}$, $\mathrm{MIN\_SAD}\le 2^{(n+q)}$, and $q=\lceil\log_2(M\times N\times(2^n-1))\rceil-n$.

Proof: The smaller one of the operands $\mathrm{SAD}_{N-1}(=\mathrm{SAD\_S}_{N-1}+\mathrm{SAD\_C}_{N-1})$ and MIN_SAD is detected with the following inequality checking:

$$\begin{aligned}&\mathrm{SAD\_S}_{N-1}+\mathrm{SAD\_C}_{N-1}<\mathrm{MIN\_SAD}&(10)\\&\Leftrightarrow\mathrm{SAD\_S}_{N-1}+\mathrm{SAD\_C}_{N-1}<2^{(n+q)}-\mathrm{MIN\_SAD}''&(11)\\&\Leftrightarrow\mathrm{SAD\_S}_{N-1}+\mathrm{SAD\_C}_{N-1}+\mathrm{MIN\_SAD}''<2^{(n+q)}.&(12)\end{aligned}$$

□

IV. Proposed Hardware Implementation

The proposed units are designed for the system in which $N=M=16$ and $n=q=8$. However, the units can be modified to support other bit widths and levels of parallelism as well.

A. Proposed Absolute Difference Unit

The *proposed absolute difference unit* is depicted in Fig. 1. The unit implements the functionality presented in (3) for the pixels $A(a_{7..0})$ and $B(b_{7..0})$, where bit widths are marked with MSB..LSB notation. If $A\le B$, the adder in Fig. 1. yields a result $S=A+B'=s_{8..0}<2^n$(5). Hence, no carry bit

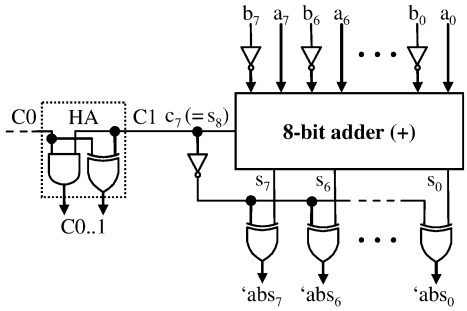

Fig. 1. Proposed absolute difference unit.

is generated $(c_7=\text{'0'})$ and the inverted c_7 bit is used to bit-invert (one's complement) the result $s_{7..0}$ to a correct absolute difference. Otherwise, $A>B$ and the result $S=A+B'=s_{8..0}\ge 2^n$ (4) implies that $c_7=\text{'1'}$. In that case, the result $s_{7..0}=A-B-1$ and c_7 bit can be forwarded as a correction bit. Notation $\text{'ABS}l=(\text{'abs}_{7..0})l$ $(l=0,1,\ldots,15)$ signifies that the correction bit (= '0'/'1') is missing from the result.

As in Chen's implementation, a low-performance end-around carry chain used in Jehng's absolute difference unit is eliminated in the proposed unit. However, contrary to Chen's unit, the carry-in of the adder can also be removed in the proposed implementation. In addition, the structure of the proposed unit enables immediate correction bit addition without increased delay. As depicted in Fig. 1, the correction bits (C0 and C1) of the two adjacent absolute difference units are added together with a half adder (HA) in parallel with output XORs of the units. The generated 2-bit correction bit vectors $(\mathrm{C0..1},\ldots,\mathrm{C14..15})$ are well suited to the proposed compression array in the subsequent stage.

B. Proposed Compression Array Unit

The *proposed compression array unit* presented in Fig. 2 implements the functionality of (8). It executes absolute difference and correction bit vector accumulations using separate CSA trees. A 6-stage CSA tree is designed to receive and compress all the 'ABSl values, as well as C0..1, $\mathrm{SAD\_S}_{k-1}$, and $\mathrm{SAD\_C}_{k-1}$ vectors. A 3-stage CSA tree is targeted for partial accumulation of the other correction bit vectors. In all the CSAs, the outputs on the left produce partial carries, whereas the right-hand outputs are for partial sums.

Before a new SAD value accumulation can begin, $\mathrm{SAD\_S}_{k-1}$ and $\mathrm{SAD\_C}_{k-1}$ vectors are initialized to zero. In Fig. 2, an initialization structure controlled by a single signal (INIT) is depicted inside the square.

During operation, the sum and carry bits produced by the 3-stage CSA tree are inserted one by one into proper CSA inputs available in the 6-stage CSA tree. In Fig. 2, each letter (A to F) indicates a connection required between the two CSA trees. The bit stages are indicated with a bold font. In turn, LSBs of $\mathrm{SAD\_S}_{k-1}$ $(\mathrm{ss}_{7..0})$ and $\mathrm{SAD\_C}_{k-1}$ vectors $(\mathrm{sc}_{7..3})$ are immediately inserted to the first stage of the 6-stage CSA tree. The intermediate bits of these vectors $(\mathrm{ss}_{11..8}$ and $\mathrm{sc}_{11..8})$ are individually inserted into the tree, whereas the MSBs of these vectors $(\mathrm{ss}_{15..12}$ and $\mathrm{sc}_{15..12})$ are added in the final stage.

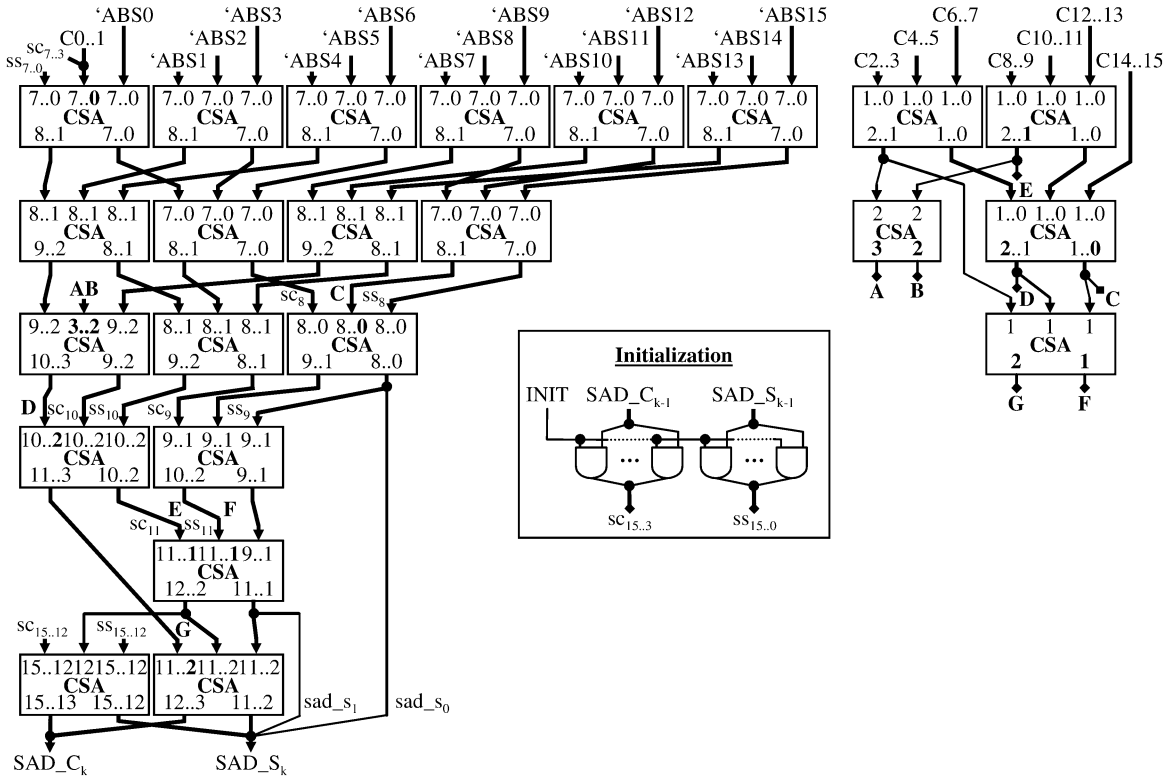

Fig. 2. Proposed compression array unit.

The proposed compression array reduces the number of operands at the earliest opportunity. Although this causes some area overhead, it also diminishes *common bit positions* involved in $\mathrm{SAD\_S}_k$ and $\mathrm{SAD\_C}_k$ vectors. In this context, a bit position is common for two vectors, if the bit position belongs to the index ranges of both vectors. Due to the enhanced correction bit accumulation, the proposed compression array is capable of compressing the number of common bit positions to 13 (i.e., $\mathrm{sad\_s}_{15..3}$ and $\mathrm{sad\_c}_{15..3}$). Without the correction bit accumulation, the proposed compression array could reduce the number of common bit positions to 12. Hence, the inclusion of the correction bits involves only one additional common bit position.

C. Proposed Minimum SAD Determination Unit

The *proposed minimum SAD determination unit* in Fig. 3 includes a novel structure for a parallel SAD value comparison and SAD value calculation. Contrary to Chen's approach, the proposed unit produces a complete minimum SAD value which can be exploited by other parts of the video encoder including INTRA/INTER mode decision [1]. The motion vector determination is excluded from this paper.

The unit performs the comparison presented in (9) for the operands $\mathrm{SAD\_S}_k$, $\mathrm{SAD\_C}_k$ and MIN_SAD. A 2-input OR gate manages the constant one bit addition which is required to convert the bit-inverted MIN_SAD value to two's complement representation. The CSA compression yields two difference vectors ($\mathrm{dif\_s}_{15..1}$ and $\mathrm{dif\_c}_{16..1}$) which are added together. The adder outputs only the most significant sum bit (s_{16}) which signifies the result of the inequality checking.

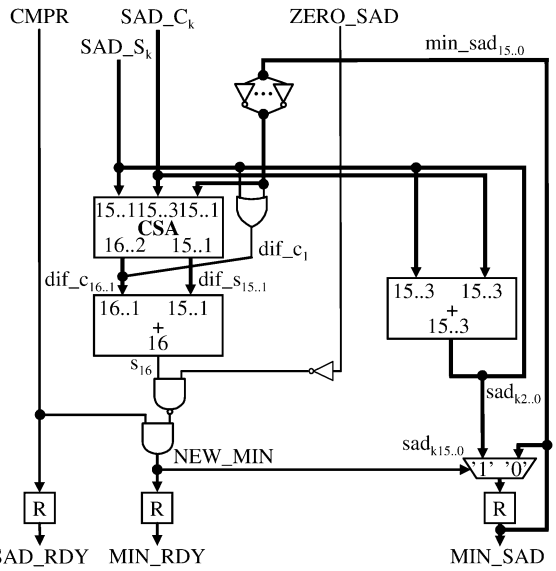

Fig. 3. Proposed minimum SAD determination unit without TM0-TM3.

In parallel with SAD value comparison, SAD_k value is calculated by adding $\mathrm{SAD\_S}_k$ and $\mathrm{SAD\_C}_k$ values together. A complete SAD value (SAD_{15}) computation requires 16 passes through the unit. Since an operationally decisive SAD value comparison is performed between SAD_{15} and MIN_SAD values, a specific control input (CMPR) indicating the proper time for comparison is only asserted during the final pass. If $s_{16} =$ '0' after the final pass comparison,

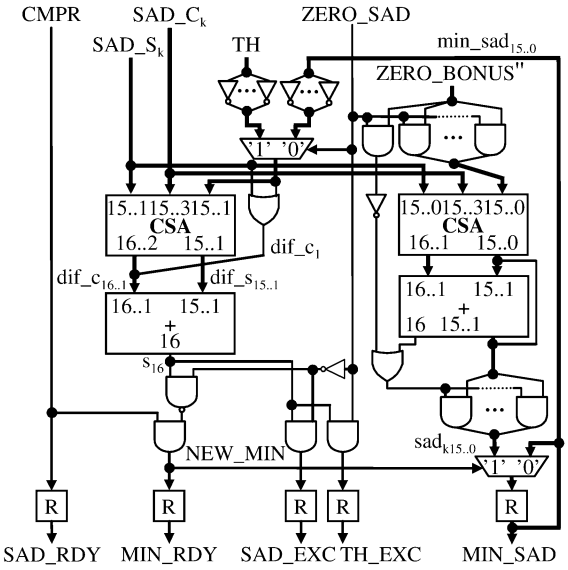

Fig. 4. Proposed minimum SAD determination unit with TM0-TM3.

$\mathrm{SAD}_{15} < \mathrm{MIN\_SAD}$ and SAD_{15} value is selected by a multiplexer ($\mathrm{NEW_MIN} = $ '1') and stored in the registers as the new minimum SAD value.

The first SAD value computed for a zero motion vector (zero MV) is processed differently due to the absence of the minimum SAD value. This special case is indicated by one of the input control signals (ZERO_SAD). Regardless of the comparison result, the calculated SAD value for the zero MV is selected as the new minimum SAD value. In the considered cases, a new minimum SAD value activates two output control signals (SAD_RDY and MIN_RDY). Otherwise, the minimum SAD value is maintained in registers and only one control signal (SAD_RDY) is active after the comparison.

D. Implemented Early Termination Mechanisms

Early termination mechanisms eliminate unnecessary calculations in SAD computation. The proposed architecture implements four different early termination mechanisms (TM0-TM3). TM0-TM2 are also considered in [11], but the implementation techniques differ from the proposed one.

TM0 monitors the temporary SAD value accumulated by the compression array. The mechanism interrupts the ongoing SAD computation if the accumulated SAD value exceeds a predetermined threshold value. In the proposed compression array, a desired output bit of $\mathrm{SAD\_C}_k$ vector can be selected as an interrupt signal. As discussed in [11], the utilization of the threshold value can enable a narrower compression array implementation. However, the bit width reduction without over restricting SAD values causes diminutive area savings in the proposed compression array. In addition, the bit width reduction maintains the array height, so the delay of the array remains the same. Thus, a full width compression array is a reasonable solution. The other mechanisms are included in the proposed minimum SAD determination unit shown in Fig. 4.

TM1 monitors the most significant sum bit (s_{16}) of the adder. $s_{16} =$ '1' before or during comparison denotes that $\mathrm{SAD}_k > \mathrm{MIN\_SAD}$. The violation is indicated by a single control signal

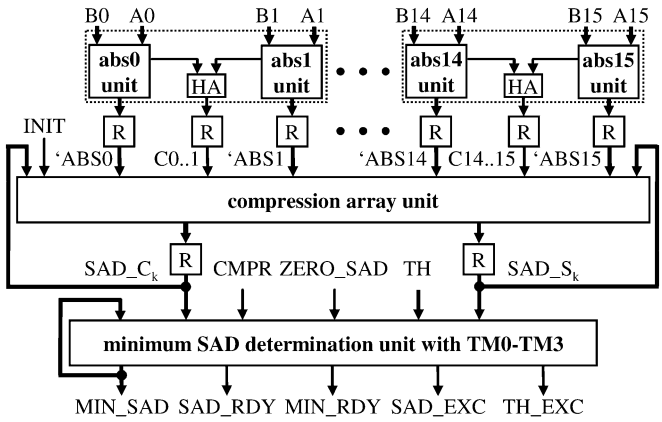

Fig. 5. Proposed 3-stage SAD architecture.

($\mathrm{SAD\_EXC}$). Due to the absence of the valid minimum SAD value, the violation is ignored during SAD computation of the zero MV.

TM2 examines whether the current SAD value is under the predetermined threshold value. The mechanism is only supported within SAD computation for the zero MV. Threshold exceeding detection is accomplished by the logic used normally for the comparison of SAD_k and $\mathrm{MIN\_SAD}$ values (9). Hence, an additional multiplexer is required to select either $\mathrm{MIN\_SAD}$ value or threshold (TH) value for the comparison logic. In turn, an activated control signal ($\mathrm{TH\_EXC}$) indicates an exceeded TH value.

The proposed unit also supports the use of multiple threshold values. After the current TH value is exceeded, the subsequent, larger, TH value can be fed in during the next cycle. The one cycle delay between successive TH values prevents the examinations of the remaining larger TH values if the TH value is exceeded during the final cycle before the SAD value completion. Unexamined, larger, TH values may, in the worst case, induce unnecessary calculations, but a proper result is still returned. On the other hand, the detection of the exceeded threshold value could also be performed completely in parallel by duplicating the comparison part of the unit. Parallel threshold value detection would reduce the unit delay due to the avoided multiplexer and would remove the delay between successive threshold value examinations. However, the area increase would be relatively high due to the duplicated comparison logic.

TM3 favors SAD value for the zero MV. In practice, a predefined constant referred to as *zero bonus* is subtracted from SAD value for the zero MV. Here, the two's complemented zero bonus ($\mathrm{ZERO\_BONUS''}$) is assumed to be determined at design time, although it could also be one of the unit inputs being adjustable at run time. Before calculating SAD_k value for the zero MV, an additional CSA performs an addition between $\mathrm{SAD\_S}_k$, $\mathrm{SAD\_C}_k$, and $\mathrm{ZERO\_BONUS''}$. The SAD_k value is clipped to 0 if $\mathrm{SAD\_S}_k + \mathrm{SAD\_C}_k + (\mathrm{ZERO\_BONUS''}) < 0$.

E. Proposed Overall SAD Architecture

Fig. 5 depicts the high-level structure containing all the proposed units as well as all the discussed early termination mechanisms. The architecture is divided into three pipeline stages. The first pipeline stage is composed of 16 proposed absolute

TABLE I
AREA AND DELAY ESTIMATES FOR THE REFERENCED AND PROPOSED UNITS OF THE SAD ARCHITECTURE

Unit Label	Unit Description	Compatible Units	Delay(τ) RCA	Delay(τ) CLA	Area(CU) RCA	Area(CU) CLA
ABS_0	Vassiliadis' smaller operand inverter	-	18	8	2320	2776
ABS_1	Jehng's absolute difference unit	-	35	14	2832	6416
ABS_2	Chen's absolute difference unit	-	19	9	2353	5769
ABS_3	Proposed absolute difference unit	-	18	8	2305	4761
ACC_0	Tailored adder tree unit	ABS_0	31	37	3625	10273
ACC_1	CSA tree unit	ABS_0	39	30	3506	3660
ACC_2	Tailored compression array unit	ABS_0	25		3853	
ACC_3	Jehng's adder tree unit	ABS_1	27	27	1714	4827
ACC_4	Tailored CSA tree unit	ABS_1	32	24	1736	1890
ACC_5	Tailored compression array unit	ABS_1	19		1944	
ACC_6	Chen's adder tree unit	ABS_2	29	28	1834	6183
ACC_7	Tailored CSA tree unit	ABS_2	38	25	1960	2319
ACC_8	Chen's compression array unit	ABS_2	19		2148	
ACC_9	Tailored adder tree unit	ABS_3	29	28	1834	6183
ACC_10	Tailored CSA tree unit	ABS_3	36	25	2018	2325
ACC_11	Proposed compression array unit	ABS_3	19		2182	
MIN_0	Chen's MV determination unit	ACC_8	48	20	953	1147
MIN_1	Proposed MIN SAD unit	ACC_11	39	16	643	1292
MIN_2	Proposed MIN SAD unit (TM0-TM3)	ACC_11	42	19	956	1786

difference units. In turn, the second and third stages include the proposed compression array and the minimum SAD determination unit, respectively. The pipeline registers are depicted only after the first two stages, since the registers of the last stage are included in the minimum SAD determination unit.

V. PERFORMANCE ANALYSIS

This analysis is restricted to SAD implementations which operate 16 pixels in parallel.

A. Theoretical Analysis

As in [15], the area cost for s-input basic gates is assumed to be s cost-units (CUs), expect for XOR and XNOR gates having $2s$ CUs. In turn, the delay for all the gates is supposed to be 1 τ. Registers, interconnections, and fan-out/fan-in related issues are excluded from the calculations.

Table I tabulates the theoretical delay and area estimates for the evaluated units. The units are divided into three groups: absolute difference units ($\mathrm{ABS\_0} - \mathrm{ABS\_3}$), accumulation units ($\mathrm{ACC\_0} - \mathrm{ACC\_11}$), and minimum SAD determination units ($\mathrm{MIN\_0} - \mathrm{MIN\_2}$). Compatibility between successive units is tabulated in the third column, e.g., the $\mathrm{ACC\_11}$ unit is compatible with the $\mathrm{ABS\_3}$ unit.

Available 2-operand adders in the units are analyzed as being implemented with ripple-carry adders (RCAs) and carry-lookahead adders (CLAs). Theoretical performance estimates for CLAs are averages of 2-input and s-input gate versions of CLA. For the compression array units ($\mathrm{ACC\_2}, \mathrm{ACC\_5}, \mathrm{ACC\_8},$ and $\mathrm{ACC\_11}$), only single performance metrics are reported, since these units are analyzed as being implemented completely with CSAs.

Let us first consider the reported results of the absolute difference units. In all the cases, the area cost is calculated for 16 units. The proposed $\mathrm{ABS\_3}$ unit attains higher performance and evidently better area efficiency over Jehng's ($\mathrm{ABS\_1}$) and Chen's ($\mathrm{ABS\_2}$) implementations. Performance of the proposed ABS_3 unit is even equal to Vassiliadis' $\mathrm{ABS\_0}$ unit, which only detects the larger one of the input operands. Hence, the CLA-based $\mathrm{ABS\_3}$ unit is a desirable solution for high-performance applications, whereas the RCA-based $\mathrm{ABS\_3}$ unit is targeted for area efficient systems.

Three types of compatible accumulation units are evaluated per each absolute difference unit: adder tree, CSA tree, and compression array. For example, the $\mathrm{ACC\_0}$–$\mathrm{ACC\_2}$ units are particularly designed for the $\mathrm{ABS\_0}$ unit. Word "tailored" attached to the accumulation unit description denotes that the unit is modified from the original implementation in order to be compatible with the targeted absolute difference unit. In addition, the functionally essential initialization logic for the sum and carry vector is attached to Chen's $\mathrm{ACC\_8}$ unit. Although some of the adder tree and CSA tree configurations have higher area efficiency than the compression arrays, the $\mathrm{ACC\_2}$, $\mathrm{ACC\_5}$, $\mathrm{ACC\_8}$, and $\mathrm{ACC\_11}$ units evidently outperform respective approaches in terms of execution speed. Compared to the other compression arrays, the $\mathrm{ACC\_2}$ unit has significantly lower performance. In turn, although the $\mathrm{ACC\_5}$ unit is as fast as the $\mathrm{ACC\_8}$ and $\mathrm{ACC\_11}$ units, the performance of the compatible $\mathrm{ABS\_1}$ unit is not sufficient. Hence, the $\mathrm{ACC\_8}$ and $\mathrm{ACC\_11}$ units with the compatible absolute difference units are the most competitive approaches. Compared to the $\mathrm{ACC\_8}$ unit, the number of common bit positions involved in the output vectors is reduced in the proposed $\mathrm{ACC\_11}$ unit. The common bit position reduction enables the usage of narrower adders and registers in the following minimum SAD determination unit. Hence, the $\mathrm{ACC\_11}$ unit is the best solution since a diminutive area overhead of it is more than compensated by area savings in the minimum SAD determination unit.

The analyzed minimum SAD determination units are particularly tailored for the preferred $\mathrm{ACC\_8}$ and $\mathrm{ACC\_11}$ units. In addition, the logic required for the motion vector determination

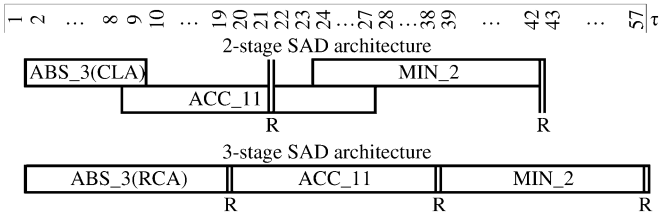

Fig. 6. Pipeline schemes for the proposed SAD architecture.

TABLE II
AREA AND DELAY ESTIMATES FOR THE MOST COMPETITIVE SAD ARCHITECTURES

SAD Architecture	Stages	Delay(τ)	Area(CU)
Chen's	2	24	9956
Proposed	2	21	9051
Proposed (TM0-TM3)	2	21	9569
Chen's	3	20	8112
Proposed	3	19	8071
Proposed (TM0-TM3)	3	19	8589

is excluded from Chen's $\mathrm{MIN\_0}$ unit. According to reported results, the proposed $\mathrm{MIN\_1}$ unit provides the fastest implementation. As shown by the $\mathrm{MIN\_2}$ unit, the presented early termination mechanisms can be included in the proposed unit very efficiently. In addition, delay increase in RCA-based implementations tends to be far more significant than area savings. Hence, the CLA-based $\mathrm{MIN\_2}$ unit is considered the best.

B. Pipelining

Fig. 6 depicts 2- and 3-stage pipelining schemes for the best overall SAD architecture $(\mathrm{ABS\_3} + \mathrm{ACC\_11} + \mathrm{MIN\_2})$. For illustration purposes, "as late as possible" scheduling is used in the schemes. The pipeline stages (R) are theoretically balanced in adder level granularity. In the proposed 2-stage scheme, the optimal location for the intermediate pipeline stage is after the fourth CSA stage of the compression array. In turn, the 3-stage pipelining follows the implementation in Fig. 5. The RCA-based ABS_3 units are adequately fast to be used in 3-stage schemes, but faster CLA-based units are required for 2-stage schemes.

The theoretical area and delay metrics for the proposed SAD architecture are tabulated in Table II, which also summarizes performance metrics for the most competitive SAD architectures: Chen's architecture $(\mathrm{ABS\_2} + \mathrm{ACC\_8} + \mathrm{MIN\_0})$ and the proposed one without early termination mechanisms $(\mathrm{ABS\_3} + \mathrm{ACC\_11} + \mathrm{MIN\_1})$. Both 2- and 3-stage pipelining schemes are analyzed. Registers are also included in the calculations in order to take pipelining overheads into account.

Compared to Chen's 2- and 3-stage architectures, the performance improvement of the proposed architectures is over 10% and 5%, respectively. The performance gap is wider with 2-stage architectures, since the proposed architectures can efficiently overlap the execution of successive units (Fig. 6). The inclusion of early termination mechanisms causes a slight area overhead in 3-stage architecture. In other cases, the proposed architecture is also more area efficient than Chen's approach.

The importance of the proposed optimizations is emphasized by the major role of SAD computation in video encoding. E.g., if DS algorithm is applied for full-pixel block-matching, encoding 16CIF (1408×1152) format at 30 frames per second requires approximately 3 million SAD operations per second. The proposed 3-stage architecture completes the 3 million SAD values $50\,\mathrm{M}\tau$ (5%) faster than corresponding Chen's architecture. The delay gap is widened to $150\,\mathrm{M}\tau$ (10%) between 2-stage architectures. Inclusion of fractional pixel estimation would increase the delay gap further.

TABLE III
SYNTHESIS RESULTS FOR THE PROPOSED SAD ARCHITECTURE

SAD Architecture	Stages	Freq(MHz)	Area(Cells)
Proposed (TM0-TM3)	2	730	7509
		575	5462
		420	4085
Proposed (TM0-TM3)	3	774	5624
		581	4377
		420	4078

C. Synthesis Results

The area and timing results based on logic synthesis are provided in Table III for the proposed 2- and 3-stage architectures $(\mathrm{ABS\_3} + \mathrm{ACC\_11} + \mathrm{MIN\_2})$. The applied technology is $0.18\,\mu\mathrm{m}$ CMOS process. The area values (gate count) are based on equivalent 2-input NAND gates, whereas the delay values represent the critical path in the pipelined architectures. Registers are included in the results.

With the selected technology, the proposed 3-stage SAD architecture is capable of operating at a frequency of 770 MHz, and costs the equivalent of 5600 NAND gates. Correspondingly, the 2-stage architecture can be clocked at 730 MHz at a cost of 7500 NAND gates. Lowering the operating frequency requirements could be used to decrease the silicon area.

To conclude the analysis, the proposed 3-stage architecture is a very good implementation for high throughput and area efficient systems in which three cycle latency is acceptable. In turn, the proposed 2-stage architecture is a feasible solution for low-latency systems as well as for 400-MHz systems and below. Very irregular block-matching algorithms and efficient use of early termination mechanisms favors the selection of 2-stage architecture.

VI. CONCLUSION

In a video encoder, motion estimation can consume up to half of the execution time. To reduce resources and computation power required for motion estimation, an optimized and efficient SAD architecture was presented in this paper. In addition, sophisticated control and several early termination mechanisms were presented for the architecture. The theoretical analysis of the paper illustrate that the proposed SAD architecture outperforms contemporary approaches. Furthermore, the synthesis results verify that the proposed architecture achieves very high execution speed at a cost of manageable silicon area.

REFERENCES

[1] V. Bhaskaran and K. Konstantinides, *Image and Video Compression Standards: Algorithms and Architectures*, 2nd ed. Amsterdam, The Netherlands: Kluwer Academic, 1997.

[2] P. Kuhn, *Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation*. Amsterdam, The Netherlands: Kluwer Academic, 1999.

[3] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion compensated interframe coding for video conferencing," in *Proc. Nat. Telecommun. Conf.*, New Orleans, LA, USA, 1981, pp. G5.3.1–5.3.5.

[4] S. Zhu and K. K. Ma, "A new diamond search algorithm for fast block-matching motion estimation," *IEEE Trans. Image Process.*, vol. 9, no. 2, pp. 287–290, Feb. 2000.

[5] O. Lehtoranta and T. D. Hämäläinen, "Complexity analysis of spatially scalable MPEG-4 encoder," in *Proc. Int. Symp. System-on-Chip*, Tampere, Finland, Nov. 2003, pp. 57–60.

[6] Y. S. Jehng, L. G. Chen, and T. D. Chiueh, "An efficient and simple VLSI tree architecture for motion estimation algorithms," *IEEE Trans. Signal Process.*, vol. 41, no. 2, pp. 889–900, Feb. 1993.

[7] S. Vassiliadis, E. A. Hakkennes, S. Wong, and G. G. Pechanek, "The sum-absolute-difference motion estimation accelerator," in *Proc. 24th Euromicro Conf.*, Västerås, Sweden, Aug. 1998, pp. 559–566.

[8] Q. Shu and H. Chen, "An efficient implementation of motion estimation algorithms," in *Proc. 4th Int. Conf. Solid-State and Integr. Circuit Technol.*, Oct. 1995, pp. 697–699.

[9] H. Chen and Q. Shu, "Apparatus for implementing a block matching algorithm for motion estimation in video image processing," U.S. Patent 5 864 372, Jan. 26, 1999.

[10] H. Chen and Q. Shu, "Adaptive block-matching motion estimator with a compression array for use in a video coding system," U.S. Patent 5 838 392, Nov. 17, 1998.

[11] D. Guevorkian, A. Launiainen, and P. Liuha, "Method for performing motion estimation in video encoding, a video encoding system and a video encoding device," U.S. Patent Appl. Publication 2003/0043911 A1, Mar. 6, 2003.

[12] C. Wallace, "A suggestion for parallel multipliers," *IEEE Trans. Electron. Comput.*, vol. EC-13, pp. 14–17, 1964.

[13] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford, U.K.: Oxford Univ. Press, 1999.

[14] L. Dadda, "Some schemes for parallel multipliers," *Alta Freq.*, vol. 34, pp. 349–356, May 1965.

[15] A. R. Omondi, *Computer Arithmetic Systems: Algorithms, Architecture and Implementations*. Hertfordshire, U.K.: Prentice Hall Int., 1994.