Journal of Signal Processing Systems 51, 77–98, 2008
© 2007 Springer Science + Business Media, LLC. Manufactured in The United States.
DOI: 10.1007/s11265-007-0143-9

# Serial and Parallel FPGA-based Variable Block Size Motion Estimation Processors

BRIAN M. H. LI AND PHILIP H. W. LEONG
*Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, NT, Hong Kong*

*Received: 16 April 2007; Revised: 28 July 2007; Accepted: 24 August 2007*

**Abstract.** H.264/AVC is the latest video coding standard adopting variable block size motion estimation (VBS-ME), quarter-pixel accuracy, motion vector prediction and multi-reference frames for motion estimation. These new features result in much higher computation requirements than previous coding standards. In this paper we propose a novel most significant bit (MSB) first bit-serial architecture for full-search block matching VBS-ME, and compare it with systolic implementations. Since the nature of MSB-first processing enables early termination of the sum of absolute difference (SAD) calculation, the average hardware performance can be enhanced. Five different designs, one and two dimensional systolic and tree implementations along with bit-serial, are compared in terms of performance, pixel memory bandwidth, occupied area and power consumption.

**Keywords:** motion estimation, FPGA, systolic, bit serial, video compression

## 1. Introduction

H.264/AVC [1] is a recent video coding standard developed by the Joint Video Team (JVC) of ITU-T VCEG and ISO/IEC MPEG. It is suggested that it can provide two times better performance than the previous coding standard MPEG-2, in terms of compression efficiency and picture quality [2]. Like previous coding such as H.263 and MPEG-4, it employs block-based motion estimation to reduce temporal redundancy between frames. In H.264, block-matching efficiency is further enhanced by advanced features such as variable block size motion estimation (VBS-ME), multi-reference frames and motion vector prediction. Due to these features, the computational complexity of H.264/AVC is increased by a factor of four, creating challenges to achieve real time performance.

Many hardware motion estimation architectures [2–6] have been proposed in the literature and most can achieve real time encoding. In many cases a full search strategy is chosen for reasons of regularity, lack of data dependencies and optimality. Fast algorithms are suboptimal but significantly reduce the search space. These are frequently adopted in software implementations but data dependencies hinder pipelined hardware designs. We focus on full search (FS) hardware designs although we also consider the performance of the three-step search (TSS) [7, 8] with our architectures.

Many previously reported FS architectures were implemented using bit-parallel operations since they have the advantages of better performance, easier control and simpler design than a bit-serial approach and most of these were for ASIC rather than FPGA technology. Of previously reported FPGA implementations [9–16], only two [15, 16] support VBS-ME, and both are bit-parallel. A most significant bit (MSB)-first bit-serial design with early termination was proposed for QCIF resolution video [13] which employed a FS within the range −15 to +16. Their experiments showed that on average, 50% of the computation, can be saved when an early termination scheme is employed, the savings depending on the video scene. A sum of absolute difference (SAD) engine employing on-line arithmetic was also reported [14]. This design has improved area-time product over previous bit-serial architectures, but only supports one block size and cannot be used for H.264/AVC or later standards.

In this work we propose a novel MSB-first bit-serial architecture for VBS-ME that efficiently utilizes the high ratio of registers to logic present in FPGA devices which can be employed in H.264/AVC. The architecture makes it possible to eliminate certain unnecessary computations which are unavoidable in a bit-parallel implementation. The total execution cycle savings over a scheme that does not use the early termination scheme is dependent on the nature of the video but is on average 36.5%. The introduction of H.264 motion vector prediction mode further improves the saving by 3–5%. The proposed architecture not only reduces the computation time via optimizations in the arithmetic and early termination schemes, but also reduces resource utilization through a bit-serial architecture and power by eliminating unnecessary calculations. The new architecture results in performance comparable to bit-parallel implementations but with greatly reduced area.

We further present bit-parallel one and two dimensional systolic implementations. These achieve high performance and require low memory bandwidth because of efficient pipelining and local interconnection. Local accumulation and global accumulation strategies to sum the absolute difference results are considered. We compare the bit-serial design with these bit-parallel ones.

The rest of this paper is organized as follows. Section 2 gives an overview of motion estimation and the new features present in H.264/AVC. Section 3 presents four parallel systolic implementations which are used for comparison with our work. In Section 4 the architecture of our serial implementation with early termination technique is described. Results and a comparison with other reported work will be given in Section 5 and conclusions are drawn in Section 6.

## 2. Motion Estimation Algorithm

In digital video, consecutive picture frames are combined to form a scene. The redundancy between frames is usually large due to a relative high frame rate to scene motion relationship in normal videos. Motion estimation (ME) techniques have been adopted since the first generation of digital video coding standards to reduce temporal redundancy between frames, hence improving compression rates. Block based matching techniques have been used because of their simplicity and high efficiency. Although, due to its limited search range, an optimal solution is not guaranteed, its hardware-friendly nature makes it the most common scheme for video coding standards.

### *2.1. Block-based Motion Estimation*

A description of block-based matching techniques follows. Each picture is divided into a fixed number of square non-overlapping blocks, called macroblocks (Mblock). Typically, the block size is 16-pixels by 16-pixels. Each block in the current frame is compared to blocks in the reference frame within a predefined search window and the best match is found (Fig. 1). A SAD between the current block and reference block, as described in Fig. 1, is commonly used as the distance metric since it can be implemented efficiently.

In Fig. 1, $c(i,j)$ and $r(i,j)$ represent pixels in the current and reference block respectively. $m, n$ are the horizontal and vertical displacement of the current block within search window. $P, Q$ is the index indicating which subblock/macroblock SAD is calculated. In total there are seven types of subblocks/macroblocks as shown in Fig. 2. The search window is confined to $-16$ to $+15$ in this example and the rest of this paper. Pixels are positive 8-bit integers, so the absolute difference is also 8-bit. Summing $16 \times 16$ of them without any rounding leads to a 16-bit sum and summing only $4 \times 4$ of them leads to a 12-bit sum. Lastly, $MV_{min(P,Q)}$ is the motion vector with the minimum SAD value.

### *2.2. Variable Block Size Motion Estimation*

In traditional motion estimation such as used in MPEG-1, only one motion vector is generated for a macroblock and the computational complexity is relatively low. In recent advanced coding standards such as H.264/AVC, the motion estimation process has been improved to exploit temporal redundancy as much as possible. Additional features add significant demand to hardware requirements, e.g. in H.264/AVC, VBS-ME increases the number of motion vectors produced per macroblock, and the quarter-pixel and multi-reference frame features add addi-

$$SAD_{P,Q}(m,n) = \sum_{i=0}^{P-1} \sum_{j=0}^{Q-1} |c(i,j) - r(i+m, j+n)|$$

$$-range \leq m,n < +range,$$

$$P,Q \in \{4,8,16\},$$

$$u = min_{m,n}\{SAD_{P,Q}(m,n)\}$$

$$MV_{min(P,Q)} = (m,n)$$

where *range* is the search range having values ±16 or ±32.

*Figure 1.* Variable block motion estimation.

tional search points to the original searching algorithm. In total, the overall complexity of H.264/AVC is raised by a factor of 4 compared to the MPEG-2 standard and most of this is due to increased complexity in the motion estimation process.

In H.264/AVC, each picture (frame) is segmented into macroblocks. Each macroblock is further divided into sub-blocks with seven different types of block sizes (4×4, 4×8, 8×4, 8×8, 8×16, 16×8 and 16×16) as shown in Fig. 2. Each macroblock has in total 41 types of sub-blocks to cover the whole macroblock. In VBS-ME, for each type of subblock a motion vector (MV) is produced so in total 41 MVs are calculated per macroblock.

The VBS-ME feature is the most challenging hardware implementation issue added in H.264/AVC. The multi-reference feature can be implemented at the algorithm level and by data scheduling techniques. Quarter-pixel accuracy can be performed after the integer motion estimation process in a post-processing unit. Thus in this paper we restrict ourselves to the problem of how to generate 41 MVs within a given search window and current block.

## 2.3. Search Algorithms

In the FS algorithm, optimality can be guaranteed by exhaustively finding the absolute minimum SAD within a search area, typically ranging from −16 to +15. Thus, in our case, FS involves a total of (15−(−16)+1)× (15−(−16)+1)=1,024 search positions.

The three-step search (TSS) algorithm, proposed by Kola [7] and implemented by Lee [8] makes an assumption that the residue values increase radially from the absolute minimum point within the search area. In the first step, TSS compares the nine search points surrounding the center point with step size *p*

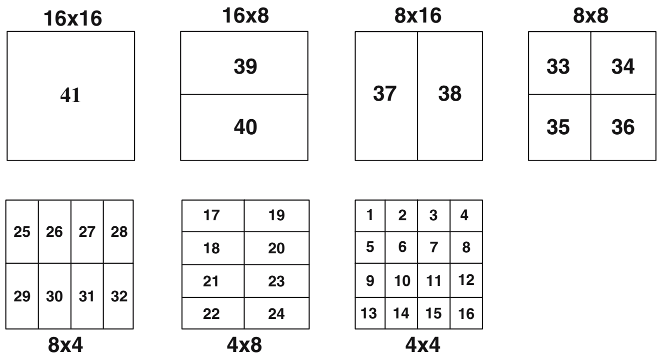

*Figure 2.* Sub-macroblock partitions in H.264/AVC.

equal to or larger than half of the maximum search range *r*. Among the nine search points, a minimum is selected and becomes the center of the next step. Next, the step size is halved and eight new search points (excluding the center) are searched and again a minimum is selected. The step size is halved again and search continues until the step size is equal to one. The minimum search point is found at this stage.

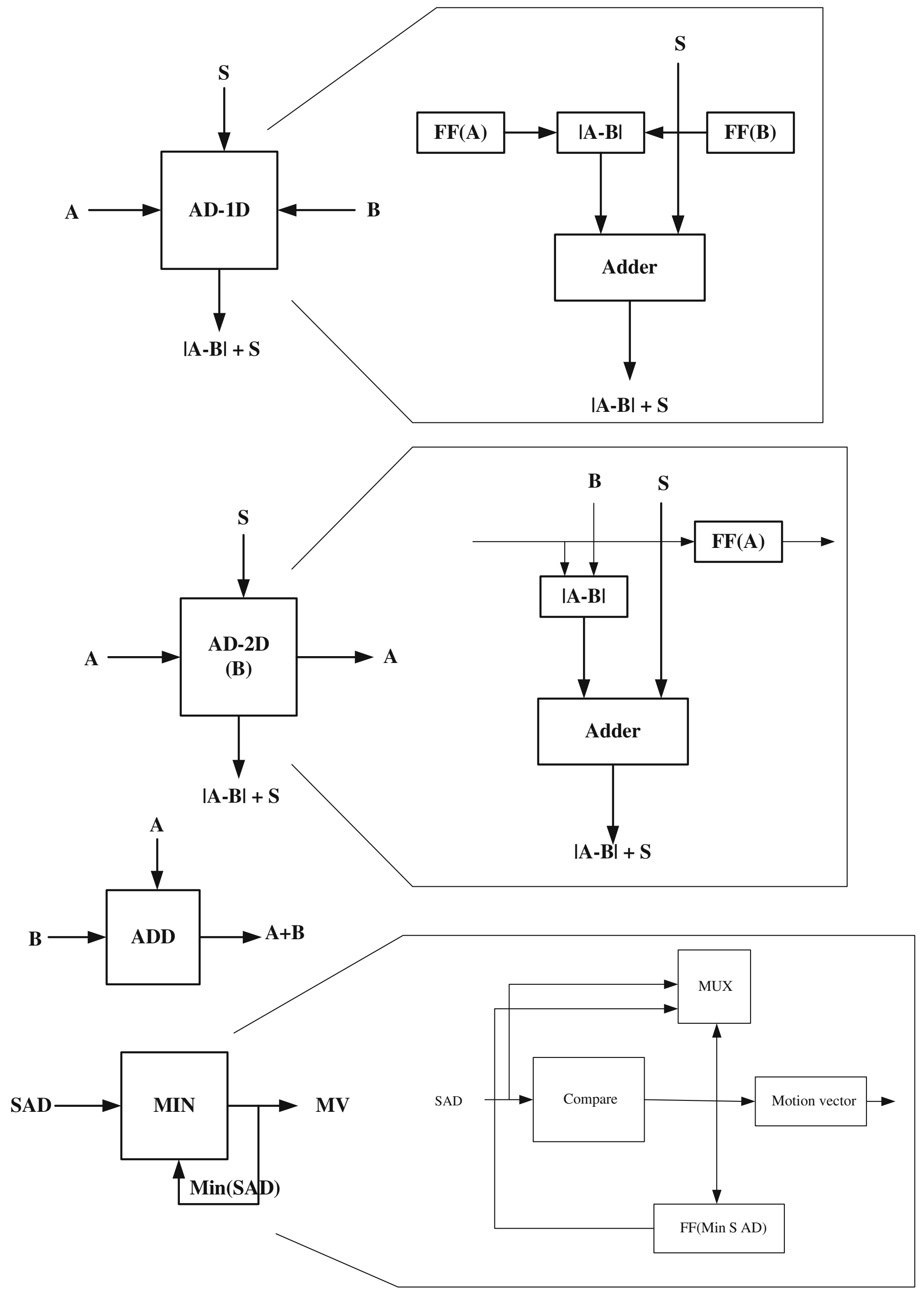

*Figure 3.* Primitive elements for the systolic and tree architectures.

The TSS constantly divides the search step size by two and is therefore a logarithmic search. The total number of search points is $[1 + 8log(p)]$. Except for the first step, the eight search points are calculated in each iteration and the algorithmic complexity for TSS is $O(8log(p)) = O(log(r/2))$ where $p$ is the initial step size and typically equals to $r/2$. This algorithm has advantages of much lower complexity than FS in terms of number of candidates to evaluate and efficient implementations in both software and hardware. Even for software implementations, this algorithm [17] can offer real time encoding. Compared with FS, techniques such as the three-step search, log-search [18] and diamond search [19] have reduced computational complexity but their hardware implementations involve data dependencies which makes it difficult to parallelize, and furthermore, they can be trapped in a local minimum and lead to poor results. The same datapath can be used to implement both FS and TSS and the performance of both will be reported in Section 5.2

## 3. Bit-parallel Systolic Implementations

In motion estimation, SAD is computed over a four-dimensional index space $i, j, m, n$ for each macroblock in a frame, as shown in Fig. 1 . This can be considered as two 2-D index spaces, the first being generated by the indexes $i,j$ for calculating $SAD_{P,Q}(m,n)$ and the second by $m, n$. The minimum SAD is found and a motion vector deduced after exploring all $m, n$ pairs in the full search. The indexes $i,j$ can be projected onto a 1D or 2D systolic array and the number of computation nodes depends on the block size. Another way is to project $i, m$ onto the systolic plane and results in an array dependent on block size and search range. Parallelism can be higher if the search range is larger than the macroblock size. A number of ways to map motion estimation algorithms to systolic arrays have been suggested [3]. Each computation node or processing element (PE) is built from subtraction, absolute value, accumulation and flip flop (FF) primitives as illustrated in Fig. 3. Additional ADD and MIN blocks, also shown in Fig. 3 are used to perform the sum and find the minimum SAD respectively.

In this section we present four systolic architectures for implementing SAD. We also estimate the number of cycles required to process a 16×16 macroblock, this figure not including overheads introduced by pipeline flushing, ADD delay and MIN delay. The actual number of clock cycles for each design is given in Section 5.

Figure 4 depicts the local accumulation 1D systolic array (SA-1D). $r(x, y)$ and $c(x, y)$ are the reference and current block respectively. It is termed local as it involves summation within the processing node. This design requires external memories to store the search area and current block, resulting in a high memory bandwidth requirement. Each absolute difference (AD) PE calculates the absolute difference of two pixels from the reference block and current block, adds the result to the already calculated partial sum for the same search position given by the neighbor PE, and passes the result to the next PE. At the end of the chain of PEs, the SAD calculation is finished and the result is compared with the previous minimum SAD result within the MIN element. The SA-1D design consists of 16 PEs, 1 ADD and 1 MIN component. Including the pipeline speedups, it takes 16 cycles on average to calculate 1 SAD search position. Since a 16×16 macroblock has 1,024 search positions, the number of cycles per macroblock is $1024 \times 16 = 16384$. The ADD and MIN processing is done in parallel with pixel shifting in all our systolic designs. This architecture is scalable for search range and block size. It has small area but the performance is low compared to 2D-implementations.

The 2D systolic array (SA-2D) is a two dimensional version of the SA-1D architecture. Figure 5 shows its datapath and timing of the data flow. The reference data is passed horizontally from one AD-2D element to the next. Data reuse is possible by making use of delay lines and by moving data from one PE to the next. This architecture offers the advantage of reducing memory bandwidth compared to the SA-1D architecture as the current data is initially shifted to each PE and will be stored and reused until the current block motion vector is found. The SA-2D design consists of $16 \times 16 = 256$ PEs, 16 ADD and 1 MIN component. The pipelining introduces 16 cycles of latency to the SAD calculation. Full search can be pipelined and after the pipeline is filled, only 1 cycle is required per SAD. There are 48 columns of reference pixels for a (−16,15) vertical search range. As a result, the approximate number of cycles is $48 \times 32 = 1536$ per 16×16 macroblock. This architecture requires large area but the performance is high because of the number of parallel computations.

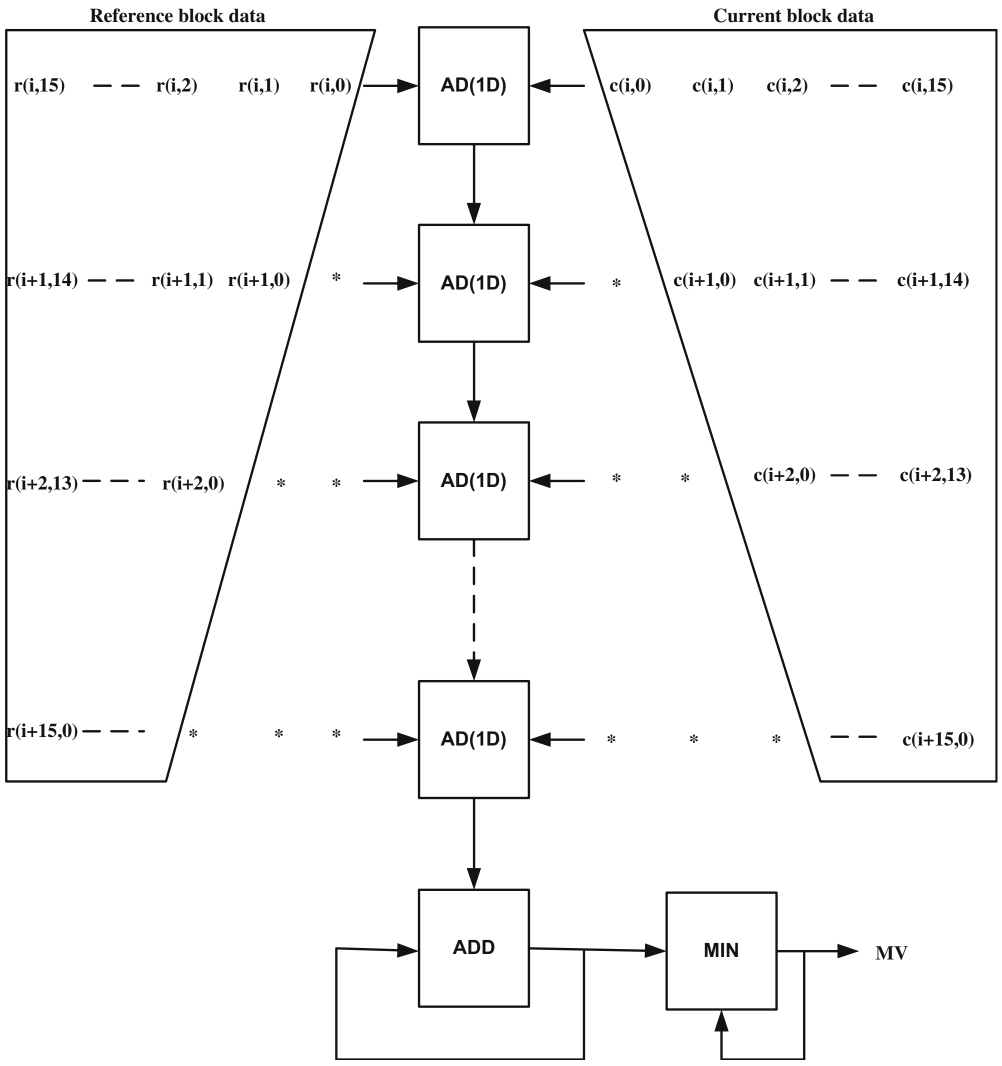
Reference block data
Current block data
r(i,15) — — r(i,2) r(i,1) r(i,0)
c(i,0) c(i,1) c(i,2) — — c(i,15)
r(i+1,14) — — r(i+1,1) r(i+1,0) *
* c(i+1,0) c(i+1,1) — — c(i+1,14)
r(i+2,13) — — - r(i+2,0) * *
* * c(i+2,0) — — c(i+2,13)
r(i+15,0) — — - * * *
* * * — — c(i+15,0)
AD(1D)
AD(1D)
AD(1D)
AD(1D)
ADD
MIN
MV

*Figure 4.* 1-D systolic architecture (SA-1D). The *asterisk* indicates that the data is delayed for these rows.

The global accumulation based architecture, also referred to as a "tree architecture," TA-1D, is shown in Fig. 6 for a 4×4 macroblock. The absolute difference of a reference and current pixel is calculated in the PE and the result is accumulated in an adder-tree external to the PE array. The adder tree is usually implemented as a non-redundant or carry-save adder tree with pipeline registers inserted between the stages. The reference data are fed every cycle to the PEs whereas the current block data is loaded when the current block is changed and are kept in registers if local caches are added. The TA-1D for a 16×16 block size consists of 16 PEs, each handling an absolute difference operation, 16 ADDs and 1 MIN. TA-1D takes approximately $1024 \times 16 = 16384$ cycles.

The two dimensional tree architecture, TA-2D, depicted in Fig. 7 is the two dimensional extension of the TA-1D architecture, in which $N \times N$ PEs are used, as illustrated for the 4×4 block size case in Fig. 7. Search area pixels are fed every cycle to the PEs whereas the current block data is loaded only once when the current block is changed. The local communication between PEs reduces memory bandwidth on reference data compared with TA-1D. TA-2D for a 16×16 block size consists of 256 PEs, each handling an absolute difference operation, 255 ADDs and 1 MIN. It processes a search point per cycle (1,024 search points in total), with additional overhead of $32 \times 16$ cycles to change rows (e.g. moving from SAD(0,15) to (1,0)), so the approximate

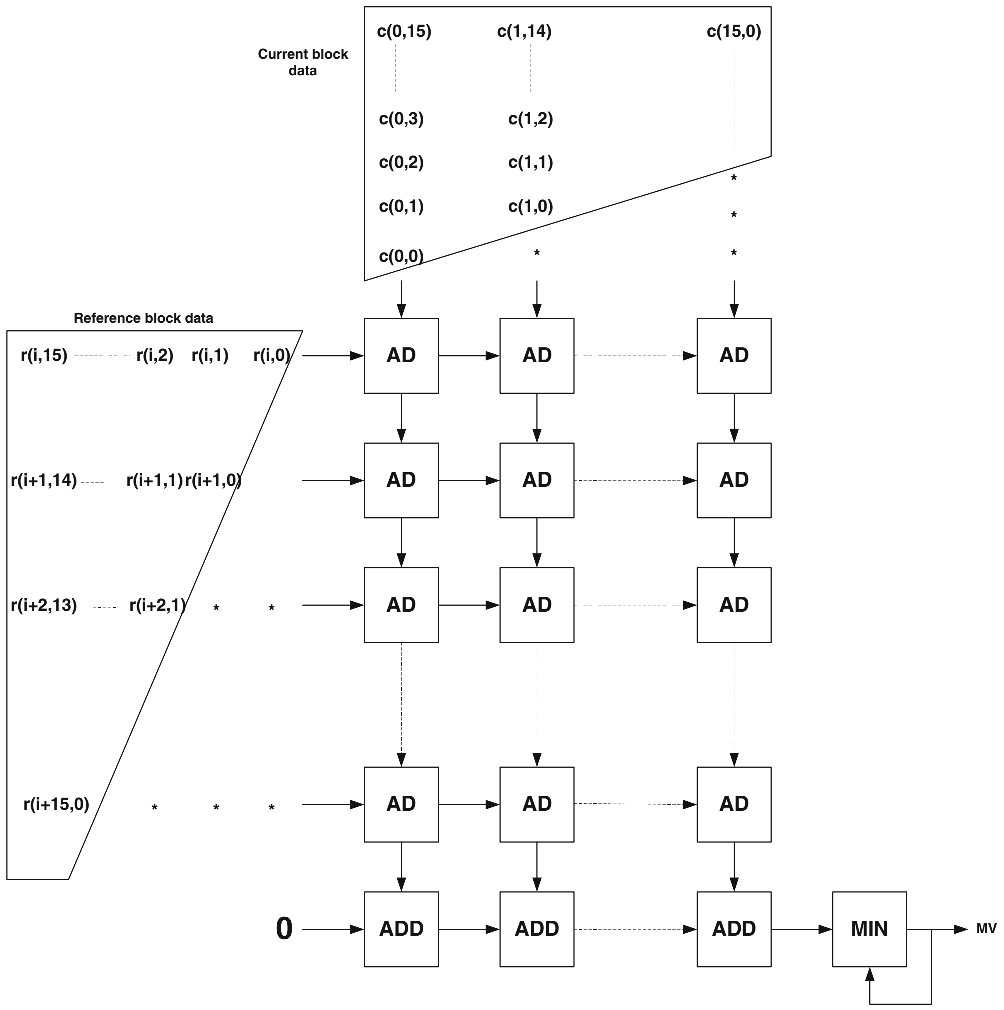

*Figure 5.* 2-D systolic architecture (SA-2D).

number of cycles is *1024* + *512* = *1536* to process a 16×16 macroblock. Since it avoids local accumulation and can be fully pipelined, the performance of this architecture is the best among the four implementation alternatives.

## *3.1. VBS-ME Support*

There are two methods to support VBS-ME in bit-parallel architectures. For local accumulation architectures like SA-1D and SA-2D, one can include 16 partial SAD registers in each node for 4×4 subblocks within a 16×16 macroblock. Each register stores its corresponding subblock SAD [20]. The second method is to divide the systolic array into its smallest possible block size architecture (sub-systolic array). For example, a 16×16 systolic array is divided into sixteen 4×4 systolic arrays, each handling a 4×4 SAD. The sixteen 4×4 SADs are then combined to form a large SAD via a SAD-reuse technique using an adder tree [21]. The first method imposes a large register overhead on each processing node and can have a large impact on silicon area. The second may increase the required bandwidth as the local communications between sub-systolic arrays are broken.

In tree architectures, the implementation is easier since there is no partial SAD stored in the systolic elements. The support of VBS-ME can be done by adding intermediate registers connected to comparison units in different stages of the adder tree. Similarly, using the SAD-reuse technique, VBS-ME

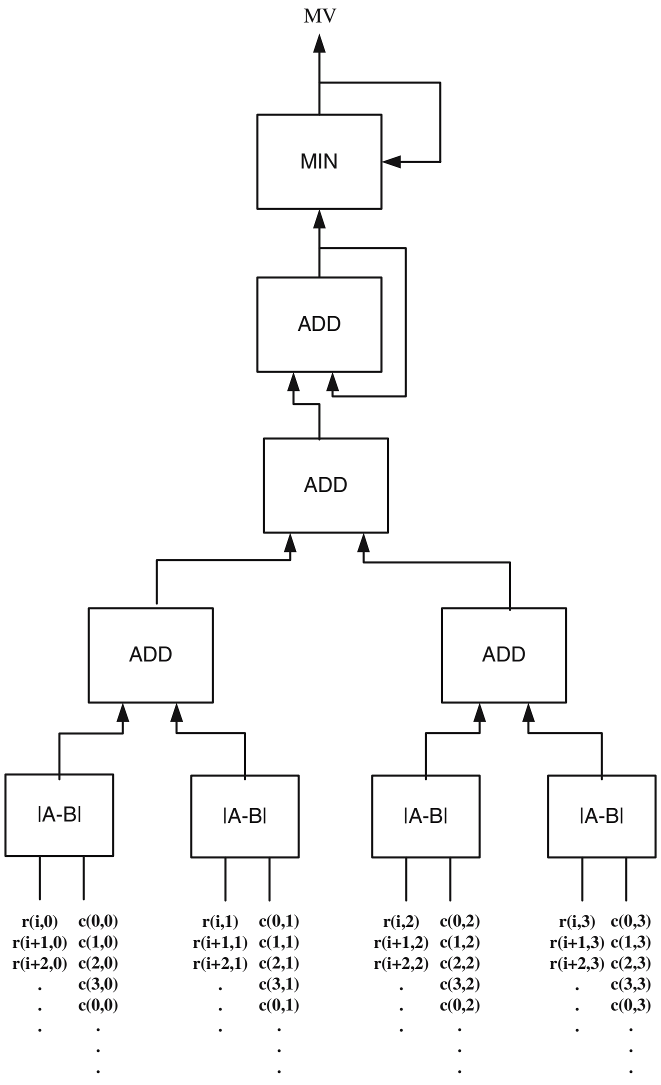

*Figure 6.* 1-D tree architecture (TA-1D).

is supported without significantly sacrificing memory bandwidth and silicon area.

For the SA-1D architecture, VBS-ME is handled by the conversion of one 16-PE array into four 4-PE arrays. An adder tree is connected to output of these four arrays to produce SADs for larger block sizes. In this design, we have to add 3 ADD and 3 MIN components, together with registers for intermediate SAD storage and control logic for 4×4 SAD comparisons. It takes approximately $1024 \times 16 = 16384$ cycles to process a 16×16 macroblock.

In the SA-2D architecture, VBS-ME is handled via the conversion of a 16×16-PE array into sixteen 4×4-PE arrays. An adder tree is connected to the output of these 16 arrays to form SADs of larger block size. In this design, we need to add 48 ADD, 15 MIN units and an adder tree. Thirty-two rows by 36 columns of pixels are processed for 4×4 blocks so it takes approximately $36 \times 32 = 1152$ cycles to process a 16×16 macroblock.

For VBS-ME in TA-1D, four 4×1 trees are needed in addition to 3 ADDs and 3 MIN units. Some registers and control logic are also needed to store intermediate SAD results. It takes approximately $1024 \times 16 = 16384$ cycles to perform a full search of a 16×16 macroblock.

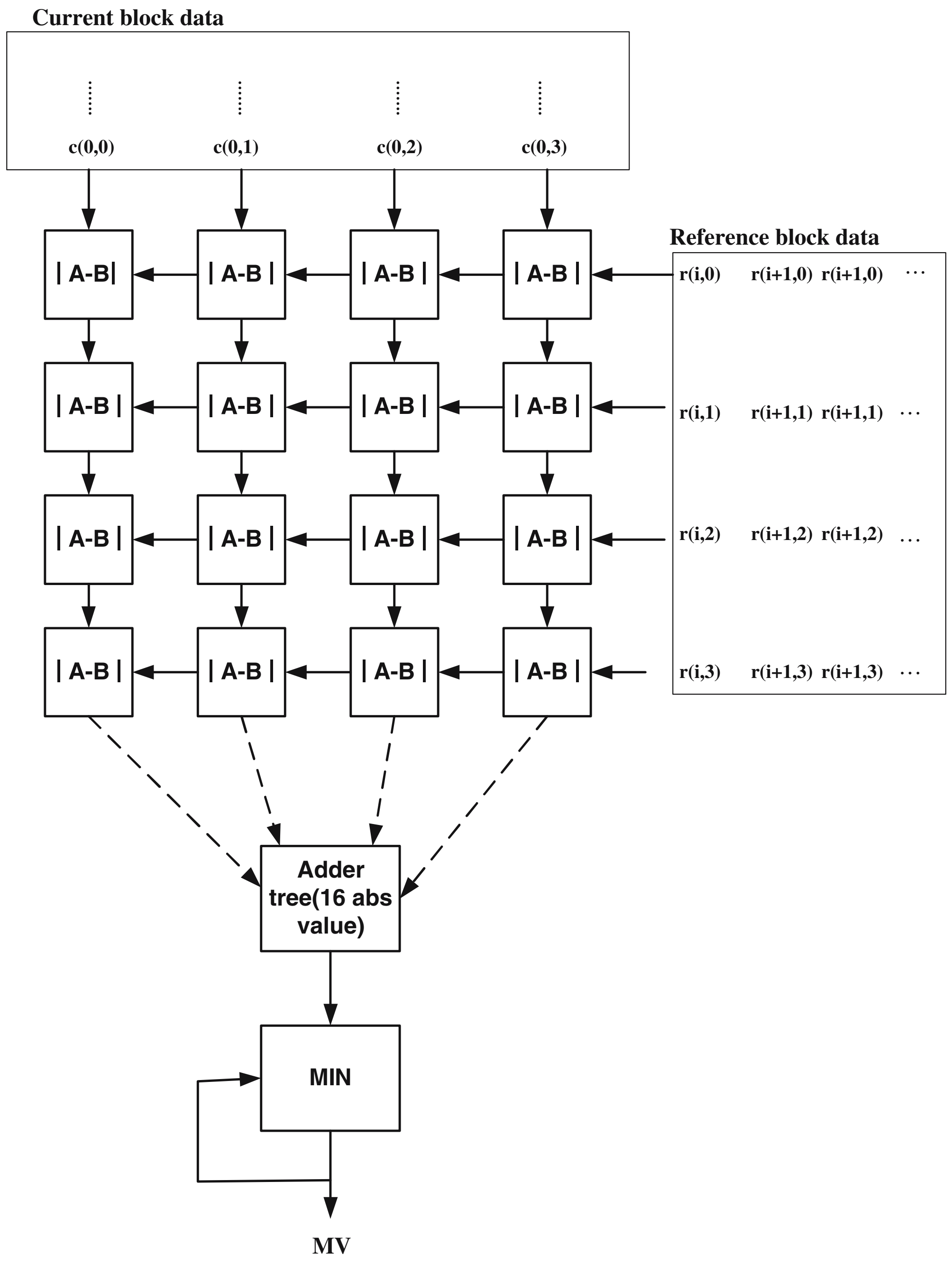
Current block data
c(0,0)
c(0,1)
c(0,2)
c(0,3)
Reference block data
r(i,0) r(i+1,0) r(i+1,0) ⋯
r(i,1) r(i+1,1) r(i+1,1) ⋯
r(i,2) r(i+1,2) r(i+1,2) …
r(i,3) r(i+1,3) r(i+1,3) ⋯
| A-B |
Adder tree(16 abs value)
MIN
MV

*Figure 7.* 2-D tree architecture (TA-2D).

Finally, for TA-2D, VBS-ME support is handled by attaching MIN elements to each level of the adder trees. In total 41 MINs are need for 41 motion vectors. The variable block size support does not change the number of cycles, so it takes approximately 1,536 cycles.

## 4. Bit-serial Implementation

In the bit-parallel motion estimation schemes just described, the SAD comparison must be made after the summation of all pixel differences. Thus, the 16-bit SAD (for a 16×16 Mblock) must be produced before any comparison can be made. Even if the current SAD is much larger than the current minimum SAD, it is not possible to terminate the SAD operation before the 16-bit result is produced. This results in wastage of hardware resources and power consumption. The bit-parallel partial distortion elimination technique [6, 16] has been proposed for the early termination of the SAD computation.

The disadvantage of this approach is a large increase in hardware requirements and reduction in the maximum operating frequency of the design. Also, the design must operate in a row-serial manner and maximum parallelism is not achieved. In this section we describe an MSB-first bit-serial technique that can address this problem.

### 4.1. *MSB-first Bit-serial Implementation of SAD*

MSB-first arithmetic [22], also called on-line arithmetic, is a bit-serial arithmetic technique in which all operations start from the most significant bit. It is particularly efficient for operations such as square root, division and comparison. This can be used advantageously in motion estimation as some comparisons can be made without examining all the bits involved, saving computation.

Compared to least significant bit (LSB) first techniques, the MSB-first approach produces results with higher latency, this delay being called on-line delay. The number of delay cycles depends on the number of operands. Redundant number systems such as carry-save or signed-digit representations are normally employed.

We use a radix-2 signed-digit format with digit set {−1,0,1} in our bit serial design [22]. The absolute difference computation can be implemented efficiently and all arithmetic operations, such as addition, subtraction and multiplication are implemented using this scheme.

Motion estimation involves the calculation of SAD values between current block and reference block as shown in Fig. 1. By rewriting this equation in a bit-serial fashion, we get Eq. (1) which has a triple summation.

$$SAD_{P,Q}(m,n) = \sum_{i=0}^{P-1} \sum_{j=0}^{Q-1} \left| \sum_{k=0}^{7} 2^k \times (c(i,j,k) - r(i+m,j+n,k)) \right| \quad (1)$$

The double summation over (P,Q) is mapped to the signed-digit adder tree and computed spatially while the innermost summation (0–7) of bit-serial part is computed iteratively. The remaining problem is how to generate signed-digit numbers from current and reference pixel values.

Recall that both current and reference pixels are positive 8-bit integers. The computation of their difference in signed-digit representation is done by making the current pixel positively weighed and the reference pixel negatively weighed. To compute the absolute value, a sign-detection on the difference is first performed. The positive and negative parts of the signed-digit number are compared MSB-first until non-equal bits are detected. If the positive bit is 0 and the negative bit 1, the difference was negative and a sign change must be performed. Otherwise, the difference was positive or zero and no change is made. A sign change is done by swapping the positive and negative digits in the number from and including the first non-equal bit. For example, if $x_+ = 10110000$ and $x_- = 10111010$, swapping is required, after which $x_+ = 10111010$ and $x_+ - = 10110000$. The absolute value operation is done on-the-fly and an 8-bit signed-digit result is produced.

### 4.2. *Early Termination Scheme and an Enhanced Method*

There are two related advantages to having a good initial value for the minimum SAD. The first is that early termination of comparisons to the current minimum can be effected more frequently. The second is that updates to the minimum SAD value take extra cycles, and initialization can serve to reduce their occurrence. H.264/AVC uses MV prediction mode (Fig. 8) and initializes the search to the predicted location. In the typical case, this serves to reduce the number of SAD updates as the search is started with a near-minimum value. Table 1 gives our simulation results showing the number of clock cycles needed to complete the comparison operation for different video scenes with different motion vector initialization strategies, a non early termination implementation requiring 16 cycles for the three initialization schemes. The news example is almost-still motion and zero-assumed (Initial MV=$0, 0$) motion and predicted MV initialization performs better than a standard sequential scheme (Initial MV=$-16, 15$). In fast motion scenes, such as flower and Stefan, the

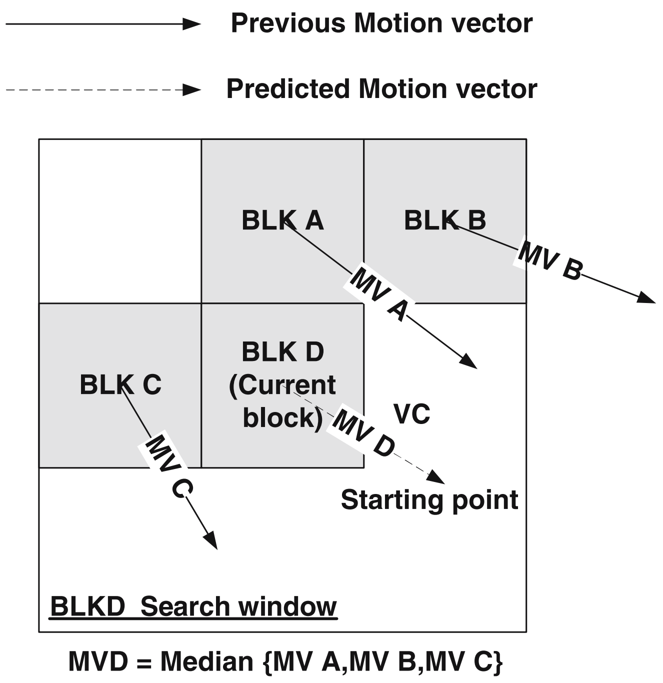

*Figure 8.* H.264/AVC motion vector prediction.

H.264/AVC predicted MV initialization scheme performs the best and has an average of 5.78 cycles. On average our early termination scheme with predictive MV initialization offers a (16−5.79)/16=63.8% savings in comparison operations. For the entire motion estimation computation, in total 28 cycles (Fig. 15) are required in the worst case, and on average our early termination scheme with predictive MV initialization offers a 36.5% improvement. Our bit-serial implementation employs this early termination scheme to eliminate wasted cycles in bit-parallel computations.

In terms of processor throughput, 100% speed-up can be achieved when 50% of calculations can be eliminated. In our case, we have to deal with the variable block size effect, which affects our early termination scheme. Since we have to compute 41 parallel comparisons, some can be terminated earlier than others. There exist dependencies between successive types of SADs, e.g. 8×4 depends on 4×4, so we cannot terminate the 4×4 summation process even if we know the current 4×4 SAD is larger than the current minimum. Thus termination only occurs when all SADs have finished, this being detected by ANDing all of the comparator results. Following termination, the next search is started.

*Table 1.* Number of cycles to complete comparison stage for different scenes using different starting strategy (16 cycles for no early termination).

| Video type | Sequential | Zero MV | Predicted MV |
|---|---|---|---|
| News | 6.95 | 5.39 | 5.4 |
| Flower | 6.64 | 5.83 | 5.5 |
| Stefan | 7.26 | 6.54 | 6.46 |

## 4.3. *Datapath*

(1) Multi-Operand SD-Adder Tree: The macroblock size of H264/AVC is 16 by 16 pixels with a 4×4-block as its smallest sub-block. To find all the minimum motion vectors of a 16×16-block and its subblocks, we employ a SAD-reuse strategy [21]. Since the different macroblock modes are overlapped in the spatial domain (Fig. 2), the SAD can be calculated using 4×4 SADs and a sequence of merging steps applied to obtain other SADs. For example, an 8×4 and 4×8-SAD can be formed by combining corresponding values of 4×4-SADs (e.g. 4×4-SADs (Block 1, 2) in Fig. 2 are combined to form an 8×4-SAD (Block 17)).

Similarly, an 8×8-SAD can be formed from 4×8-SADs; 16×8 and 8×16-SADs can be formed from 8×8-SADs; and finally, a 16×16-SAD is formed from 16×8-SADs. As a result, the 4×4-SAD computation becomes our primitive element and is reused to form other SADs. The top level adder tree for generating all 41 SADs is shown in Fig. 9.

The SAD for a 4×4 subblock is produced by 16 pairs of operands summed in signed-digit format, implying we need to add 32 bit operands in our adder tree. In our adder tree for SAD, we make use of two kinds of adders, namely online carry-save full adders (ol-CSFA) and online signed-digit adders (ol-SDFA) [23] (Fig. 10). Specifically, we use a 16-operand signed-digit adder tree based on two ol-CSFA trees (Fig. 11) which are combined using an ol-SDFA [24] as illustrated in Fig. 12. For a 4×4 subblock, this consists of eight levels with 8 cycles of on-line delay. The total number of cycles to calculate the 12-bit summation including the on-line delay is $8+8=16$ cycles. The output of a SAD4×4 adder tree is the SAD value of a 4×4-subblock in signed-digit format. This value is passed to the SAD Merger unit to calculate other larger SADs.

(2) SAD merger: In our design we need sixteen SAD4×4 adder trees to compute the SAD of 16 subblocks in parallel. The sixteen SAD4×4 values computed are passed to the SAD merger as inputs (Fig. 13). The sixteen 4×4-SADs are fed to a series of ol-SDFAs, i.e. SAD merger, and combined to form 4×8, 8×4, 8×8, 16×8, 8×16 and 16×16 SADs. The number shown in Fig. 13

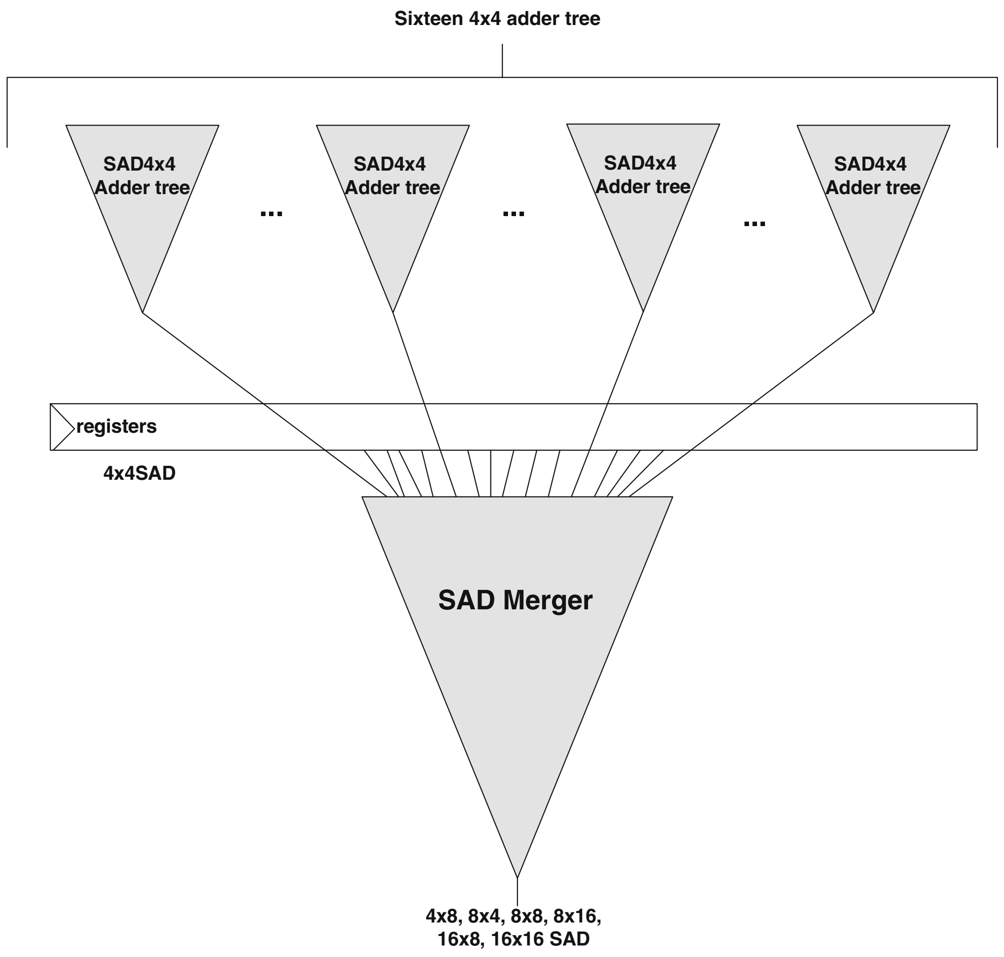

*Figure 9.* Signed-digit adder tree that generates 41 SADs.

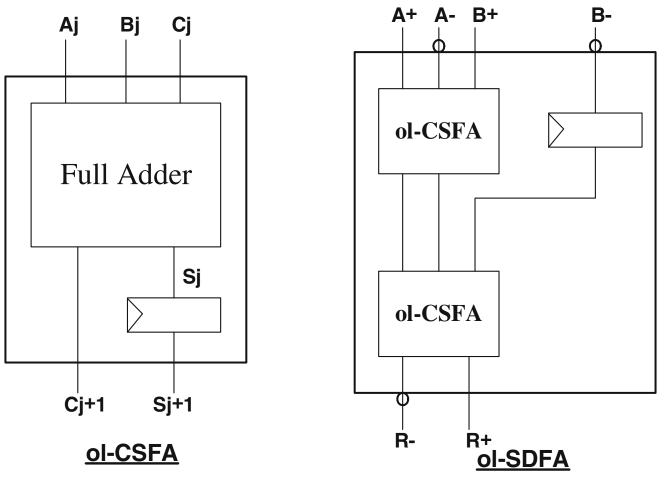

*Figure 10.* On-line carry save and signed digit adders.

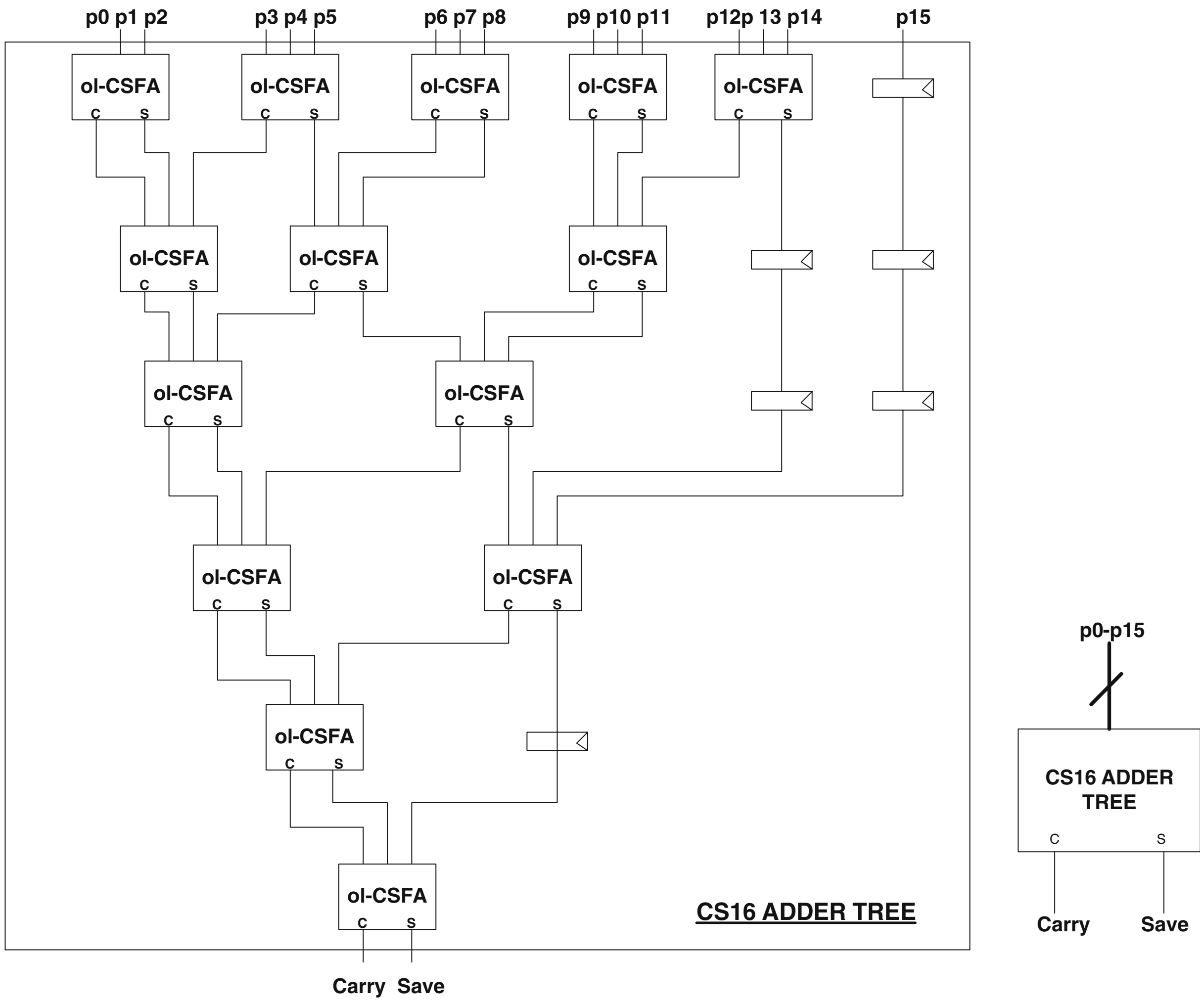

*Figure 11.* A 16-operand carry save adder tree.

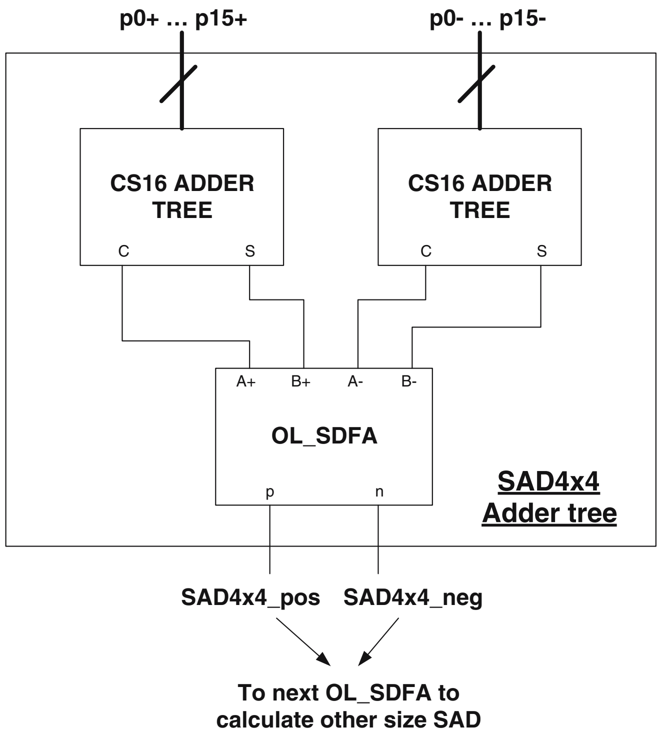

*Figure 12.* 16-operand signed-digit adder tree for 4×4 SADs.

indicates which block's SAD is calculated at that node. The block index is shown in Fig. 2. In total, the number of ol-SDFAs in SAD merger is 8+8+4+2+2+1=25. Pipelining registers are added between SAD4×4 adder trees and the SAD merger to split the combinatorial path and boost the operating frequency. In our FPGA prototype, one pipeline register obtains a good balance between maximum frequency and latency.

Finally, the 41 SAD values are passed to an on-line comparator. Since the arrival times of different SAD results are different, the completion times to determine the minimum SAD vary. Table 2 shows the delay for each type of SAD.

(3) Signed-digit comparator: In the comparison stage, we compare the current SAD to the current minimum SAD for each subblock type in a MSB-first manner. A signed-digit comparator is used for this purpose. The architecture of the comparator suggested in [5] is shown in Fig. 14. If the number being compared has a difference of two or more, we can determine which SD number is bigger. The on-line comparator will stop when this situation arises, a proof being given in [5]. The on-line comparator can determine the result in a minimum of 2 cycles.

### 4.4. Other Details

In a bit-serial based architecture, we also need to handle word-to-serial conversion which is unnecessary in a bit-parallel design. In addition, we have to handle extra scheduling brought upon by MSB-first arithmetic. For example, summation of 16 8-bit signed-digit numbers results in a 12-bit result, which involves 8 cycles of on-line delay. Hence we have to generate eight consecutive cycles of all-zero operands to feed into the adder tree to compensate the online delay. Similarly, a 16×16 SAD requires 12 consecutive cycles of zeros as shown in Fig. 15. The 16-bit 16×16-SAD result is calculated in 28 cycles for the worse case where the last 2 cycles in Fig. 15 are required for the online comparison.

The allocation of picture pixels in memory is different to that normally used in a bit-parallel case. 256×9-pixels of the search window are stored in 4 block RAMs, each being an 18 kB memory bank in the FPGA. The RAM address indexes the bit position

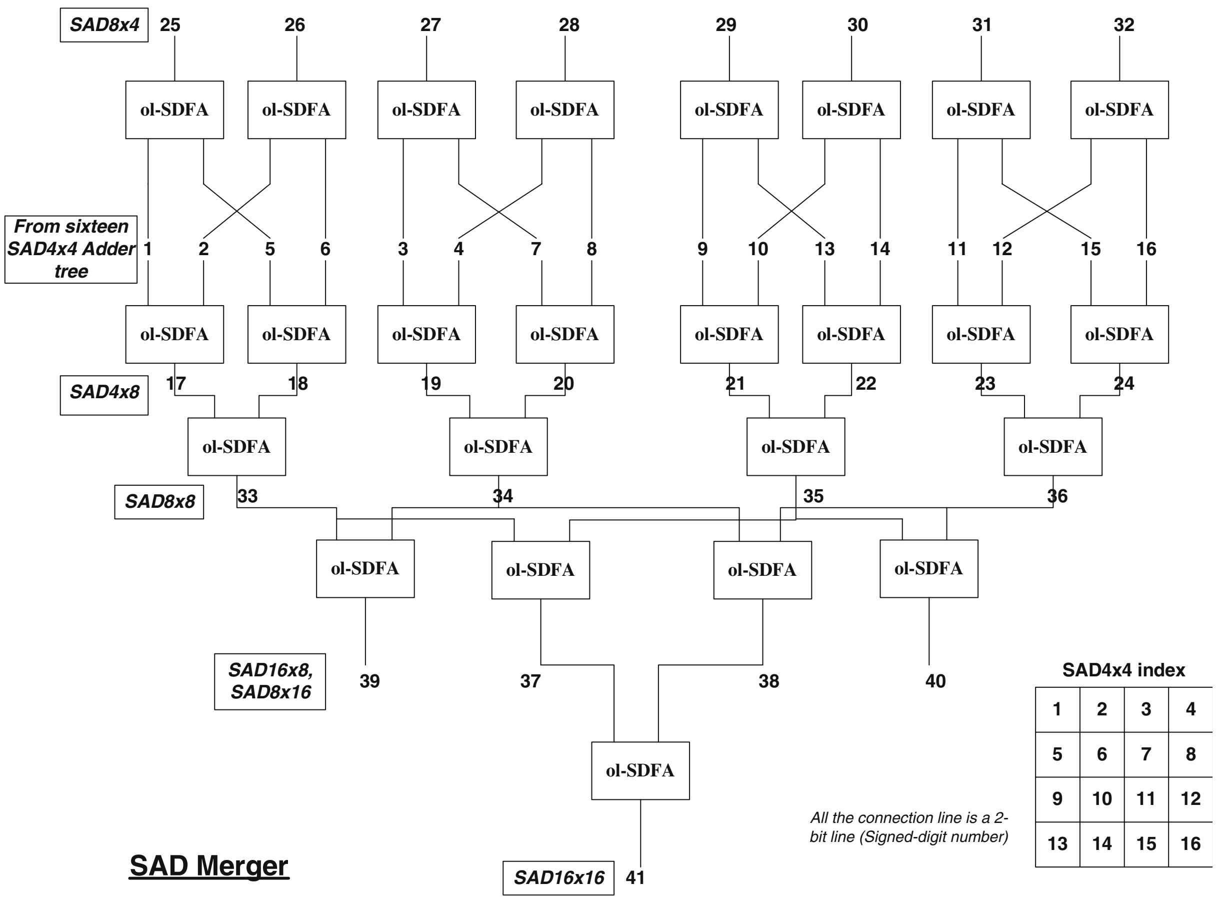

*Figure 13.* SAD merger.

instead of the pixel location. Before feeding the reference block pixels into the SD adder tree, 1-bit from 32×32 pixels are loaded from 4 block RAMs to 4-to-1 multiplexers. The multiplexers select the correct reference block from the search window, and their values are loaded in a bit-serial fashion, MSB-first. The drawback of this approach is that we need preprocessing to fetch search window pixels from the external bus to block memories, requiring shift registers before the block RAMs. The current block is stored similarly but no multiplexers are required.

*Table 2.* On-line delay (latency) of different SAD types.

| SAD type | Delay (cycles) |
|---|---|
| 4×4 | 16 |
| 4×8, 8×4 | 19 |
| 8×8 | 21 |
| 8×16, 16×8 | 23 |
| 16×16 | 25 |

## 5. Results

The proposed motion estimation cores are written in VHDL hardware language, implemented, simulated and verified on a Xilinx Virtex-II Pro −6 speed grade device. They are synthesized using Simplicity Pro 8.4. Place-and-route was done and the power consumption estimated using XPower provided by the Xilinx ISE tools. The area utilization, maximum frequency and estimated power consumption as reported by the mentioned CAD tools are used for the results presented in this section. In order to compare different architectures, "performance per slice" and "power consumption per slice" are also given as a measure of motion estimation processor efficiency. In the Virtex-II Pro, each slice includes two 4-input function generators,

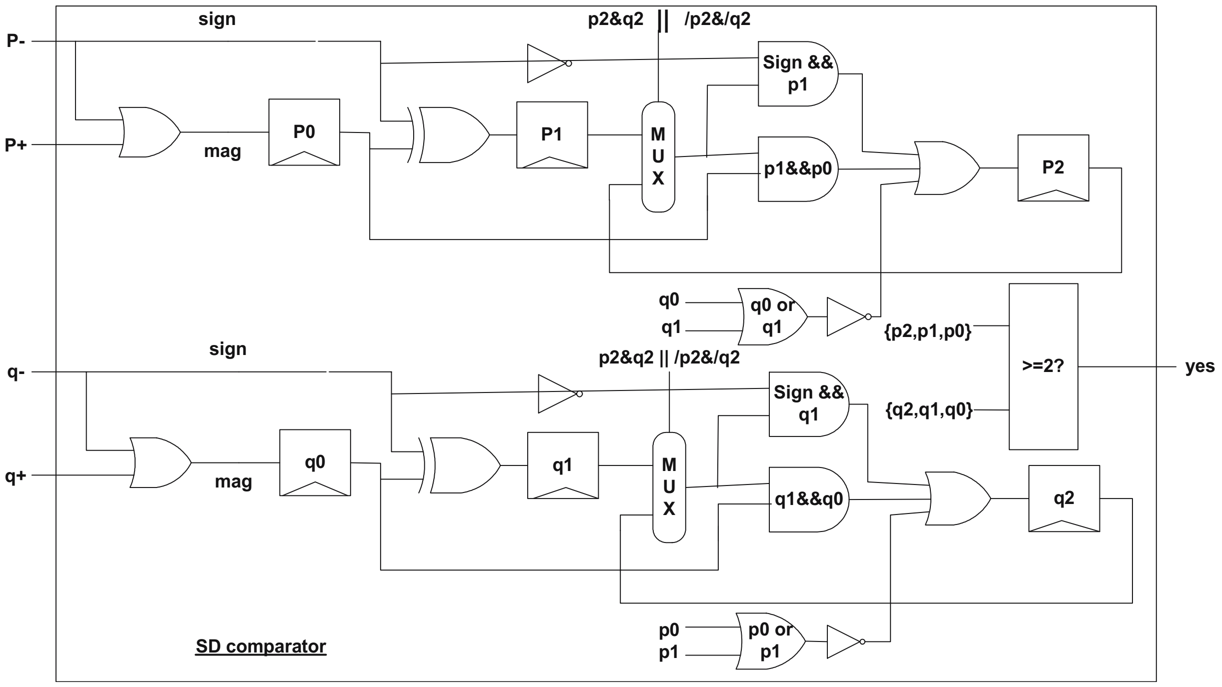

*Figure 14.* Architecture of on-line comparator.

carry logic, arithmetic logic gates, wide function multiplexers and two storage elements.

We compare both implementations of the architectures supporting fixed block size and VBS-ME. Both FS and TSS performance is compared. The search range is fixed to −16 to +15 so that the search area is *48 × 48* pixels in size, and a 16×16 macroblock is used. The bit-serial design uses the enhanced early termination scheme with predicted MV. To measure the efficiency of different architectures, we ignore the

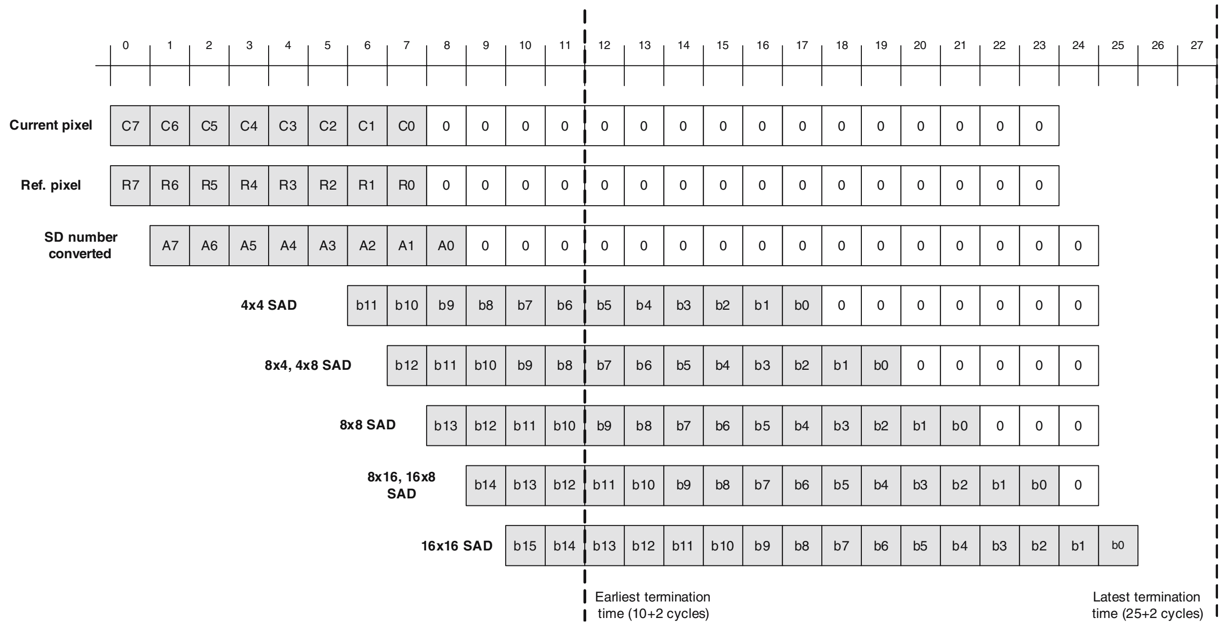
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
Current pixel
C7 C6 C5 C4 C3 C2 C1 C0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Ref. pixel
R7 R6 R5 R4 R3 R2 R1 R0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SD number converted
A7 A6 A5 A4 A3 A2 A1 A0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4x4 SAD
b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0 0 0 0 0 0 0 0
8x4, 4x8 SAD
b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0 0 0 0 0 0
8x8 SAD
b13 b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0 0 0 0
8x16, 16x8 SAD
b14 b13 b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0 0
16x16 SAD
b15 b14 b13 b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0
Earliest termination time (10+2 cycles)
Latest termination time (25+2 cycles)

*Figure 15.* Timeline of bit-serial design for whole motion estimation computation process.

*Table 3.* Fixed block size results for systolic array processors (best performance in bold).

| Design | SA-1D | SA-2D | TA-1D | TA-2D |
|---|---|---|---|---|
| Max frequency (MHz) | 239 | 232 | **240** | **240** |
| Clocks per 16×16 Mblock | 16,401 | 1,600 | 16,390 | **1,546** |
| Area (Slices) | 836 | 9,478 | **350** | 5,789 |
| Area (Gate) | 16,788 | 193,345 | **8,281** | 142,234 |
| Throughput in FS (Mblock/s) | 14,572 | 145,000 | 14,643 | **155,239** |
| Max req bandwidth (MB/s) | 7,615 | **3,712** | 7,680 | 3,920 |
| Throughput/Slice | 17.4 | 15.3 | **41.8** | 26.8 |
| Total power (mW) | 1,344 | 26,755 | **1,160** | 19,324 |
| Energy per Mblock (mJ/Mblock) | 0.092 | 0.185 | **0.079** | 0.124 |
| Power/Slice (mW/Slice) | **1.61** | 2.82 | 3.31 | 3.34 |

area occupied by data storage for current block pixels, reference block pixels, motion vectors and minimum SADs since they are required in other parts of a H.264/AVC coder and normally stored in external memories. For 2D architectures, as the data storage for the current block is implemented inside the systolic array, they are counted in the area measures. The throughput of architectures is given by $\frac{max\ frequency}{cycles\ per\ macroblock}$.

## 5.1. Fixed Block Size

Table 3 summarizes implementation results obtained for the systolic array implementations without VBS-ME.

Comparing the 1D architectures, TA-1D has a similar performance to SA-1D but uses far less resources. This improves its throughput/slice. In addition it has higher pixel memory bandwidth compared to the SA-1D architecture because of higher operating frequency. The TA-2D architecture uses the lowest number of clock cycles and has the highest throughput among all architectures. However, together with SA-2D, they store the current block pixels in each of its PEs, making its area larger. This is a tradeoff as memory bandwidth is greatly reduced. Among the bit-parallel architectures, the most power efficient architecture is TA-1D. Generally 1D architectures use less energy per macroblock since the PEs are better utilizied.

## 5.2. Variable Block Size

Table 4 summarizes implementation results obtained for all VBS-ME implementations. As more computation is required, the area is increased and throughput decreased compared with fixed block size implementations.

*Table 4.* Variable block size results (best performance in bold).

| Design strategy | SA-1D | SA-2D | TA-1D | TA-2D | BS |
|---|---|---|---|---|---|
| Max frequency (MHz) | 230 | 227 | 240 | 239 | **420** |
| Clocks per 16×16 Mblock | 16,393 | **1,172** | 16,392 | 1,546 | 18,432 |
| Area (slices) | 1,457 | 10,794 | **925** | 8513 | 2,133 |
| Area (gate) | 25,158 | 215,315 | **20,614** | 184,329 | 55,301 |
| FS throughput (Mblock/s) | 14,030 | **193,686** | 14,641 | 154,592 | 22,786 |
| Max req bandwidth (MByte/s) | 7,360 | 14,528 | 7,680 | **3,920** | 13,440 |
| Throughput/slice (Mblock/s/slice) | 9.6 | 17.9 | 15.8 | **18.1** | 10.7 |
| Total power (mW) | **1,794** | 29,875 | 2,834 | 32,337 | 13,919 |
| Energy per Mblock (mJ/Mblock) | **0.128** | 0.154 | 0.194 | 0.209 | 0.611 |
| Power/slice (mW/slice) | **1.23** | 2.77 | 3.06 | 3.79 | 6.5 |

BS refers to the FS bit-serial implementation with early termination and predictive MV initialization.

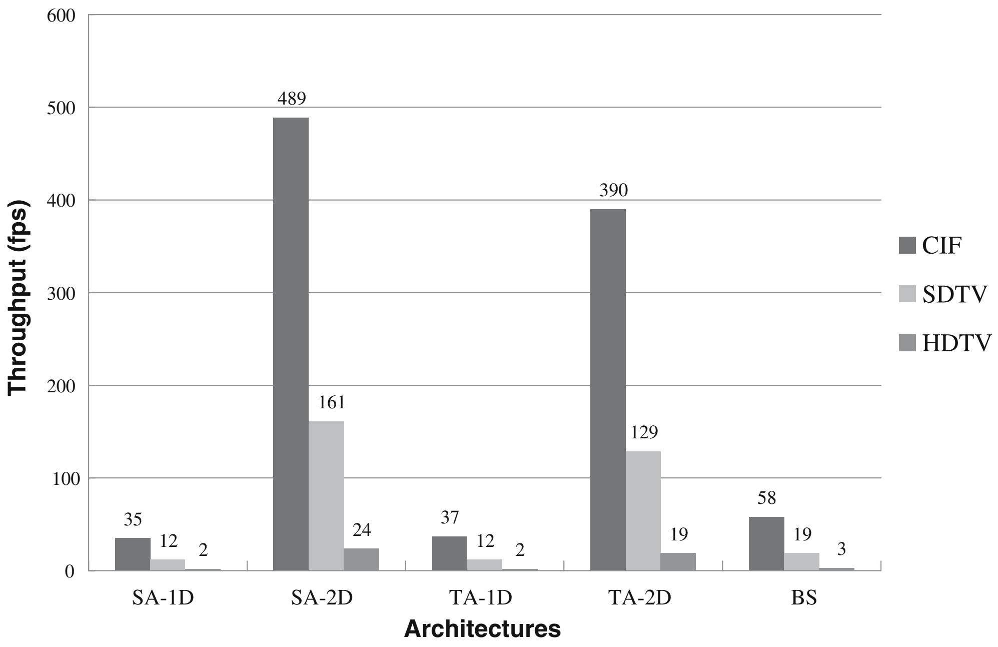

*Figure 16.* Throughput of different motion estimation architectures at different resolutions.

The lowest area is achieved by the TA-1D design and the highest throughput is achieved by SA-2D. The best throughput per gate is achieved by TA-2D. These designs are able to achieve high performance and efficiently utilize hardware resources. The bit serial design sits between the 1D and 2D architectures and balances throughput, bandwidth and area. Its performance is helped by its much higher clock frequency compared to the other bit-parallel implementations. The lowest power and energy consumption is achieved by the SA-1D design. Bit-parallel designs are much more power efficient than bit-serial

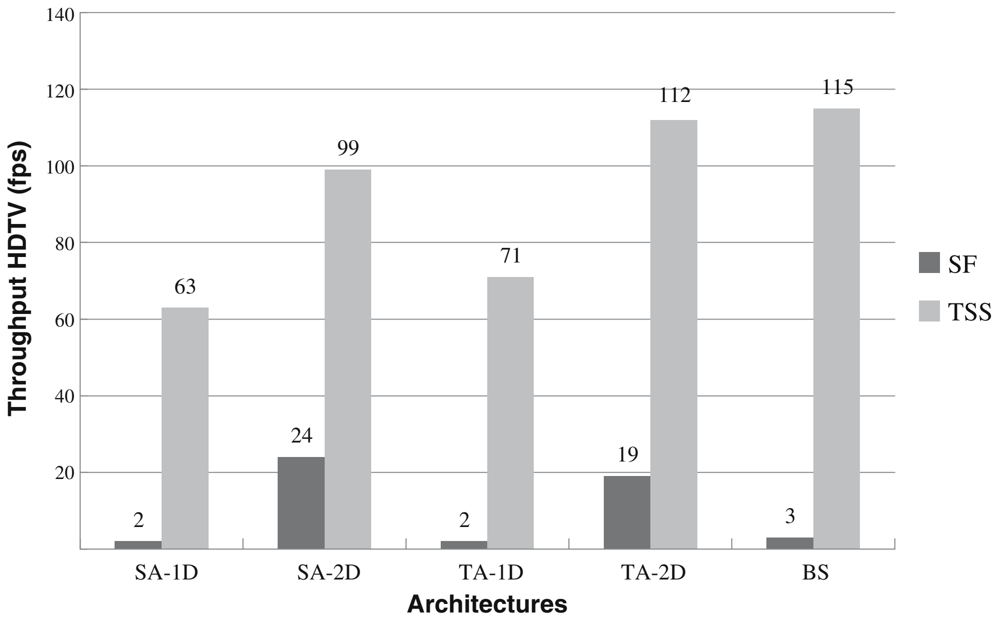

*Figure 17.* Throughput of different architectures for different motion estimation algorithms.

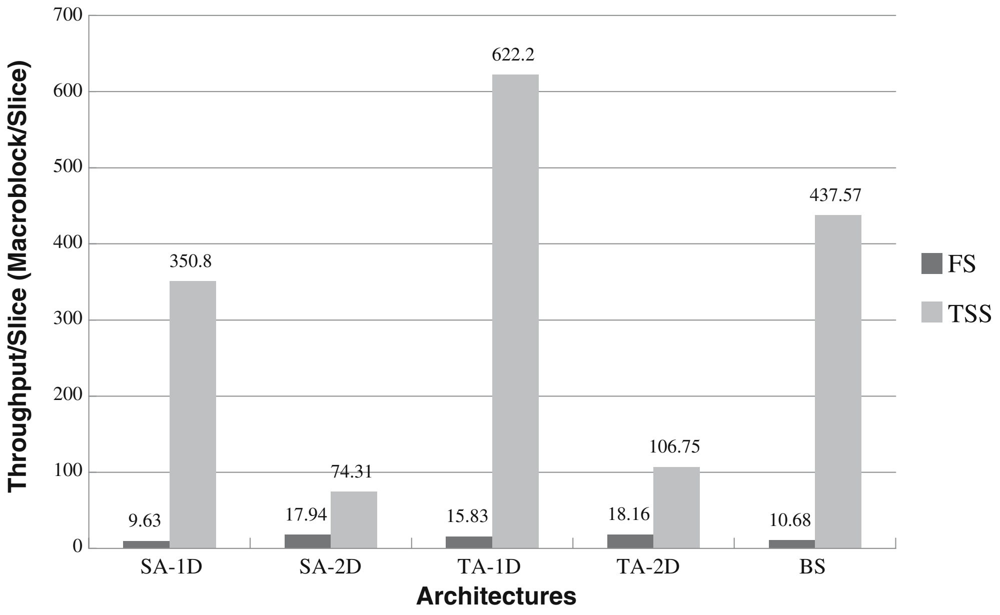

*Figure 18.* Maximum throughput per slice of different motion estimation architectures.

due to their lower operating frequency and less storage elements.

A throughput comparison of different architectures at different resolutions is shown in Fig. 16. The 2D designs clearly have the highest performance among all alternatives due to their pipelined, parallel architecture. For VBS-ME, SA-2D has the highest throughput. Two dimensional architectures are the most suitable for high throughput applications such as full HD video encoding, cinema quality video creation, etc. Low throughput applications, e.g. video conferencing, are best handled with 1D systolic and bit serial architectures.

In terms of area (Table 4), 2D architectures occupy the most resources, 1D the least and the bit-serial implementation are in-between. Area is strongly related to its performance and throughput/slice is a better measure of an architecture's efficiency. As area directly affects the cost of the hardware, a suitable architecture should be selected to minimize production cost. In modern technology, implementation of 2D architectures on FPGA devices is still expensive as high-end FPGA devices must be used. 1D and bit-serial architectures may be more suitable for cost-constrained applications.

The required memory bandwidth for different architectures at their maximum operating frequency is shown in Table 4. The SA-2D has the largest bandwidth requirement as 16 PEs must be supplied with 8 B/cycle. The bit-serial design also has high bandwidth due to its higher operating frequency. The SA-1D, TA-1D and TA-2D designs have reduced memory bandwidth requirements, TA-2D being the lowest by far. Memory bandwidth significantly affects the power consumption and in battery-powered applications, high bandwidth architectures should be avoided. On the other hand, memory is slow compared to logic and smaller bandwidth requirements often allow us to process data at a higher throughput.

Power consumption is due to four main factors. The area occupied, operating frequency, bandwidth requirement, and algorithm involved. Bandwidth can be reduced by introducing more local memories, but the area is increased. The power and energy consumption and power per slice are shown in Table 4,

*Table 5.* Summary of throughput for full search and three-step search.

| Architecture | Worst case delay (cycles) | Throughput (FS) (SAD/cycle) | Throughput (TSS) (SAD/cycle) |
|---|---|---|---|
| SA-1D | 33 | 16 | 18 |
| SA-2D | 33 | 1 | 10.4 |
| TA-1D | 22 | 16 | 16.6 |
| TA-2D | 26 | 1 | 9.8 |
| BS | 28 (18 with ET) | 18 | 20 |

ET stands for early termination.

*Table 6.* Results and comparison of VBS-ME processors on FPGA devices.

| Device | Altera EP20K200E [15] | Xilinx XC2V6000 [16] | Xilinx XC2VP100-6 (this work) | Xilinx XC2VP100-6 (this work) |
|---|---|---|---|---|
| Design strategy | Bit-parallel | Bit-parallel | Bit-serial | SA-2D |
| Max frequency(MHz) | 120 | 51.49 | **420** | 227 |
| Area (slices) | 3,690 | 9,788 | **2,133** | 10,794 |
| Throughput (Mblock/s) | 29,296 | 3,036 | 23,068 | **193,686** |
| Throughput/Slice | 7.9 | 0.31 | 10.8 | **17.9** |
| Average cycles/SAD | 40 | 16.4 | 27 | **1.14** |

One slice is considered to be two logic cells in a Xilinx part and 2 logic elements for Altera parts.

the SA-1D being superior in all these metrics. The high power consumption of the bit-serial design is due to the high frequency required by bit-serial architectures. Employing fast algorithms other than full search can greatly reduce its power consumption, albeit with some loss of quality. This may be acceptable in many low-end applications.

Figure 17 shows the throughput of different architectures for both full and three-step searches, assuming HDTV resolution (1080p) and Fig. 18 shows the throughput per slice. The bit-serial design performs the best for TSS as the data dependencies in bit-parallel designs lead to data hazards, causing the pipeline to stall. This overhead does not exist in our bit-serial design. A high level summary of the performance for full and TSS is given in Table 5. In 1D or 2D systolic arrays, 16 and 32 cycles are wasted respectively when a data hazard occurs. As an example, a three-step search in a 2D systolic array requires about 260–280 cycles to calculate SADs for 27 search points and the two redundant search points in TSS cannot be eliminated. The number of cycles per search point is increased from 1 to around 10 cycles/search point. In our bit-serial architecture, due to lack of systolic data dependencies, wasted cycles due to pipeline flushes are minimized and the efficiency for TSS and other fast search algorithms can be much higher. It is able to process three-step search in around 450 cycles compared to 18,432 for FS, and the number of cycles per search point is almost kept constant.

## 5.3. *Comparison to Previous Work*

In this subsection we compare our bit-serial and SA-2D implementations to previously reported FPGA and ASIC VBS-ME designs. The differences in FPGA architectures, CAD tools and VLSI technology used make direct comparisons between the different implementations difficult. For example, the same design would have higher throughput if a newer device is used, and it is difficult to separate improvements due to technology from improvements due to the design.

In Table 6, we compare our work with previously reported VBS-ME implementations. The bit-serial design operates at the highest frequency among all architectures due to its bit-serial design and the 130 nm technology of the FPGA. Furthermore, its throughput per slice is comparable to the best previously reported bit-parallel design. The SA-2D design achieves the lowest average number of cycles per SAD and best overall throughput and throughput per slice of all implementations studied.

*Table 7.* Results and comparison of VBS-ME processors on ASIC devices.

| Design strategy | Bit-parallel [25] | Bit-parallel [26] | Bit-parallel [27] | Bit-parallel [28] | Bit-serial (this work) | SA-2D (this work) |
|---|---|---|---|---|---|---|
| Num. PEs | 256 | 256 | 16 | 16 | N/A | 256 |
| Max frequency (MHz) | 200 | 100 | 100 | 294 | 420 | 227 |
| Area (Gate) | 597k | 154k | 108k | 61k | 55k | 215k |
| Throughput (Mblock/s) | 195,313 | 97,560 | 5,560 | 17,820 | 23,068 | 193,686 |
| Performance/gate | 0.327 | 0.634 | 0.051 | 0.292 | 0.417 | 0.90 |

Since we can obtain an equivalent gate count from Xilinx ISE tools, we are also able to compare our architectures to ASIC implementations. The gate count collected from the Xilinx ISE tool is likely to overestimated. Tables 6 and 7 show the results for FPGAs and ASICs respectively. A comparison of our designs with previously reported ASICs is given in Table 7. The number of PEs indicates the architecture, typically 16-PE architectures being 1D systolic and 256-PE architectures being 2D. In typical 1D implementations, performance per slice is lowest while fully parallel 2D architectures obtain the highest scores. Our bit-serial design lies in-between.

## 6. Conclusion

We have presented FPGA-based systolic and bit-serial implementations of VBS-ME processors for H.264/AVC. For the bit-serial implementations, on-line arithmetic, careful initialization and early termination are combined to achieve small area with high performance. Of the VBS-ME designs tested, the bit-serial design achieves highest throughput for fast search algorithms such as the three-step search. For full search, the two dimensional architectures were superior in terms of performance to the one dimension ones and best absolute throughput was achieved by the SA-2D design but requiring very high memory bandwidth. Best performance per slice was achieved by the 2D designs and for power consumption per slice, best results were achieved with the SA-1D design. Our motion estimation processors offer different tradeoffs among performance, required memory bandwidth, area and power. Depending on the design criteria, the most suitable can be selected according to the results presented in this work.

## References

1. T. Wieg and Ed. Pattaya, "Draft ITU-T Recommendation H.264 and Draft ISO/IEC 14496-10 AVC," in *JVC of ISO/IEC and ITU-T SG16/Q.6 Doc. JVT-G050*, 2003, Mar.
2. Y. Kamaci and N. Altunbasak, "Performance Comparison of the Emerging H.264 Video Coding Standard with the Existing Standards," in *ICME'03*, 2003, pp. 345–348.
3. T. Kormarek and P. Pirsch, "Array Architectures for Block Matching Algorithms," *IEEE Trans. Circuits Syst.*, vol. 36, no. 10, 1989, pp. 1301–1308.
4. K. Yang, M. Sun and L. We, "A Family of VLSI Designs for the Motion Compensation Block-matching Algorithm," *ACM Trans. Comput. Syst.*, vol. 36, 1989, pp. 1317–1325, Oct.
5. C. L. Su and C. W. Jen, "Motion Estimation Using On-line Arithmetic,". in *Proc. IEEE Intl. Symp. Circuits System*, vol. 1, 2000, pp. 683–686.
6. S.-S. Lin, P.-C. Tseng and L.-G. Chen, "Low-power Parallel Tree Architecture for Full Search Block-matching Motion Estimation," in *Proc. IEEE Intl Symp. Circuits and Systems*, vol. 2, 2004, pp. 313–316, May.
7. T. Koga, K. Iinuna, A. Hirano, Y. Iijima and T. Ishiguro, "Motion Compensated Interframe Coding for Video Conferencing," in *Proc. of National Telecomm. Conf*, (New Orleans), 1981, pp. G531–G535, Nov.
8. W. Lee, Y. Kim, R. J. Gove, and C. J. Read, "Media Station 5000: Integrating Video and Audio," *IEEE Trans. Multimedia*, vol. 1, no. 2, 1994, pp. 50–61.
9. H. Loukil, F. Ghozzi, and A. Samet, "Hardware Implementation of Block Matching Algorithm with FPGA Technology," in *IEEE Int. Conf. Microelectronics*, vol. 16, 2004, pp. 542–546.
10. M. Mohammadzadeh, M. Eshghi, and M. Azadfar, "An Optimized Systolic Array Architecture for Full Search Block Matching Algorithm and its Implementation on FPGA Chips," in *IEEE Int. Conf. NEWCAS*, vol. 3, 2005, pp. 327–330.
11. S. Wong, B. Stougie, and S. Cotofana, "Alternatives in FPGA-based SAD Implementations," in *IEEE Int. Conf. Field Programmable Logic*, 2002, pp. 449–452, Dec.
12. S. Wong, S. Vassiliadis, and S. Cotofana, "A Sum of Absolute Differences Implementation in FPGA Hardware," in *Proc. 28th Euromico Conf.*, 2002, pp. 183–188, Sept.
13. C. L. Su and C. W. Jen, "Motion Estimation using MSD-first Processing," in *Proc. IEEE Circuits, Device and Systems*, vol. 150, no. 2, 2003, pp. 124–133.
14. J. Olivares and J. Hormigo, "Minimum Sum of Absolute Differences Implementation in a Single FPGA Device," in *IEEE Int. Conf. on Field Programmable Logic*, vol. 3203, 2004, pp. 986–990.
15. C. Wei and M. Z. Gang, "A Novel SAD Computing Hardware Architecture for Variable-size Block Matching Estimation and its Implementation with FPGA," in *Proc. 5th Int. Conf. ASIC*, vol. 2, 2003, pp. 950–953.
16. S. Lopez, F. Tobajas, A. Villar, V. de Armas, J. Lopez, and R. Sarmiento, "Low Cost Efficient Architecture for H.264 Motion Estimation," in *Proc. IEEE Int. Symp. Circuits and Systems*, vol. 1, 2005, pp. 412–415.
17. P. M. Kuhn, G. Diebel, S. Herrmann, A. Keil, H. Mooshofer, A. Kaup, R. M. Mayer, and W. Stechele, "Complexity and PSNR Comparison of Several Fast Motion Estimation Algorithms for MPEG-4," *Proc. SPIE*, vol. 3460, 1998, pp. 486–489.
18. J. R. Jain and A. K. Jain, "Displacement Measurement and its Application in Interframe Image Coding," *IEEE Trans. Commun.*, vol. 29, no. 12, 1981, pp. 1799–1808.
19. 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, 2000, pp. 287–290.
20. C. Y. Chen, S. Y. Chien, Y. W. Huang, T. C. Chen, T. C. Wang, and L. G. Chen, "Analysis and Architecture Design of Variable Block Size Motion Estimation for H.264/AVC," *IEEE Trans. Circuits Syst.*, vol. 53, no. 3, 2006, pp. 578–593.
21. C. Y. Cho, S. Y. Huang, and J. S. Wong, "An Embedded Merging Scheme for H.264/AVC Motion Estimation," in *IEEE Int. Conf. Image Proc.*, vol. 3, 2005, pp. 1016–1019, Sept.

22. M. D. Ercegovac and T. Lang, *Digital Arithmetic*, Morgan Kaufmann, 2004.
23. M. D. Ercegovac and T. Lang, “On-Line Arithmetic: A Design Methodology and Applications,” in *Proc. IEEE workshop. VLSI Signal Processing*, 1988, pp. 252–263.
24. J. Villalba, J. Hormigo, J. M. prades, and E. L. Zapata, “On-line Multioperand Addition Based on On-line Full Adders,” in *IEEE Intl. Conf. on App. Specific systems*, 2005, pp. 322–327, July.
25. C. Ou, C. F. Le, and W. J. Hwang, “An Efficient VLSI Architecture for H.264 Variable Block Size Motion Estimation,” *IEEE Trans. Signal Process.*, vol. 51, no. 4, 2005, pp. 1291–1299.
26. M. Kim, I. Hwang, and S. I. Chae, “A fast VLSI Architecture for Full-search Variable Block Size Motion Estimation in MPEG-4 AVC/H.264,” in *Proc. ASP-DAC*, vol. 1, 2005, pp. 631–634, Jan.
27. S. Y. Yap and J. V. McCanny, “A VLSI Architecture for Advanced Video Coding Motion Estimation,” in *Proc. IEEE Intl. Conf. application-specific systems, arch., processors*, 2003, pp. 293–301, June.
28. S. Y. Yap and J. V. McCanny, “A VLSI Architecture for Variable Block Size Video Motion Estimation,” *IEEE Trans. Circuits Syst.*, vol. 51, no. 7, 2004, pp. 384–389.

**Brian M. H. Li** is a software engineer in Entone Technologies (HK) Limited. He completed the work described in the article while he was a graduate student at the Chinese University of Hong Kong. He received Mphil in Computer Science and Engineering and B.Eng in Computer Engineering from the Chinese University of Hong Kong. His research interests include high performance FPGA applications and video compressions.

**Philip Leong** received the B.Sc., B.E. and Ph.D. degrees from the University of Sydney in 1986, 1988 and 1993 respectively. In 1993, he was a consultant to ST Microelectronics in Milan, Italy working on advanced flash memory-based integrated circuit design. From 1994–1997, he was a lecturer at the University of Sydney. Since 1997, he has been with the Department of Computer Science and Engineering at the Chinese University of Hong Kong where he is a Professor and the director of the Custom Computing Laboratory. Dr. Leong is also Visiting Professor at Imperial College, London and the Chief Technology Advisor to Cluster Technology. He was program co-chair of the FPT and FPL conferences and is an associate editor for the ACM Transactions on Reconfigurable Technology and Systems. The author of more than 100 technical papers and 4 patents, Dr. Leong was the recipient of the 2005 FPT conference best paper and 2007 FPL conference Stamatis Vassiliadis outstanding paper awards. His research interests include reconfigurable computing, signal processing, computer architecture, computer arithmetic and biologically inspired computing.