FPGA-BASED MSB-FIRST BIT-SERIAL VARIABLE BLOCK SIZE MOTION ESTIMATION PROCESSOR

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
email: {mhli,phwl}@cse.cuhk.edu.hk

ABSTRACT

H.264/AVC is the latest video coding standard adopting variable block size, 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 (FSBM) variable block size motion estimation. 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. The architecture has been simulated, synthesized and implemented on a Xilinx Virtex-II XC2V6000 FPGA. The maximum frequency achieved is 340 MHz and the throughput rate is around 18674 macroblocks per second within a -16 to 15 search range. The resource utilization is 3345 LUTs and it can encode CIF resolution video in real time.

1. INTRODUCTION

H.264/AVC [1] video coding standard is the latest 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 [9]. Like previous coding standards, e.g. 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 sizes, multi-reference frames and motion vector prediction. Because of these features, the computational complexity of H.264/AVC is increased by a factor of four, creating challenges for engineers to achieve real time performance.

Many motion estimation architectures [2][3][7][9][16] have been proposed in the literature and most can achieve real time encoding. In many cases a full search strategy is chosen because of regularity, no data dependencies and it finds the optimal solution. Fast algorithms are employed in some cases, but the speedup is often not significant compared with full search, particularly for hardware implementations. For this reason we focus on full search (FS) hardware designs.

Most of the 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 [13,6,14,11,5,12,8,15], only two [8,15] support variable block sizes, and both are bit-parallel. A most significant bit (MSB)-first bit-serial design with early termination was proposed for QCIF resolution video [5] 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 SAD engine employing on-line arithmetic was also reported [12]. 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 variable block size motion estimation 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, and has a higher performance per look-up-table (LUT) than all previously reported implementations.

Furthermore, the new architecture results in performance comparable to bit-parallel implementations but with greatly reduced area. 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. This makes it feasible to implement a motion estimation processor on a small FPGA device with high performance and low power dissipation. A similar scheme could be used for other algorithms that search an input space to optimize a given metric.

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 the early termination technique employed in this work. Section 4 explains the construction of a sum of absolute difference (SAD) adder tree which uses a signed-digit representation. Section 5 presents our 4-stage top level architecture and section 6 describes details concerning data allocation and scheduling. Results and a comparison will be given in section 7 and conclusions are drawn in section 8.

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. Typically, the block size is 16-pixel by 16-pixel. 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. A sum of absolute difference (SAD) between the current block and reference block is commonly used as the distance metric since it can be implemented efficiently. A SAD is calculated using equation (1) below.

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

$$-16 \geq m,n \geq +15, P = 4,8,16, Q = 4,8 \, or \, 16 \quad (1b)$$

$$u = \min_{m,n}\{SAD_{P,Q}(m,n)\} \quad (1c)$$

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

In equation (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 subblocks/macroblock's SAD is calculated. In total there are 7 types of subblocks/macroblocks shown in figure 1. The search window is confined to -16 to +15 in this example and the rest of this paper. Lastly, MV_{min} is the motion vector with the minimum SAD value.

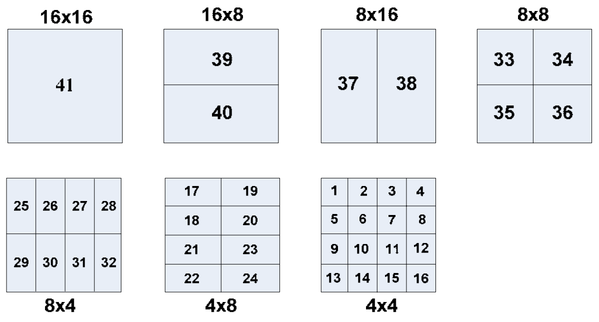

Fig. 1. Supported block sizes in H.264/AVC

2.2. Variable block size motion estimation

In traditional motion estimation such as 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, variable block sizes increase the number of motion vectors produced per macroblock and the quarter-pixel and multi-reference frame features add additional 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 7 different types of block sizes (4x4, 4x8, 8x4, 8x8, 8x16, 16x8 and 16x16) as shown in figure 1. Each macroblock has in total 41 types of sub-blocks to cover the whole macroblock. In variable block size motion estimation, for each type of subblock a motion vector (MV) is produced so in total 41 MVs are calculated per macroblock.

The variable-block-size motion estimation feature is the most challenging hardware implementation issue added in H.264/AVC. The multi-reference feature can be solved 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

Table 1. Online carry save adder example {A=$00010110_2(22_{10})$, B=$10101011_2(171_{10})$, C=$11111010_2(250_{10})$, Result[C+S]=$110111011_2(443_{10})$}

Cycle	Aj Bj Cj	Cj+1	Sj+1
0	0 1 1	1	0
1	0 0 1	0	0
2	0 1 1	1	1
3	1 0 1	1	0
4	0 1 1	1	0
5	1 0 0	0	0
6	1 1 1	1	1
7	0 1 0	0	1
8	0 0 0	0	1

Table 2. Online signed-digit adder example $\{A=\bar{1}0\bar{1}1\bar{1}1\bar{1}\bar{1}(-151_{10}), B=11111\bar{1}1\bar{1}(245_{10})$
$R=00011000\bar{1}0(94_{10})\}$

Cycle	A+ A- B+ B-	R-	R+
0	0 1 1 0	0	0
1	0 0 1 0	0	0
2	0 1 1 0	1	1
3	1 0 1 0	0	1
4	0 1 1 0	0	1
5	1 0 0 1	0	0
6	0 1 1 0	1	1
7	0 1 0 1	0	0
8	0 0 0 0	1	0
9	0 0 0 0	0	0

Table 1.

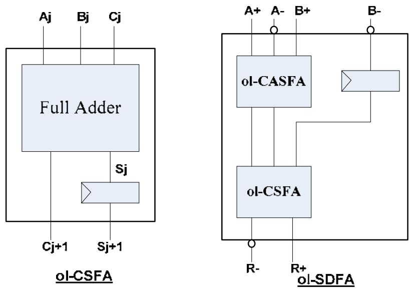

Fig. 2. On-line Carry save and signed-digit adders

generate 41 MVs within given search window and current block.

2.3. Search algorithm – full search

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 1024 searching positions. Compared with FS, techniques such as the three-step search, log-search and diamond search have reduced computational complexity but their hardware implementations involve data dependencies which makes it difficult to parallelize, and furthermore, they produce sub-optimal results.

3. EARLY TERMINATION SCHEME IN BIT-SERIAL MOTION ESTIMATION

In bit-parallel motion estimation, the SAD comparison must be made after the summation of all pixel differences. Thus, the 16-bit SAD (for a 16x16 MV) 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 (PDE) [15][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 may not be achieved. In this section we describe an MSB-first bit-serial technique that can address this problem.

3.1. MSB-first arithmetic (on-line arithmetic)

MSB-first arithmetic [17], also called on-line arithmetic, is a bit-serial arithmetical technique in which all operations start from the most significant bit. It is particularly efficient for operations such as square root, division and comparisons. This can be used advantageously in motion estimation as some comparisons can be made without examining all the bits involved. Such early termination schemes can save computation.

Compared to least significant bit (LSB) first techniques, the MSB-first approach produces results with more delays, 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 present an example of on-line addition using carry-save operands. The addition consists of three operands and is done using the on-line full adder proposed in [17], as shown in figure 2. Referring to table 1, a 3 operand 8-bit addition can be done in 9 cycles with 1 cycle of on-line delay using an online carry-save full adder (ol-CSFA).

In our adder tree for SAD, we make use of two kinds of adders. The first is the online carry-save full adder (ol-CSFA) presented above. The other is an online signed-digit adder (ol-SDFA) to handle signed-digit addition. The ol-

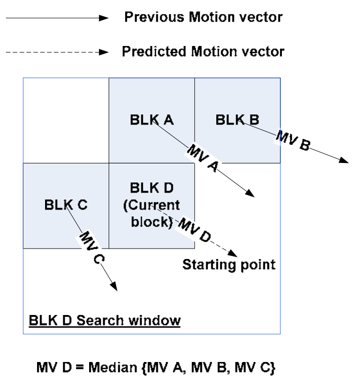

Fig. 3. Motion vector prediction in H.264/AVC

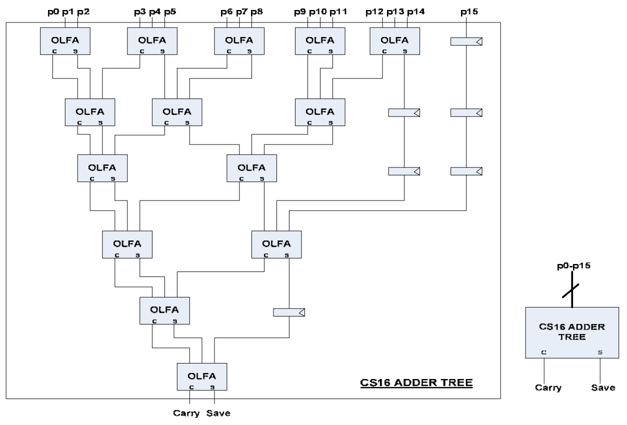

Fig. 5. A 16-operand carry save adder tree

SDFA architecture is shown in figure 2. It can be constructed from ol-CSFAs with some inputs and outputs inverted. An example of signed-digit addition is shown in table 2 with 2 signed-digit pairs. The signed-digit result can be produced with 2 cycles of on-line delay.

3.2. 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, and 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 (figure 3) 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 3 shows our simulation results showing the number of clock cycles needed to complete the comparison operation for different video scenes with different motion vector

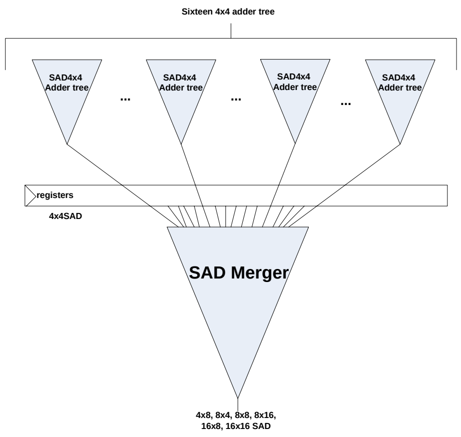

Fig. 4. Signed-digit adder tree that generates 41 SADs

Table 3. Number of cycles to complete comparison stage for different scenes using different starting strategy (16 cycles for no termination scheme)

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

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 H.264/AVC predicted MV initialization scheme performs the best and has an average of 5.78 cycles. On average our scheme offers a (16-5.79)/16=63.8% savings in comparison operations. For the entire motion estimation computation, in total (12+16)=28 cycles are required in the worse case, and on average our scheme offers a 36.5% improvement.

4. MULTI-OPERAND SD-ADDER TREE

The macroblock size of H264/AVC is 16 pixels by 16 pixels with a 4x4-block as its smallest sub-block. To find all the minimum motion vectors of a 16x16-block and its subblocks, we make use of a SAD-reuse strategy [4]. As a result, the 4x4-SAD computation becomes our primitive element and this is reused to form other SADs. Since the different macroblock modes are overlapped in spatial domain (Figure 1), the SAD can be calculated using 4x4 SAD and a sequence of merging steps to obtain other mode's SADs. For example, 8x4-SAD and 4x8-SAD can be

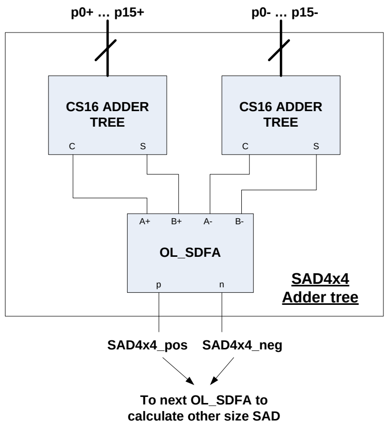

Fig. 6. 16-operand sign-digit adder tree for 4x4 SAD.

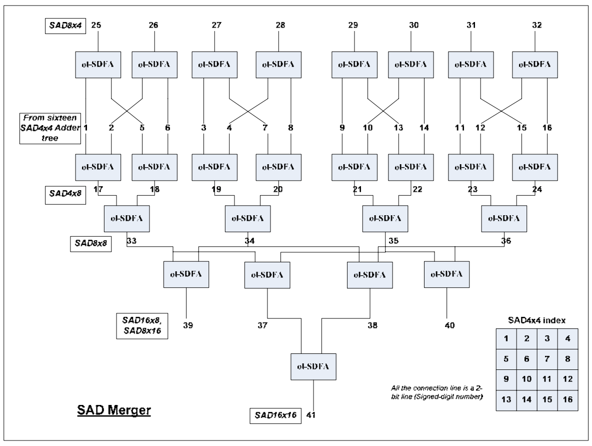

Fig. 7. SAD merger

formed by combining corresponding value of 4x4-SAD (e.g. 4x4-SAD(Block 1, 2) in figure 1 to form 8x4-SAD(Block 17)). Similarly, 8x8-SAD can be formed by 4x8-SAD. 16x8-SAD, 8x16SAD can be calculated from 8x8-SAD and finally 16x16-SAD is combined from 16x8-SAD. The top level adder tree is shown in figure 4.

4.1. SAD4x4 signed-digit adder tree

The SAD of a 4x4-subblock involves a 16 pairs of operand summation in signed-digit format. Thus, effectively we need to add 32 bit operands in our adder tree. According to [10], we could implement a 16-operand signed-digit adder tree based on double ol-CSFA trees and ol-SDFA. Together, the hardware utilization is minimized [10]. This is illustrated in figure 5 and figure 6. It consists of 8 levels with 8 cycles of on-line delay. The total number of cycles to calculate the 12-bit summation result including the on-line delay is 8+8=16 cycles. The output of SAD4x4 adder tree is the SAD value of 4x4-subblock in signed-digit format. This value is passed to SAD Merger to calculate other necessary SADs.

4.2. SAD Merger

In our design we need sixteen SAD4x4 adder trees to compute the SAD of 16 subblocks in parallel. The sixteen SAD4x4 computed are passed to SAD merger as inputs shown in figure 7. The sixteen 4x4- SAD is fed to a series of ol-SDFAs called SAD merger and combined to form 4x8, 8x4, 8x8, 16x8, 8x16 and 16x16 SADs. The number shown in figure 7 indicates which block's SAD is calculated at that node. The block index can be referred to figure 1. In total, the number of ol-SDFAs in SAD merger i is 8+8+4+2+2+1=25. Pipelining registers are added between SAD4x4 adder trees and SAD merger to split the combinatorial path and boost the operating frequency. In our FPGA platform, one pipeline register obtains a good balance between maximum frequency and latency.

Table 4. On-line delay of different SAD types

SAD type	Delay (cycles)
4x4	16
4x8, 8x4	19
8x8	21
8x16, 16x8	23
16x16	25

Finally, the 41 SAD values are passed to an on-line comparator for next stage processing. Since the arrival times of different SAD results are different, the completion times to determine the minimum SAD vary. Table 4 shows the delay for each type of SADs.

5. TOP LEVEL ARCHITECTURE

Motion estimation involves the calculation of SAD value between current block and reference block as shown in equation (1). By rewriting equation (1) in a bit-serial representation, we get the equation (2) with a triple summation.

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

The double summation (0 to P,Q) are mapped to the adder tree and computed spatially while the innermost summation (0 to 7) of bit-serial part is computed iteratively. The remaining problem is how to generate signed-digit numbers from current and reference pixel values. Both current and reference pixels are positive 8-bit integers. The computation of their difference in signed-digit representation can be done easily by making the current pixel positive weighed and the reference pixel negative weighed. The absolute value operation can be done by on-the-fly checking of the

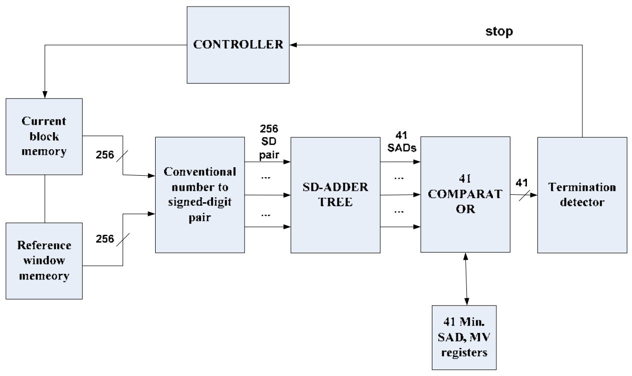

Fig. 8. SD-adder tree that generate 41 SADs

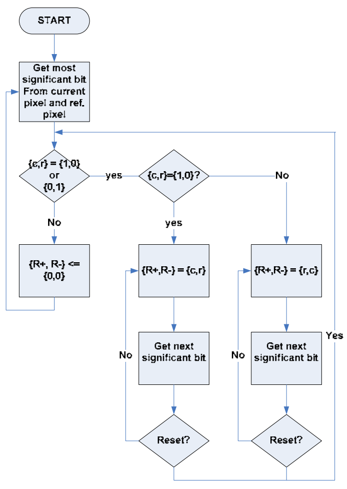

Fig. 9. Flow chart of conventional number to signed-digit conversion

signed-digit number until 1 or -1 is detected for the first non-zero digit, then the positive weighing is interchanged with the negative weighing part to complete the absolute operation if negative.

In the following sub-session we describe the entire motion estimation process in 4 stages: conventional number to signed-digit number conversion, summation, comparison and early termination stage. The top level system is shown in figure 8.

5.1. Conventional number to S.D. conversion stage

As described above, the |ci – ri| operation, where ci and ri are 8-bit positive integers from current and reference blocks, can be converted to SD representation by setting ci and ri as being positively and negatively weighed respectively and finally doing a sign-detection to check if changing the sign of result is necessary. The circuit that implements this functionality requires few hardware resources and little computation delay is introduced. A finite state machine

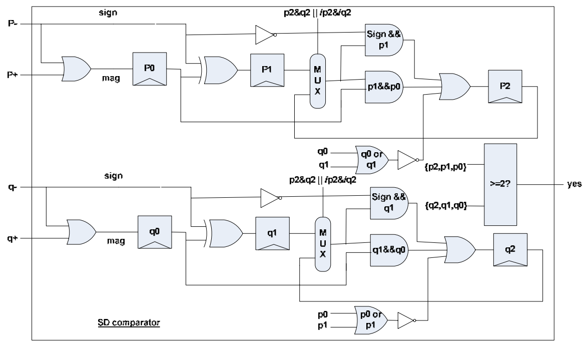

Fig. 10. Architecture of on-line comparator

which detects the first non-zero digit is required for the absolute value. Together with a pair of multiplexers for interchanging the signed-digit, |ci – rj| in signed-digit form is produced. Figure 9 shows the flow chart for sign-detection of the signed-digit number. In total there are 16x16=256 absolute difference stages in our motion estimation processor.

5.2. Summation stage

The datapath for directly feeding all 256 pairs of signed-digit numbers into our signed-digit adder tree is described in section 4 and figure 4. 41 SADs are calculated at the end of the adder tree stage.

5.3. Comparison stage

In the comparison stage, we compare the current SAD to 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 [7] is shown in figure 10. 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 for this algorithm is given in [7]. The on-line comparator can determine the result in 2 cycles at a minimum.

5.4. Early termination stage

Early termination of the SAD computation allows the avoidance of redundant calculations. 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 the others. There exists dependencies between successive types of SADs, e.g. 8x4 depends on 4x4, and we cannot terminate

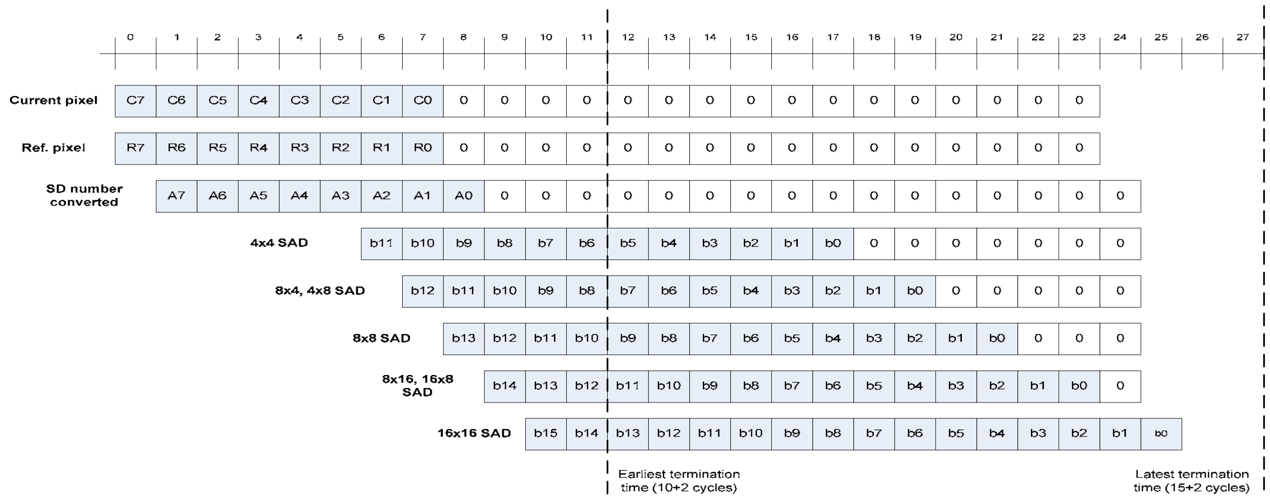

Fig. 11. Timeline for whole motion vector computation process

the 4x4 summation process even if we are sure the current 4x4 SAD must be larger than the minimum. For the sake of simplicity, we check for early termination for all SADs and when all have terminated, the current summation can be terminated and begins next searching position. Termination can be detected by OR-ing all the comparator results.

6. DATA SCHEDULING AND ALLOCATION TECHNIQUE

In a bit-serial based architecture, we 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. We have to generate 8 consecutive cycles of all-zero operands feeding into adder tree to compensate the online delay. Similarly, a 16x16 SAD requires 12 consecutive cycles of zeros as shown in figure 11. The 16-bit 16x16-SAD result is calculated in 28 cycles in the worse case.

The allocation of picture pixels in memory is different to that normally used in a bit-parallel case. 256x9-pixels of the search window are stored in 4 block rams. The ram address indexes the bit position instead of the pixel location. Before feeding the reference block pixels into the SD adder tree, 1-bit from 32x32 pixels are loaded from 4 block rams to 4-to-1 multiplexers. The multiplexers select the correct reference block from the search window. The reference block values from MSB to LSB are loaded in each cycle. The drawback of this approach is 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.

7. RESULTS AND COMPARISON

The proposed bit-serial architecture was written in VHDL, implemented, simulated and verified for a Xilinx Virtex-II XC2V6000 -6 speed grade device. The ME processor was synthesized using Simplicity Pro 8.4 and place-and-route performed using Xilinx ISE tools. The area and speed obtained is shown in table 5. Performance per LUT is also indicated to show our improved efficiency on the FPGA. The LUT per flip-flop for this design is 3345/2733=1.22 which matches the logic ratio in FPGA slices.

A comparison is also made between bit-parallel and bit-serial implementations in table 5. Note that some of these designs do not support variable block sizes, e.g. [5][6][11][12][13][14], and hence occupy significantly less area than variable-block-size motion estimation processors. This is because they require only one comparator and their adder tree can optimized for one specific block size.

Compared with those designs that fully support H.264/AVC [8,15], our architecture occupies less area and has a performance per LUT 1.4 and 36 times higher respectively. In the worst case, our design needs 28x1024=28672 cycles to complete a full search of a macroblock with -15 to +16 search range, which implies our architecture can process at least 11858 macroblocks per second. By employing early termination, our architecture can process 18674 macroblocks per second, achieving a 36.5% savings on average over an approach that does not use early termination. The overall performance achieved is comparable to reported bit-parallel implementations. Thus we are able to process CIF images (352x288 resolutions) at

Table 5. Results and comparison of ME processor

	[13]	[6]	[14]	[11]	[5]	[12]	[8]	[15]	Our design
Design strategy	Bit-parallel	Bit-parallel	Bit-parallel	Bit-parallel	Bit-serial	Bit-serial	Bit-parallel	Bit-parallel	Bit-serial
Supported block size	16x16	8x8	16x16	16x16	4x4	16x16	4x4,4x8,8x4, 8x8,8x16, 16x8,16x16	4x4,4x8,8x4, 8x8,8x16, 16x8,16x16	4x4,4x8,8x4, 8x8,8x16, 16x8,16x16
Area (LUT)	1654[1]	3752[2]	31060[1,3]	1699[1]	1510[5]	1945	7381[1]	19576	3345
Speed (MHz)	103.84	191	380.7	197	352	425	120	51.49	340
Throughput (MB[16x16]/sec)	18519	4752[2]	371513	7125[4]	5078[2]	17456	29296	3036[6]	18674
Performance/LUT	11.2	1.265	12	4.19	3.36	8.97	3.97	0.155	5.59

[1] Altera STRAIX family [2] Throughput normalized to 16x16 block size. [3] No comparator included
[4] Fully parallized approach [5] Area in terms of gate [6] Normalized to search range (-15 +16)

30 fps. The performance per LUT is comparable to bit-parallel implementations while having greatly reduced absolute area requirements, demonstrating the advantage of a variable block size.

8. CONCLUSION

We employ on-line arithmetic in an FPGA-specific implementation of variable block size motion estimation for H.264/AVC. The performance per LUT is improved over bit-parallel approaches; the small area and high speed nature of this motion estimation processor enables real time encoding of small frame sizes, and an early termination scheme serves to reduce power consumption. These combined features make the architecture particular suitable for FPGA-based implementations on mobile devices.

9. REFERENCES

[1] "Draft ITU-T Recommendation H.264 and Draft ISO/IEC 14496-10 AVC" *in JVC of ISO/IEC & ITU-T SG16/Q.6 Doc. JVT-G050*, T.Wieg, Ed., Pattaya, Thailand, Mar. 2003.

[2] Thomas Komarek, Peter Pirsch, "Array arhcitectures for Block Matching Algorithm," *IEEE Trans. On Circuits and Systems*, vol. 36, no. 10, pp. 1301–1308, Oct. 1989.

[3] K.M. Yang, M.T. Sun, L. We, "A Family of VLSI Designs for the motion compensation Block-Matching Algorithm," *IEEE Trans. On circuit and systems*, vol. 36, no. 10, pp. 1317–1325, Oct. 1989.

[4] C.Y. Cho, S.Y Huang, J.S. Wong, "An Embedded Merging Scheme for H.264/AVC Motion estimation," *IEEE Int. Conf. on Image Processing*, vol. 3, pp. 1016–1019, Sept, 2005.

[5] C.L. Su, C.W. Jen, "Motion estimation using MSD-first processing", *in Proc. IEEE Circuits, device and systems*, vol. 150, Issue 2, pp. 124-133, Apr. 2003.

[6] M. Mohammadzadeh, M. Eshghi, M.M. Azadfar, "An optimized Systolic Array Architecture for Full Search Block Matching Algorithm and its Implementation on FPGA chips," *IEEE the 3rd Intl. Conf. NEWCAS*, pp. 327-330, 2005.

[7] SU, C.L., JEN, C.W., "Motion estimation using online arithmetic," Proc. of IEEE Intl. Symp. on Circuits and Systems, vol. 1, pp. 683-686, 2000

[8] C. Wei, M.Z. Gang, "A Novel SAD Computing Hardware Architecture for Variable-size Block Motion Estimation and Its Implementation with FPGA," *on Proc. 5th Intl. conf. on ASIC*, vol.2, pp. 950-953, Oct. 2003.

[9] Y. Kamaci, N. Altunbasak, "Performance comparison of the emerging H.264 video coding standard with the existing standards", *ICME'03*, vol. 1, pp.345-348, July 2003

[10] J. Villalba, J. Hormigo, J.M. Prades, E.L. Zapata, "On-line multioperand addition based on on-line full adders," *IEEE Intl. Conf. on App. Specific systems*, pp. 322-327, July 2005.

[11] S. Wong, S. Vassiliadis, S. Cotofana, "A Sum of absolute differences implementation in FPGA hardware," *on Proc. 28th Euromico Conf.*, pp. 183-188, Sept. 2002.

[12] J. Olivares, J. Hormigo, "Minimum Sum of Absolute Differences implementation in a single FPGA device," *Prof. FPL.*, pp. 986-990, 2004.

[13] H. Loukil, F. Ghozzi, A. Samet, "Hardware Implementation of Block Matching Algorithm with FPGA Technology," *IEEE 16th intl. conf. on microelectronics*, pp. 542-546, 2004.

[14] Stephan Wong, Bastiaan Stougie, Sorin Cotofana, "Alternatives in FPGA-based SAD Implementation," *Proc. FPL*, pp. 449-452, Dec. 2002.

[15] S. Lopez, F. Tobajas, A. Villar, V. de Armas, J. Lopez, R. Sarmiento, "Low cost efficient architecture for H.264 Motion estimation," *IEEE Intl. Symp. On Circuits and Systems*, vol. 1, pp. 412-415, May 2005.

[16] Siou-Shen Lin, Po-Chih Tseng, Liang-Gee Chen, "Low-power parallel tree architecture for full search block-matching motion estimation," *Proc. IEEE Intl Symp. Circuits and Systems*, vol. 2, pp. 313–316, May. 2004

[17] M.D. Ercegovac and T. Lang, "Digital Arithmetic," Morgan Kaufmann, San Francisco, 2004.