A VLSI Architecture for Variable Block Size Video Motion Estimation

Yap, S. Y., & McCanny, J. (2004). A VLSI Architecture for Variable Block Size Video Motion Estimation. *IEEE Transactions on Circuits and Systems II: Express Briefs*, *51(7)*(7), 384-389.
https://doi.org/10.1109/TCSII.2004.829555

Published in:
IEEE Transactions on Circuits and Systems II: Express Briefs

A VLSI Architecture for Variable Block Size Video Motion Estimation

Swee Yeow Yap and John V. McCanny, *Fellow, IEEE*

***Abstract*—With the advent of new video standards such as MPEG-4 part-10 and H.264/H.26L, demands for advanced video coding, particularly in the area of variable block size video motion estimation (VBSME), are increasing. In this paper, we propose a new one-dimensional (1-D) very large-scale integration architecture for full-search VBSME (FSVBSME). The VBS sum of absolute differences (SAD) computation is performed by re-using the results of smaller sub-block computations. These are distributed and combined by incorporating a shuffling mechanism within each processing element. Whereas a conventional 1-D architecture can process only one motion vector (MV), this new architecture can process up to 41 MV sub-blocks (within a macroblock) in the same number of clock cycles.**

***Index Terms*—Advanced video coding (AVC), sum of absolute difference (SAD), variable block size motion estimation (VBSME), very large-scale integration (VLSI) architecture.**

I. Introduction

THERE HAS BEEN a growing interest in the use of advanced video coding (AVC) for temporal prediction 1) in order to obtain higher compression ratios, and 2) to improve video quality in low-bit rate video systems. In particular, a video frame is segmented into smaller and variable block sizes (VBSs) to accommodate different changes in object movement within a video frame. One way to achieve this is by splitting the video frame using conventional fixed size macroblocks. Each macroblock is then further segmented into VBSs. A typical macroblock has a dimension of 16×16 pixels, with the smallest segmented block size (base block) being 4×4. In this case, a macroblock contains 16 base blocks corresponding to 16 motion vectors (MVs). Other VBSs correspond to derivatives of the base block. Newer video applications such as H.264 [1] include such schemes in their standard specifications.

The purpose of this paper is to present a new one-dimensional (1-D) very large-scale integration (VLSI) architecture for implementing full-search VBS video motion estimation (FSVBSME). An important aspect of this architecture is that it is able to perform a full motion search on integral numbers of 4×4 blocks sizes. As will be discussed, this requires the same number of clock cycles as previous 1-D architectures [2], [3]. However, this is capable of performing searches of up to 41 submotion displacements within a macroblock, as compared to one motion displacement, in previous 1-D systems. The architecture presented achieves this by incorporating multiplexers and latches plus a small additional amount of computational hardware in the processing element (PE) data path.

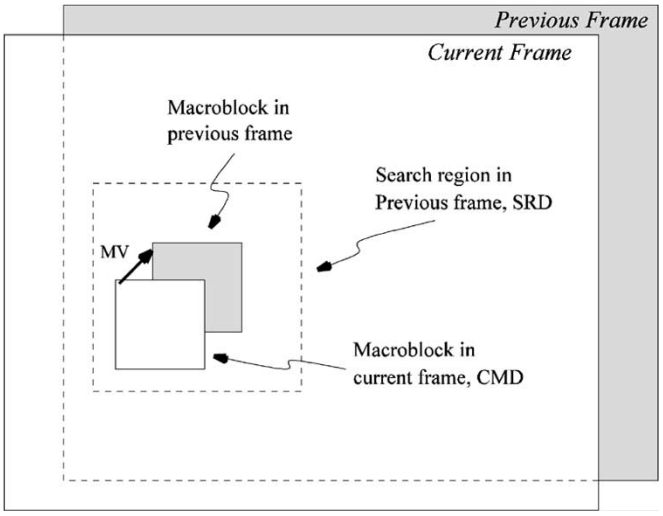

Fig. 1. Block matching.

The structure of the paper is as follows. Section II provides a brief overview of previous research on ME architectures and builds on this to develop a new architecture for a full search VBSME. The proposed architecture is then presented in more detail in Section III. The results of silicon design studies based on this are then given in Section IV, with the main conclusions presented in Section V.

II. Background

ME algorithms exploit the temporal redundancy of a segmented video sequence, as described by Jain and Jain [4]. Among all the estimation algorithms, the full-search block-matching algorithm has been shown to produce the best results in terms of finding displacement vectors (MVs), as depicted in Fig. 1. Such algorithms are implemented in two stages, namely the calculation of the sum of absolute differences (SAD) for each displacement vector, followed by methods for finding the smallest SAD values. This is summarized by (1) and (2)

$$\mathrm{SAD}(i,j) = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} |C(k,l) - R(i+k, j+1)| \quad (1)$$

$$\mathrm{SAD}_{\mathrm{MIN}} = \min(\mathrm{SAD}(i,j)). \quad (2)$$

Manuscript received August 28, 2003; revised November 25, 2003. This work was supported in part by Amphion Semiconductor Ltd., and in part by Queen's University Belfast under a Research Studentship. This paper was recommended by Associate Editor M. Flynn.

The authors are with the Institute of Electronics, Communications, and Information Technology, School of Electrical and Electronic Engineering, Queen's University of Belfast, Belfast BT9 5AH, U.K. (e-mail: s.y.yap@ee.qub.ac.uk; j.mccanny@ee.qub.ac.uk).

Digital Object Identifier 10.1109/TCSII.2004.829555

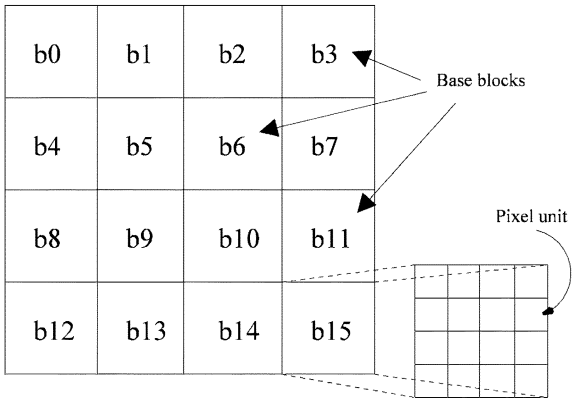

Fig. 2. Segmented macroblock.

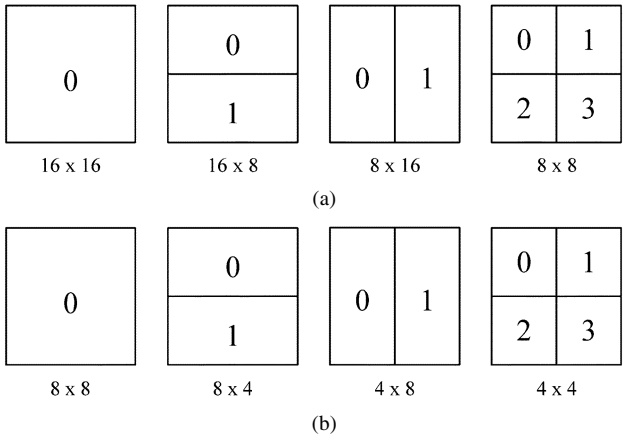

Fig. 3. (a) Macroblock mode. (b) 8 × 8-mode.

Here, $C\ (k,l)$ and $R(i+k, j+l)$ represent the current picture frame and search region's macro-block displacements, respectively.

The computational requirements for block matching are high and a real-time video application usually requires a direct mapped hardware architecture. Direct mapped architectures also have important advantages in terms of reduced power dissipation. Full-search algorithms, typically, can be implemented using regular 1-D or 2-D systolic or systolic-like architectures as described by Pirsch [5]. 1-D systems offer a number of attractive features over their full 2-D counterparts, in particular much less complex data scheduling and simpler structures. These architectures are therefore attractive for portable devices because of their lower silicon area and thus size. Kuhn [3] has also demonstrated that flexible 1-D systems can be used to implement other fast matching algorithms, such as a three-step search (TSS) and pel subsampling.

To date, conventional VLSI architectures for computing VBSME have been based on 2-D processor systems. For example, the architecture by described Vos [6] uses a 2-D array with appropriate through masking of PEs. However, this results in low processor utilization. Shen's architecture [7] uses a smaller 2-D array with partial-sum SAD calculations performed sequentially using the smallest block size, 8×8. However, these architectures do not incorporate the capability to process all the VBSs that the architecture presented in this paper does.

In AVC, a macroblock is further segmented with the smallest block size being 4×4, as shown in Fig. 2. This has two modes, the macroblock mode and the 8×8 mode, as illustrated in Fig. 3(a) and (b), respectively. VBSs must be accommodated, namely 4×4, 4×8, 8×4, 8×8, 16×8, 8×16, and 16×16. Referring to Fig. 3(b), it will be noted that there are four quarter-blocks in a macroblock, each of which contains nine block patterns i.e., a total of 36 block patterns. However, as will be observed in Fig. 3(a), each macroblock contains another nine block patterns, with four of the 8×8 blocks common with the equivalent 8×8 blocks in Fig. 3(b). Therefore, the total number of block patterns, to be processed is $36 + 9 - 4 = 41$ i.e., a total of 41 MVs.

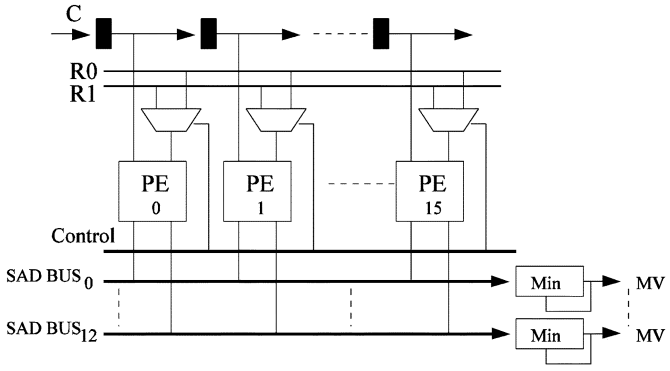

Fig. 4. One-dimensional array VBSME architecture.

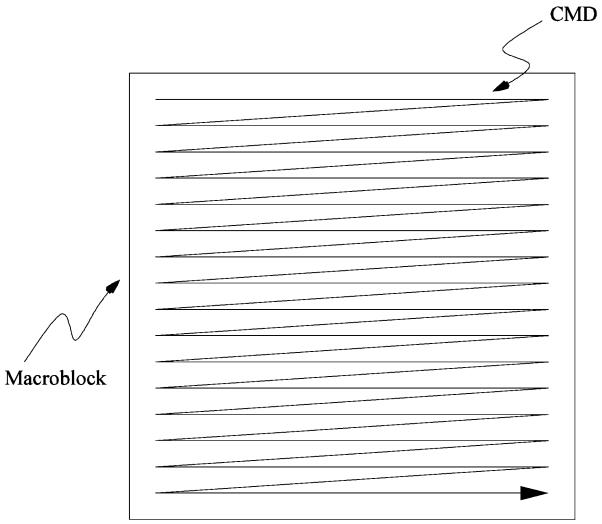

Fig. 5. CMD raster scan.

III. Proposed Architecture

The architecture presented in this paper is based on 1-D array processor, in this case containing 16 PEs, in general, N for an N × N macroblock. This is summarized in Fig. 4. A key aspect of the approach proposed is that it incorporates within the basic PE the means to accumulate the partial SAD values through shuffling. The scheduling of the current macroblock data (CMD) and search region data (SRD) is similar to a

TABLE I
DATA FLOW SCHEDULE

Clk	PE_0 (t-0)	PE_1 (t-1)		PE_{14} (t-14)	PE_{15} (t-15)
0	C(0,0)-R(0,0)	C(0,0)-R(1,0)		C(0,0)-R(14,0)	C(0,0)-R(15,0)
1	C(1,0)-R(1,0)	C(1,0)-R(2,0)		C(1,0)-R(15,0)	C(1,0)-R(16,0)
…	…	…	…	…	…
14	C(14,0)-R(14,0)	C(14,0)-R(15,0)		C(14,0)-R(28,0)	C(14,0)-R(29,0)
15	C(15,0)-R(15,0)	C(15,0)-R(16,0)		C(15,0)-R(29,0)	C(15,0)-R(30,0)
			…		
240	C(0,15)-R(0,15)	C(0,15)-R(1,15)		C(0,15)-R(14,15)	C(0,15)-R(15,15)
241	C(1,15)-R(1,15)	C(1,15)-R(2,15)		C(1,15)-R(15,15)	C(1,15)-R(16,15)
…	…	…	…	…	…
254	C(14,15)-R(14,15)	C(14,15)-R(15,15)		C(14,15)-R(28,15)	C(14,15)-R(29,15)
255	C(15,15)-R(15,15)	C(15,15)-R(16,15)		C(15,15)-R(29,15)	C(15,15)-R(30,15)

conventional 1-D architecture [2] with the CMD arranged in a raster scan sequence and the SRD arranged in a dual raster scan sequence, as shown in Fig. 5. Applying this approach to the macroblock shown in Fig. 2 results in 16 SADs being computed, each with block size 4×4. The stored SADs are then re-used to compute SAD values for other block sizes. This is done by shuffling and combining the computed sub-block SAD values appropriately to derive SADs for each of the other larger blocks sizes. For example, the results of two 4×4 sub-block computations can be combined to derive results for an 4×8 or 8×4 computation, and so on. This avoids the need to compute each of these from scratch and allows the overall computational requirements to be significantly reduced by avoiding the need to derive sub-block computation values that already have been established. As discussed below, this allows up to 41 VBS SAD values to be processed in a single processor.

The circuit shown in Fig. 4 operates by scheduling the CMD through a delay line and broadcasting two sets of SRD data on each clock cycle. The PEs accumulate the absolute difference (AD) between the CMD and SRD on every clock cycle, with the CMD and SRD data flow within each PE summarized in Table I, the CMD being denoted by $C\,(x, y)$, and the SRD data being denoted by $R(u, v)$.

If the pixel values in the CMD are labeled p_0 to p_{255} then it will be noted that the computation of the SAD value for block b0 involves pixels p_0 to p_3, p_{16} to p_{19}, p_{32} to p_{35} and p_{48} to p_{51}. In the case of block b1, this involves pixels p_4 to p_7, p_{20} to p_{23}, p_{36} to p_{39} and p_{52} to p_{55}, and so on (see Fig. 6).

These computations are performed using the internal PE circuitry, details of which are shown in Fig. 7. This uses a three-stage process, provides 100% PE efficiency and allows SAD value computation to be choreographed directly with the data flow within the image. The first stage in the PE contains hardware to derive absolute difference values between the CMD and the SRD. These values are then latched to a second stage where they are multiplexed appropriately and stored in one of eight registers. The function of the registers and Mux C is to ensure that once computations have been performed these are stored and fed back in the correct order to compute the overall AD

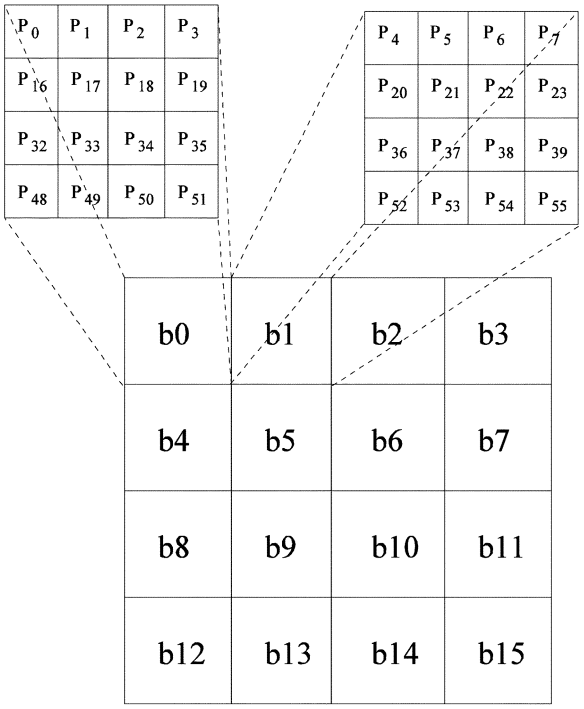

Fig. 6. CMD pixel values in base blocks b0 and b1.

values for each of the sub-blocks b_0 to b_{15}. For example, the AD value involving p_0 and the corresponding pixel in the SRD are fed back after the first cycle and accumulated with the AD value involving pixel p_1 and the corresponding SRD value. The result is then passed to register 0 in the first stage of the PE. This process then repeats for p_2 and p_3. In the following cycle, the AD value involving pixel p_4 and the corresponding SRD pixel is computed. However, these values correspond to sub-block b_1 rather than b_0 and thus these must be accumulated and stored separately. This is done by assigning these values to register 1. The process then repeats for p_5, p_6, and p_7. Having done so, the AD value derived from the next set of four pixels is then assigned to register 2, the next set of four to register 3, and so on, up to register 7. This data shuffling process then repeats

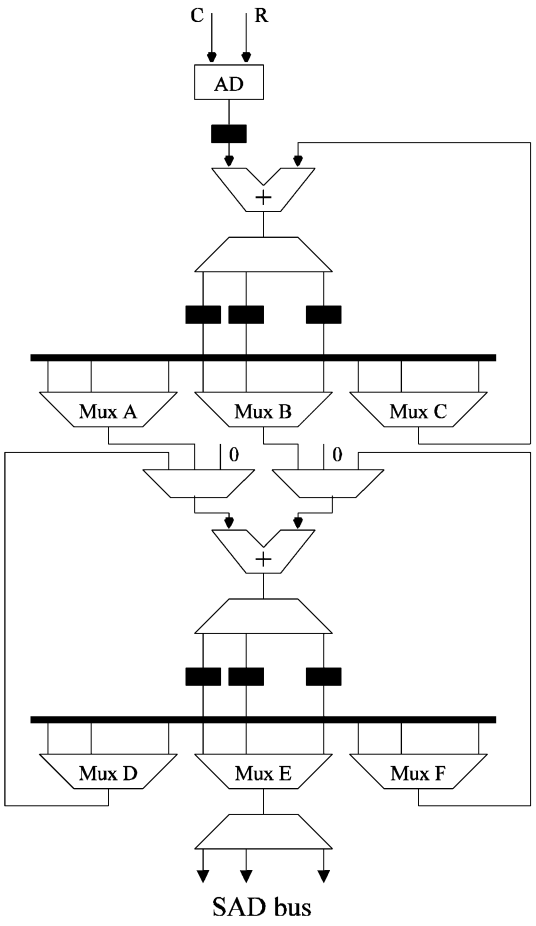

Fig. 7. PE.

for the next row of the image, with the AD values involving the first four pixels in the second row being accumulated and stored in register 0. The same applies to the next four pixels in this row, with these results being stored in register 1, and so on. This process then repeats with AD results for sub-blocks b_0 to b_7 being assigned to registers 0 to 7, respectively. With this approach, and ignoring processor latency, the base block SAD values {b0, b1, b2, b3, b4, b5, b6, b7}, (Fig. 2) then become available on clock cycles {51, 55, 59, 63, 115, 119, 123, 127}, respectively. Once computed, each of the values for these 4×4 sub-blocks is then immediately latched down to the second stage within the PE. This frees up the first stage and on successive cycles allows it to perform identical operations to derive SAD values for the 4×4 blocks in the second half of the array in Fig. 2, i.e., {b8, b9, b10, b11, b12, b13, b14, b15}. These results then become available on clock cycles {179, 183, 187, 191, 243, 247, 251, 255}, respectively.

The function of the second stage of the array is twofold. The first is simply to pass, on successive cycles, the values b0 to b15 downwards through the PE cell. The second is to combine these values appropriately to compute results for larger block sizes such as 8×4, 4×8 etc. For example, as discussed above and summarized in Table II, the SAD value for b0 becomes available on clock cycle 51 and that for b1 on clock cycle 55. These can therefore immediately be combined to derive a SAD value for MV 3. The same applies to MV 6 which can be computed on cycle 64 following the availability of the values for b2 and b3. This is done in a similar manner to stage 1 i.e., shuffling and combining results using a combination of multiplexing and adder circuitry, with results and intermediate results, in this case, being assigned to one of six registers, and so on.

TABLE II
BUS LINE ALLOCATION

Clk	Stage 2 reg	A	B	Stage 3 reg	Blk	Size	Vector	Bus
51	0	0	-	13	0	4x4	1	0
55	1	1	-	13	1	4x4	2	1
56	2	0	1	13	0,1	8x4	3	2
59	2	2	-	13	2	4x4	4	3
63	3	3	-	13	3	4x4	5	4
64	4	2	3	13	2,3	8x4	6	5
115	4	4	-	13	4	4x4	7	0
116	5	0	4	8	0,4	4x8	8	1
119	5	5	-	13	5	4x4	9	2
120	6	1	5	9	1,5	4x8	10	3
121	6	4	5	13	4,5	8x4	11	4
122	6	8	9	10	0,1,4,5	8x8	12	5
123	6	6	-	13	6	4x4	13	6
124	7	2	6	8	2,6	4x8	14	7
127	7	7	-	13	7	4x4	15	8
128	0	3	7	9	3,7	4x8	16	9
129	0	6	7	13	6,7	8x4	17	10
130	0	8	9	11	2,3,6,7	8x8	18	11
131	0	10	11	8	0,1,2,3,4,5,6,7	16x8	19	12
179	0	0	-	13	8	4x4	20	0
183	1	1	-	13	9	4x4	21	1
184	2	0	1	13	8,9	8x4	22	2
187	2	2	-	13	10	4x4	23	3
191	3	3	-	13	11	4x4	24	4
192	4	2	3	13	10,11	8x4	25	5
243	4	4	-	13	12	4x4	26	0
244	5	0	4	8	8,12	4x8	27	1
247	5	5	-	13	13	4x4	28	2
248	6	1	5	9	9,13	4x8	29	3
249	6	4	5	13	12,13	8x4	30	4
250	6	8	9	12	8,9,12,13	8x8	31	5
251	6	6	-	13	14	4x4	32	6
252	7	2	6	8	10,14	4x8	33	7
253	7	10	12	10	0,1,4,5,8,9,12,13	8x16	34	8
255	7	7	-	13	15	4x4	35	9
256	0	3	7	9	11,15	4x8	36	10
257	0	6	7	13	14,15	8x4	37	11
258	0	8	9	13	10,11,14,15	8x8	38	12
259	0	12	13	8	8,9,10,11,12,13,14,15	16x8	39	0
260	1	11	13	11	2,3,6,7,10,11,14,15	8x16	40	1
261	1	10	11	8	Full MB	16x16	41	2

The sequence of events is as follows. At the end of cycle 51, the b0 SAD value is moved from register 0 in stage 1 to the adder in stage 2 where it undergoes a null operation, i.e., added to zero indicated by the "0" value. This is then piped to stage 2 where it is stored in, in this case, register 13 and then output via bus 0. The b0 SAD value then follows four cycles later. The availability of both the b0 and b1 SAD values then means that it is then possible to derive values for the first 8×4 block by simply adding these values together. This is achieved on clock cycle 56 by using Mux A to output the value on register 0 and Mux B to output the value form register 1, and these are then added and output via bus number 2. This sequence of events then continues as summarized in Table II.

TABLE III
REGISTER SELECTION

Clk	Counter 6543210	Register	Clk	Counter 6543210	Register
0	0000000	0	64	1000000	4
1	0000001	0	64	1000001	4
2	0000010	0	66	1000010	4
3	0000011	0	67	1000011	4
4	0000100	1	68	1000100	5
5	0000101	1	69	1000101	5
6	0000110	1	70	1000110	5
7	0000111	1	71	1000111	5
8	0001000	2	72	1001000	6
9	0001001	2	73	1001001	6
10	0001010	2	74	1001010	6
11	0001011	2	75	1001011	6
12	0001100	3	76	1001100	7
13	0001101	3	77	1001101	7
14	0001110	3	78	1001110	7
15	0001111	3	79	1001111	7
16	0010000	0	:	:	:
:	:	:	111	1111111	7
:	:	:	:	:	:
:	:	:	255	1111111	7

The third stage in the PE has a similar function to the second stage, but in this case feeding back the SAD values stored in the stage 2 registers via Mux D and Mux F. As an example, consider clock cycle 122. In this case, register 8 contains the SAD value for b0 and b4 (block size 4×8) and register 9 contains the SAD value for b1 and b5 (also 4×8). These are then combined to derive results for the appropriate 8×8 block, with this being made available via bus 5. As before, these processes continue according to Table II with other 8×8 blocks and the 16×16 block derived in a similar manner. The net result is that by clock cycle 261 (256 cycles plus 5 cycles internal cell latency) all 41 candidate MVs are available from each PE.

Once all the values from an image block have been input then the data from a new block can immediately be input to each PE. This thus provides a continuous streaming process that directly synchronizes with a constant flow of image data and means that each PE is 100% utilized.

With 16 PEs working concurrently, the architecture described allows a total of 256 candidate MVs (16×16 search region) per sub-block to be processed in parallel with each PE producing all the information needed for a full search every 256 clock cycles—the same as existing architectures. However, in this case, this is done through the derivation of 41 MVs rather than one for each macroblock. Repeating this a further 16 times means that up to 4096 clock cycles are required to complete a full search.

The determination of the most appropriate MV is achieved using the buses shown in Fig. 4. These are used to perform simultaneous and adjacent comparison of SAD values. The best vectors from each bus are thus established and can then be supplied to appropriate post-processing circuitry. On the face of the control required for this architecture may appear to be quite complex. However, a detailed examination of the shuffling required shows that this is highly regular. In the case of Mux C (Fig. 7), which is used to reshuffle the accumulator registers, the data flow is as described in Table III. This scheduling can be implemented using a 7-bit modulo counter, which counts from 0 to 127. Three of the counter bits $\{[6], [3], [2]\}$ then control the top level registers used for accumulation and MUX C. The operation of the other multiplexers can then be programmed using look-up tables (LUTs). The same approach applies for all the other PEs. However, because the operation of each of these is delayed by one cycle with respect to one another, this can be implemented using a simple delay line.

TABLE IV
VBSME CHIP PERFORMANCE

Algorithm	Variable block size full search motion estimation
Number of PE	16 (1-D array)
Searching range	16 x 16
Block size	4x4, 4x8, 8x4, 8x8, 8x16, 16x8, 16x16
Technology	TSMC 130nm CMOS
Gates count	61k
Max frequency	294 MHz
Normalized dissipation	0.008 mW/MB/fps
Normalized speed	181 fps/CIF

IV. SILICON DESIGN

The architecture described has been captured using VERILOG in a manner that allows it to be easily ported to a range of silicon fabrication technologies. For the purpose of this research, we have used this to synthesize an ASIC demonstrator design based on the TSMC's 130 nm CMOS standard cell technology (1.2 V). The circuit design is based on a 16 PE 1-D array, has a search range of 16×16 $(-8, 7)$ and can handle the VBSs listed in Table IV. If a wider search range of 32×32 $(\{-16, 15\})$ is required, then a $4 \times$ search can readily be performed. The input wordlengths used were 8 bits. This is consistent with common video standards. The memory scheme used is similar to that described in [2] and [3].

An important aspect in terms of silicon area is the wordlength used at different stages in the PE. In the case of the first stage, this involves the computation of the AD between 8 bit pixels and thus can be accommodated with 8 bits. The second stage computation involves the accumulation 16 ADs, and thus the wordlengths grow to 12 bits. Up to 16 bits are then required in the third stage with the exact number varying from register to register depending on the number of computations required. In the most general case, it might be assumed that a total of 16 busses would be required to handle the SAD values emerging from each processor cell. However, it will be observed from Table II that such values only become available on specific cycles. This can therefore be exploited to reduce the number of busses needed. Specifically, it will be observed from Table II that in a 16 clock cycle, the worst-case scenario that occurs is when SAD values are output on 13 out of the 16 cycles i.e., between cycles 243 and 258. As a result, the number of busses needed can be reduced from 16 to 13. This is obviously beneficial in terms of reducing silicon area.

TABLE V
TRADEOFFS BETWEEN PROCESSING RATES AND NUMBER OF PEs

No. of PEs	Normalized Max. Speed	Max fps		
		QCIF	CIF	4CIF
1	11	44	11	2
2	22	88	22	5
4	45	180	45	11
8	90	360	90	22
16	181	724	181	45

The design created contains 61-K gates and can operate at frequencies of up to 294 MHz. Typical video applications cover a range of specifications in terms of resolution and frame rates. In order to determine the performance of the motion estimator described two normalized units have been derived. The first is the normalized power consumption, which is defined in terms of the power dissipation (in microwatts) per macroblock of data processed (MB) per frame per second (fps). The second determines the number of fps that can be processed at a specific resolution. For the circuit designed, these values were determined to be 0.008 mW/MB/fps and 181 fps/CIF, respectively.

For a typical video application, requiring QCIF at 30 fps, then, the circuit will dissipate 23.76 mW. Alternatively, 11.88 mW at 15 fps. Conversely, if operated at maximum clock speed then up to 181 fps can be processed in a system with CIF video resolution or 45 fps in a 4-CIF system. The focus to this point has been on a system in which full block searches are undertaken. This is typically required in applications requiring high-quality video e.g., digital TV/HDTV. For some applications, where very low power dissipation is a key requirement (e.g., for portable devices) then an important trade off that can be made, that significantly reduces computational complexity and thus power dissipation, is to use a reduced complexity search algorithm such as TSS. The basic circuit presented and its principles of operation can readily be adapted to incorporate this rather than a full search algorithm.

The presentation to this point has also focused on a full 16 (in general, N) PE linear array. The hardware complexity of such a system can, of course, be reduced by mapping the computational functions described onto a folded array. This provides a mechanism to provide tradeoffs between hardware complexity and performance, albeit with the expense of additional multiplexing and scheduling circuitry. For practical applications, this provides the means to minimize the hardware requirement needed to achieve a desired performance, for example for a standard video specification. Table V provides a guide to the performance achievable using a reduced number of processors. More specific figures require detailed chip designs to be undertaken.

A comparison between this circuit and previous VBSME circuits is presented in Table VI. An exact comparison is complicated by the fact that these have been implemented with different technologies and exhibit variations in their specifications and capabilities. Nevertheless, it will be noted that the design presented exhibits the highest level of flexibility in terms of block sizes catered for. It offers the highest clock rates and has a gate count which is roughly a quarter that of the most flexible alternative—that of Vos [6]. In addition, it should be pointed out that the flexibility of the architecture presented means that it is easy to reprogram the latches to cater for other block sizes, should these be needed in future video standards.

TABLE VI
COMPARISON WITH OTHER VBSME CHIP DESIGNS

	Vos '95[6]	Fujita '97[8]	Shen '01[7]	This work
PE number	16x16	---	64	16
Search range	16x16	---	16x16, 32x32	16x16, 32x32
Block size	2nx2n, n>=1, (masking)	8x8, 16x16	8x8, 16x16, 32x32	4x4, 4x8, 8x4, 8x8, 8x16, 16x8, 16x16
Process	0.6um	0.35um	0.6um	0.13um
Frequency	72MHz	15MHz	60MHz	294MHz
Gate count	263k	12k	67k	61k

V. CONCLUSION

In this paper, a new 1-D VLSI architecture for FSVBSME is presented. This architecture can process up to 41 variable block MVs in a macroblock in the same number of clock cycles as conventional 1-D architectures i.e., 256 (in general $N \times N$ cycles) A key aspect is the incorporation within each PE of mechanisms for shuffling, and combining the partial SAD values. This allows SADs for larger block sizes to be computed using the results derived for 4×4 blocks and avoids having to compute these from scratch. Design studies show that this is very suitable for the next generation of AVC. The concepts presented can be extended to half and quarter pixel ME for FSVBSME. They can also be extended to systems in which reduced complexity search algorithms (e.g., TSS) are employed, for example to reduce power dissipation. Research on this is currently underway and will be discussed in a future paper.

REFERENCES

[1] *Coding of Moving Pictures and Audio*, ISO/IEC Std. 14 496-10, 2002.
[2] K. M. Yang and L. Wu, "A family of VLSI designs for the motion compensation block-matching algorith," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 1317–1325, Oct. 1989.
[3] P. M. Kuhn, "Fast MPEG-4 motion estimation: Processor based and flexible VLSI implementations," *J. VLSI Signal Processing Syst. Signal, Image, Video Technol.*, vol. 23, pp. 67–92, Oct. 1999.
[4] J. R. Jain and A. K. Jain, "Displacement measurement and its application in interframe image coding," *IEEE Trans. Commun.*, vol. COM-29, pp. 1799–1808, Dec. 1981.
[5] P. Pirsch, "VLSI architectures for video compression—A survey," *Proc. IEEE*, vol. 83, pp. 220–246, Feb. 1995.
[6] L. de Vos and M. Schobinger, "VLSI architecture for a flexible block matching processor," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 5, pp. 417–428, Oct. 1995.
[7] J. F. Shen *et al.*, "A novel low-power full-search block-matching motion-estimation design for H.263+," *IEEE Trans. Circuits Syst. Video Technol.*, no. 7, pp. 890–897, July 2001.
[8] G. Fujita *et al.*, "A new motion estimation core dedicated to H.263 video coding," in *Proc. 1997 IEEE Int. Symp. Circuits Systems (ISCAS'97)*, vol. 2, 1997, pp. 1161–1164.