Cell Based Motion Estimators for Reconfigurable Platforms

Wim J. C. Melis[1], Kieron Turkington[1], Alexander Whitton[1],
Wayne Luk[2], Peter Y. K. Cheung[1] and Paul Metzgen[3]

[1]Department of Electrical & Electronic Engineering
Imperial College London, Exhibition Road
London SW7 2BT, England
[2]Department of Computing, Imperial College London
180 Queen's Gate, London SW7 2BZ, England
[3]Altera Europe, Holmers Farm Way, HP12 4XF High Wycombe, England

Abstract

Many scientific and engineering applications are data intensive. Their data-flow is often the performance limiting factor. When these applications are implemented on a reconfigurable system, the use of long routing resources usually limits the performance. This paper compares cellular design methods for reconfigurable devices which reduce the need for long paths. Two different approaches are presented: a heterogeneous approach with centralised control, and a homogeneous approach with distributed control, both performing full search motion vector estimation. Our heterogeneous approach, which is based around the NIOS II processor, is easy to develop; when extended with custom instructions, it can process a 16 by 16 pixel macroblock in about 23 msec and uses 5099 LEs (Logic Elements). Our homogeneous approach, which contains one or more copies of an optimised cell and a system control block, can process a macroblock with larger search area in 4.6 msec using only 950 LEs. Additionally, the homogeneous approach can exploit more parallelism and shows nearly linear scaling to cope with larger frame-sizes, search-area and frame-rate.

1 Introduction

The computational demand and data throughput of high performance video applications like full search motion vector estimation is a well known niche area for reconfigurable computing. Many successful implementations towards this type of devices have been presented. However, the continued drive towards increased frame size and frame rate requires ever faster and more accurate motion estimation designs to cope with new application domains such as HDTV. Increasing the clock speed could produce faster motion estimators. However, this increased computational load cannot easily be matched with an improved clock speed, especially when real time performance is a necessity. Therefore to further increase computational speed, more parallelism should be exploited, which consequently also leads to higher data-flow demands. Unfortunately, these larger data-flow demands also increase the requirements on the routing resources, which are known to be the performance limiting factor for many designs implemented on reconfigurable devices. Therefore, to overcome this performance bottleneck it would be necessary to keep the communication as local as possible, based on using small cells.

This paper contributes to design and evaluation of motion estimators, an important operation for many applications. In particular, it covers: (a) a heterogeneous approach based on a processor extended with various custom instructions (Section 3), (b) a homogeneous approach based on one or more copies of an optimised cell (Section 4), (c) implementation and evaluation of the above two approaches, showing for instance that the homogeneous design can achieve a 3-times speedup with an increase in area of 3.4 times (Section 5).

2 Overview and Related Work

It is well known that the underlying logic structure of reconfigurable architectures is formed by an array of cells. This structure is normally exploited during synthesis and placement. In addition to this exploitation, this paper uses cellular design techniques [1] to localise the needed routing resources from a design level. The division of a design into smaller cells each with a specific function creates a certain virtual boundary on the silicon and therefore shortens

the critical path. A global control unit and inter-cell communication might be needed, however, the design should be performed in such a way that neither are part of the critical path. As a result, this approach should be linearly scalable without affecting the clock-speed, but the overall performance increases due to the larger parallelism. The combination of cells can be based on a repetition of different or similar cells, leading to a heterogeneous or homogeneous approach.

Motion vector estimation searches for the best match of a reference block of pixels in the object frame [2]. Normally an SAD (Sum of Absolute Differences) algorithm is used to determine the similarity of the blocks. The resulting value for each position is then used to determine the optimal motion vector. Since a full search requires a large amount of data, alternative methods have been presented that reduce the amount of comparisons [3] and consequently also the data-flow. Most of these implementations target ASICs [4], which do not suffer as intensively from long routing delays in comparison to reconfigurable systems.

Alternative implementations based on the combination of a processing unit extended with custom instructions are e.g. presented by S. Khawam et al. [5]. This work augments a DSP with off-chip custom instructions implemented on an FPGA. They target a low power implementation and achieve a 23% timing improvement in comparison to hardwired ASIC implementations.

Implementing a motion vector estimation has two crucial parts to achieve high performance. The first is the level of data reuse exploited within the application, the second is the number of calculations needed to perform the comparison. S. Wong et al. [6] present a fully pipelined and highly parallel SAD implementation. The hardware unit performs a 16×1 SAD operation, by bit-inverting the smallest item and calculating the absolute value, which is then fed into an adder tree. This structure is then repeated 16 times and again combined using a final adder tree. Due to the large parallelism 512 data inputs are needed per cycle for full functionality. Hence, a large number of pins (over 4000) are needed, which requires the use of multiple devices. The homogeneous design presented in this paper will be compared with this approach in Section 5.3.

Gokhale et al. [7] present a hardware/software partitioning method for image based processing. They compare the use of two different processors extended with hardware custom instructions. One of these processors used is the ARM, the other is NIOS, however the custom instructions are implemented as a systolic array. Due to the intensive memory accesses within the application, the ARM based system has a significant advantage due to its larger number of memory ports.

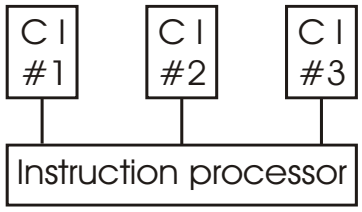


Figure 1. Structure of design with heterogeneous cells; CI denotes Custom Instruction

3 Heterogeneous Approach

The heterogeneous cellular architecture within this paper is based around the use of a soft-core processor. This soft processor allows for an easy implementation of data-flow to external memory, as well as the actual application. The processor model can easily be expanded with custom instructions to elevate the performance bottlenecks caused by critical instructions, resulting in a structure as shown in Figure 1. This figure shows the system based around the host processor extended with additional custom instruction blocks. On a system level the structure of Figure 1 could be seen as multiple heterogeneous cells, but it could also be seen as a single cell within a larger system.

The baseline implementation, used as a comparison for the other designs, only uses Altera's NIOS II as the instruction processor. This NIOS II processor runs the motion vector estimation sequentially and collects any required data from external memory. Due to this sequential execution the achievable performance is limited, which automatically keeps the data-flow requirements low. This data-flow requirement can easily be achieved using the off-chip memory, however this approach has a rather high power consumption due to the numerous off-chip accesses. To speed up this approach, the processor is extended with custom instructions to reduce the number of clock cycles required for key operations in the application. This should move the bottleneck operations towards hardware and therefore speed up the performance of the overall design. The individual custom instructions added are discussed in the next subsections.

3.1 Custom Absolute Difference Calculation

The first bottleneck within the single processor implementation is due to the actual Absolute Difference (AD) calculation. Therefore a hardware AD unit is added as a custom instruction. This allowed for the AD to be calculated in a single cycle and therefore improves performance. Consequently, this also increases the data flow requirements and necessitates on-chip buffering to exploit data reuse.

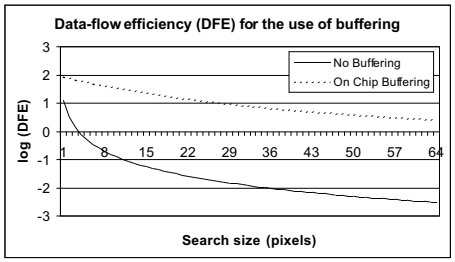


Figure 2. Data-Flow Efficiency of on chip buffering (macroblock of 16 by 16)

3.2 Custom Cell for Data Buffering

Motion vector estimation inherently has a large capability of data reuse, which is exploited in most implementations. To judge the efficiency of this buffering a DFE (Data-Flow Efficiency) parameter is introduced. This parameter as defined in Equation (1) describes the reuse of data on-chip. In the ideal case, this should be 100 %, which means that all the buffered data are reused before being discarded. Figure 2 shows the buffering efficiency of this approach in comparison with the approach without buffering. The graph clearly indicates the increased efficiency due to buffering when the search size increases.

$$DFE = \frac{\text{Theoretical minimum reads from memory}}{\text{Actual reads made by system}} \times 100\% \quad (1)$$

The custom instruction for data buffering is made autonomous [8] to distribute the control within the system. This means that the actual address calculation is not performed by the processor, however the processor still signals the need for read and write operations. To buffer the data, two sets of swinging buffers are used, each set called Autonomously Addressed Caches (AAC). One set is used to store the macroblock, whereas the other is used to store the reference image. The swinging buffer principle is used to allow appending of new data into one of the buffers while using data from the other buffer for the SAD calculations.

In addition to the reduced number of instructions for the processor, the buffering allows data to be accessed in less clock cycles when available on chip. This leads to an additional performance benefit. However, the central controlling NIOS II processor still forms the bottleneck to the overall performance of the system, therefore another custom instruction is added.

3.3 Custom Cell for Off-chip Memory Access

Since using a custom instruction to perform the on chip buffering is beneficial, the system is extended with another address calculation unit to perform the accesses to external memory. At this point all address calculations are performed outside the processor. This approach is similar to a DSP system with dedicated address generation unit, where the address calculation is often initiated from the main processor.

The addition of this unit leads to another performance improvement, similar to the one achieved with the previous custom instruction. The improvement is due to the address values now being calculated in one cycle and the possibility of removing a set of loops and conditional statements from the processor's program.

3.4 Further Extension of the System

At this stage, several tasks have been allocated to different hardware cells and the processor mainly serves as a controller. However the performance of the system is still determined by this central controlling unit. This could be overcome by generating additional custom instruction units, which eventually bypass the processor and communicate among one another as in the homogeneous approach. Alternatively, multiple copies of these cells based around the NIOS II could be combined, with appropriate task splitting.

4 Homogeneous Approach

The approach presented in this section eliminates the need for an overall controlling unit that drives all functionality by using autonomous blocks. These blocks are then combined into a cell which operates on a particular area of the image. This cell determines the optimum vector within its specific region of the search area. Depending on the used number of cells, more or less parallelism can be achieved, which requires the system to be flexible towards data buffering. In comparison to the design in Section 3, which contains the Altera specific NIOS II processor, this design is platform independent.

4.1 Data splitting for Buffering

For a system with n cells, the search area is split into n equal size areas as shown in Figure 3. Each cell then buffers the area allocated for its processing. In this particular implementation the splitting is performed column wise, and the vector calculation moved down the column before being moved to the next column (Figure 4).

As shown in the previous section, buffering of data provides a large advantage. In this case the buffering consisted of three sets of image data. There is the macroblock, which is needed by each of the individual cells, and then there is the search area data. In this case the search area for both forward and backward vector estimation is stored. Since each

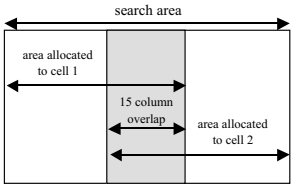


Figure 3. Division of search area for two cells

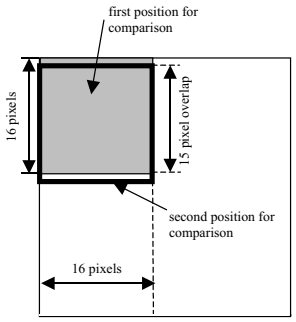


Figure 4. Movement of macroblock through search area

cell performs the estimation for a particular set of columns, a certain number of columns need to be buffered within the local memory. Figure 3 shows this splitting in the case of a system with two cells. The amount of data to be stored within one cell comprises of the number of columns exclusively assigned to this cell for SAD calculation. In addition another 15 more columns are needed to cover the full macroblock for the last column of the stored search area. These 15 columns are also the start location for another cell and are therefore duplicated.

In this implementation, the only function of the overall controller is to access the external memory, for which a simple address generation unit is needed. This unit accesses the memory and distributes the received data among the cells. The received data are sent along an on-chip bus which is shared by a number of cells. In addition to the actual data the column number of the provided data are also distributed. This allows each cell to select its data and perform appropriate local buffering.

4.2 Cell Structure

Each cell is composed of a number of basic components, which are made as autonomous as possible to overcome central control problems. A block diagram of a single cell is provided in Figure 5. This diagram shows the local control unit, two autonomous memory blocks [8] and a processing unit. Each of these components only have a small number of signals that connect them to one another. Only the controlling unit communicates with each of the other components since it is the access point to the other cells and the shared data bus. The buffers are used to buffer the respective image data. The processing unit is used to calculate the SAD as well as decide on the optimum vectors (forward and backward) for this search area. When the first cell finishes its search area it passes its optimum vector value to the second cell which then compares that with its own optimum. This continues by moving down the chain of cells until the last

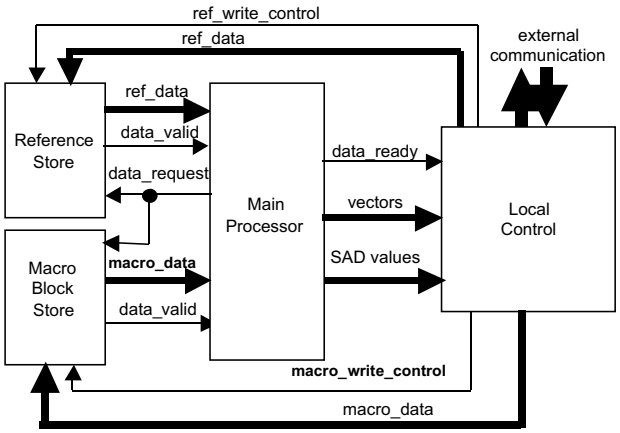


Figure 5. Block diagram of homogeneous cell

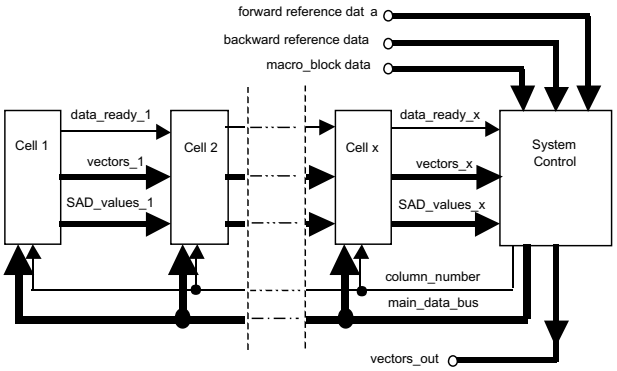


Figure 6. Combination of multiple homogeneous cells

cell sends the optimum motion vectors to the outside world. The resulting concatenation of cells is shown in Figure 6.

4.2.1 Memory Structure

The buffering in this design also makes use of the swinging buffer principle for both of the storage buffers [8]. This approach again allows for buffering new data while processing a current set of data. The actual number of simultaneous accesses to any of these buffers is limited to the number of ports. An increase in parallelism which also increases the data-flow requirements will therefore need to exploit the data reuse within the processing unit.

4.2.2 Processor Unit

The processing unit within the cell is responsible for the SAD calculation as well as determining the optimal vectors for the specific search area. This cell could make use of a sequential SAD calculation performing one SAD calcula-

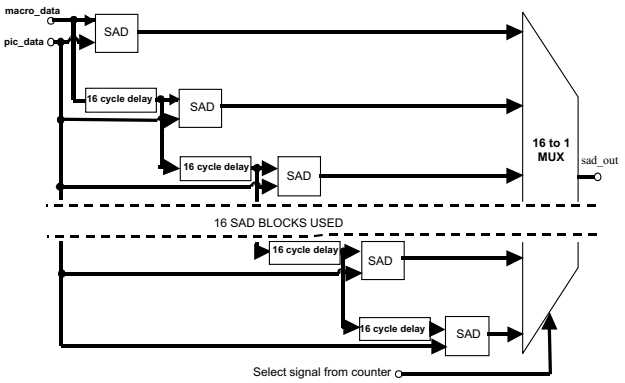


Figure 7. Buffering when performing a parallel SAD calculation

tion per cycle, or more parallelism can be exploited. The latter exploits data reuse within the actual processing unit.

To exploit this data reuse in the parallel unit, extra buffering is needed within the processing unit. The data reuse occurs in the macro block as well as the reference image. The macro block is reused for every new search calculation, whereas for the reference image only a limited area is reused. This design exploits only the macroblock reuse by adding shift registers to the design as shown in Figure 7. The shift registers act as 16 cycle delays on the macroblock data. This effectively results in a shift of one row between each of the SAD blocks. Each of the SAD units is however working on a different block comparison. When the full calculation of this block is finished, the SAD value is sent out. Since there is a delay of 16 cycles between each of the individual blocks, the results for the different blocks become available with the same delay. The register chain needs to be filled with one pixel per clock cycle, additionally, only two pixels per cycle (forward and backward reference) need to be supplied.

A cell for colour motion estimation (using luminance, blue and red chrominance data per pixel) is also developed along the same principals using the parallel SAD calculation by duplicating the appropriate blocks within the cell.

5 Results

Each of the systems is implemented in VHDL for the hardware, and C-code runs on the NIOS II processor. For each implementation the same Altera Stratix device (EP1S40) is targeted. The implemented motion vector estimation uses a macroblock size of 16 by 16. The normal implementations operate on gray-scale images, storing one byte per pixel, for the colour images, three bytes per pixel are used. Frame rate, frame size and search area are variable to investigate the scalability of the system.

5.1 Heterogeneous Approach

Results for the heterogeneous implementation are shown in Table 1. This table indicates that extending the baseline NIOS II system with custom instructions requires less time for the completion of one macroblock, but the necessary amount of resources increases. In addition to the number of LEs another 8 DSP units are used in each of the cases. The reduction in number of cycles for completion is related to the hardware performing most tasks in a single clock cycle in comparison to the processor. The latter needs several cycles for the same task. This also means that the program to be executed on the processor becomes shorter. Nevertheless, the performance of the overall system is still limited by the processor as a controller.

Increasing frame-rate, search area or frame size for this implementation would increase the time necessary to finish the calculation. Alternatively, a system with multiple NIOS II processors could be build. Since these processors would need to share the bus to external memory, their load in controlling this would increase. On the other hand, the parallelism would improve the workload on the computational part. The scalability of the system is considered by investigating the occupation of the bus to external memory. This indicated an occupation of $\pm 40\%$, which means that one extra NIOS II system can easily be added. When extending beyond this number of cells, the number of accesses to external memory cannot be fulfilled without extending the used number of ports.

The implemented heterogeneous system requires 22 msec in its fastest implementation to calculate the motion vector of a single macroblock. This value needs to be multiplied by the number of macroblocks per frame to give an idea of the time required to process one frame. For the NTSC and PAL standards, a frame rate of 30 or 25 frames per second is used, which means that every 33 or 40 msec a new frame is supplied. This clearly indicates that this implementation will not achieve real-time performance. It can also be noticed that the throughput per LE, which is independent of the search size, is rather low.

5.2 Homogeneous Approach

In comparison with the heterogeneous approach, the cells of this approach search in a larger search area and calculate the forward as well as backward motion vector with a search size of ± 24. The VHDL implementation of this design uses parameters to be adaptable to the number of cells, search area and other shared parameters. The resultant code is then mapped onto the Stratix device.

Table 2 and Table 3 show area and performance numbers for calculating the motion vectors of a luminance image using homogeneous cells. Table 2 provides data for the cell

Table 1. Performance and Area for heterogeneous implementation (search-size $\pm$ 4)

Implemented architecture	Max Clock Rate (MHz)	Area (# LE)	Memory (Bits)	Processing time per macroblock (msec)	Throughput per LE
NIOS II only	50	2411	46592	50	0.21
NIOS II + AD unit	50	2547	46592	39	0.25
NIOS II + AD + AAC	50	3605	101376	22	0.31
NIOS II + AD + AAC + External Memory Controller	50	5099	101376	23	0.21

Table 2. Homogeneous cellular design with sequential SAD calculation (luminance image)

Number of cells	Max clock rate (MHz)	Area (# LE)	Memory (Bits)	Processing time per macroblock (msec)	Throughput per LE
1	133	950	10752	4.6	143
2	129	2322	172032	2.3	117
3	134	3252	208896	1.5	128

Table 3. Homogeneous cellular approach with parallel SAD calculation (luminance image)

Number of cells	Max clock rate (MHz)	Area (# LE)	Memory (Bits)	Processing time per macroblock (msec)	Throughput per LE
1	125	3511	136842	0.401	444
2	127	7382	175392	0.201	421
3	125	10440	213936	0.139	431
4	123	13886	252480	0.108	417
5	121	18271	284880	0.084	407

with sequential SAD calculation, hence, it does not exploit data reuse within the processor block. As expected, for a same number of cells the required area is smaller, but the amount of time to calculate a macroblock is larger in comparison to the cell with parallel SAD (Table 3). The small cells do not need the extra buffering and hence also use less memory. As a result of the cell with sequential SAD unit being smaller, the maximum clock rate is also slightly higher.

When considering the area taken by each of the blocks within a cell, then the cell with sequential SAD calculation uses 84 LE for the SAD-block and a further 1066 for address generation and control. The balance between processing logic and control is however better when the parallel SAD-block is used, since this one is approximately 1520 LEs.

The results in Table 4 provide an idea of the amount of logic required to implement the cell with parallel SAD for a colour image. It can be noticed that area and memory requirements increase which is obviously due to each pixel now consisting of three bytes. In fact the system scales

Table 4. Homogeneous cellular approach with parallel SAD calculation (colour image)

Number of cells	Max clock rate (MHz)	Area (# LE)	Memory (Bits)	Processing time per macroblock (msec)	Throughput per LE
1	124	5637	273328	0.401	276
2	123	11193	350048	0.201	278
3	125	16672	426768	0.139	270
4	120	22114	503488	0.108	262
5	121	27747	578160	0.084	268

fairly conveniently towards colour images since the individual bytes can be treated separately and therefore the blocks within a cell simply need to be replicated.

In each of the cellular systems presented, the performance does not have large variations in relation to the number of cells. The small variations in maximum clock rate are actually related to the placement of the cell components, since the critical path is always within one of the cell boundaries, but varies when a different number of cells are used for the implementation. In summary this linear scalability promises easy expansion of the system to deal with larger frame and search sizes.

The relation between number of cells and the amount of time necessary to complete a macroblock is not completely linear either. This is due to the loading of the cells. In some cases it is namely impossible to balance the load of the cells. Hence, some cells require more cycles to finish in comparison to other cells. Therefore it is essential to preferably have a linear relation between the number of cells and the search size to be able to balance the cell's load.

This design is clearly capable of achieving real time performance, since the amount of time to calculate the motion vector of one macroblock is much smaller than the actual frame rate. In case of NTSC or PAL standard, the frame size is 352 by 240 or 352 by 288 respectively. This means that the largest number of macroblocks is 396. Hence, using 5 cells allows real time performance for NTSC or PAL.

The throughput per LE allows for a comparison of both methods, independent of their search size. The results clearly indicate a large difference between the heterogeneous and homogeneous approach. This difference becomes even larger when the parallel SAD is considered

for the homogeneous approach. If the amount of memory would be included in this measure, then similar trends can be noticed, but the difference factors are smaller.

The scalability of the system for a frame size of 720 by 576 and frame rate of 30 frames per second is also investigated. The search-size is still ± 24. The implementation of this design requires a total of 15 cells resulting in the need for 50991 LEs on a Stratix EP1S80. The maximum clock rate is 119 MHz and there is a need for slightly more input pixels per clock cycle, namely 11 input ports are required. This is due to the need for increased parallelism, which leads to a rise in data-flow requirements. As a result of this, more items per clock cycle are needed to fill the buffers.

5.3 Comparison To Other Implementations

The work presented in [6] uses a fully pipelined highly parallel SAD calculation, this results in a high clock rate of 380 MHz. This high parallelism comes at the cost of needing 512 input pixels each clock cycle and requiring 31060 Stratix LEs. In comparison, the homogeneous design presented in this paper, requires less external ports but exploits more on-chip memory for the data buffering. The design using 5 cells only requires 18271 LEs but needs more cycles to complete at a lower clock-speed. However, the clock rate could be improved if the design would be fully pipelined, which is not the case for the current implementation. This pipelining can be performed at no hardware cost since each LE contains a register, however there will be a slight increase in latency for completion. Alternatively the number of cells could be increased. Since this system scales linearly in several aspects a total of 10 cells could be used which would result in the need for approximately the same amount of logic as in [6]. The number of cycles required for completion using these 10 cells is close to half that of the case using 5 cells and therefore smaller than the design presented in [6]. Due to the increased number of cells and therefore processing speed and data-flow, an extra 2 ports to access reference data are required.

6 Conclusions and Future Work

This paper presents different cellular design alternatives for motion vector estimation. The NIOS II based system suffers from a performance which is about 5 times worse than the homogeneous single cell without parallelism. On the other hand, the NIOS II system achieves good performance in comparison to design effort and allows for easy design exploration. In particular, the custom instructions shows clearly which design blocks to speed up when higher performance is required. The use of NIOS II also benefits applications with large amounts of sequential execution and therefore less need for distributed control. On the other hand, the full hardware implementation with distributed control indicates an additional nearly linear scalability with an increase in search or frame size.

In addition, this paper has indicated that using cellular design techniques for reconfigurable designs has several benefits. Although the synthesis tools attempt to localise circuit parts, for a system using a global design technique synthesis becomes harder, whereas when the design is cellular it becomes more straightforward to exploit the features of the underlying architecture. Hence, local routing resources are exploited and performance suffers less from long routing delays.

Future work will focus on investigating several parameters in relation to these cellular architectures, e.g. trying to determine the optimum cell size as a function of the application requirements. This involves finding the optimum balance between the size of each of the autonomous subparts within a cell.

References

[1] M. Sipper, "Simple + parallel + local = cellular computing," in *Parallel Problem Solving from Nature PPSN V. 5th International Conference. Proceedings*, Berlin, 1998, vol. 1498 of *Lecture Notes in Computer Science*, pp. 653–662.

[2] B. G. Haskell, A. Puri, and A. N. Netravall, *Digital Video: An Introduction to MPEG-2*, New York: Chapman and Hall, 1996.

[3] M.-J. Chen, L.-G. Chen, and T.-D. Chiueh, "One-dimensional full search motion estimation algorithm for video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 4, no. 5, pp. 504–509, 1994.

[4] S. H. Nam, J. S. Baek, and M. K. Lee, "Flexible VLSI architecture of full search motion estimation for video applications," *IEEE Transactions on Consumer Electronics*, vol. 40, no. 2, pp. 176–184, 1994.

[5] S. Khawan, T. Arslan, and F. Westall, "Embedded reconfigurable array targeting motion estimation applications," in *IEEE International Symposium on Circuits and Systems (ISCAS) 2003*, 2003, vol. II, pp. 760–763.

[6] S. Wong, B. Stougie, and S. Cotofana, "An investigation of FPGA based hardware implementations," in *13th Annual Workshop on circuits, systems and signal processing*, Veldhoven, The Netherlands, 2002, pp. 568–573.

[7] M. Gokhale, J. Frigo, K. Mccabe, et al., "Experience with a hybrid processor: K-means clustering," *The Journal of Supercomputing*, vol. 26, pp. 131–148, 2003.

[8] W. J. C. Melis, P. Y. K. Cheung, and W. Luk, "Autonomous memory block for reconfigurable computing," in *International Symposium on Circuits and Systems, 2004. ISCAS 2004.*, Vancouver, BC, Canada, 2004, vol. II, pp. 581–584.