A Fine-grained Parallel Pipelined Karhunen-Loève Transform

Martin Fleury, Bob Self, and Andy Downton
University of Essex
Electronic Systems Engineering Dept.
Colchester, Essex, CO4 3SQ, UK
Tel.: +44 1206 872817
Fax.: +44 1026 872900
fleum@essex.ac.uk

Abstract

A high-performance Karhunen-Loève transform for multi-spectral imagery suitable for remote-sensing applications has been prototyped on a platform FPGA, by means of a PC-based development board. Performance estimates suggest that the design will already outperform implementation on a high-end microprocessor, given due attention to I/O (Input/Output). General conclusions are reached for the utility of this architecture for fine-grained parallel processing, when the design is extended to massively parallel processing.

1 Introduction

The discrete time Karhunen-Loève Transform (KLT) [2] is a staple image-processing algorithm, usually applied to a highly correlated image ensemble (set of related images). Unfortunately, the KLT kernel is data dependent with the result that, in general, no fast algorithm exists, as the transform coefficients are not known in advance. To improve processing speed the KLT might be parallelized. Alas, there is an impediment, because, though two of the three processing stages can be readily parallelized, due to a scaling discontinuity, the middle stage cannot. The final stage cannot begin before the transform coefficients are output from the middle stage. Therefore, a synchronous pipelined solution is sought in which the last (or first) two processing stages run in parallel with the other stage. A processing architecture with a high-data bandwidth is also required, as all images in an ensemble are input to find the transform kernel. As the two parallelizable stages involve spatially-localized processing, a fine-grained architecture is ideal.

In this paper, we prototype the KLT on an FPGA (Field Programmable Gate Array), and show that the current generation of FPGA's can effectively compete with a high-performance microprocessor, when processing image ensembles of size $6 \times 512 \times 512$ gray-scale pixels. Further, as the algorithm is deterministic and data-parallel, it can be readily scaled up to achieve arbitrary throughput in a massively parallel design. The arrival of platform FPGAs implies there is now a mass-produced device that supports fine-grained hardware parallelism. On these devices, the I/O bandwidth is an important feature. In the Virtex-I from Xilinx [12], there are 512 I/O pads, and in the later version, the Virtex-II there are 1,108 buffered I/O pins on the largest member of the

0-7695-1926-1/03/$17.00 (C) 2003 IEEE

family of ICs. Additionally, on the Virtex-II Pro [13] between zero and twenty-four Rocket serial transceivers with individual data rates of 3.125 Gbps can be added as cores. The Virtex-II Pro is, at the time of writing, at an advanced specification stage. In the work reported in this paper, the Virtex-I was employed.

A pipelined design allows data transfer to be overlapped with computation. Such a design assumes a continuous flow of image ensembles into the KLT engine. This is the form in which the KLT is likely to be applied in practice, for instance, if the algorithm is used to compress multiresolution satellite imagery, when there are usually large databanks of images available. For example, the Landsat satellite was capable of producing hundreds of image ensembles per day, each ensemble consisting of ten different spectral channels at 512×512 resolution. Note that for such imagery standard codecs such as JPEG (Joint Photographic Experts Group) are likely to lose high-frequency features such as building edges [9].

Section 2 of this paper describes the KLT algorithm, and shows how the order of computation is arranged for a parallel FPGA implementation. Section 3 introduces the FPGA development board and the constraint that it imposes on prototyping large-scale solutions. Section 4 is an analysis of the fine-grained architecture of the prototype KLT pipeline. Section 5 presents results, timings, and scaled performance if the design is extended from a single FPGA board to a massively parallel KLT engine. Section 6 considers forthcoming developments that will ease I/O on platform FPGAs. Finally, Section 7 draws some conclusions.

2. KLT algorithm

2.1. Overview

Unlike other well-known orthogonal transforms, such as the Fourier transform, the KLT is commonly applied to a collection or ensemble of images [1], rather than a single image. Each image can be viewed as a single realization of a stochastic process. In multi-spectral imagery, each image is the same scene viewed at different frequencies, but equally in microscopy, the same material at different resolutions is imaged, and in [4] a set of face images is transformed, each face being regarded as a single realization of a face-producing process.

In Fig. 1, the image intensities for the equivalent pixel in each image serve to form a multi-spectral data vector. The sample covariance matrix [3] is formed out of the zero-meaned data vectors by correlating each pixel in an image with the equivalent pixel in every image. It turns out that the eigenvectors of the covariance matrix are orthogonal and thus, when projected (through zero-meaning) onto the original basis, have the effect of an axis rotation. Moreover, the image data tends to be aligned within the reduced dimensional space formed by a few axes in the new basis. Thus, whereas in Fig. 1 just two images have been sampled, if a number of images were sampled then the number of eigenvector axes (marked _klt in the figure) would be the same as the number of images, and the clustering would be in the sub-space formed by a few but not all of the eigenvectors. It is also apparent from Fig. 1 that, if additive noise is present, the Signal-to-Noise Ratio (SNR) will be improved after transform (*i.e.* projection onto each in turn of the new axes) [6] in the higher-component (indexed by the eigenvalue magnitude) images, as no realignment takes place. (Noise-dominated image sets may be analyzed through the low-component images.)

2.2. Detailed description

Consider a sample set of real-valued images taken from an ensemble of images:

- Create vectors with the equivalent pixel taken from each of the images. If there are D images each of size $N \times M$, then form the

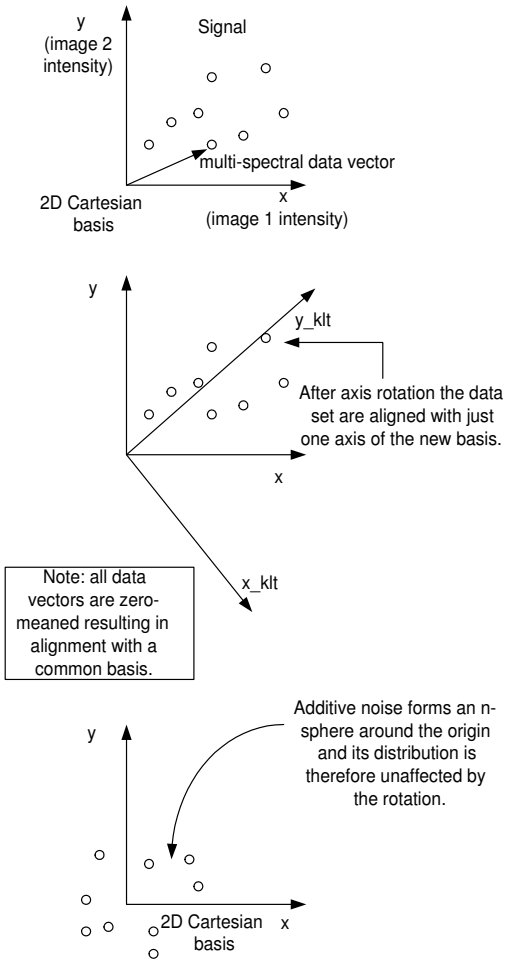

Figure 1. Effect of the KLT change of basis on signal dimensionality, and noise (additive)

column vectors $x_k = (x_{ij}^0, x_{ij}^1, \ldots, x_{ij}^{D-1})^T$ for $k = 0, 1, \ldots, MN-1, i = 0, 1, \ldots, M-1$ and $j = 1, 2, \ldots, N-1$.

- Calculate the sample mean vector:

$$m_x = \frac{1}{MN} \sum_{k=0}^{MN-1} x_k$$

- Use a computational formula to create the sample covariance matrix:

$$C_x = \frac{1}{MN} \sum_{k=0}^{MN-1} x_k x_k^T - m_x m_x^T,$$

which is appropriate if the image ensemble is formed by a stochastic process that is wide-sense stationary in time. It is important to note that C_x has rank D, which being the number of images in an ensemble is, in practice, usually below ten.

- Form the eigenvector set:

$$C_x u_k = \lambda_k u_k, \quad k = 0, 1, \ldots, D-1,$$

where $u_k$are the eigenvectors with associated eigenvalue set λ_k.

- Finally, the KLT kernel is a unitary matrix, V, whose columns, vectors u_k (arranged in descending order of eigenvalue amplitude), are used to transform each zero-meaned vector:

$$y_k = V^T(x_k - m_x)$$

The time complexity of the operations is analyzed as follows, where no distinction is made between a multiplication and an add operation:

- Form the mean vector with ***O(MND)*** element-wise operations. Calculate the set of outer products and sum, $\sum_{k=0}^{MN-1} x_k x_k^T$, in $O(MND^2)$.

- Form $m_x m_x^T$; subtract matrices to find C_x; and find the eigenvectors of C_x. The eigenvector calculation is $O(D^3)$. Convert the x_k to zero-mean form in $O(MND)$.

- Form the y_k by $O(MND^2)$ operations.

As the covariance matrix is generally too small to justify parallel decomposition in general or hardware implementation on an FPGA in particular, the total implementation complexity is $O(MND + MND^2)$.

2.3. Computational considerations

Notice that in Section 2 it has been tacitly assumed that the variance is calculated by a computational formula rather than by the direct calculation, $i.e.$

$$\frac{1}{MN}\sum_{k=0}^{MN-1}(x_k - m_x)^2$$

Using the direct calculation of the variance causes data zero meaning to be performed at an early stage of the calculation, and it is no longer necessary to form the square of the mean vector. In a sequential implementation, direct calculation would be the normal procedure. However, calculation of the square of the mean vector has trivial computational cost compared to calculation of the mean vector itself and the summation leading to the covariance matrix, and what is more these two operations can be performed in time parallel fashion. Calculating by the direct method requires the zero-mean operation to take place in strict sequence and hence no parallelism is possible.

However, zero-meaning the data at an early stage does reduce the data width on fine-grained implementations. This is largely a question of system gate usage on an FPGA.

There is another significant problem for an implementation by direct calculation on an FPGA, as calculation of the mean vector requires normalization, whereas this operation can be postponed if the computational formula is employed. Normalization for arbitrary image dimensions requires fixed-point operation, and hence all operations would take place in fixed-point form. A commitment to fixed-point operation at an early stage both complicates the implementation and results in increased word length, to minimize arithmetic errors. Increased word length impacts on the sizing of multiplier hardware and leads to a rapid increase in gate count. For example, a word length of 5 results in 680 gates usage for a multiplication on the Virtex-I, whereas a word length of 25 results in a gate usage of 16,120.

3. I/O constraint on the parallelization

The critical part of the KLT algorithm is the middle stage, eigenvector extraction from the image ensemble covariance matrix, as it is unsuitable for parallelization because of the low dimensionality of the matrix, and therefore forms a bottleneck in a pipeline because of the need to perform I/O between stages.

In the prototype, the middle stage was performed on a Pentium host connected through a PCI (Peripheral Component Interconnection) bus to the RC1000-PP system development board from Celoxica Inc. [8]. The on-board PCI bus was clocked at 33 MHz and was 32 bits wide, not the 66 MHz 64 bits width of the system bus on the PC host. The RC1000-PP board, Fig. 2, has four banks of 2 MB SRAM (Static RAM) memories, but these are accessible only four bytes individually (or one word) at a time. In [10], the bandwidth bottleneck on this and other boards is essentially recognized, as a task-parallel processing model is implemented for the RC1000-PP. These comments do not in any way constitute a criticism of the RC1000-PP, but merely reflect the constraint imposed when prototyping on such boards, which effectively color the results that can be obtained.

In fact, an on-chip PowerPC405 core (running at 350 MHz) would remove any I/O bottleneck between the stages, though the proposed version of the PowerPC core for the Virtex-II Pro supports fixed-point not floating-point operations. A further possibility is to include a specialist eigenvector engine, and indeed systolic designs have long existed for symmetric matrices, for example [7]. Naturally, this latter proposal would result in a customized board, and not a development board. The issue of streaming data on and off a development board still remains and this issue is returned to in Sections 4.3 & 6.

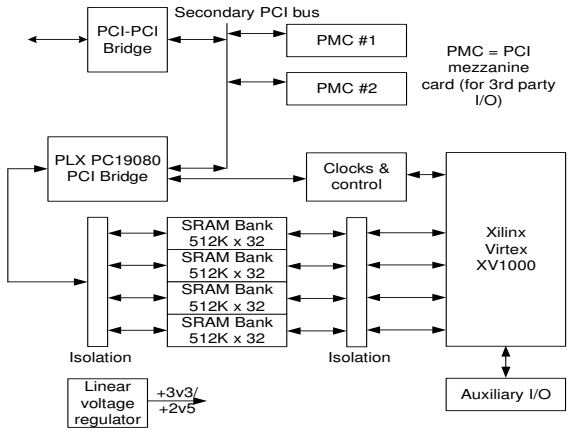

Figure 2. RC-1000PP board

4. KLT pipeline

The pipeline design is fully synchronous, corresponding to the systolic model of processing whereby data are 'pumped' across a processing array. The KLT pipeline comprised three distinct processing stages: covariance matrix formation, eigenvector calculation, and eigenspace mapping.

A small pilot implementation of the KLT was completed first using a 20×6 matrix with data and a set of proven results, corresponding to six images with just 20 data items in each. This enabled the results of the implementation to be verified to ensure correct working, after which gate usage for a much larger image size could be estimated simply by replicating the basic KLT building block. One of the reasons for the small size of the pilot implementation was the I/O bottleneck on the RC1000-PP development board already referred to in Section 2. Another reason is that performing place-and-route of FPGA components on a full-scale design, even with a high-performance host, took about one hour. Repeatedly changing a design and testing it can be a time-consuming activity. Results from scaling the design are reported in Section 5.2.

Parallelism arises in various ways in the design, as noted in Section 1. First and obviously the stages of the KLT pipeline can be partially overlapped in time. In fact, the second stage on a Virtex-II Pro could be elided with the third and final stage as it consumes little time, making for a reasonably balanced pipeline. Secondly there is internal parallelism in the first stage between the covariance and mean-vector sub-stages. Thirdly and most importantly, once the pilot design is replicated then there is data parallelism across the images' width.

4.1. Covariance matrix formation and eigenvector calculation

As already mentioned, the main issue in designing the first two stages of the pipeline, as shown in Fig. 3, was data management. In general, input is available from the memory banks in the form of 4×8-bit data samples, representing one-byte samples from each of the four memory banks. As noted in Section 2.3, the order of operations, from the two alternative ways of calculating the variance, is selected to avoid fixed-point calculations.

In the KLT prototype, the Mean Vector and Covariance summation sub-tasks processed data in rows of 6-byte blocks. This required data buffering and data alignment between memory I/O and processing. Once the test data had been copied from the Pentium host to the SRAM bank connected in turn to the FPGA then each 6-byte block took 1.5 clock cycles to pass to the parallel subtasks. Note carefully that the 1.5 cycles refers to a per-byte average, as the number of bytes in the image ensemble direction, six, is not a multiple of the word size. Synchronization between the parallel covariance and mean-vector sub-stages was by a 48-bit wide channel, from which data input was copied into two buffers for each of the sub-stages. Covariance calculations include a multiplication whereas calculation of the mean vector involves addition. As the Virtex family of FPGAs include fast add carry-chains in the vertical direction, there is a significant advantage given to addition and subtraction operations, which explains the relative time scaling.

Upon completion of processing of all rows, the first stage terminates causing a notification to be relayed to the host processor. The host processor then forms the covariance matrix and produces the eigenvectors, which are subsequently copied to an SRAM bank in order that the final hardware processing stage can be completed.

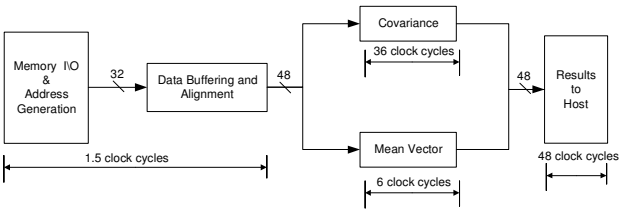

Figure 3. KLT stages 1 & 2 data flow

4.2. Eigenspace mapping

The layout of the eigenspace mapping stage is shown in Fig. 4. The computation performed by the pipeline consists of mean-normalization of vector elements, transformation of elements into eigenspace, and summation of elements in an accumulator. Source input and result output data pipeline stages manage the flow of data between external SRAM banks and the KLT pipeline processor. Normalization- and eigenvector data are stored in distributed RAM elements present on the FPGA selectRAM.

At the normalization sub-stage, incoming data is transformed by subtraction from a normalization vector stored in distributed RAM. The normalized data is then transformed into eigenspace by multiplication with eigenvector data, also stored in distributed RAM memory. Finally, the summation sub-stage accumulates the partial results generated by the multiplier into a result data element for storage in the result SRAM data bank. Counters, not shown in Fig. 4, are used to generate the distributed RAM lookup table addresses and sequencer control logic.

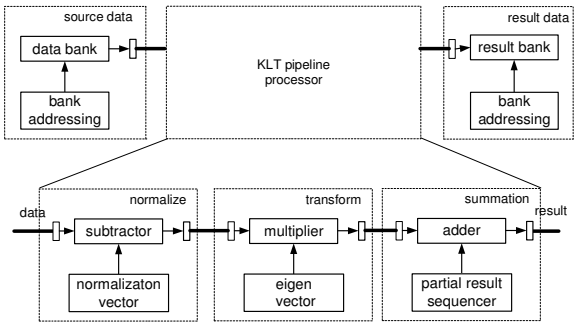

Figure 4. Normalization and eigenspace transformation stage

4.3. Data I/O

Data I/O was a relatively important part of the design, and hence is examined in detail in this Section. The application test data was arranged as a twenty-row by six-column matrix, with each data element being 8-bits wide. These data were mapped into the FPGA data space as thirty rows by 32-bits. The data were organized in this fashion in order to maximize data transfer between the FPGA and external SRAM memory banks. The memory banks can be configured for 8- or 32-bit transfers - clearly, 32-bit data accesses offered a bandwidth advantage over the 8-bit alternative.

The memory management task was to extract the required data element from the appropriate memory bank 32-bit word. Data bytes were accessed by reading SRAM bank data words into a register and writing data placed at the first-byte position into the pipeline data stream. Performing an 8-bit right shift for three subsequent clock cycles realigns higher-order bytes into the lower-byte position. An SRAM read operation was then generated in order to access the next data word.

The ability to read from the third-byte position was necessary when reading from odd (application data) rows. This was because the alignment of successive data rows in SRAM memory results in the first data element of odd rows being located at the third byte within a data word. Selecting between the first and third bytes was handled by

a multiplexor controlled by a signal produced by the SRAM Address Generator.

The heart of the Data Pump was the Address Generator. Data were managed in blocks of 12 byte-sized data elements. Each block corresponded to two rows of application data, where Row0 refers to the first row and Row1 is the second row.

The job of the Address Generator was to:

- Generate an SRAM address pointing to the base of a data block
- Generate an SRAM address offset in order to select the appropriate 4-byte word within the data block. Valid offsets are 0, 1, and 2.
- Schedule the writing of data read from the SRAM memory bank into the shift register.
- Schedule the reading of either the first or third data bytes from shift register by selecting the appropriate multiplexor port.

These requirements were achieved using the design shown in Fig. 6. The objective was to create a design, which was simple though sufficiently general to be easily changed as it evolved. The approach taken was to store the SRAM address offsets and various control signals as a lookup table in ROM. A counter provides for basic scheduling, while a 'row0\row1' register provides the ROM address offset that selects between the scheduling requirements of even (ROM addresses 0-5) and odd rows (ROM addresses 8-13).

The ROM table approach was particularly useful since it enabled scheduling of control functions to be easily changed by simply updating the contents of the ROM. In addition, the 16×1 ROM lookup tables were FPGA resource efficient to implement as each maps into a single CLB LUT. The 16×1ROM used in this design occupied a single CLB in a Virtex device.

5. Results

5.1. Optimization of the pilot design

Table 1 shows initial results from the small pilot study. The system gate count is a Xilinx metric based on the notional cost of implementation on an ASIC. Details of the accounting procedure for this metric are given in [11], and it should be noted that designs that include on-chip (FPGA) memory attract relatively large gate counts. There are two identical slices per CLB on a Virtex FPGA, and in general 'slices' are a more reliable metric. Together both stages occupy approximately 7% of the Virtex-I capacity. The clock times given, though taken from the output of the place-and-route tool (to position and connect the netlist components on the FPGA) rather than the implementation, were found to be very reliable.

The clock speed of the stage 3 design was lower than expected. From earlier timings made on arithmetic operators a clock rate in the range 20-26 MHz was thought to be a realistic target. It should be noted that others report similar clock speeds in this range for applications on the RC1000-PP board, for example in [10]. One problem was thought likely to be caused by the fact that the path through stage 3, which dominated the timings at this stage in the development, is composed solely of combinational logic. Additional pipeline registers to limit propagation delay were, therefore, included in the design. Registers can also remove positional dependencies between components.

In Table 2, the successive optimization steps are detailed. In this table, the statistics refer to both stages 1 and 3. Step 1 is the result of inserting registers between successive stages. Step 2 decouples selectRAM from the stage 3 multiplier and caused the main step up in performance. Replacing the in-house multiplier with a CoreGen Multiplier, which is soft IP (Intellectual Property), has the effect of removing any unnecessary

spatial separation between the multiplier's components. Finally, as the CoreGen Multiplier is already buffered, the now redundant registers were removed to produce the result in step 4.

Stage	Gate count	Number of slices	Clock speed (MHz)
1	18,288	561	33.37
3	13,402	397	13.66
1 & 3	30,567	909	13.28

Table 1. Initial place-and-route statistics

Step	Gate count	Number of slices	Clock speed (MHz)
1	31,087	986	17.511
2	31,287	1,001	26.773
3	31,632	972	28.373
4	31,432	953	30.418

Table 2. Optimization steps with place-and-route statistics

5.2. Scaling to full size

Table 3 records the results when the optimized pilot KLT pipeline, consisting of first and third stages was replicated, $n$times. In these measurements, no account is taken of coordination of data I/O, as only the first pipeline in every replication set is properly connected to memory banks. For the same reason, the clock frequency should only be taken to indicate an approximate speed of 20 MHz. The 16-replication build failed during place-and-route; the tool was unable to find sufficient paths to route all of the signals.

What the results do show is that the number of replications is limited to twelve. This represents an image of just 240 bytes, whereas a representative image is obviously either 128×128 or 512×512 pixels. To accommodate images of these dimensions, many times what can be accommodated in one pass on the Virtex, it is necessary to pass repeated streams of data many times through the KLT engine, accumulating results either on the host processor or on the FPGA, before the eigenvector calculation is performed.

When the mean vector and covariance totals are accumulated on the FPGA, then the data width will be larger, respectively increasing from 11 to 26 bits and 17 to 34 bits for a $6 \times 512 \times 512$ dataset. Table 4 records the results of accounting for the increase in dynamic range, establishing that the number of replications that can be supported, in fact, remains the same.

Replic-ations (n)	Gate count	Number of slices	Capa-city (%)	Clock speed (MHz)
1	31,608	1,016	8	27.243
8	204,942	7,582	62	23.974
10	253,096	9,313	76	21.560
12	302,738	11,211	91	19.529
16 (failed)	400,804	12,286	100	14.658

Table 3. FPGA usage from place-and-route statistics

5.3. Overall timing and scaling

Timings for a realistically-sized image ensemble are shown in Table 5. Taking absolute timings for the first stage for 12 iterations of the pilot design resulted in a time of 54 μs and for the third stage 38 μs. The first stage is now dominant, and scaling this result to the full image ensemble, requiring 1,093 passes of data, gives the result in Table 5. The accuracy of the timings was established by averaging over many thousands of tests. The MS Windows High-Resolution Timer API, which in turn examines a hardware register on the Pentium, was called to make the timings. What is also apparent from these timings is that two linear arrays of 512 FPGAs, representing the first and third processing stages, could accomplish processing of image ensembles at the rate of one per 200 μs, given appropriate image stream-

Replications (n)	Gate count	Number of slices	Capacity (%)	Clock speed (MHz)
1	39,705	1,077	9	25.295
8	270,126	8,065	66	22.444
10	334,306	9,920	80	22.077
12	400,334	11,921	97	20.115

Table 4. FPGA usage with increased data widths

ing hardware, and given an hierarchical result accumulation hardware to form the input to the second stage. Obviously, intermediate scalings between 1 and 1024 are also possible, as this is a fine-grained, data parallel algorithm.

Description	Timing (ms)
Pentium IV (1.7 GHz)	301.0
Virtex-1(20 MHz)	59.2

Table 5. KLT execution times for a $6 \times 512 \times 512$ pixel image ensemble

6. Discussion of I/O issues

The scaled timing for the FPGA is an order of magnitude faster than the Pentium IV, but this pre-supposes that there is a streaming I/O subsystem to sustain the Virtex data requirements. In the RC1000-PP board, access to memory is limited by the clock speed of the FPGA, whereas a stream sub-system can independently control memory access. In general, dual-ported selectRAM blocks act as a buffer, so that data can be accumulated before being read-out for consumption by the FPGA engine. This requires two clock domains. However, up to 12 parameterizable digital clock managers with maximum frequency of 400 MHz are available on the Virtex-II for this purpose.

In the case of the KLT, the Virtex-1 design operated at 20 MHz allowing six updates for reasonable DRAM access speeds of 120 MHz. In fact, there is a greater latency because the KLT pipeline takes about sixty 20 MHz clock cycles per stage, allowing about 360 DRAM accesses before a new set of data is required. The data width for one stage of the KLT is 6×12 bytes. Each of the twelve data streams requires a 48-bit width, typically formed from three 16-bit wide DRAM memory modules. A write of the original images from an external source and two reads for each of the KLT stages is required for each of the twelve data streams, making 36 accesses in all, certainly allowing a further set of writes if the processed image ensemble were to be written to sufficiently large memory modules. In fact, a streaming sub-system has been constructed for a different purpose [5] with four 8 Mb $\times 16$ bits access width for Virtex 1 modules.

On the Virtex-II a limitation is likely to be the availability of selectRAM blocks as the maximum port width per block is 32 bits, requiring two blocks per stream, and hence 24 blocks per stage, 48 in all. The XC2V1500 with 1.5 M system gates, has exactly 48 selectRAM blocks, with a maximum of 192 blocks on the largest device in the family. Though it is possible that additionally or instead distributed RAM could be used in combination with external buffering, this would present a typical developer's trade-off. If Rocket I/O transceivers were to be used, then the transceiver includes deserializer and decoder, while buffering would take place in a similar way to the use of selectRAM. An FPGA can deliver four eight bit or encoded 10 bits for transmission by a transceiver.

7. Conclusions

For multi-spectral and multi-resolution imagery, the Karhunen-Loève transform is an indispensable tool for which no 'fast' algorithm exists. Until the advent of platform FPGAs, and since the

demise of SIMD architectures, no cost-effective architecture has existed to perform parallel processing with the KLT. Indeed, in common with other algorithms more suitable for fine-grained processing, real-time processing in the recent past took place on medium- and coarse-grained parallel architectures. Unlike some of these algorithms, the KLT is still difficult to parallelize because of discontinuities in the scale and type of processing across the three stages that the algorithm can be divided into.

The paper reports that

- The Virtex family of platform FPGAs, running at low clock speeds, compete favorably with high-end general-purpose microprocessors, if sufficient data parallelism can be exploited.
- The addition of an embedded RISC core would significantly enhance the range of such algorithms that are realistic on a platform FPGA. The same may apply to other hybrid architectures that include a reconfigurable element.
- Even with a very large FPGA, there will be insufficient gates to allow one pass image processing.
- Therefore, if high throughput is to be accomplished, it will be necessary
  - to multiply pass image segments through such a device, requiring careful design of an I/O sub-system; and
  - use many FPGAs in a massively parallel pipelined design.

References

[1] P. A. Devijver and J. Kittler. *Pattern Recognition: A Statistical Approach*. Prentice-Hall, London, 1982.

[2] J. J. Gerbrands. On the relationship between SVD, KLT, and PCA. *Pattern Recognition*, 14(1-6):375–381, 1981.

[3] A. N. Jain. *Fundamentals of Digital Image Processing*. Prentice-Hall, London, 1989.

[4] M. Kirby and L. Sirovitch. Application of the Karhunen-Loève procedure for the characterization of the human face. *IEEE Transactions on Pattern Recognition and Machine Intelligence*, 12(1):103–108, 1990.

[5] D. Protnove, A. Sulejmanpasic, and C. Ebeling. An emulator for exploring RaPiD configurable computing architectures. In *Field-Programmable Logic and Applications, FPL2001*, pages 17–26. Springer, Berlin, 2001. LNCS 2147.

[6] P. J. Ready and P. A. Wintz. Information extraction, SNR improvement, and data compression in multispectral imagery. *IEEE Transactions on Communications*, 31(10):1123–1130, 1973.

[7] R. Schreiber. Solving eigenvalue and singular value problems on an undersized systolic array. *SIAM Journal of Scientific and Statistical Computing*, 7:441–451, 1986.

[8] C. Sweeney and B. Blyth. *RC1000-PP Hardware Reference Manual*. Celoxica Ltd., Didcot, UK, 2001.

[9] J. Vaisey, M. Barlaud, and M. Antonini. Multispectral image coding using lattice VQ and the Wavelet transform. In *IEEE International Conference on Image Processing*, 1998. Paper 712.

[10] M. Weinhardt and W. Luk. Task-parallel programming of reconfigurable systems. In *Field-Programmable Logic and Applications, FPL2001*, pages 172–181. Springer, Berlin, 2001. LNCS 2147.

[11] Xilinx Inc., San Jose, CA. *Gate Count Capacity Metrics for FPGA's Application note*, 1997.

[12] Xilinx Inc., San Jose, CA. *Virtex^{tm} 2.5 V Programmable Gate Arrays Datasheet*, 2001.

[13] Xilinx Inc., San Jose, CA. *Virtex-II Pro^{tm} Platform FPGA Handbook*, 2002.

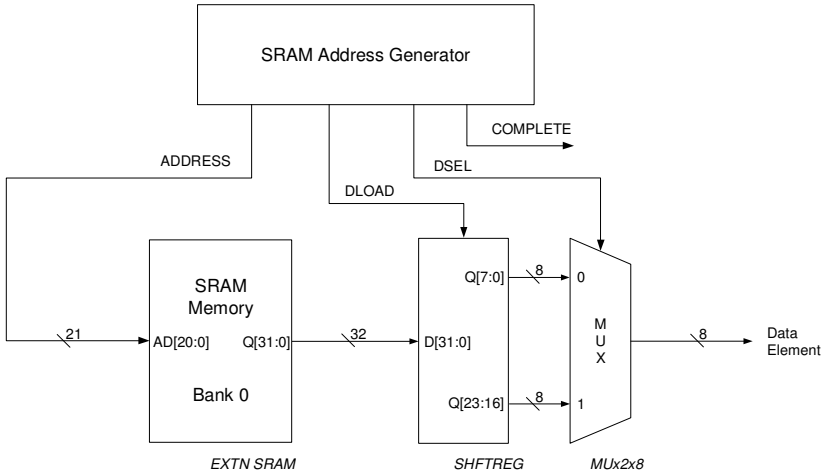

Figure 5. Data pump

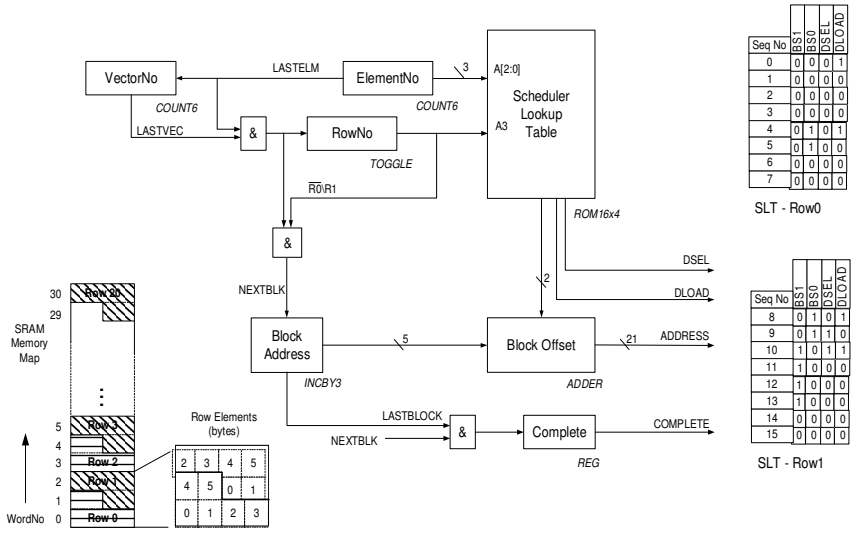

Figure 6. SRAM address generator

Acknowledgements

This work is being carried out with assistance from an EPSRC/DERA CASE studentship 9930329X.