Image Processing Applications Using a Novel Parallel Computing Machine Based on Reconfigurable Logic

N M Allinson, N J Howard, A R Kolcz and A M Tyrrell
Department of Electronics, University of York, York, YO1 5DD

Zelig is a 32 physical node fine-grained computer employing field-programmable gate arrays. Its application to the high speed implementation of various image pre-processing operations (in particular binary morphology) is described together with typical speed-up results.

Introduction

There has traditionally been a trade-off between flexibility and high performance in electronic systems. Custom hardware allows high performance at the cost of inflexibility and long development times and high cost. Software implementations are highly flexible but relatively slow. Field-Programmable Gate Arrays (FPGAs) permit the effective realisation of reconfigurable hardware [6]. FPGAs contain a two-dimensional array of cells that each implement a small amount of logic and the ability to configure the routing between them. Some FPGAs have PROM-like fuses to define this routing configuration, but many use fast static RAM and so are fully reconfigurable. New configurations which define the FPGA's functionality can be downloaded to the device in a few milliseconds. FPGAs effectively form an extra layer in the system between the hardware and the software [1].

FPGAs are not a replacement for all custom hardware, and their limitations should be respected. As a general rule of thumb, FPGAs can achieve about 10% of the functionality and 30% of the speed of custom VLSI devices of the same die size and fabrication technology. Suitable applications are those which display the following characteristics:–

- **Computationally irregular** – functions that can be implemented vary in shape and form. Otherwise, it is more efficient to use a coarser-grained, more dedicated hardware.
- **Non-mathematical** – functions that employ Boolean operators or small integer parameters are preferable. Mathematical operations consume a prohibitively large number of cells.
- **Repetitive** – such functions are preferred since the time to download new configurations to the FPGAs should be small when compared with the active processing time.
- **Easily segmentable** – to minimise the difficulties of partitioning algorithms between multiple FPGAs, those that can be fitted within a single FPGA are preferred.

The Zelig Architecture

Zelig was designed initially designed for the fast processing of a range of cellular automata (CA) algorithms. CA are relatively large arrays of homogeneous simple processes that are updated in discrete synchronous steps [1, 5]. Individual cell *rules* are based upon the current state of a cell and those of its local neighbourhood. From the application of such simple rules, complex local and global behaviour can result over many updating generations.

Such problems conform well with the list of preferred FPGA problems outlined above. A cell update rule which would require some 40 instructions on a conventional microprocessor can be processed in reconfigurable hardware in under 100 ns. A hundred copies (say) of this rule may be fitted into the reconfigurable logic and executed in parallel. A finer-grained task would require less logic; more copies can then be fitted into the hardware and so a higher degree of parallelism would result. In this way, billions of instructions can be executed per second.

The processing core of Zelig (Fig. 1) consists of 32 Xilinx XC3090-50PC84 FPGAs, each with an associated 64K by 8 static RAM. This core can be considered as a single logic surface with a 64K by 256 RAM for storing node data, and it should be regarded as one synchronous state machine. Each node update-rule copy is a physical node, and larger automata are created by time-multiplexing many virtual nodes through one physical node. The logic surface provides one physical axis for the computation space. Higher dimensions are implemented by time-multiplexing to create virtual axes. Many combinations of computation space are possible; for example:–

- 65,536 list of 256-bit vectors
- 512 x 512 plane of 8-bit nodes (formed from 16 swaths of 32 by 512 nodes)
- 128 x 65,536 plane of 2-bit nodes
- 256 x 256 x 256 cube of 1-bit nodes.

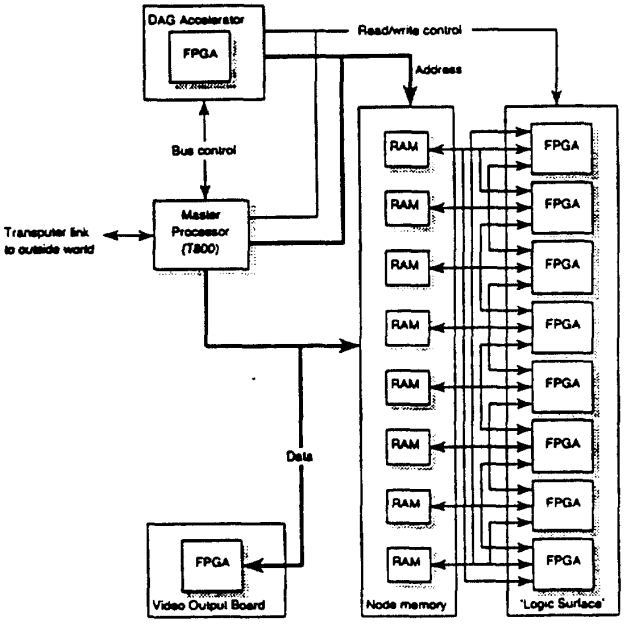

Fig. 1 Block Diagram of Zelig

Each FPGA can *see* the node data of the two FPGAs directly above and the two directly below. By suitable sequencing of the loading of node data into the FPGAs, any virtual node can *see* any other virtual node that shares its same physical node. The dimensionality and size of the computation space, and the maximum neighbourhood size of each virtual node, is determined by the sequencing of the addresses for reading from and writing to the FPGAs. The architecture is therefore capable of accommodating the extremes of small- or large-grain multidimensional neighbourhoods.

All data movements are generated and controlled by an Inmos T800 Transputer or by Data Address Generator (using further reconfigurable logic). Adjacent virtual nodes will typically possess a large number of common neighbours whose values are retained with the logic surface between successive node updates. This is achieved by shift registers internal to the FPGAs. The process logic may, in fact, need to be pipelined to reduce the processing time for complex tasks. The system's video board is capable of displaying a non-interlaced 512 x 512 image with three 8-bit pseudo-colour resolution. It based around the Inmos IMSG176 and is directly controlled by the master processor.

Software Environment

All software for Zelig is written in Occam and runs within the TDS3 Transputer development system. The two TDS3 Transputers (host and intermediate) communicate via a C004 crossbar switch to the master Transputer on Zelig. The host directs all processing. Together with global commands (e.g., update a specified frame, display a specified frame), the host can issue commands for the master to enable, disable and reconfigure FPGAs, and modify the contents of the node memory.

FPGA configurations are defined by ViewLogic schematic drawings with some lower level blocks defined by ABEL hardware description language descriptions. Placement and routing was achieved using Xilinx XACT software, though for some applications this was not entirely successful and manual placing had to be resorted to. Typical CLB utilisation were binary morphology (47%), grayscale morphology (61%) and Monte Carlo optimisation (64%).

Binary Image Morphology

As an example of an application domain, the implementation of binary morphological operations are presented in some detail [3]. The host-level morphology operations available include all the common low-level operations (MAX, MIN, TRANSLATE, DILATE, ERODE, OPEN, CLOSE, COPY, COMP, HIT, THIN, THICK); together with a number of more general commands (e.g., FILL – fills a specified portion of an image; PEEK – reads the image data at a specified location; VIDEO – displays a specified image). The Zelig node memory can store 64 512 x 512 binary images (the first image plane is used by some operations as an intermediate working register). Each FPGA has eight copies of the processing logic, so that the 32 FPGAs update 256 pixels in parallel.

The morphological operations require the use of a structuring element, which is defined as a nine-bit integer. The weightings for each element in the 3 x 3 set are:–

$$\begin{bmatrix} 1 & 2 & 4 \\ 8 & 16 & 32 \\ 64 & 128 & 256 \end{bmatrix}$$

For example, the structuring element:–

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \rightarrow 186$$

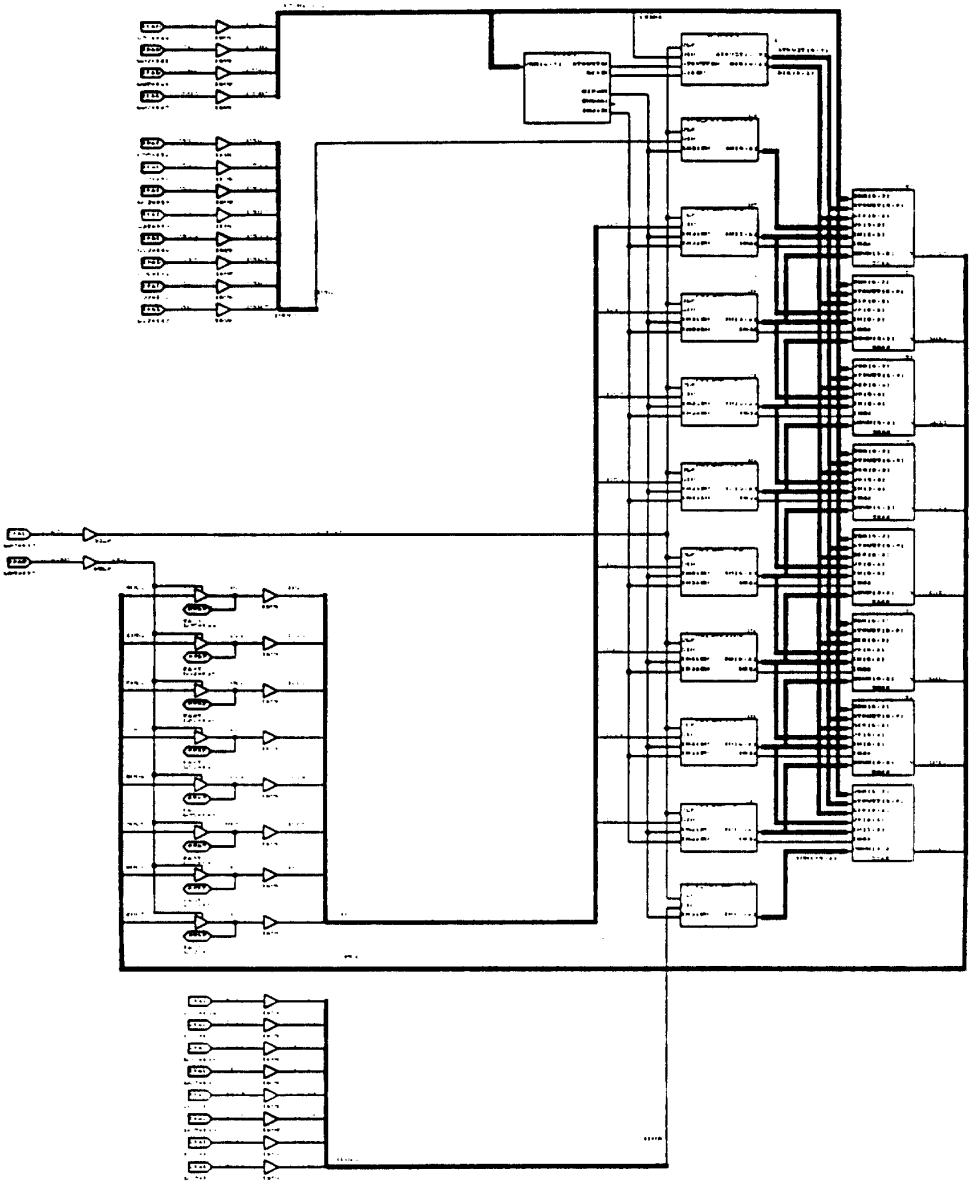

Fig.2 Top-Level Schematic for the BITMORFB FPGA Configuration

There are three FPGA designs – BITMORFA (for FPGA 0), BITMORFB (for FPGAs 1 to 30) and BITMORFC (for FPGA 31). The two end configurations differ due to the way data is loaded into the neighbouring blocks. The hierarchical design for BITMORFx is:–

BITMORFx
- ENDEC
- GLOBREG
- ENDNEIGHBOURS
- NEIGHBOURS
- CORE
  - MINKOWSKI
  - MINKOB

The top-level schematic for BITMORFB is shown in Fig. 2.

The NEIGHBOURS and ENDNEIGHBOURS blocks form a 3 x 10 pixel pipleline store for the current frame, around the eight pixels to be updated. The states of the nine neighbours are sent to the CORE block for each of these centre pixels. The ENDEC block performs the command address decoding to generate the various enable signals. The CORE block (Fig. 3) has sections to calculate the outputs for dilation, copy, maximum and translation. The final XOR inverts the output to convert dilation, copy and maximum to erosion, complement and minimum respectively. Generating the minimum from the maximum is achieved using De Morgan's Law and inverting the inputs as well. Erosion is generated from the dilation operation using the relationship $(A \otimes B)^c = A \oplus (B^T)^c$ (i.e., dilating the complemented structuring element rotated through 180° yields the complement of erosion). The five MINKOWSKI blocks, the MINKOB block and the OR gate in the CORE deal with dilation and complement of the erosion for minimal delay and ease of routing. The blocks are all five input, one output blocks that are implemented as distinct Xilinix CLBs.

Other IP Tasks and Performance

Other IP operations that have been implemented in Zelig include a set of grayscale morphological operators, neighbourhood filters (including rank-filtering) and local histogram modification. One meaningful indicator of performance is to compare the performance of Zelig with the corresponding pure-software implementation running on the T800-25 host processor. All software implementations have been optimised and make extensive use of look-up tables and minimise external memory accesses. Zelig operates at a modest memory cycle of 100 ns and uses slow-grade FPGAs. In terms of technology and cost, Zelig is approximately equivalent to 32 T800 Transputers. The highly parallel nature of the applications means that speedups of 32 would be expected. Table 1 gives some examples of speedups actually obtained.

Table 1 Example Speedup Results

Application	DAG Controlled
One-bit totalistic automata – 512 x 512	10,900
Binary erode/dilate – 512 x 512	7,750
Gray-scale erode/dilate – 512 x 512	630
Median filtering – 512 x 512 – 3 x 3 window	1,870
Local histogram equalisation – 512 x 512 – 5 x 5 window	2,890
Monte Carlo yield modelling	1,940

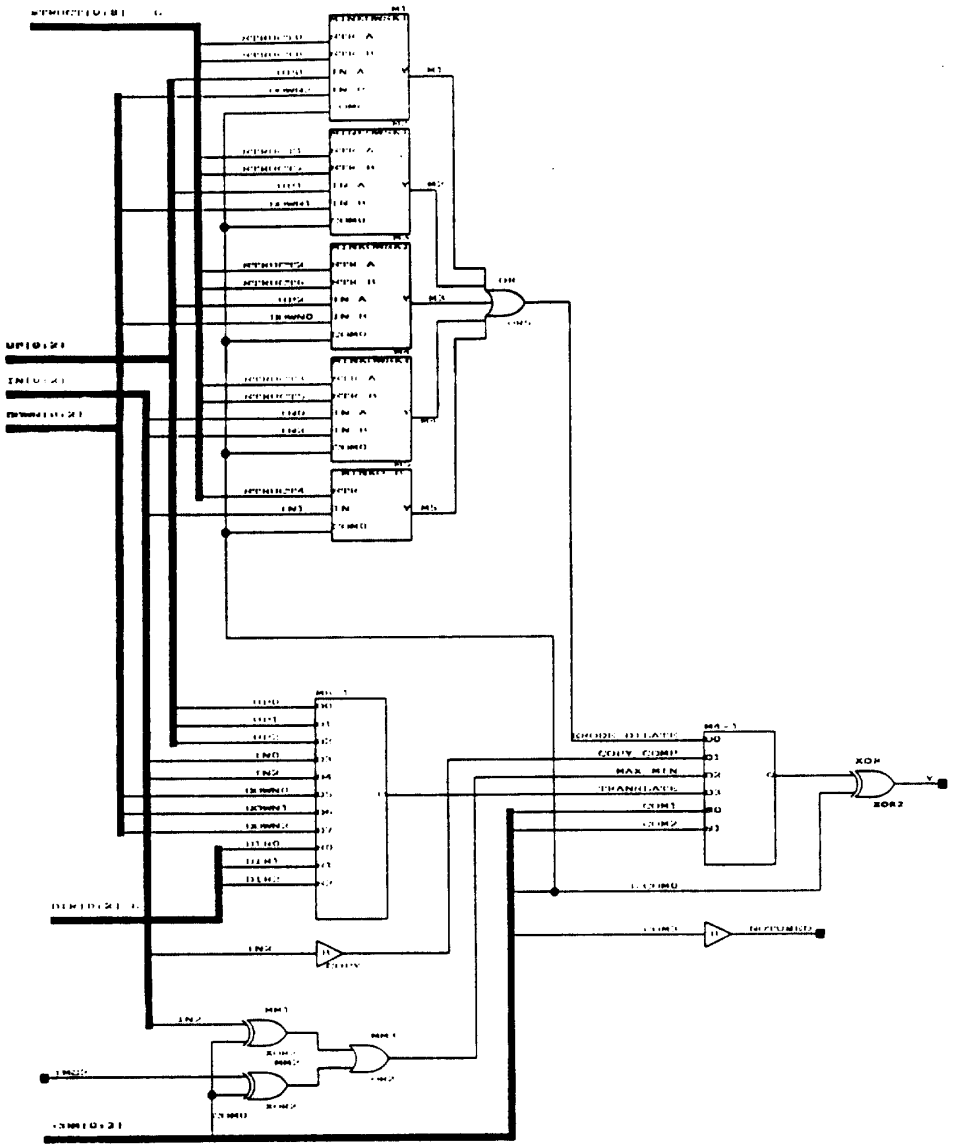

Fig.3 Top-Level Schematic for the CORE FPGA Configuration

The automata figure is given to show the performance level achievable for the application for which Zelig was especially designed. The Monte Carlo yield modelling is an example of a more complex application where the operations are repeated many thousands of times until convergence is reached [4].

Conclusions

The normal rigid architectures of custom hardware means that applications can either run very fast or not at all. Zelig offers the gradual degradation normally associated with software implementations:–

- Number of physical nodes $\propto$ size of reconfigurable logic
- Number of nodes $\propto^{-1}$ node data size
- Time overhead $\propto$ number of physical nodes
- Time overhead $\propto$ number of virtual nodes
- Time overhead $\propto$ neighbourhood size.

Zelig was primarily designed for investigations into probabilistic cellular automata, but its fine-grained architecture has proved appropriate for a range of other tasks including various image pre-processing applications. Future work will concentrate on other architectures that permit longer-range interconnections and improvements to the overall software environment. At present, Zelig is programmed by the separate writing of Occam code for the master Transputer and the designing and simulating of schematic diagrams for the FPGA designs. The ultimate aim is to be able to pass a single program to a compiler that identifies those segments of code to accelerate on FPGAs and then produce a netlist (and configuration bitstream) for the FPGAs and a controlling program for the master processor.

Acknowledgement

This work was funded by SERC/DRA project grant GR/F 92152 "The Development of a Fast Probabilistic Automata Machine".

References

[1] N M Allinson and M L Sales (1992), *CART - A cellular automata research tool*, Microprocessors and Microsystems, **16**, 403-415

[2] P M Athanas and H F Silverman (1993), *Processor reconfiguration through instruction-set metamorphosis*, IEEE Computer, **26**, 11-18

[3] R M Haralick and L G Shapiro (1992), *Computer and Robot Vision*, Vol. 1, Addison-Wesley, Mass., Chp. 5

[4] N J Howard, A M Tyrrell and N M Allinson (1994), *The yield enhancement of field-programmable gate arrays*, IEEE Trans. on VLSI Systems, **2**, 115-123

[5] T Toffoli and M Margolus (1987), *Cellular Automata Machines*, MIT Press, Mass.

[6] Xilinx Inc. (1991), *The Programmable Gate Array Data Book*, Xilinx Inc., San Jose, Cal.

*****