A Programmable-Logic Based Multiprocessor Engine for Real-Time Vision Preprocessing

Simon Freeman, Libor Spacek, Victor Callaghan, Paul Chernett[1]

Real-time vision is central to many embedded applications (e.g. vehicle guidance). It is a computationally intensive task well beyond current general purpose computing platforms such as PCs and workstations. Thus, most real time vision systems need special high performance computing platforms, commonly provided in the form of parallel processing engines or dedicated hardware.

It is well known that dedicated hardware has the potential to provide the fastest execution speeds but its rigidity often deters potential users. They prefer the economies of scale and flexibility which programmable systems offer. The proposed architecture uses new generations of re-programmable logic devices and modularised hardware, thereby gaining the performance advantage of hard-wired logic with the flexibility and associated economies of programmable systems. The architecture takes the form of an extensible processing hierarchy consisting of a set of tightly coupled parallel processors, each processing a portion of the image using a classic pipeline arrangement. A programmable image splitting (and reconstruction) engine feeds this array and offers the potential of further enhancing the performance of the engines by restructuring the pixel distribution (bit-shuffling) so as to match the requirements of the executing algorithms. The physical implementation will be based on a modularised bus system together with EPLD processing devices.

The project is currently in a simulation phase and this paper will report the predicted performance of low level vision functions running on this architecture.

Objectives of Work. The motivation for this work was twofold. Firstly, our links with Altera provided the stimulus to explore the potential of programmable logic to aid the development of programmable vision architectures and secondly we had an internal need to create a reprogrammable and scalable *real-time* image processing engine to support experimental *embedded* vision and system architecture work being conducted at the University of Essex. The initial goal of this work is to produce an embedded *pre-processor* which could perform the sort of "first steps" computations which most visual methods have in common such as convolution, segmentation, grouping, thinning, and morphology (e.g. to produce a general representation of the image such as David Marr's Primal Sketch (Marr 80)). Other goals are to facilitate experimentation in the area of (re)active vision and motion (e.g. programming a robot to move towards moving objects).

Programmable and scalable systems are desirable to researchers who constantly need to change the system characteristics as part of their experimentation procedures. They also frequently produce commercially enduring hardware and software architecture by enabling systems to grow with changing needs and adapt to ever-advancing technology.

Embedded Vision Processors. Vision is undoubtedly one of our most useful senses and it is hardly surprising that we find many machines which benefit from having an ability to view objects. Such machines, that have built into them an ability to automatically extract and act on image data, are referred to as embedded vision systems (e.g. industrial inspection, robotics). Real-time vision describes systems where the image data capture and processing are conducted in sufficiently short time to allow their results to be usefully acted on.

Emulating the performance of human vision is a significant challenge to the scientific community and whilst much commendable progress has been made, the level of vision performance we enjoy still remains well beyond current technological capabilities. Even simple vision systems can involve prohibitive amounts of computations for traditional architecture. For example, a 512x512 image has a quarter of a million pixels which, if operated on by a single basic imaging algorithm such as convolution, may involve multiplications or additions to at least each of the pixel's 8 neighbours. This results in a requirement for more than 7 million operations to be performed in less than 1/25th sec (i.e. if every data frame is to be processed) or over 170 mega-operations per second (MOPS). Typically, many operations may need to be performed on an image to deduce the required information so it is easy to see that massive computing power is required to implement useful vision engines. As the functionality of the machine is frequently dedicated to some given application, then so too is the vision domain often limited. Limiting the application domain often allows the vision problem to be considerably simplified, resulting in working applications. Thus, whilst there may be no general purpose

1. The authors are employed by the University of Essex, Colchester, England (email robots@essex.ac.uk)

solution, working systems have been built within limited application domains. In addition to designing architectures to satisfy such computational criteria, embedded systems frequently impose design constraints related to physical and electrical requirements of the embedded environment. As stated in the introduction, the initial thrust of this work is directed at the production of an embedded programmable *pre-processor* which could be used to perform the sort of "first steps" computations which most visual methods have in common. Longer term goals are to develop more fully integrated vision systems which could be used in embedded applications, such as small mobile robots of the type used at Essex.

Other Work. The proposed architecture makes use of some classic image processing architecture structures such as SIMD and pipelining. As has been noted by many researchers (Brumfitt 89), vision systems need deal with at least two distinct architectural needs, low-level pixel operations, such as edge detection which can exploit the intrinsic image parallelism and higher level processing such as image understanding, needed to produce intelligent systems. As feature extraction can be supported at either level it often falls into a slightly grey area between the two.

Architectures that deal with pixel level operations can be highly optimised to take advantage of the intrinsic image, neighbourhood and algorithm parallelism associated with pixel arrays. These architectures are sometimes referred to as vision *pre-processors* and often are constructed from hard-wired logic or ASIC devices such as the Plessey edge detection chip (Beedie 86). The architecture being described belongs to this class of processor.

Image understanding is clearly a complex task and the focus of much AI research work. However within constrained domains (e.g. traffic flow) workable systems can be produced (D'Agostino 92). An introduction to some of the problems associated with object recognition and image understanding can by found in texts such as Fairhurst (Fairhurst 88) or Sonka, Havac and Boyle (Sonka 93).

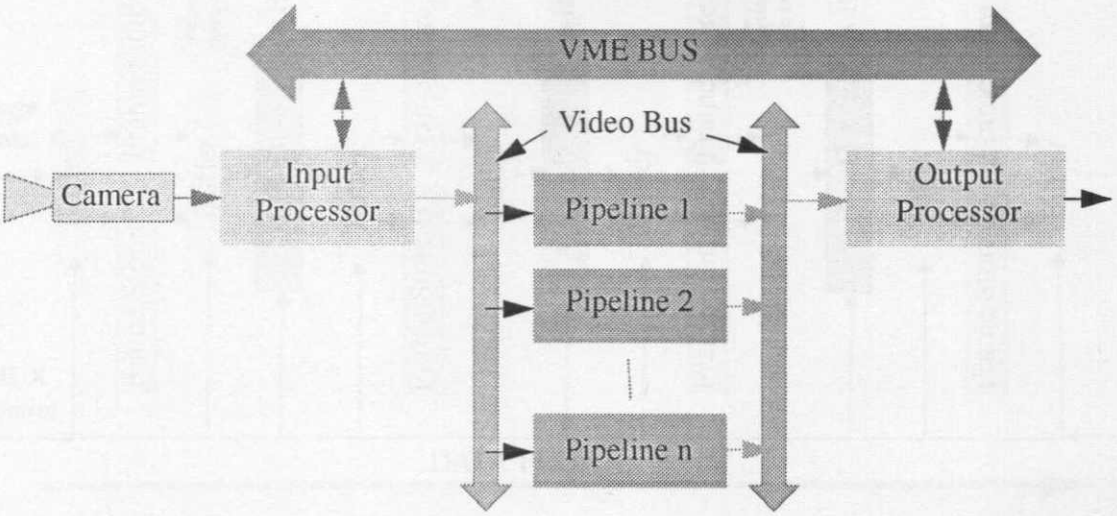
The coarse structure of the proposed engine is that of an SIMD machine (Fountain 86), the DAP being an example of this class of architecture. In the proposed architecture a programmable pipeline based on EPLD's forms the processor elements. The Cytocomputer (Brumfitt 89) is the best early example of a programmable pipeline but differs in that it uses LUTs as the programmable elements. Programming of the proposed architecture is accomplished by reprogramming the EPLDS (logically equivalent to exchanging boards or VLSI chips). A notable example of a system using this approach is the CRS 1000 (Alsford 85, D'Agostino 92) whose vision functionality may, in part, be customised via the use swapping of VLSI chips. Incidentally, in keeping with many other embedded vision systems, CRS use a VME bus in conjunction with a dedicated image bus. As was mentioned at the outset of this section processors optimised for pixel level processing are not always suitable for higher level processing and vice-versa. For instance, Intel i860, which has established itself as favourite choice amongst some vision practitioners (as it has many features that have been optimised for graphics work), provides extensive floating point operations but lacks parallel structure to execute such algorithms as convolve, dilate, neighbourhood operations (Yencharis 91). Thus, many commercial products will opt to combine a dedicated pixel processor for low end processing with RISC technology for the higher levels. For example, the Max860, is a combination of Datacube's (of Danvers, MA) pipeline image processor maxVideo MaxVideo200 (7000MOPS) and CSPI (of Billerica, MA) i860 based Supercard-2XL/VME (160MFLOPS).

There are many other architectures not mentioned in this short note. For information on these, reference to one of the many excellent texts in this area such as Kittler and Duff (Kittler 85) and Clark (Clark 91) or on-line archives[1] is recommended.

Architecture Overview. It is well known that dedicated hard-wired logic can provide much higher throughputs than conventional von-Neumann processors. Unfortunately, the complexities associated with realizing algorithms in hardware and the resultant rigid nature of such implementations has always acted as a significant deterrent to those considering adopting this approach. However, in recent years, advances in programmable logic device (PLD) technology have reached the point where they contain sufficient logic within a single device to allow them to implement many popular image operators (e.g. convolution, filters etc.). This architecture is an attempt at gaining the best of both of these approaches by

1. There are two public information servers located at Essex. PEIPA (Pilot European Image Processing Archive) is a dedicated image processing archive, largely funded by the British Machine Vision Association (BMVA), which will, eventually, form the basis of a pan-European archive in association with DG 13 of the EC. The usual anonymous FTP access is provided (peipa.essex.ac.uk or numeric address 155.245.115.161). ROBOTS is a small archive dedicated to storage of robotic related information. It can be accessed via Janet or Internet at ftp.essex.ac.uk. The robot archive is located in pub/robots.

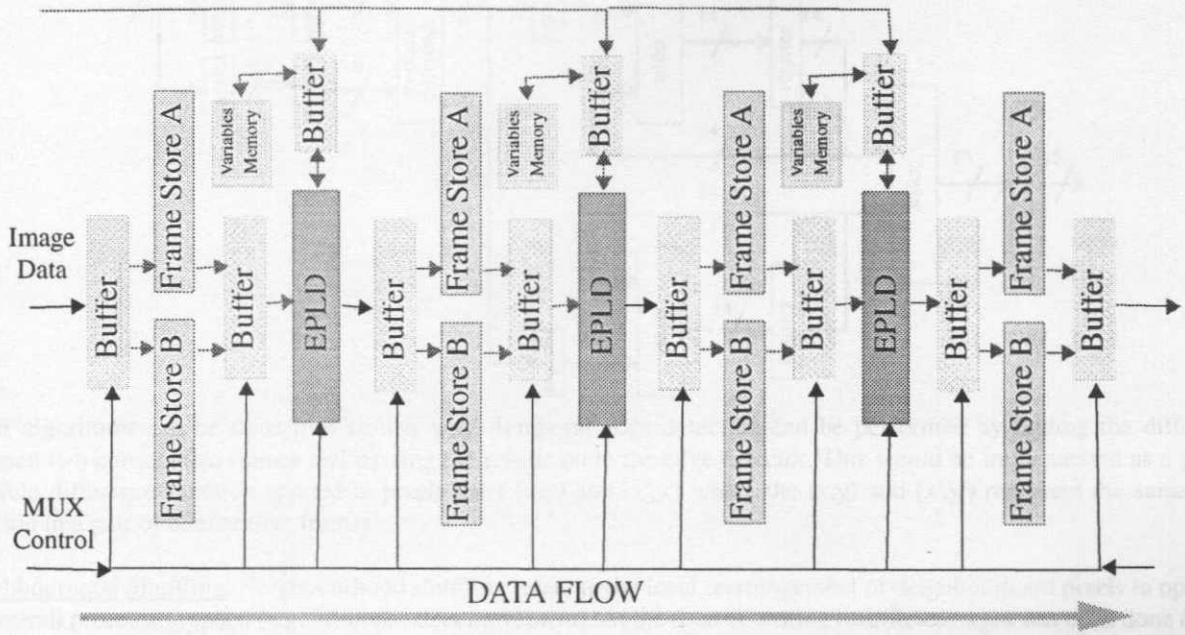
basing the design around PLDs so as to obtain the performance advantages of hard-wired logic with some of the flexibility of programmable systems. The basis of the architecture is an image processing engine which takes the form of a conventional SIMD parallel architecture, where each processor is a SIMD pipeline. Each SIMD engine processes a segment derived from a symmetrical partition of the image which provides the processor scalability. Images are normally held in simple arrays (Callaghan 93).



An unusual, if not novel, feature of the architecture is that the frame grabber can be programmed to output an unconventional pixel map. For example, pixels may be duplicated or swapped, so as to reduce the need to move them within the engine, thereby increasing the processing speed. This method is referred to as neighbourhood shuffling and is discussed in a following section. To accommodate the potential for structure growth the image processor image memory is 4 times that of the digitised picture. Finally, the image segments are delivered from the image processing engine to the output processor, where they are recombined to produce the whole image description.

Processor. In the planned implementation, each SIMD processor operates on segments of up to 128 x 128 pixels and is implemented using Altera EPLDs in a 3-line MISD pipeline. Currently, the EPM72256 is the largest member of the Altera PLD family. It offers some 5000 usable gates and initial estimates indicate two of these devices per pipeline processor will be needed to implement basic operators such as convolution. However, EPLDs have been announced that are more than 4 times the size of these devices. Whilst a number of alternative manufacturers produce devices that might have been considered for this design, Altera were considered most suitable due to their comparative size (large component count), quality of the design tools, and their support. In the prototype, the image resolution has been set to a maximum of 768 x 768 x 8bits. This figure was set by the selection of components making up the input and output processors (e.g. ADCs) which need to cope with a digitisation rate of 14.75 Mpixels/sec. The amount of memory for each pipeline image store has been set at 64Kbytes. This is 4 times the size of the image segment (128x128), a factor that was selected by considering the expansion effects of some algorithms.

As each SIMD processor operates on a segment of 128 x128 pixels the architecture is scalable by adding more processors to deal with larger pictures. For example, to process a 256 x 256 picture, 4 SIMD processors are needed in the imaging engine. Each EPLD will have programmed within it a computer vision algorithm. Algorithms can be changed by doing no more than re-programming EPLD chips. The EPLD's also control the memory reading and writing. Each EPLD has two framestores associated with it together with a small RAM for the storage of local variables. Using two frame stores between each EPLD pipeline section enables the memory to be accessed continually without having to wait for the other device.



Vision Algorithms. There has been a lot of work done over the years on edge detection and boundary grouping (or skeletonisation) algorithms. The amount of effort expended on these tasks indicates their continuing importance to Computer Vision. The higher level visual tasks, such as object recognition, depend crucially on the quality of the output from the early visual processing. It is not the purpose of this paper to discuss these tasks in much detail, rather to choose a simple but effective algorithm and to see whether it can be executed in real time.

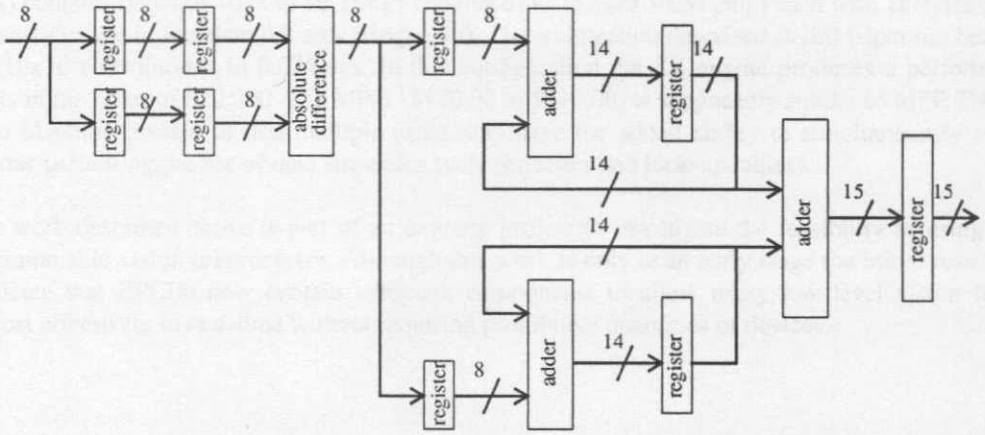
The edge detection algorithm used for this experiment is described in greater detail in (Spacek 93). It is partly inspired by earlier work on edge detection, particularly (Spacek 86) and (Fleck 92). The main idea is to compute the finite differences of symmetrically opposed pixels across an inter-pixel point, and to sum the squares of the differences in all possible directions. This fulfills admirably the criteria of a simple, fast, yet effective edge finder and, furthermore, is quite similar to the kind of locally parallel computations that are performed by the retina. The value returned at each point by this initial convolution (with a non-linear filter) is a good approximation to the magnitude of the image gradient. These values are typically normalised to lie in the range 0 to 1.

The next step, non-maximum suppression (or boundary thinning, skeletonisation), consists of identifying the boundary points as the ridge points on the gradient magnitude map. It appears that one of the best solutions to this problem is again a very simple one: mark all points which are a local maximum in at least two directions (assuming eight-connected grid). Typically one of these directions is normal to the extended local boundary.

The final (third) stage of the Spacek Boundary Detection algorithm consists of simple encoding to represent the connectivity of the boundary points, specifically the count (number) of adjacent boundary points. This operation is again local, and hence suitable for parallel execution. It is particularly convenient for detecting end-points and junction points of extended boundaries, which are perceptually significant, and hence very useful for tasks such as motion computation and autonomous vehicle guidance applications.

Translating Algorithms for EPLD Implementation. The usual way to implement any algorithm into hardware is to translate each part of the algorithm into a suitable mathematical function and then implement each of the functions as hardware. Algorithms such as convolution of an image area are easily put into hardware. The numbers used to convolve an image are usually small and for most of the time, not unique in the convolution square. This means the multipliers can be implemented as adders which are ideally suited to this architecture. An example algorithm has been developed to prove the theoretical system performance. This algorithm (described in the previous section) is an edge detection and thinning algorithm. The first part of the algorithm has already been implemented onto EPLD as follows. Within an area of image data that has equal and even pixel dimensions we find the sum of the absolute differences of all opposing pairs of pixels whose joining line (pixel centre to pixel centre) passes exactly through the centre of the image data square. So for a 4*4

image data square we have the sum of 8 absolute differences.



Other algorithms can be done in a similar way. Temporal edge detection can be performed by finding the difference between two consecutive frames and passing the results on to the edge detector. This would be implemented as a simple absolute difference function applied to pixels pairs (x,y) and (x',y') where the (x,y) and (x',y') represent the same pixel location in a pair of consecutive frames.

Neighbourhood Shuffling. Neighbourhood shuffling refers to the local rearrangement of neighbourhood pixels to optimise the overall processing speed (e.g. minimise data movement). At the time of writing insufficient work has been done on this technique to give a proper assessment of its worth. However, clearly its use has to be considered very carefully as the time and memory overheads to do the shuffling can quickly outweigh the benefits of doing such a task. Nevertheless it would seem that there are performance gains to be had by this method in certain tasks. Take for example the temporal edge detector described above. In order to find the absolute difference between two pixels we must make two memory accesses. One to pixel (x,y) in frame 1 and another to (x,y) in frame 2. For most of the time the image data will be 8 bits but the system has the option to work on 16 bits. By loading the first pixel into the 8 MSB's and the second pixel into the 8 LSB's when the picture is being loaded to the processor boards, by the input board, the EPLD will only require one memory access to get the pixel pair, thus doubling the performance.

Programming The System. The pre-processor vision engine being described is intended to be a component making up a larger embedded system whose operation is managed by a conventional von-Neumann based processor system. Thus, when considering the system as a whole, there are many levels of programmability. High level system management (e.g. set up modes, convolution size, start acquisition) is provided by the multi-tasking, real-time operating system VxWorks. This is a cross development system and all program development takes place on a variety of Unix workstations including PCs (running NeXTStep), HPs and Suns. The programming language is ANSI C. Once program modules are down-loaded via ethernet the network cable can be detached for complete autonomy. Should a program reach maturity it could be installed in the boot ROMs (which can be up to 1 MB in total) to form a stand-alone embedded system. It is intended that the vision preprocessor will usually be programmed by inserting combinations of common vision function modules obtained from a prefabricated library into the preprocessor. This can be regarded as analogous to the use of C functions and OOPs objects in conventional software development. The implementation of EPLD modules requires knowledge of logic and PLD design. Simple algorithms can be converted and programmed with relative ease but the larger the algorithm the higher the degree of skill required to the design of the EPLD. Programming at this level is not appropriate for those without hardware skills.

Performance. The system has been designed to work at 25fps. The engine characteristics in terms of procssing time and operations per second are dependent on the algorithm being implemented. Provided the processing time is less than 40ms (25fps) the system will function properely. For instance, an 8x8 finite difference followed by two 3x3 algorithms (see Spacek algorithm above) can be executed in 37ms. By means of a comparison to dedicated hardware, the Plessey PDSP16401 edge detection chip (a fixed 3x3 convolution) provides a performance of 225MOPS, a throughput of 450Mbits/s and a pixel latency of 1.3×10^{-6}s. Considering the equivalent part of the SX architecture (the pipeline processor), and running what we believe is a similar algorithm on a 512x512 image, we estimate it would achieve a performance of the order 1GOPS, a throughput of 503Mbits/s, a pixel latency of 420ns and a frame processing time of about 5.5ms. By way of another example, the Alacron "add-on" image processing card (i860 based) performs a 3x3

convolution on a 1kx1k image in about 198.9ms. Considering programmable super-processors, the MP-1 from MasPar (of Sunnyvale, CA) contains between 1024 to 16,384 processing units (32 per VLSI chip) each with 1kBytes can do a 10x10 convolution on a 1k image in less than 0.2 secs (Hogan 90). The architecture described in this paper has been estimated to perform three 10x10 convolutions in 0.074secs. In this configuration the SX engine produces a performance of 14.31 GOPS and costs in the order of $20,000. The MP-1 ($170,00 to $810,00) is supposedly similar to MPP, DAP, Blitzen and the Connection Machine (except in that multiple processors have the added ability to simultaneously access different memory locations permitting the use of data structures such as queues and look-up tables).

Summary. The work described above is part of an ongoing project to investigate the feasibility of using EPLDs as the basis of a programmable vision preprocessor. Although this work is only at an early stage the initial results obtained via simulation indicate that EPLDs now contain sufficient components to allow many low level vision functions to be implemented cost effectively in real-time without requiring prohibitive quantities of devices.

References.

Alsford J, "*CRS Image Processing Systems with VLSI Modules*" (p49, chapter in Kittler book, see below)
Beedie M, "*Image IC Detects Edges in Real-Time*", Electronic Design, May 15th 1986, p58-58
Callaghan V, Prettyjohns K, Alvarez A, "*Structures & Metrics for Image Storage & Interchange*", Journal of Electronic Imaging, Vol.2 No.2, April 1993, ISSN 1017-9909, pp126-137
Clark A et-al. "*Image Processing Architectures*" & "*Parallel Architectures for Image Processing*", two chapters (8 & 10) that appear in book edited by Pearson, see below)
D'Agostino S, "*Machine Vision and Intelligent Highways*", Photonics, April 1992, p109-112
Fairhurst M C, "*Computer Vision For Robotic Systems: An Introduction*", Prentice-Hall, 1988, ISBN 0-13-166927-3
Fleck, M M, "*Multiple Width Yield Reliable Finite Differences*", IEEE PAMI, vol.14,no.4 (1992), 412-429]
Kittler J, Duff M J B (eds), "*Image Processing System Architectures*", Wiley, 1985 (ISBN 0-471-90681-6)
Marr D, 1980. "Visual Information Processing: The Structure and Creation of Visual Representations", Phil. Trans. R. Soc. London B 290: pp199-218.
Maskaki I (ed), "*Vision Based Vehicle Guidance*", Springer-Verlag, 1992 (ISBN 0-387-97553-5
Pearson D, "*Image Processing*", McGraw-Hill, 1991, ISBN 0-07-707323-1
Seitzler T "*Industrial Vision Tackles Web Inspection*", Advanced Imaging Feb. 93, p32
Spacek L "*Edge Detection and Motion Detection*", Image and Vision Computing Vol.4,1 (1986), pp.43-56.
Spacek L "Thinning Image Boundaries", Research Report CSM-187, Department of Computer Science, University of Essex, UK, 1993.
Volker G, Kuhnert K-D, "*Vision Based Autonomous Road Vehicles*" p1-29 (in Maskaki book)
Yencharis L "Imaging Chip Overview 91: Into the Risky 90's"Advanced Imaging Oct. 90, p23

Acknowledgements.
We are pleased to acknowledge the support of Altera Inc. of San Jose, California