Real-Time Imaging **8**, 345–356 (2002)
doi:10.1006/rtim.2001.0273, available online at http://www.idealibrary.com on IDEAL®

A New FPGA/DSP-Based Parallel Architecture for Real-Time Image Processing

In this article, we present a new reconfigurable parallel architecture oriented to video-rate computer vision applications. This architecture is structured with a two-dimensional (2D) array of FPGA/DSP-based reprogrammable processors P_{ij}. These processors are interconnected by means of a systolic 2D array of FPGA-based video-addressing units which allow video-rate links between any two processors in the net to overcome the associated restrictions in classic crossbar systems such as those which occur with butterfly connections. This architecture has been designed to deal with parallel/pipeline procedures, performing operations which handle various simultaneous input images, and cover a wide range of real-time computer vision applications from pre-processing operations to low-level interpretation. This proposed open architecture allows the host to deal with final high-level interpretation tasks. The exchange of information between the linked processors P_{ij} of the 2D net lies in the transfer of complete images, pixel by pixel, at video-rate. Therefore, any kind of processor satisfying such a requirement can be integrated. Furthermore, the whole architecture has been designed host-independent.

J. Batlle[1], J. Martí[1], P. Ridao[1] and J. Amat[2]

[1]Computer Vision and Robotics Group,
Institute of Informatics and Applications,
University of Girona, Av. Lluís Santaló s/n,
17071 – Girona, Catalonia, Spain
E-mail: {jbatlle,joanm,pere}@eia.udg.es
[2]ESAII, Polytechnical University of Catalonia,
C/Pau Gargallo, 5-08028 Barcelona,
Catalonia, Spain
E-mail:amat@esaii.upc.es

Introduction

Computer vision presents us with a widefield of research incorporating a great amount of data and, consequently, requires a high level of processing capacity, mainly in parallel computing. Stated simply, computer vision can be separated into three main levels. The first one, the lower level, deals with pixel-operation functions such as differences between images and neighborhood filtering.

Table 1. A parallel machine classification applied to computer vision systems, proposed by Sima *et al.* [1]

Parallel architectures	Data-parallel architectures	Vectorial		
		Associative and neuronal		
		SIMDs		
		Systolic		
	Function-parallel architectures	Instruction level (ILPs)		Pipeline processors
				VLIWs
				Superscalar processors
		Thread-level	MIMDs	Distributed memory MIMD (multicomputers)
		Process-level		Shared memory MIMD (multiprocessors)

At the intermediate level, we kept in mind tasks such as segmentation, motion estimation and feature extraction or matching, all of which need pipelined processes.

At the upper level of computer vision we deal with interpretation. This function usually requires Artificial Intelligence tools as well as previous knowledge of the environment.

The challenge of pipelining these three levels in real-time can be undertaken using parallel architectures.

Table 1 shows a well-known classification presented by Sima [1] in which the most representative groups of families are shown.

This table identifies some of the major families which represent key junctures in the evolution of parallelism oriented to image processing. To date, these premises are still quite valuable.

A great deal of work on parallelism has been carried out over the past decade. However, most approaches, even the most recent ones, have been based on a handful of historical architectures. In view of this, we should mention the systems based on a net of interconnecting modules which process the whole image by local operation on a pixeled neighborhood [2–6]. These architectures are generally based on geometric parallelism operating in a single instruction, multiple data (SIMD) mode. Moreover, pipelined systems and systolic nets of processors are based on multiple input processes whose outputs constitute the inputs of the next operation [7, 8], scanning the whole image and processing it piece by piece. Thirdly, pyramidal systems compute complex operations by using the *divide and conquer* paradigm [9, 10]. These architectures are frequently used on multi-scalar or recursive operations such as image grabbing at different resolution levels. Furthermore, there are some architectures which are internally organized without taking into account the image structure or their operations [11–13]. Multiple instructions, multiple data (MIMD) architectures, such as digital signal processors (DSPs), permit the design of several types of parallelism. Finally, hypercube processors combine the advantages of pyramidal structures and meshed nets [14,15].

Since architectures for real-time image processing need to manage a large amount of data and work within real-time requirements, the use of parallelism is a fundamental part of most of these systems. An interesting project considering specific architectures for parallelism was carried out by Raman and Clarkson [16]. Their article describes a parallel architecture composed of several specific, non-identical modules which can work concurrently with only one shared memory.

Another interesting proposal was presented by Young [17], a DSPs-based structure which computes in parallel to solve tasks with a high computational cost, as in real-time image processing. He gives a few examples using several DSPs from Texas Instruments, but most specifically the C40. Srinivasan and Govindaraj [18] have also used these devices (DSPs) in multi-processor networks. To increase the process speed, they split the original image into several independent blocks. Each block is then processed concurrently by the DSPs. They also use Texas Instruments DSPs, in this case the C25.

Other projects, like Chen and Jen [19], show a special processor for video signals. This processor is composed of several functional units which work in parallel. However, each unit in turn specializes in one kind of computation (arithmetic unit, multiplicative unit, discrete cosine transform unit, and so forth). Another project by Wu *et al.* [20], presents a co-processor built of several basic modules interconnected by means of a programmable network.

The article by Turton *et al.* [21] is interesting since they attempt to apply genetic algorithms to computer vision studies. To take advantage of this kind of algorithm they had to use parallelism and decided on parallel algorithms genetic (PAG), specifically the Fine Grained version, which suggests that it is better to work with a larger number of simple processors than with only a few complex processors. Finally, the studies by Bertozzi and Broggi [22] describe a stereo vision system for obstacle and lane detection. The kernel of this system is a parallel architecture called *Paprica* 3 which is composed of 256 processing elements (PEs) which work in an SIMD manner. We found that the GIOTTO system, an architecture proposed by Cucchiara [23] used in robotics applications, was similar to the previous work. This is also a parallel computer based on an SIMD reduced-size array processor with a novel organization of the memory sub-system. Several of these architectures are modular, which means that the system can be extended with more PEs or basic cells, depending on the application desired.

As far as our DSP/FPGA-based parallel architecture is concerned, this article is organized as follows. Firstly, in Section 3, the architecture and its performance are discussed in general. Analog I/O video signal devices as well as the systolic crossbar are briefly described in Section 4. Section 5 presents the architecture of the elemental processors and its hardware implementation. A brief overview concerning the control and synchronization of the whole architecture, as well as communications with the host, is presented in Section 6. Finally, conclusions and further work are presented.

The Proposed Architecture

Figure 1 presents our new architecture, which can cope with the majority of problems which crop up in real-time computer vision application, mainly when dealing with more than one RGB camera simultaneously.
The most important features of this proposed architecture can be summarized as follows:

- Each processor P_{ij} operates on an input image supplied by any one of the other processors belonging to the net and supplies a new output image to be processed by the next module.

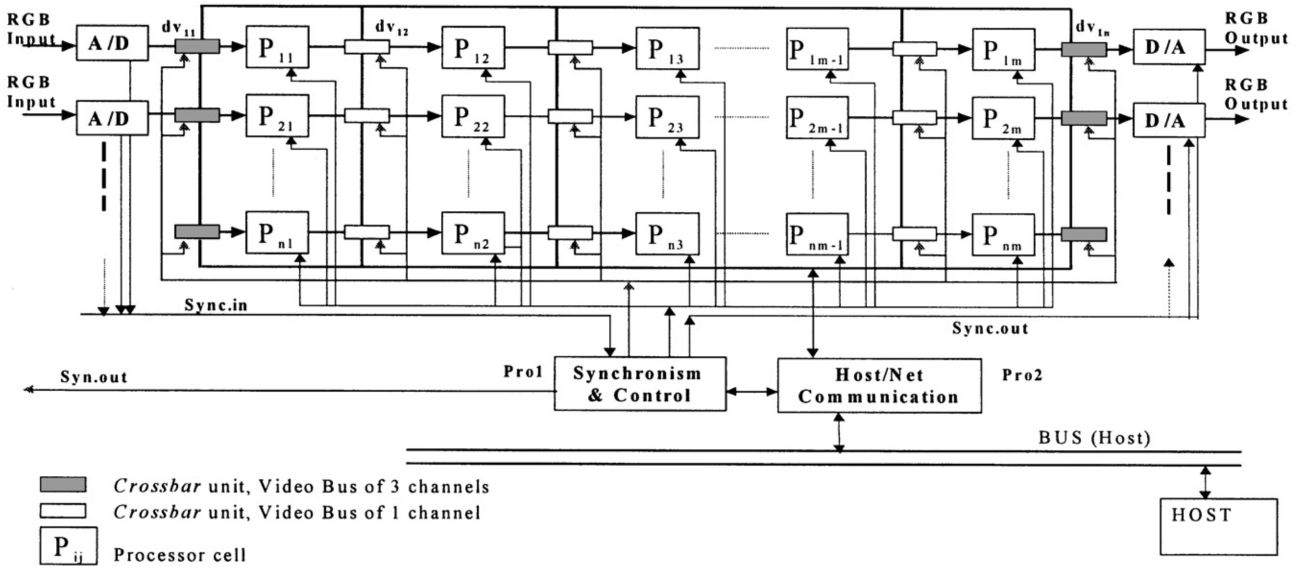

Figure 1. The proposed architecture which constitutes a configurable parallel computer.

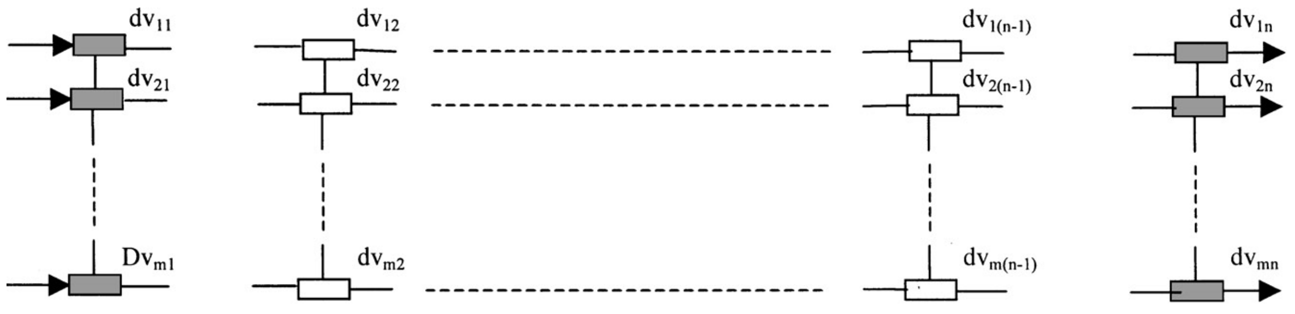

Figure 2. Systolic network of *dv* units which drives the video signal through the architecture.

- The data BUS between processors have been designed for byte-by-byte transmission, that is pixel by pixel.
- A set of digital video multiplexors (dv_{ij}) allows parallel and pipeline connections between processors depending on the final application.
- Since only video-sync information and enable/disable control signals are considered, an FPGA-based processor (Pro1) controls the architecture.
- Processors are independent and perform different image-processing functions. Each processor controls its own memory module. The same function can be paralleled as many times as needed.
- RGB images can be used as input/output.
- All the processors can operate as a switch, allowing the image to pass through without any inconvenience. In such cases, the video input is transferred straight away to the output of the module.

We placed a processor (Pro1) in the overall control of the whole cell architecture, while another processor (Pro2) was in charge of communications with the host. Finally, a matrix of elemental processors [P_{ij}] interconnected through a crossbar built with video-addressing processors called dv_{ij} to deal with the initial stages of the computing process was used. This crossbar construction allows for all sorts of interaction among the basic processors [P_{ij}] which can send or receive images pixel by pixel using a digital 8-bits BUS. Through Pro2, this BUS is used to send information to the main host as well.

In the following, each module of the architecture will be described in further detail, pointing out which kind of devices have been or will be used in future implementation.

Video Transmission

Analog input/output video signal

As mentioned before, multiple RGB input/output signals are possible when dealing with parallel processing of various input images. As examples of probable applications requiring multiple parallel input images, we tested for three-dimensional (3D) image processing, tracking, and object recognition.

In this architecture, every A/D module incorporates a Philips TDA8709A converter and an LM1881 synchronism extractor. These synchronizational signals are supplied to Pro1, the processor controlling the basic processors making up the net. The D/A conversion is performed using TDA8702 devices.

The systolic crossbar

The video-addressing units [dv_{ij}] alone constitute a 2D net of FPGAs (see Figure 2) with the purpose of taking care of video-transmission through the various basic processors [P_{ij}] in the main architecture.

In such a net, we can differentiate between the input/output units addressing a 3×8 video BUS from the rest of the units addressing an 8-bit video BUS.

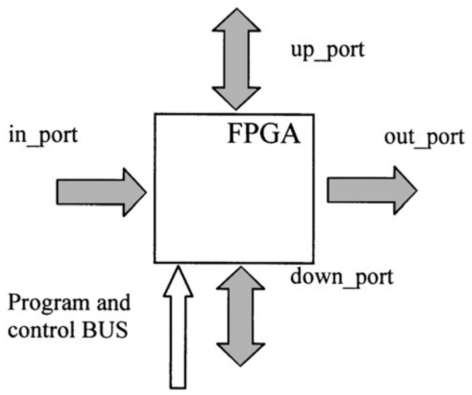

Figure 3. A simplified architecture of the I/O buses for a video-addressing cell [dv_{ij}].

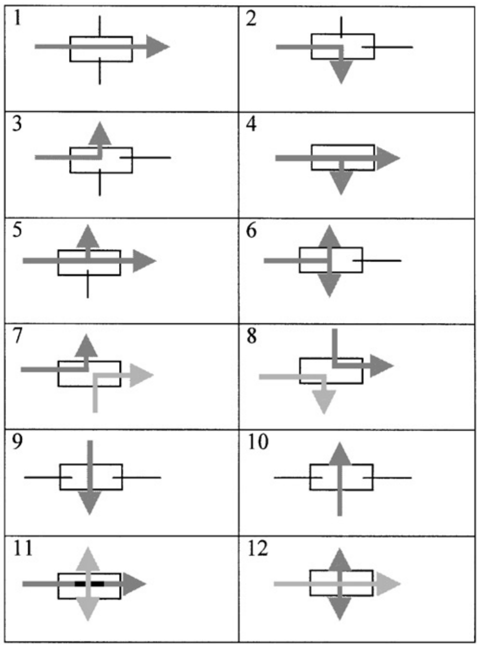

Figure 4. Different options for video multiplexing.

For the implementation of the dv_{ij} *crossbar cells* we opted for an Altera FPGA model FLEX 10K100ARC240-2. Figure 3 shows a simple sketch giving an idea of the number of I/O pins needed. The program and control word is 10 bits in size, four of which are used to select any one of the 12 configurations possible shown in Figure 4.

If an architecture with 64 elemental processors is to be connected, then the 6 remaining bits will be used to identify the position (x,y) of the selected cell $[dv_{ij}]$ from among the 64 possibilities.

The power of the FPGA used allows control tasks to be carried out on a local level, that is, two or more grouped devices $[dv_{ij}]$ could, if necessary, collaborate independently from the general architecture BUS.

Figure 5 shows a time diagram with some examples of video multiplexing tasks. The nomenclature used is as follows:

- *clk_pixel*: A pixel synchronization signal which reacts to the FPGA by means of a rising scale.
- *control*: 4 bits of the control BUS which select one of the 12 multiplexing options.
- *in_port*: Input BUS to the device (8 bits).
- *out_port*: Output BUS to the next processor (8 bits).
- *up_port*: 8 I/O bits from the upper row of processors.
- *down_port*: 8 I/O bits from the lower row of processors.

Figure 5 also shows the time consumption for reprogramming any one of the 12 video-multiplexing options for an individual cell dv_{ij}. The timing shows the following operations:

*Previous period to T*1:

- The 4 first bits of the 10-bits control word indicate that we have selected the video-addressing function-1 represented in Figure 3. This option allows the digital signal to pass through the selected dv_{ij} cell from left to right.
- In in_port we have the gray level of the input byte; for example, 10 h.
- In out_port we have the same 10 h value.
- up_port and down_port are in three-state level.

At instant T_1:

- The value of the entering byte changes from 10 to 20 h.

At instant T_2:

- The value of the input byte 20 h is available at the output pin “out_port”.

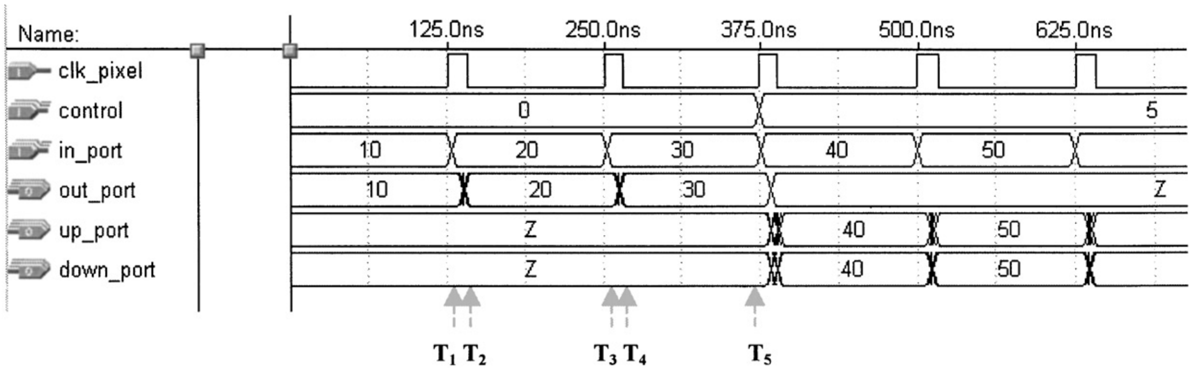

Figure 5. Some examples of multiplexing and its time consumption.

At instant T_3:

- The value of the entering byte changes from 20 to 30 h.

At instant T_4:

- The value of the input byte 30 h is available at the output pin “out_port”.

At instant T_5:

- A change in the control signal produces the new function-6 configuration of the video-addressing cell to that represented in Figure 3.

The current prototype integrates the FPGA FLEX10-K250A.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity multip is
   port (
               control: in std_logic_vector(3 downto 0);
               in_port: in std_logic_vector(7 downto 0);
               up_port: inout std_logic_vector(7 downto 0);
               down_port: inout std_logic_vector(7 downto 0);
               out_port: out std_logic_vector(7 downto 0)
   );

end multip;

architecture multip_arch of multip is
begin

      process(control, in_port)
      begin
               case control is
               when "0000" =>
                       out_port <= in_port;
                       up_port <= "ZZZZZZZZ";
                       down_port <= "ZZZZZZZZ";
               when "0001" =>
                       down_port <= in_port;
                       up_port <= "ZZZZZZZZ";
                       out_port <= "ZZZZZZZZ";
               when "0010" =>
                       up_port <= in_port;
                       down_port <= "ZZZZZZZZ";
                       out_port <= "ZZZZZZZZ";

               end case;
      end process;
end multip_arch;
```

Figure 6. VHDL software oriented to program the video-addressing cells.

The facility of programming such cells dv_{ij} using VHDL can be seen from the program sample in Figure 6, which shows how to manage the first three video-addressing functions of the 12 possibilities from Figure 4.

Finally, in Figures 7 and 8, some schematic examples are presented to indicate the possibilities of interconnecting the processors using the FPGA-based crossbar.

The Basic Processors Cells P_{ij}

As can be seen in Figures 7 and 8, the proposed architecture allows the linkage of an unlimited number of processors compatible with I/O requirements of the video-addressing cells [dv_{ij}]. However, as mentioned before, and with the goal of facilitating the programming task, we suggest the use of identical processors.

Figure 9 shows the developed architecture for the basic processor cell which is composed of the following modules:

- A ping-pong memory.
- A P_1 processor mainly oriented to computational functions.
- A P_2 processor basically addressed to communications and low-level image processing tasks.

In the current prototype, P_1 is a DSP TMS320C51 device chosen for its high computational capabilities. As far as P_2 is concerned, it will be in charge of initial loading program functions, intercommunications with the rest of the architecture and memory management. The addressing needs suggest the use of an FPGA device like the Altera series FLEX which will give computational support to the DSP as well.

The cell itself constitutes a powerful tool oriented to real-time image processing. As far as the P_2 control signals are concerned, a 3-bit BUS was used: a video-synchronism bit and two binary bits to control the four functions: enable, disable, program load and execution.

Figures 10 and 11 show two basic examples of low-level parallelism. In Figure 10, processor P_1 computes with the data stored in M_2 while P_2 loads the input image into M_1. In Figure 11, processor P_1 deals with the input image at video rate and provides an output image to the next cell. Figure 12 shows the first hardware prototype of the basic cell (P_{ij}). Figure 13 shows the first parallel architecture with two basic cells and the I/O RGB interfaces.

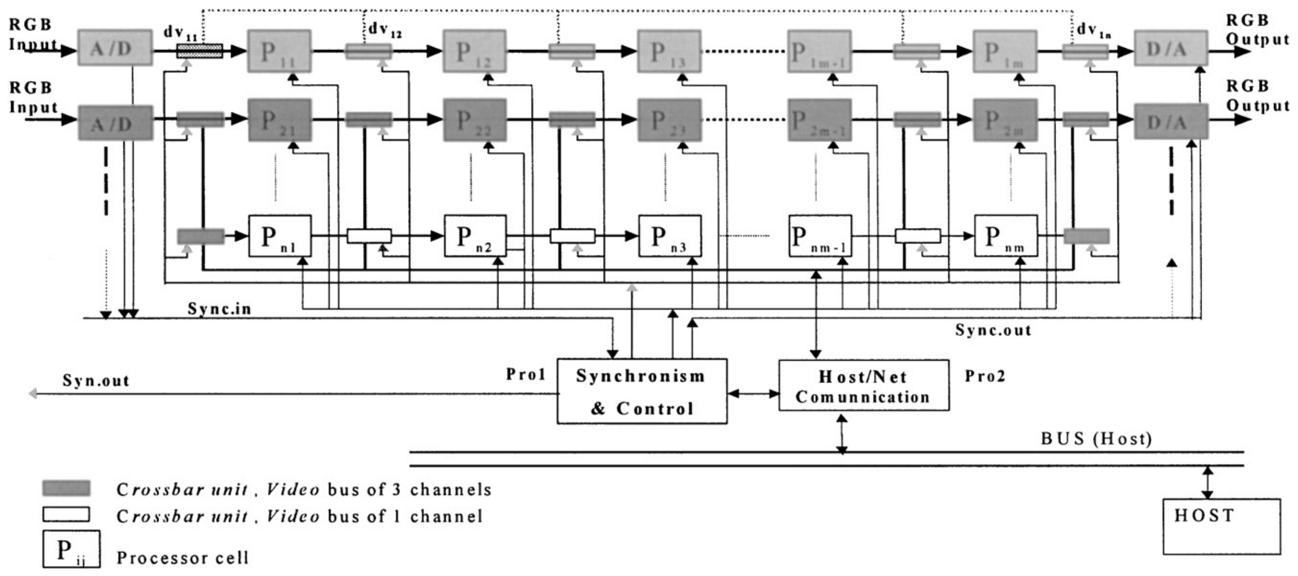

Figure 7. Pipeline and parallel operations.

To end this section, we would like to present an example of how this basic cell (P_{ij}) can be programmed. The developed application consists of loading a frame into the M_1 memory and reading the previous frame from the M_2 memory. Figure 14 shows a chart of the VHDL program. Figure 15 shows the code program. In summary, the DIV block is a simple frequency divider, while the RAMCTRL block is in charge or read/write memory operation.

Control, Synchronization and Communication Tasks

Processor Prol – control and synchronization of the whole architecture

This processor is in charge of control and synchronizational tasks. Its principal functions can be summarized as follows:

- to supply external video-sync. signals for the video cameras,

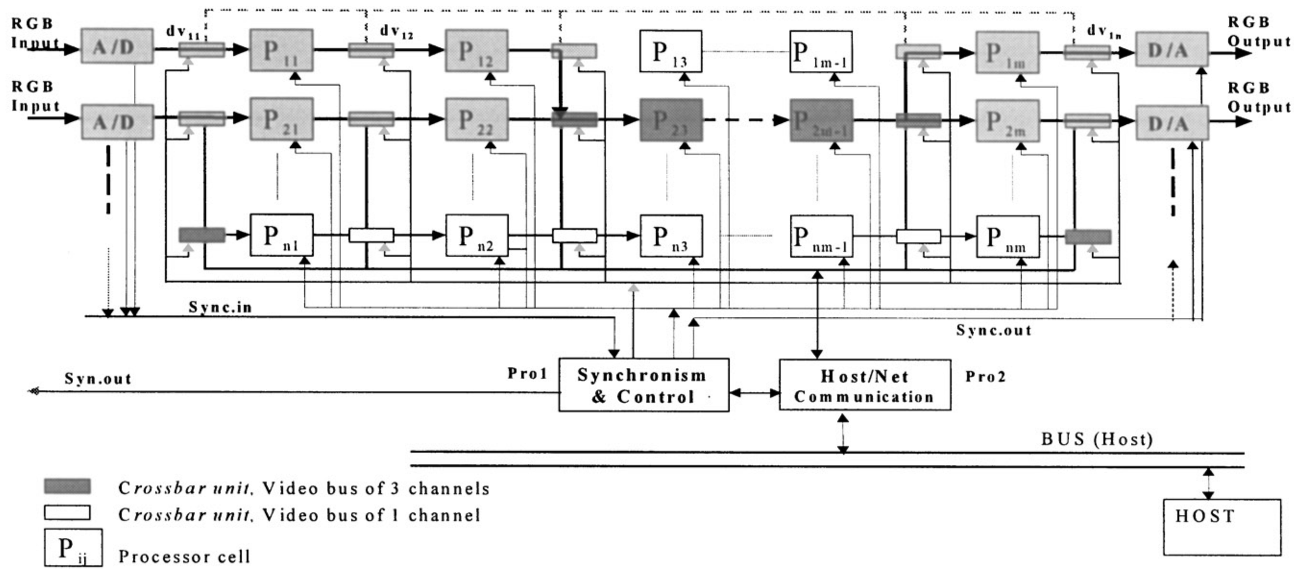

Figure 8. Pipeline and parallel operations.

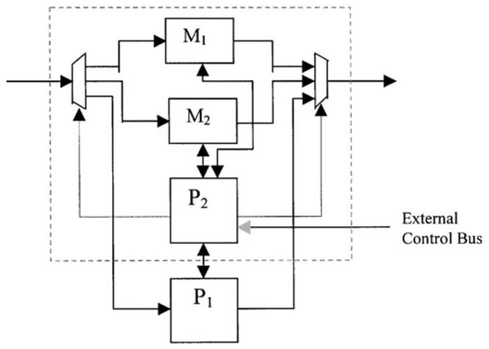

Figure 9. The proposed architecture for the basic processor cell [P_{ij}].

- to supply the sync. signals for the D/A converters,
- to deal with the sync. signals provided by the A/D converters,
- to maintain control over the Pro2 processor charged with host communications,
- the 2D crossbar programs one of the 12 video-addressing functions permitted for the dv_{ij} units (Figure 4),
- the basic P_{ij} cells enable/disable initial loading functions and video-sync. signals supply (Figure 9).

Processor Pro2 – communication with the host

The Pro2 processor manages P_{ij} program loading using its own internal dual-port memory as a shared address and communication with the host. As we suggested for any desired specification, this processor should provide host-independent features, although in the current prototype, only a PCI protocol was used. An FPGA

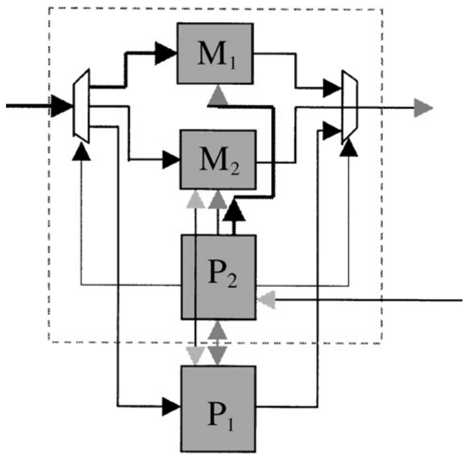

Figure 10. Another basic example of parallelism.

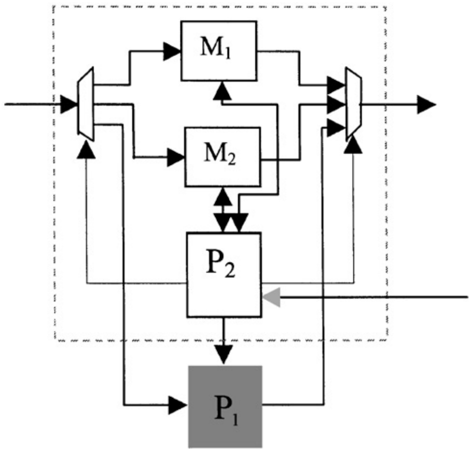

Figure 11. A basic example of parallelism.

Altera FLEX10K100A was used, mainly for its ability to be easily programmed with a 32/64-bit PCI interface.

Application and Conclusions

We have presented a highly versatile parallel architecture which allows dealing with high-level real-time image processing routines. The hardware has been designed to work co-operatively with a host, leaving the host free to deal with the final steps concerning scene understanding and interpretation tasks. The 2D FPGA-

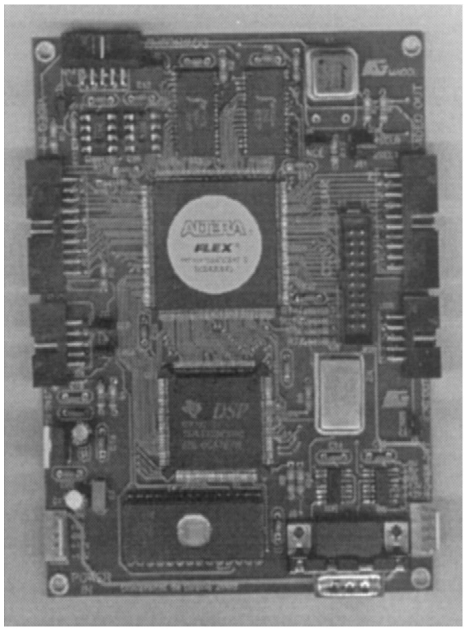

Figure 12. First hardware prototype of the basic cell (P_{ij}).

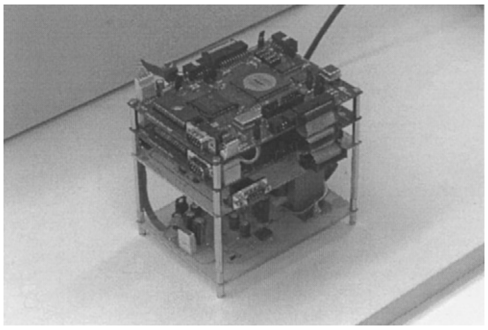

Figure 13. First parallel architecture with two basic cells and the I/O RGB interfaces.

based crossbar allows interconnecting the basic cells [P_{ij}] which, in turn, allow a free flow of pipelining and parallelism with no restrictions concerning the number of linked processors or the number of parallel input images to be dealt with at the same time. The first prototype was used as an embedded computer vision system to implement real-time underwater imaging procedures for the AUV GARBI developed in our Lab (Figure 16). It is well known that the images of the sea bottom suffer from poor light and high noise. As a result, computational time is the most important parameter to be optimized when dealing with autonomous navigation. Keeping underwater imaging in mind, our main purpose was to perform in real-time operations such as undersea pipe tracking. As can be imagined, taking parameters from such an image would not be an easy task. The proposed architecture can perform a great deal of real-time computation from preprocessing steps until final interpretation levels.

The pipeline is detected using two parallel plane laser beams and a video camera oriented to the sea bottom. Since the aim of this application is to show how the board can be programmed, an easy example dealing with the obtained image when projecting two laser beams over a cylindrical object will be presented in Figure 17. The lengths of both lines change with the modification of the distance between the robot and the pipe. Left–right movements of the underwater vehicle with respect to the pipeline can be detected by the location of the lines inside the image.

In the presented application, real-time image processing tasks are performed by FPGA, while the DSP computes parameters like angular displacement and the distance between. In fact, thresholding the image of the projection of the laser beams is the most important step in obtaining the parameters used in tracking control. Moreover, a matrix filter passes over the image and the parameters are extracted from the result. From an architectural point of view, the process is conformed by the modules executed by the FPGA presented in Figure 18 and described as follows:

- BINARY does threshold tasks. The binary image is shown in Figure 19.
- FILMAX3 performs a 2D matrix filter.

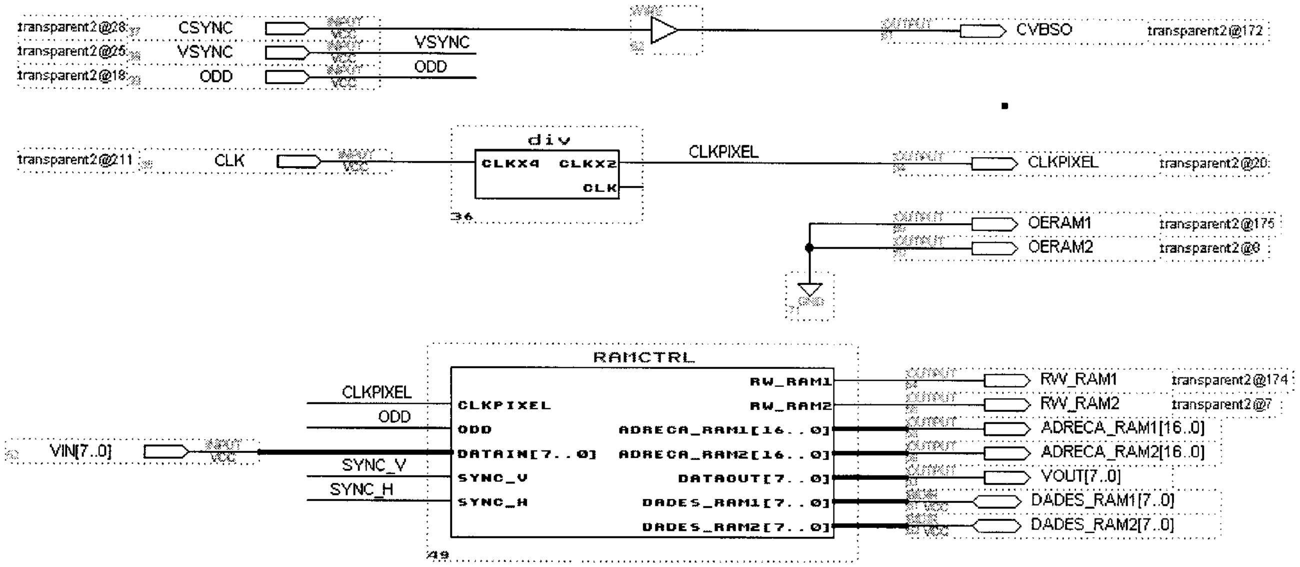

Figure 14. VHDL program for read/write memory operations.

```
RAMCTRL BLOC
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;

ENTITY ramctrl IS
        PORT(
                clkpixel        : in std_logic;
                odd             : in std_logic;
                datain          : in std_logic_vector(7 downto 0);
                sync_v          : in std_logic;
                sync_h          : in std_logic;
                rw_ram1         : out std_logic;
                rw_ram2         : out std_logic;
                address_ram1    : buffer unsigned(16 downto 0);
                address_ram2    : buffer unsigned(16 downto 0);
                dataout         : out std_logic_vector(7 downto 0);
                dades_ram1      : inout std_logic_vector(7 downto 0);
                dades_ram2      : inout std_logic_vector(7 downto 0)
        );
END ramctrl;

ARCHITECTURE arq_ramctrl OF ramctrl IS
BEGIN
        process(clkpixel)
        begin
                if rising_edge(clkpixel) then
                        if sync_v='1' and sync_h='0' then
                                if odd='0' then   -- read in RAM1 and write in RAM2
                                        rw_ram1 <= '1';
                                        rw_ram2 <= '0';
                                        dataout <= dades_ram1;
                                else
                                        rw_ram1 <= '0';
                                        rw_ram2 <= '1';
                                        dataout <= dades_ram2;
                                end if;
                        else
                                dataout <= "00000000";
                        end if;
                end if;
        end process;

        process(clkpixel)
        begin
                if rising_edge(clkpixel) then
                        if sync_v='0' then
                                address_ram1 <= conv_unsigned(0, 17);
                                address_ram2 <= conv_unsigned(0, 17);
                        elsif sync_h='0' then
                                address_ram1 <= address_ram1 + 1;
                                address_ram2 <= address_ram2 + 1;
                        end if;
                end if;
        end process;

        process(clkpixel)
        begin
                if odd='0' then
                        dades_ram1 <= "ZZZZZZZZ";
                        dades_ram2 <= datain;
                else
                        dades_ram1 <= datain;
                        dades_ram2 <= "ZZZZZZZZ";
                end if;
        end process;
end arq_ramctrl;
```

Figure 15. Code program for read/write memory operations.

- CSERIE2 does the communication protocol RS232 with the external devices, like PC.
- C_SERIE_N controls the input data flow from the PC, communication being done using the serial port.
- CONTROL_SERIE collects the signals to be sent through serial port and sends them to the previous block.
- DSPCTRL does the communication function with the DSP.

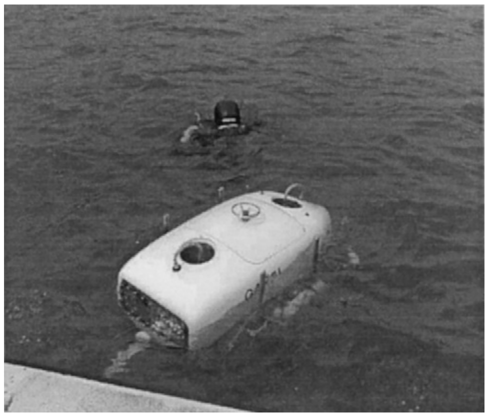

Figure 16. The GARBI underwater robot.

- GETOPTI is a buffer for initialization data. The information taken from the image processing is compared with the pre-set one.
- R_CONTROL2 processes the binary image. It computes the distance between the lines appearing in the binary image, the position of the center of the first line and detects if one of the lines has disappeared from the image.

The system compares the information obtained by processing the image with the pre-set values for the optimum distance to be maintained between the robot and the pipeline. The DSP is used as a complementary processor for computation of the displacement angle and the distance between Figure 20 shows the implemented software in C-language for DSP.

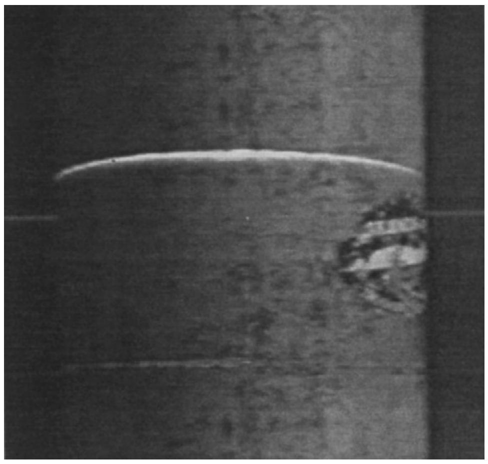

Figure 17. Images taken by the camera from 2 m.

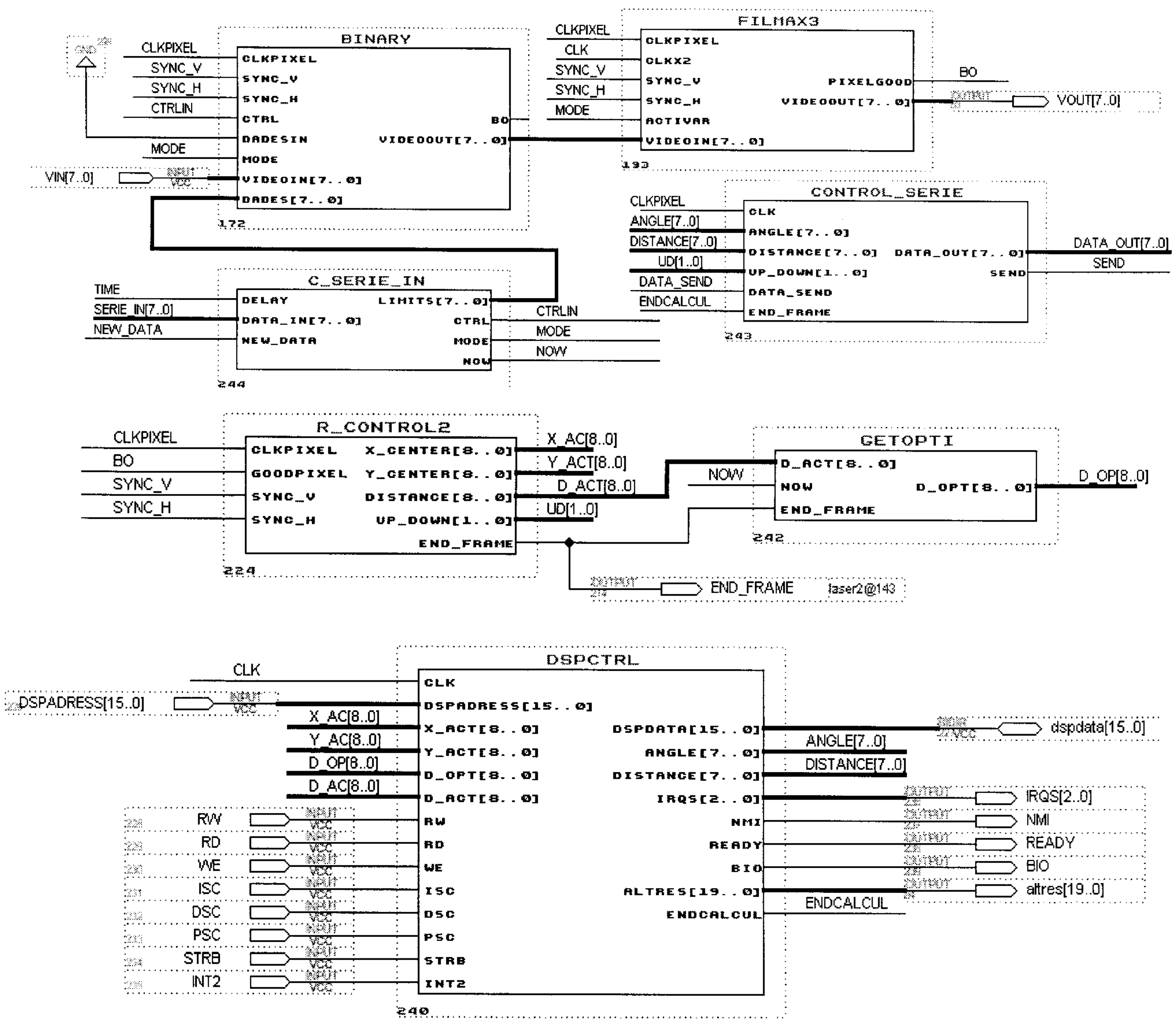

Figure 18. The modules of the application.

This processor is capable of dealing with noise, segmentation, edge detection, correlation, FOE detection and perspective transformation at video rate with minimal delay of a few frames. As far as further hardware work is concerned, the FPGA FLEX10-K250A will be used to implement the 2D crossbar net, since only a low number of individual dv_{ij} cells can be integrated using a single chip. As further work, the powerful processor DSP TMS320C62x will be used mainly because its facilities are oriented to mathematical computing. This processor is able to perform instructions at 5 ns, an invaluable feature when dealing with filtering, FFT operations and so forth. Furthermore, its 1-Mbit internal RAM will allow optimization of the external memory resources. Another important feature is its reduced size compared with its power, a useful characteristic when dealing with embedded systems to be located inside small autonomous underwater vehicles and other mobile robots.

Acknowledgements

This project was carried out under the research projects PETRI 95-0145-OP and PETRI2 95-0299-OP of the Spanish Research Council CICyT.

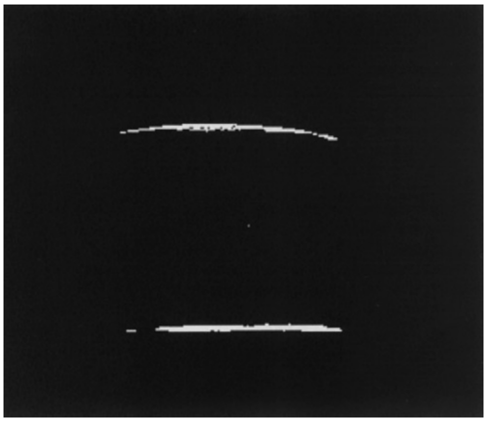
Figure 19. Binary image.

```
#include "c5xregs.h"
#include "math.h"

void main(void)
{
  volatile unsigned int x_act, y_act, d_act, d_optim, teta, dist;

  for (;;) {
     x_act = *PA0;
     y_act = *PA1;
     d_act = *PA2;
     d_optim = *PA3;
     teta=tan(x_act/y_act);
     dist=(d_act*100)/d_optim;
     *PA4 = teta;
     *PA5 = dist;
  }
}
```

Figure 20. Implemented software in C-language for DSP.

References

1. Sima, D. *et al.* (1997) *Advanced Computer Architectures: A Design Space Approach*. Reading, MA/New York: Addison-Wesley/Longman.
2. Batcher, K.E. (1987) Design of a massively parallel processor. *IEEE Transactions on Computers* **C-29**: 1523–1538.
3. Duff, M.J. *et al.* (1988) Review of the CLIP image processing system. *Proceedings of the National Computer Conference*, pp. 1055–1060.
4. Kung, S.Y. *et al.* (1988) *VLSI Array Processor*. Englewood Cliffs, NJ: Prentice-Hall.
5. Maresca, M. *et al.* (1990) Connection autonomy in SIMD computers: a VLSI implementation. *Journal of Parallel and Distributed Computing* **7**: 302–320.
6. Oldfield, D.E. *et al.* (1985) An image understanding performance study on the ICL distributed array processor. *Proceedings of the IEEE Workshop on Computer Architecture for Pattern Analysis and Image Database*, pp. 264–265.
7. Loughead, R.M. *et al.* (1980) The cytocomputer: a practical pipeline image processor. *Proceedings of the 7th Annual International Symposium on Computer Architecture*, pp. 271–277.
8. Kent, E.W. *et al.* (1985) PIPE: Pipeline Image Processing Engine. *Journal of Parallel and Distributed Computing* **2**: 50–78.
9. Mérigot, A. *et al.* (1991) Architectures massivement parallèles pour la vision artificielle. *Annales des Télécommunications* **46**: 78–89.
10. Uhr, L. (1987) *Parallel Computer Vision*. Boston, MA: Academic Press.
11. Siegel, H.J. *et al.* (1981) PASM: A partitionable SIMD/MIMD system for image processing and pattern recognition. *IEEE Transactions on Computers* **C-30**: 934–947.
12. Annaratone, M. *et al.* (1987) The Warp computer: architecture, implementation and performance. *IEEE Transactions on Computer* **C-29**: 1523–1538.
13. Borkar, S. *et al.* (1988) iWarp: an integrated solution to high-speed parallel computing. *Proceedings of Supercomputing'88*, pp. 330–339.
14. Hillis, W.D. (1985) *The Connection Machine*. Cambridge, MA: MIT Press.
15. Blank, T. (1990) The MasPar MP-1 architecture. *Proceedings of IEEE Compcon Spring*, pp. 20–24.
16. Raman, S. & Clarkson, T. (1990) Parallel image processing system – a modular architecture using dedicated image processing modules and a graphics processor. *IEEE, Conference on Computer and Communication Systems*, September 1990, Hong Kong, pp. 319–323.
17. Young, E. (1995) The application of parallel DSP networks to real time image processing. IEE Colloquium on Applications of Machine Vision, London, UK, pp. 11/1–11/6.
18. Srinivasan, S. & Govindaraj, H. (1989) Design of a multiprocessor system using TMS320C25 for real time image processing. IEEE International Symposium on Circuits and Systems, Portland, OR, USA, Vol. 3, pp. 1915–1918.
19. Chen, C.-C & Jen, C.-W. (1996) A programmable concurrent video signal processor. *Proceedings of the International Conference on Image Processing*, Vol. II, pp. 1039–1042.
20. Wu, A.-Y., Ray Liu, K., Raghupathy, A. & Liu, S.-C. (1995) Parallel programmable video co-processor design. *IEEE International Conference on Image Processing* pp. 61–64.
21. Turton, B.C.H., Arslan, T. & Horrocks, D.H. (1994) A hardware architecture for a parallel genetic algorithm for image registration. IEEE Colloquium on Genetic Algorithms in Image Processing and Vision, London, UK. pp. 11/1–11/6.
22. Bertozzi, M. & Broggi, A. (1988) GOLD: a parallel real-time stereo vision system for generic obstacle and lane detection. *IEEE Transactions on Image Processing* **7**: 62–81.
23. Cucchiara, R., Di Stefano, L., Piccardi, M. & Salmon Cinotti, T. (1997) The GIOTTO system: a parallel computer for image processing. *Real Time Imaging* **3**: 343–353.