Journal of VLSI Signal Processing 28, 259–278, 2001


# A High Speed VLSI Architecture for Handwriting Recognition

FRANCESCO GREGORETTI, ROBERTO PASSERONE, LEONARDO MARIA REYNERI
AND CLAUDIO SANSOÉ
*Dipartimento di Elettronica-Politecnico di Torino*

*Received October 1999; Revised April 2000*

**Abstract.** This article presents PAPRICA-3, a VLSI-oriented architecture for real-time processing of images and its implementation on HACRE, a high-speed, cascadable, 32-processors VLSI slice. The architecture is based on an array of programmable processing elements with the instruction set tailored to image processing, mathematical morphology, and neural networks emulation. Dedicated hardware features allow simultaneous image acquisition, processing, neural network emulation, and a straightforward interface with a hosting PC.

HACRE has been fabricated and successfully tested at a clock frequency of 50 MHz. A board hosting up to four chips and providing a 33 MHz PCI interface has been manufactured and used to build BEATRIX, a system for the recognition of handwritten check amounts, by integrating image processing and neural network algorithms (on the board) with context analysis techniques (on the hosting PC).

**Keywords:** image processing, parallel architectures, handwriting recognition, artificial neural networks, VLSI implementations

## 1. Introduction

Handwriting recognition [1–3] is a major issue in a wide range of application areas, including mailing address interpretation, document analysis, signature verification and, in particular, bank check processing. Handwritten text recognition has to deal with many problems such as the apparent similarity of some characters with each other, the unlimited variety of writing styles and habits of different writers, and also the high variability of character shapes issued by the same writer over time. Furthermore, the relatively low quality of the text image, the unavoidable presence of background noise and various kinds of distortions (for instance, poorly written, degraded, or overlapping characters) can make the recognition process even more difficult.

The amount of computations required for a reliable recognition of handwritten text is therefore very high, and real-time constraint can only be satisfied either by using very powerful and expensive processors, or by developing ad-hoc VLSI devices.

To cope with the tight cost and size constraints we had, we decided to develop HACRE, a massively parallel VLSI image processor with the instruction set dedicated to execute both traditional *image processing* (such as *filtering* and *image enhancement*) and *mathematical morphology* [4] (such as *opening*, *closing*, *skeletonization*) and several types of *neuro-fuzzy networks* [5] (such as *perceptrons, self-organizing maps*, *cellular networks*, *fuzzy systems*). We have tailored the recognition algorithms to the architecture capabilities.

HACRE is based on PAPRICA-3, a dedicated architecture deriving from enhancement of previous works [6–8]. It is designed to be used both in a minimum-size configuration (consisting of 1 chip, 1 external RAM, plus some glue logic for microprocessor interfacing) and in larger-size configurations (consisting of as many cascaded chips as required, as many external RAMs, some additional global interconnection logic and a host interface, typically a PCI).

A configurable PCI board hosting up to four HACRE chips, together with the required RAM chips (for image

and program memories), camera and monitor interfaces, controllers, PCI interface and all the required logic, has been designed, manufactured and successfully tested.

One such board, populated with two HACRE chips, plugged into a hosting PC, has been used to build and test BEATRIX, a complete system for high-speed recognition of the amount on banking checks, which mixes image processing algorithms, neural networks and a context analysis subsystem [3].

The VLSI device and the overall system (board and PC interface) have been designed in parallel with the development of the algorithms (BEATRIX), so that the hardware and software designs have influenced each other very much. That resulted into an efficient though flexible implementation for a wide class of applications.

Section 2 describes the driving application, while Sections 3 and 4 describe the PAPRICA-3 architecture and the HACRE chip, respectively. Section 5 describes the PCI board, while Section 6 briefly describes the BEATRIX check recognizer. At the end, Section 7 gives the measured performance of the system and compares them with those of a commercial PC.

## 2. Driving Application

The aim of our work was to recognize real-world checks, where handwriting is assumed to be unboxed and usually unsegmented, so that characters in a word may touch or even overlap. The amount is written twice: the *legal amount* (namely, the literal one), and the *courtesy amount* (namely, the numerical one).

The two fields are placed in well-known areas of the check, and an approximate localization of these two areas can be obtained from the information contained in the *code-line* printed at the bottom of the check.

### *2.1. Application Requirements*

Our aim was to cope with the tight cost and performance requirements which might make our system commercially relevant. From a preliminary system analysis we pointed out the following requirements: *average processing speed* of at least 3 checks per second (5 per second, peak), with a *rejection rate* significantly lower than 5% and an *error rate* approaching zero. This corresponds to a *sustained recognition speed* of about 500 characters per second (both digits and alphanumeric) and a *character accuracy rate* in excess of 99%.

*Cost requirements* imposed a higher bound of 50 US$ per chip with a *power dissipation* of at most 1 W per chip. In addition, one particular application in a low-speed but low-cost system was claiming at a *single-chip embedded solution*.

As far as the host processor is concerned, we selected a commercial PC with a standard PCI bus. The operating system could either be Microsoft Windows or Linux.

The choice of such a general-purpose hybrid architecture (a PAPRICA-3 system tightly interconnected to a host PC) was driven by the observation that in many image processing systems the first processing steps (at the bitmap level) mostly require simple operations on a large amount of data, whereas, as processing proceeds, less and less data requires more and more complex operations. In our approach, PAPRICA-3 is tailored to a fast and repetitive processing of elementary pixels of an image while the PC is best used for more complex and more symbolic-based operations on a much reduced amount of data. We feel that such a cooperation between the two systems can provide the best cost-performance ratio.

### *2.2. Chip Requirements*

The HACRE chip has been designed bearing in mind the following application-dependent constraints/requirements:

- cascadability: HACRE implements a slice of 32 Processing Elements (**PE**'s) (all that could fit in 100 $mm^2$), but additional chips can be cascaded and connected to as many RAM chips;
- neural networks mapping: one PE per each neuron, to have the highest efficiency;
- a simple 1-bit Processor Element; operations with higher resolution can be computed by means of a bit-serial approach; this provided the best complexity/flexibility/performance trade-off among all the architectures which were analyzed;
- provisions for both image processing, mathematical morphology [4], neural networks [5] emulations, image acquisition, etc.;
- easy interface with a host processor (possibly a PC), to improve overall functionality;
- the highest degree of programmability, while keeping the hardware complexity at a low level; this has been achieved by letting the host PC (instead

of HACRE) perform a number of operations which normally occur at a lower rate;
– both "local" and "global" instructions; the former are used to implement a sort of pixel neighborhood, while the latter are available to compute global operators, such as summations, maxima, minima, winner-takes-all, etc.;
– provisions for simple handling of external look-up tables (external RAM's or ROM's);
– a set of external "status registers" where HACRE can accumulate neuron outputs and which can be read (or written into) in parallel by the host PC;
– a set of direct binary I/O channels $(6 + 6)$ through which HACRE can either interrupt the host PC or activate stepper motors, CCD cameras, etc.

## 3. Architecture Description

PAPRICA-3 is the latest component of a family of massively parallel processor architectures [6–8] designed at the Politecnico di Torino. As shown in Fig. 1, the kernel of PAPRICA-3 is composed of a linear array of identical Processing Elements (**PE**s) connected to a memory via a bidirectional bus.

The memory stores the image and the intermediate results of the computation and is organized in words (each one associated to an address) whose length matches that of the array of processors. Each word in the memory contains information relative to one binary pixel plane (also called *layer*) of one line of an image. Because the width of the bus that connects the array and the memory is the same as the number of processing elements (and therefore it is the same as the word length), a single memory cycle is required to load or store an entire line of the image to/from the PE's internal registers.

A *Control Unit* executes the program stored in the instruction memory (called the *Writable Control Store*) and generates the signals that control the operations of the processing elements, the image memory, and

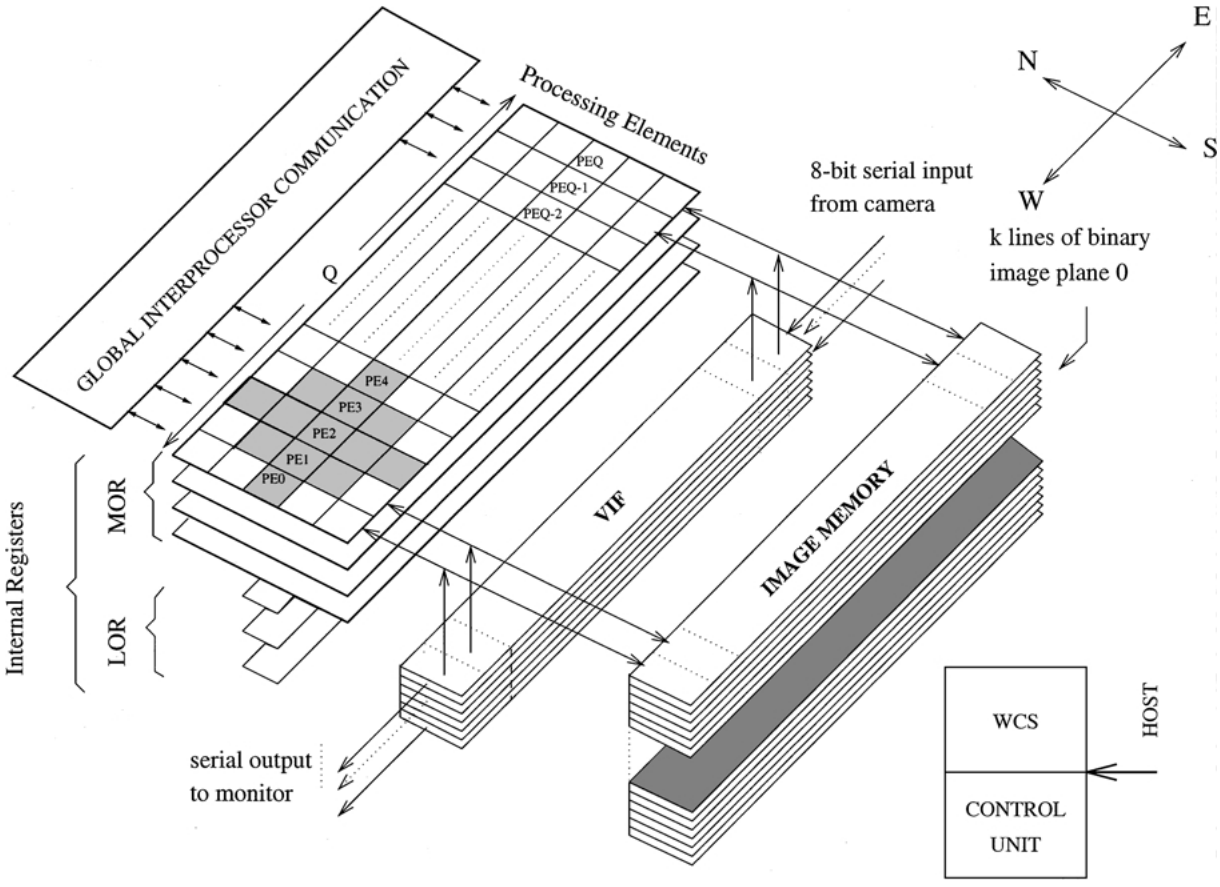


*Figure 1.* General architecture of the processor array.

of the other architectural elements that will be described below. The typical flow of operation consists of first transferring one line of the image from the memory to the array, then processing the data, and finally storing back the results into memory according to a LOAD–EXECUTE–STORE processing paradigm typical of RISC processors. The same cycle is repeated for each line until the entire image has been processed.

When executing a program, the correspondence between the line number, the pixel plane of a given image, and the absolute memory address is computed by means of data structures called *Image Descriptors*. An Image Descriptor is a memory pointer that consists of two parts: a base address that usually represents the first line of the image, and two line counters which can be reset or increased by a specified amount during the execution of the program, which are used as indices to scan different portions of the image.

The instruction set includes several ways to modify the sequential flow of control. Branches can be taken unconditionally or on the basis of conditions drawn over the control unit internal registers. In addition, any instruction can be prefixed by an enabling condition. One register in the control unit is dedicated to the implementation of hardware loops: given the iterative nature of the algorithms employed in image processing, this features greatly enhances the performance of the architecture. Two additional conventional counters can be used as indices in the outer loops; instructions are provided to preset, increase and test their value.

### *3.1. Processing Elements*

Each PE is composed of a Register File and a 1-bit Execution Unit, and processes one pixel of each line. The core of the instruction set is based on *morphological operators* [4]: the result of an operation depends, for each processor, on the value assumed by the pixels of a given neighborhood, which in the case of PAPRICA-3 is a reduced $5 \times 5$ box, as sketched by the grey squares in Fig. 1. The morphological function can be selected by changing the value of a template which encodes for each pixel of the neighborhood the kind of required boolean combination.

The instruction set includes also *logical* and *algebraic* operations (AND, OR, NOT, EXOR, etc.), which can be used either to match input patterns against predefined templates, or to compute algebraic operations such as sums, differences, multiplications, etc. As PE's are 1-bit computing elements, all algebraic operations have to be computed using a bit-serial approach.

For each Processor Element, the value of the pixels located in the East and West directions (either 1 or 2 pixels away) is obtained by a direct connection to the neighboring PE's, while the value of the pixels in the North and South directions corresponds to that of previously or to be processed lines.

To obtain the outlined neighborhood in the chip implementation, a number of internal registers (16 per each PE, at present), called *Morphological Registers* (**MOR**), have a structure which is more complex than that of a simple memory cell, and are actually composed of five 1-bit cells with a S→N shift register connection. When a load operation from memory is performed, all data are shifted northwards by one position and the south-most position is taken by the new line from memory. In this way, data from a $5 \times 5$ neighborhood are available inside the array for each PE, at the expense of a two-line latency. A second set of registers (48 per each PE, at present), called *Logical Registers* (**LOR**), is only 1-bit wide and is used for logical and algebraic operations only.

### *3.2. Video Interface*

An important characteristic of the system is the integration of a serial-to-parallel I/O device, called *Video InterFace* (*VIF*), which can be connected to a linear CCD array for direct image input (and optionally to a monitor, for direct image output). The interface is composed of two 8-bit shift registers which serially and asynchronously load/store a new line of the input/output image during the processing of the previous/next line. Two instructions activate the bidirectional transfer between the PE's internal registers and the VIF, ensuring also proper synchronization with the CCD and the monitor.

### *3.3. Inter-Processor Communication*

Two inter-processor communication mechanisms are available to exchange information among PE's which are not directly connected.

The first mechanism consists of a network (called *Status Evaluation Network*), shown in Fig. 2(a) which spans the extent of the array; each processor sends the 1-bit content of one of its registers and the global network provides a *Status Word* which summarizes the

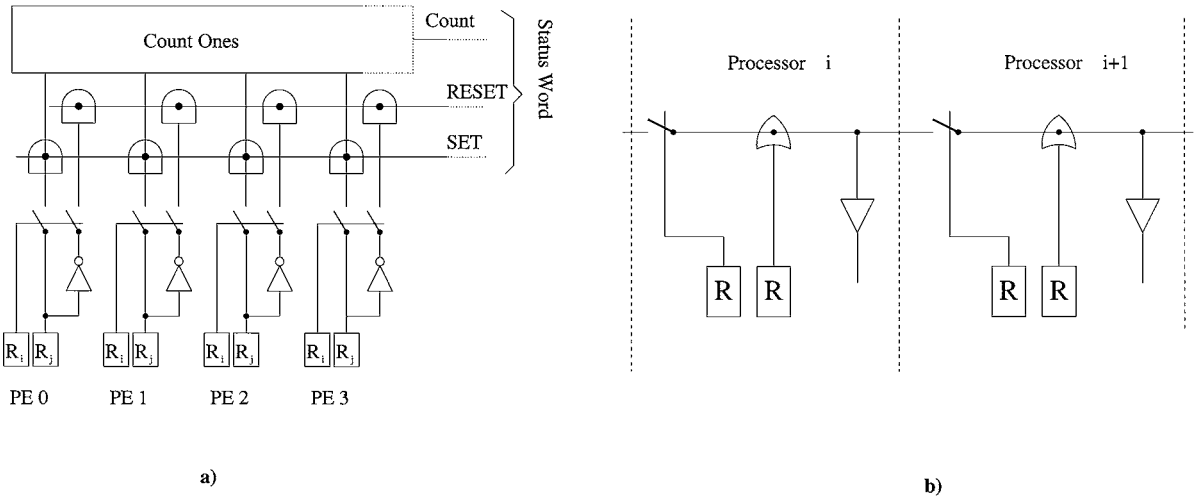


*Figure 2.* Inter-processor communication mechanisms: a) status evaluation network b) inter-processor communication network.

status of the array. The word is divided into two fields: the first field is composed of two global flags, named *SET* and *RESET*, which are true when the contents of the specified registers are all '1's or all '0's, respectively; the second field is the *COUNT* field which is set equal to the number of processing elements in which the content of the specified register is '1'. This inter-processor communication mechanism can be used to compute global functions such as maxima, minima (e.g., for emulation of fuzzy systems), logical OR and AND of boolean neurons, neighborhood communications, neuron summations in perceptrons, external look-up tables, winner-takes-all, seed propagation algorithms, and many others.

This global information may also be accumulated and stored in an external *Status Register File* and used for further processing, or to conditionally modify program flow using the mechanism of prefix conditions. Status Registers can also be read by the host processor. For instance, Status Registers have been used in the example of Section 2 to implement a neural network by computing the degree of matching between an image and a set of templates (weight and center matrices).

The second communication mechanism is an *Inter-processor Communication Network*, shown in Fig. 2(b), which allows global and multiple communications among clusters of PE's. The topology of the communication network may be varied at run-time: each PE controls a switch that enables or disables the connection with one of its adjacent processors. The PE's may thus be dynamically grouped into clusters, and each PE can broadcast a register value to the whole cluster with a single instruction. This feature can be very useful in algorithms involving *seed-propagation* techniques, in the emulation of pyramidal (hierarchical) processing and for cellular neural networks or for local communication (short range neighborhood).

### *3.4. Host Interface*

A Host Interface allows the host processor to access the WCS and a few internal configuration registers when HACRE is in STOP mode. The access is through a conventional 32-bit data bus with associated address and control lines. The same lines are controlled by HACRE in RUN mode and used to access the private external Image Memory.

Some additional control and status bits are used to exchange information with the host processor: these include a START input line and a RUNNING output line, plus another six input and six output lines called *Host Communication Channels* (**HCC's**). HCC input lines can be tested during program execution to modify program flow, while HCC output lines can be used as flags to signal certain conditions to the host processor (for instance, interrupts).

## 4. Chip Description

The kernel of the hardware implementation of the PAPRICA-3 system is the *HACRE* integrated circuit whose block diagram is shown in Fig. 3.

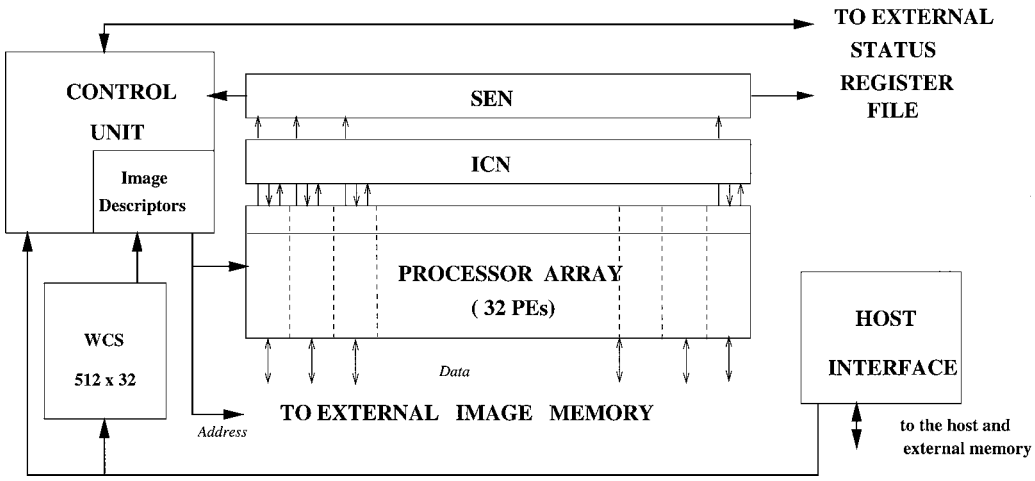


*Figure 3*. Block diagram of the chip.

The main components are:

- The *Processor Array* (PA) which is composed of 32 processing elements; the internal architecture and the main features of the PA and of the PE's are described in detail in Section 4.1.
- The PA has a direct and fast parallel communication channel towards the Image Memory (IM) which in the current implementation is external to the chip. The decision to keep the IM outside of the chip has been a very critical issue. The direct and fast access to a large internal memory would have allowed to execute at each clock cycle a LOAD or STORE operation with the same timing as other instructions and with a high processing throughput but, on the other hand, the cost of implementing a large memory with a standard CMOS technology would have been too high. In fact architectures such as the IMAP system [9], which have large on chip data memories, employ dedicated, memory oriented fabrication processes. Therefore we decided to implement the IM external to the chip and to reduce the processing overhead by a heavy pipeline of internal operations which may allow the partial overlap of a memory access with the execution of subsequent instructions.
- The *Control Unit* (CU) which executes the instructions by dispatching to the PA the appropriate control signals. The choice of implementing the CU on the same chip as the PA is a key characteristic of the PAPRICA-3 architecture with respect to other mono- and bi-dimensional SIMD machines [9–13] in which the controller is a unit which is physically distinct form the array. In this case the maximum clock speed is limited to a few tens of MHz by the propagation delay of the control signal from the controller to the different processing units. This may be a non critical limit in systems with a large number of PE's, but, since our application was aimed at real time embedded systems, we preferred to push to the limit the performances even of single-chip systems by integrating the CU with the array. This means that with multiple-chip systems the CU is replicated in each chip with an obvious loss in silicon area, but in our case it has been considered a little price to pay, with respect to the possible increase in performance since our design goal was an operating frequency of 100 MHz. Section 4.2 will analyze in detail the design choices of the CU and its internal architecture.
- The *Writable Control Store* (**WCS**, 1K words $\times$ 32 bits), in which a block of instructions is stored and from which the CU fetches instructions. The choice of a WCS is a compromise between different constraints. First the goal of executing one instruction per cycle required the Instruction Memory to reside on the same integrated circuit as the CU. Since the amount of memory which may be placed on a chip with a standard CMOS technology is limited the optimal solution would have been a fast associative cache. A preliminary feasibility study showed that the cache approach would have been too expensive in terms of silicon area and too slow to match the target 10 ns cycle time. Considering that most image processing algorithms consist of sequences of low level steps, such as filters, convolutions, etc., to be performed line by line over the whole image this means that the same block of instructions has to be

repeated many times. Hence we chose to pre-load each block of instructions into the WCS, a fast static memory, and to fetch instructions from there. After the instructions of one block have been executed for all the lines of an image, a new block of instructions is loaded in the WCS. If the ratio between the loading time and the processing time is small then the performance of a fast cache with a hit ratio close to 1 can be obtained, at a fraction of the cost and complexity.

– The *ICN* and *SEN* communication networks. The former is fully distributed in the PE's; the latter is composed of two parts: the first part is distributed and is the collection of EVAL units (see Section 4.1) integrated in the PE's, while the second one is centralized and is composed of one *Count Ones* unit and two 32-AND units for the evaluation of the SET and RESET flags.
– The *Host interface* which allows a host processor to access the WCS, and a few internal configuration registers. The access is through a conventional 32-bit data bus with associated address and control lines. In addition the interface handles the external protocol for the communication between the PA and the external Image Memory.

A microphotograph of the complete chip is shown in Fig. 4. The chip has been implemented in a 0.8 $\mu$m, single poly, dual metal CMOS technology and has a total area of 99 mm$^2$. Multiple chips may be cascaded to build systems with a larger number of processing elements, as explained in detail in Section 5.

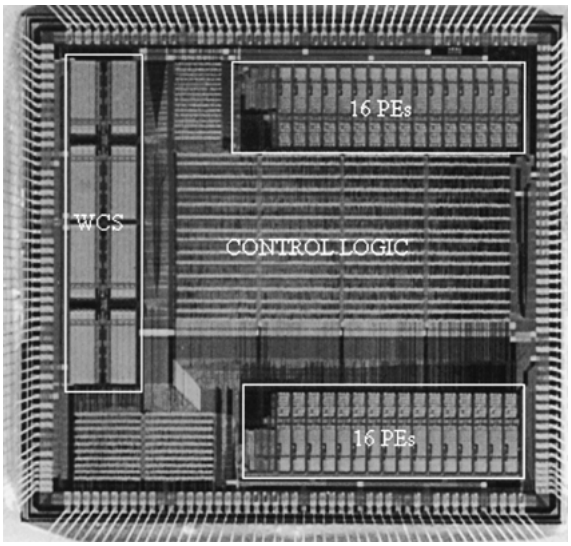


*Figure 4*. Microphotograph of the chip.

### *4.1. Processing Array*

The Processing Array has been implemented using a full custom design style in order to take advantage of its regular structure and optimize its area. In fact each Processing Element is a layout slice and, since all PE's operate according to a SIMD paradigm, the control signals are common to all of them and may be broadcast by simple abutment of PE cells. Unlike the block diagram of Fig. 3 where the 32 PE array is shown as a single entity, in the implementation the array is divided into two 16-PE sub-blocks which are clearly visible at the top and at the bottom of the photograph of Fig. 4. In this way the capacitive load and the peak current on the control lines is reduced and the delay is optimized; in addition, the 16 PE's block has a better aspect ratio and is more easily routed by automatic tools. The block diagram of a PE is shown in Fig. 5. Its main components are the Register File (RF) and the Execution Unit (EU).

The RF, introduced in Section 3.1, is composed of two sections, corresponding to the 48 LOR registers and to the 16 MOR registers. Address decoding is centralized for each 16 PE block in order to optimize its area and the decoded selection lines are run over the whole block. LOR's are implemented as 3-port static RAM cells, allowing the simultaneous execution of 3 operations (2 read and 1 write) on the RF at any given time.

Each MOR register is composed of five 1-bit RAM cells, as shown in Fig. 6. The central cell is similar to a LOR cell with 3 ports, while the other four are static RAM cells with a single read port. In addition all cells have an input port (SH) which allows data to be shifted from each cell to its right neighbor in a single clock cycle.

When executing a LOAD operation between the Image Memory or the VIF and a LOR register, the value of the 1-bit pixel is simply transferred in the register through one of the ports. When the same operation is performed on a MOR, the value from the memory (or the VIF) is loaded by the SH port into the leftmost cell and the contents of all the others is shifted one position to the right. In this way the central cell contains the current pixel of the image, the right cells the *north* neighbors with distance 1 and 2 and the left cells the *south* neighbors. When a read operation is executed on

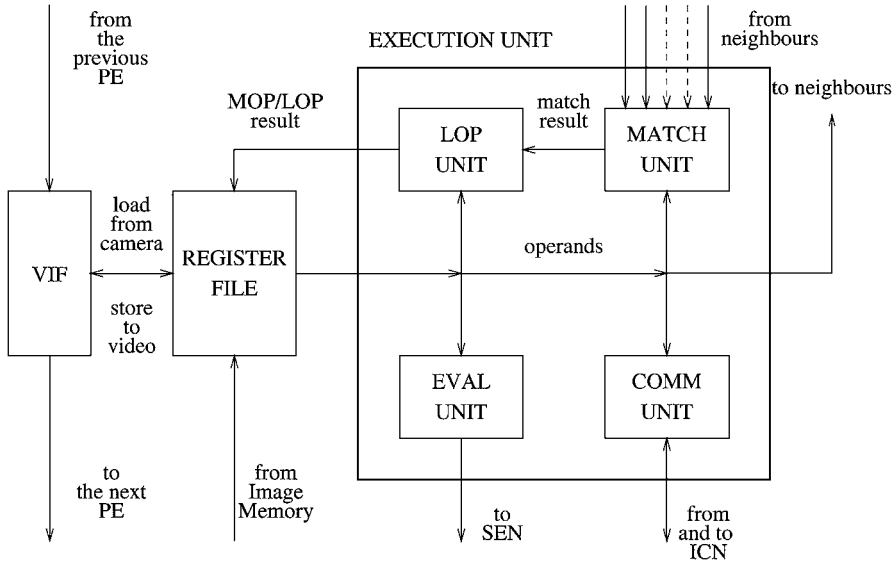


*Figure 5*. Block diagram of the processing element.

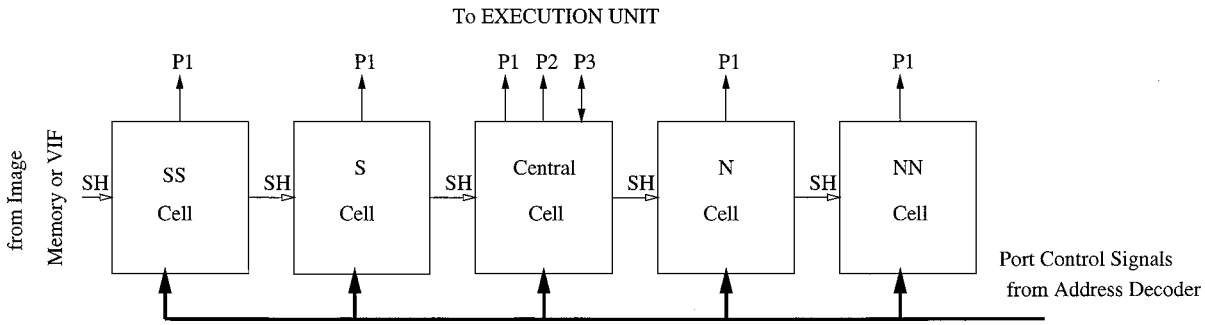


*Figure 6*. Structure of a MOR register.

port 1 of a MOR, the values of all cells are sent to the EU for the execution of neighborhood based operations. The execution time of a read/write operation from the RF is in the worst case lower than 10 ns and the data is latched at the output by the global system clock.

The register file of each PE integrates also one stage of the VIF. From a global point of view the VIF is a shift register with a number of stages equal to the number of PE's and with a 16 bit word width. It is divided in an Input and an Output section, each 8-bit wide, which are connected respectively to a pixel serial input and output device and may be clocked independently.

The operations of the VIF and of the Processor Array are independent and may proceed in parallel, overlapping I/O and processing. However they synchronize with each other when one of the two following events takes place:

- A full line of input data has been shifted in the Input section and the CU has fetched a Load From Camera instruction. In this case in each PE the 8 bits of the Input section of the local VIF stage are transferred to the southmost position of the first 8 MOR registers.
- A full line of data has been shifted out of the output section of the VIF and a Store To Video instruction has been fetched by the CU. When it happens the value of the first 8 LOR registers are transferred in parallel into the output section of the local stage of the VIF.

In order to minimize the transfer time and the interconnection area required, the Input and Output sections of the local VIF stage have been implemented directly inside the RF, close to the corresponding LOR and MOR registers.

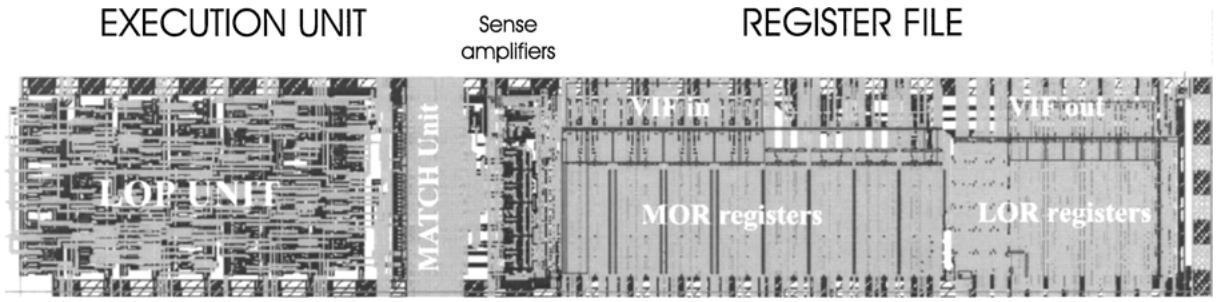


*Figure 7.* Layout of a processing element.

The EU performs the operations corresponding to the instructions of the program on the data from the RF, under the control of the CU. It is composed of 4 blocks:

- The *LOP* unit which is responsible for logical and arithmetic operations. It has been synthesized using a standard cell approach and the final layout has been routed manually to optimize the area.
- The *MATCH* unit which is responsible for MATCH operations, the base of all morphological and low level image processing operation. Its basic function is a comparison between a given neighborhood of the pixel and a vector of 3-valued templates (0, 1 and don't care) provided by the CU. The neighborhood of the pixel is obtained from the contents of a MOR register of the PE and from the four neighboring PE's. In order to reduce the silicon area and execute the operation in a single clock cycle the unit has been implemented with custom dynamic logic.
- The *COMM* unit which implements in standard cell one section of the ICN communication network depicted in Fig. 2(b).
- The *EVAL* unit which, when the corresponding instruction is executed, takes the contents of one register of the PE, masks it with the contents of another register and sends the results to the centralized part of the SEN for the evaluation of the different fields of the Status Word.

As explained in the next section, the Control Unit is able to concurrently execute more than one instruction which may in turn activate different functional units in the PE's. Hence, in order to obtain a correct execution, a data pipeline that reflects that of the CU had to be put in place to separate the units in the data path.

Figure 7 shows the layout of one PE which occupies approximately 1 $mm^2$ of silicon area. As clearly visible, all functional units have the same vertical dimension (horizontal in Fig. 6). Control signals run in the vertical direction across the PE which may be connected by abutment to its neighbors. The layout also shows how tightly the register file and the VIF I/O structure are integrated in order to obtain a high throughput in I/O operations.

### *4.2. The Control Unit*

As already mentioned, PAPRICA-3 exploits both spatial parallelism due to its massively parallel architecture, and instruction level parallelism due to the pipelined design of the control unit. Because of the very different nature of the instructions executed by the array and those executed directly by the control unit, the pipeline had to be designed with particular care both in its architecture and its implementation in order to obtain the best trade-off between complexity and performance.

In HACRE the Control Unit is located on each chip of a multi-chip implementation together with the WCS, thus decentralizing the issue of control signals. The WCS is loaded only once every algorithm while each instruction is usually executed many times (often thousands of times) per image. In most cases the overhead is thus reduced by orders of magnitude. The drawback of a similar implementation is the waste of chip area required for the duplicated logic: while this is still significant in the technology used for the design (0.8 $\mu$m), we anticipate a lower impact in deep sub-micron technologies where the interconnections, and not the logic, play the major role.

The main drive in the design of the pipeline is to obtain the highest performance for those sequences of instructions that are most used in image processing algorithms; these include, among others, sequences of morphological operations, loops on the entire image, and bit-serial computations. Although the relative

frequency of global and scalar instructions is far less than that of array instructions, their sometimes long latency due to their global nature may impose severe constraints to the execution of other instruction, negatively impacting the overall performance; neglecting their execution would therefore certainly lead to a suboptimal design.

For these reasons the pipeline has been logically divided into two sections which have been partially collapsed in the implementation to better exploit the hardware resources. The first section deals with scalar and global operations and has been designed as a conventional mono-functional queue; the second section controls the execution of array operations and employs a more complex multi-functional pipeline (which increases instruction level parallelism) and a more sophisticated conflict resolution mechanism. Despite the implementation as a single continuous structure, the two sections are designed to operate independently and they synchronize by means of mutual exclusion on shared resources during the execution of instructions that involve both array and global or scalar operations.

***4.2.1. Pipeline Implementation.*** Figure 8 shows the first section of the pipeline together with the supporting registers. Dotted lines represent data flow, while block lines represent the flow of the instruction codes.

The first stage of the pipeline (**IF**) is responsible for loading the instructions from the program memory. In order to achieve a high clock rate in spite of a relatively slow memory, instructions are fetched two at a time (technique known as *double fetch*). The entire mechanism is handled by this stage, so that to the rest of the pipeline it appears as if the memory were able to deliver one instruction per clock cycle.

The drawback of this technique is an increased penalty due to control conflicts that require to invalidate the initial stages of the pipeline, for example when a branch is taken (this problem is typical of superpipelined architectures where the queue is very deep). If the loop control in bit-serial and morphological computations were subject to this problem, the benefits obtained by the higher clock rate achievable by the double fetch technique would certainly be offset by the increased penalty. For this reason, the stage **J** of the pipeline is dedicated to the support of a hardware loop mechanism that controls the program counter and minimizes the negative effects. When the number of instructions in the loop is even (or if there is only one instruction), the pipeline is able to deliver one instruction per clock cycle with no control overhead. In all other cases the penalty is just one clock cycle per iteration. In the **ID** stage the scalar instructions get executed and leave the pipeline, while the array instructions are dispatched to the **AR** queue that computes the effective addresses for the internal registers and the image memory.

Figure 9 shows the second section of the pipeline. The stage **CD** is dedicated to the conflict resolution as will be explained later. The rest of the pipeline is

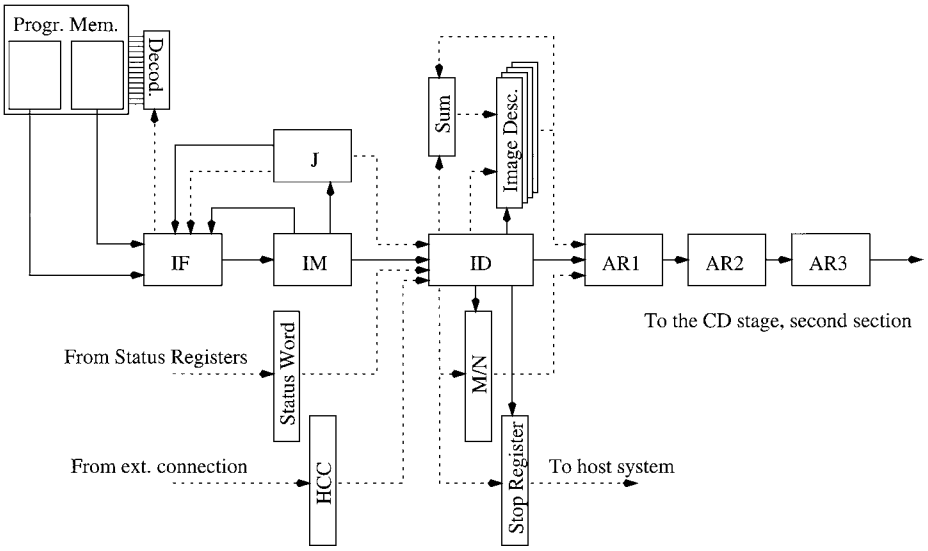


*Figure 8.* First section of the pipeline.

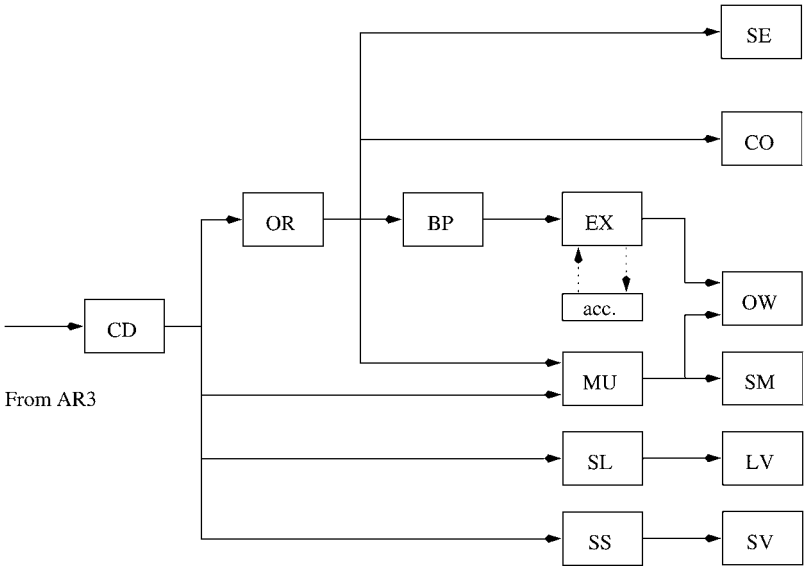


*Figure 9.* Second section of the pipeline.

a multi-functional queue, where each branch of the queue is dedicated to a particular feature of the array. In particular **SE** controls the Status Evaluation; **CO** the inter-processor communication mechanism; the sequence **OR-BP-EX-OW** executes the array operations by first reading the internal registers, then propagating the data to the neighborhood, then computing the result of the operation and finally writing back the result in the internal registers; **MU** handles the access to the image memory; and finally **SL-LV** and **SS-SV** is the path followed by the instructions that control the VIF.

Because of the multi-functional nature of the pipeline and the presence of different execution stages (including **ID** for the scalar instructions), it is possible that instructions be executed *out of order* (although the architecture still issues no more than one instruction per clock cycle). This feature enhances temporal parallelism and therefore performance. In addition, as mentioned earlier, the first and the second sections of the pipeline are decoupled, so that a stall in the second doesn't halt the execution of scalar instructions in the first. By doing this, the impact of the sequential part of the algorithm is minimized.

***4.2.2. Conflict Resolution.*** The concurrent execution of instructions gives rise to conflicts in the pipeline that must be resolved in the most efficient way. In this section we will present the mechanism employed to resolve conflicts involving the internal registers. In a multi-functional queue there are essentially two ways to detect a conflict condition:

*Scoreboard method*: In the scoreboard method, all conflicts are resolved in a single pipeline stage. A table, called the *scoreboard*, contains an entry for each resource in the architecture (e.g. registers). Every instruction, before entering the execution stages, verifies the availability of the source and destination operands, and once the permission is obtained, it gets possession of them by registering with the scoreboard. The resources are released only after the execution is complete.

*Tomasulo algorithm*: In the Tomasulo algorithm the conflict resolution is distributed in the pipeline. For example, the stage responsible for reading the operands checks their availability, as does the stage that writes them. Since the resolution is distributed, resources are reserved only when they are actually needed, so that an instruction is allowed to proceed even if only part of the required operands are available.

The advantage of the scoreboard method is a much simpler implementation, which is traded-off with the higher throughput achievable with the Tomasulo algorithm. HACRE adopts a combination of the two approaches. In a traditional implementation the scoreboard is basically a map where a set of flags signals the availability of a particular register for reading or for

writing. Given the number of registers in our architecture, the distributed access to this map that would be used in the Tomasulo algorithm results in a big and slow implementation. Our solution is to build a map that does not represent the registers, but rather the interaction among the instructions in the different stages of the pipeline.

Once in the **CD** stage, each instruction compares the effective addresses of its operands to those of the instructions in the *following* stages of the pipeline (in the picture, those to the right); in doing so, the instruction builds a map where each *stage* of the pipeline is marked with a flag: an active flag in the map means that there is a *potential* conflict between the instruction which owns the map and the instruction in the stage of the pipeline corresponding to the flag. The conflict is only potential because the instruction that builds the map has yet to be routed to the stages where the operands are needed: at that later time, the instruction that gives rise to the conflict might have already left the pipeline.

When our instruction leaves the **CD** stage, it brings the map with it; in addition, at each cycle, the flags in the map are updated to reflect the change in state of the pipeline, i.e. each flag is routed in the map to the stage where the corresponding instruction ends up. When an instruction corresponding to a flag leaves the pipeline, the flag is cleared in the maps. Note that each instruction has its very own different copy of the map, reflecting the fact that the conflicts are the expression of a precedence relation which involves two instructions at a time.

The data collected by each instruction in the map can then be used to establish whether a certain operand is available at a certain time, thus implementing the Tomasulo technique. Note however that by optimizing this approach not only is the collective size of the maps much smaller than that of a register map (approximately one third in our case), but the update of the flags is much simpler to implement than a multiple indexed access to the scoreboard in all stages.

Assuming a 2 clock cycle access time to the image memory, the pipeline has been measured to perform at about 2 clock cycles per instruction on real applications. Under these conditions, the speed-up obtained with respect to a conventional, non-pipelined controller running at a similar clock speed is close to 6. The use of more aggressive optimization techniques for the software may further improve these numbers.

## 5. The PCI Board

We designed a PCI board hosting up to four HACRE chips, a large image memory, a program memory, a system controller, the Status Register File and the PCI interface. A piggy-back connector provides a direct interface to an input imaging device, such as a linear scanner or a video camera and to an output device, such as a video monitor. Some bus drivers and the clock generation and synchronization logic are also part of the board. A block diagram of the board is depicted in Fig. 10.

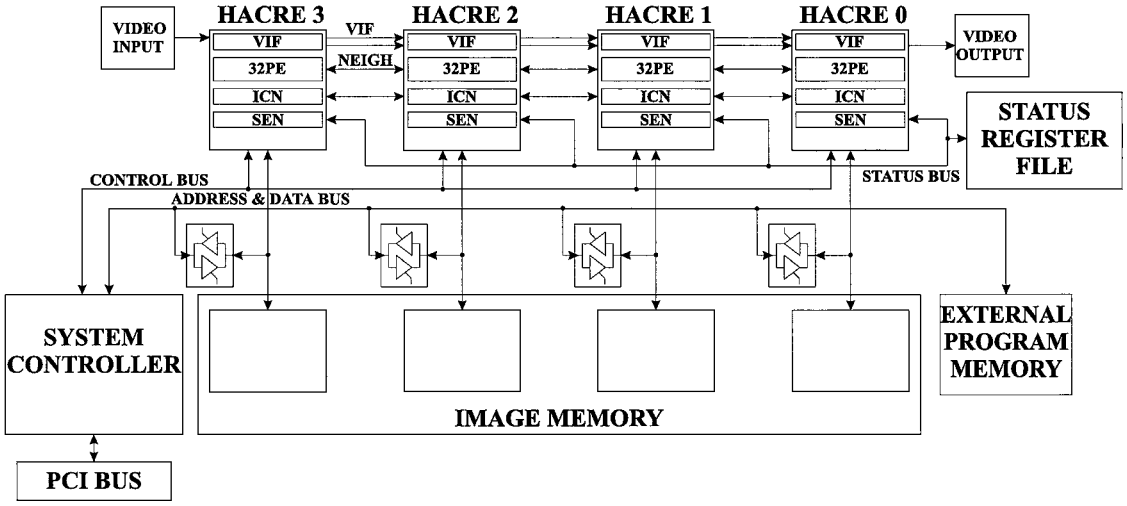


*Figure 10.* System block diagram.

The *Image Memory* is a fast static RAM (15 ns access time), $256K \times 32 \times n$ bits, where $n$ is the number of HACRE chips installed. One 32-bit memory module is associated to each of the chips. The memory is of exclusive use of the array during program execution, while it is memory mapped on the PC through the PCI interface when it is in STOP mode. Isolation is by means of bidirectional drivers.

The *External Program Memory* is a static memory module analogous to those used for the Image Memory and is used to store the complete HACRE program. It is mapped on the PC address space while in STOP mode. The System Controller autonomously transfers to the internal Writable Control Store the correct instruction block during program execution, so that a continuously repeated program, even if very long, can be down-loaded from the PC only once. The array outputs a completion word to the System Controller every time it completes the execution of the active program block. The completion word contains a code that directs the system controller to load a new block at a specific address or to signal program completion or an anomaly to the PC.

The *System Controller* provides several functions. It contains a set of registers, memory mapped on the PC address space, always accessible, that enable system management. It is also responsible of the control of the piggy-back board dedicated to image acquisition and display. When debugging an application it is sometimes desirable to test an algorithm on a well-defined set of images. To be able to do so without having to modify the HACRE program, the System Controller provides means of excluding the piggy-back I/O board and give the PC an access to the VIF. The PC writes data to a register and the System Controller shifts them into the VIF. The VIF output can also be redirected to a PC memory location for test purposes.

Different types of *Video I/O Boards* can be connected through the piggy-back video connector. This connector makes available to the interface board all video signals from the VIF structure, the Host Communication Channels and several control lines from the System Controller. We defined a simple interface both for digital video input from a camera or a scanner and digital video output to a monitor. Input data are shifted into HACRE asynchronously, using a pixel clock provided by the video source. At each end of line a handshake protocol insures that no data are missed or overwritten. Another handshake protocol allows the interface to read from the array the output video lines as soon as they are available. A frame buffer memory, a DAC and a simple scan logic are needed to provide an analog video signal for an output monitor.

The *Status Register File* (**SRF**) is used to accumulate and broadcast the global information collected by the Status Evaluation Network described in Section 3. When executing the EVAL instruction, every HACRE chip, using the Status Evaluation Network, calculates its Status Word, asserts a dedicated output line and suspends program execution. The EVAL line of the first chip is connected to the System Controller. The System Controller, using a 16 bit Status Bus, reads the Status Word from every chip. The COUNT field is accumulated, while the SET and RESET fields are ANDed to detect *all-ones* and *all-zeros* condition. The address field of the EVAL instruction is then retrieved from the first chip and used to store the result in the corresponding Status Register. The result is also propagated to the chips, which resume program execution and can test the result in conditional instructions. The same basic mechanism is used to reset, accumulate and normalize or read the contents of a specific register. A special SRF operation interrupts the PC in correspondence of a Status Register Write, thus allowing for non-linear or custom functions, implemented in software on the PC, to be used when accumulating values in a Status Register. The SRF is very useful in object classification algorithms. Every register stores the degree of matching of the image with a different template. By analyzing the SRF, the PC can interpret the image contents accordingly.

We designed the PCI board to cope with both the available chips, limited in frequency to 45–50 MHz, and the future ones running at 100 MHz. The clock is obtained from the system PCI clock. This was done to avoid synchronization problems between the PCI interface and the rest of the system, which have to interact. The array clock is derived from the PCI clock by a PLL multiplier. The PLL can generate 33, 50, 66 or 99 MHz frequencies (jumper configurable) from the 33 MHz PCI clock, with known phase relationship. This high frequency signal feeds the different HACRE chips. To be able to compensate for the different PCB line lengths of the clock signals and to comply with the stringent requirements on clock loading and delay on PCI specifications, we found very useful the adoption of skew buffers to control the clock signal feeding the different devices. These are PLL devices with four outputs at the same frequency as the input one, but with delays presettable via

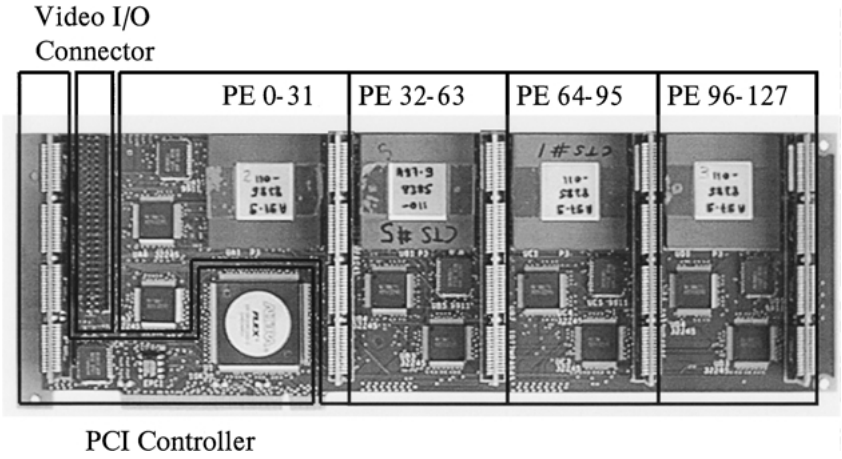


*Figure 11.* The PCI board.

jumpers. In this way it is possible to compensate for PCB transmission delays. All clock signals are one-to-one connections to avoid multiple reflections. All lines longer than few centimeters are series terminated at the source to match PCB line impedance. All high frequency signals run on internal layers of the PCB, shielded by ground or power supply planes, to limit EMI.

The PCI interface has PCI target capabilities and conforms to PCI 2.1 specifications. All of the board registers and memories are mapped on the PCI memory address space, to form a contiguous 8 MB memory block. The board can interrupt the PC to signal error conditions, end of run and to permit non-standard SRF operations. The PCI interface, the System Controller and the SRF were all implemented using a single FPGA (Altera 10K30), providing internal memory blocks and the availability of a large number of gates. This solution was adopted for both cost reasons and flexibility. A photograph of the PCI board is shown in Fig. 11.

## 6. Application Implementation

This section describes BEATRIX, that is an implementation of the proposed handwriting recognizer on the PCI board populated with: two HACRE chips (namely, 64 PE's); 2 high-speed SRAM chips (for a total of 2 MB of image memory); 256 KWord of program memory; 256 Status Registers; 6 + 6 direct I/O channels; a direct interface to an image scanner; PCI interface to a hosting 100 MHz Pentium PC.

The system has been tested and Section 7 shows its performance. See also [3] for additional details on the complete recognizer algorithm and performance.

### 6.1. System Description

BEATRIX integrates four logical subsystems which are in cascade [3] a mechanical and optical *scanner*, to acquire a bit-map image of the check; an *image preprocessor* for preliminary image filtering, scaling, and thresholding; a *neural subsystem*, based on an ensemble of morphological feature extractors and neuro-fuzzy networks, which detects character centers and provide hypotheses of recognition for each detected character; a *context analysis subsystem* based on a lexical and syntactic analyzer.

The *neural subsystem* carries out a pre-recognition of the individual characters, based on an *integrated segmentation and recognition* technique [14].

Legal and courtesy amounts are preprocessed and recognized independently (at the character level) and then the two streams of information are sent to the common *context analysis subsystem*, which exploits all the mutual redundancy.

The *context analysis subsystem* combines the candidate characters and, guided by the mutual redundancy present in the legal and courtesy amounts, produces hypotheses about the amount so as to correct errors made by the *neural subsystem* alone.

The *image preprocessor* and the *neural subsystem* are executed on the PAPRICA-3 system (namely, the PCI board), while the *context analysis subsystem* is

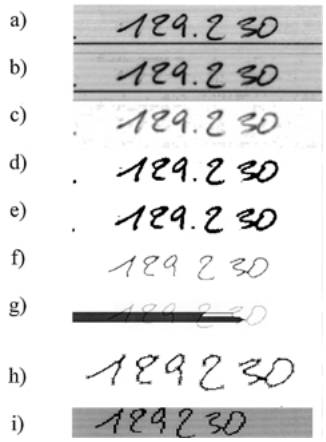


*Figure 12.* Preprocessing steps of handwritten image (an "easy" example): a) original image, 200 dpi, 16 gray levels; b) low-pass filtered image; c) compensated for brightness; d) thresholded; e) spot noise removal; f) thinned, after 6 steps; g) finding baseline (at the left side of the image); h) features detection (features are tagged by small crosses); i) compressed.

executed by an external Pentium processor, which can implement these types of algorithms more efficiently.

### 6.2. *Image Preprocessor*

The first preprocessing subsystem is the image preprocessor which consists of the blocks described below.

- A `WINDOW EXTRACTOR` acquires the input image from the `SCANNER`, at a resolution of approximately 200 dpi, 16 gray levels. The scanner is an 876-pixel CCD line camera scanned mechanically over the image, from right to left (due to practical reasons), at a speed of 2 m/s (which is equivalent to about 700 characters/s); Fig. 12(a).
  Image acquisition is performed by the VIF, in parallel with processing, and a whole image line is acquired in just one clock cycle.
- A `FILTER` block computes a simple low-pass filter with a $3 \times 3$ pixel kernel (Fig. 12(b)), while a `BRIGHTNESS` block compensates for the non-uniform detector sensitivity and paper color (Fig. 12(c)).
- A `THRESHOLD` block converts the gray-scale image into a B/W image by a comparison with an adaptive threshold (Fig. 12(d)).
- A `THINNING` block reduces the width of all the strokes to 1 pixel (Fig. 12(f)). Thinning is a morphological operator [4] which reduces the width of lines, while preserving stroke connectivity.
- A `BASELINE` block detects the *baseline* of the handwritten text, which is a horizontal stripe intersecting the text in a known position (Fig. 12(g)).
- A `FEATURES` block detects and extracts from the image a set of 12 *stroke features*, which are helpful for further character recognition. As shown in Fig. 12(h) (crosses), this block detects the four *left*, *right*, *top* and *bottom concavities*, and the *terminal strokes* in the eight main directions.
- A `FEATURE REDUCTION`, a `ZOOM` and a `COMPRESS` blocks reduce, respectively, the number of features (by removing both redundant and useless ones), the *vertical size* of the manuscript (to approximately 25–30 pixels), and the overall size of the manuscript (by a linear factor of 2), by means of ad-hoc topological transformations which do not preserve image shape, although they do preserve its connectivity. (Fig. 12(i))

After all the preprocessing steps, the B/W image is ready for the following neural recognition steps (see Section 6.3). The image is reduced both in size (down to $14 \times 18 (= 252)$ or $12 \times 21 (= 252)$ pixels for the courtesy and the legal amounts, respectively), in number of gray levels (2), and in stroke thickness (1 pixel), and noise is removed. Table 7 lists execution times of individual blocks.

### 6.3. *Neural Subsystem*

The *neural subsystem* is made of two cascaded subsystems, namely a `CENTERING DETECTOR` and a `CHARACTER RECOGNIZER`. See [3] for further details on the algorithms and the implementation of each block. Table 1 lists execution times of individual blocks.

- The `CENTERING DETECTOR` scans the preprocessed and compressed image from right to left (for mechanical reasons) and extracts a *sliding window* of fixed size. It then tries to locate the characters, by detecting the position of their center, based on the type, quantity and mutual positions of the detected features.
  Note that windows without strokes are immediately skipped, as they contain no useful information. Lines can be skipped in as low as one clock cycle.
- The `CHARACTER RECOGNIZER` recognizes each individual character, using a hybrid approach, which mixes feature-based [2] and neural [1] recognizers.

*Table 1.* Average execution times of the various processing steps, while processing the courtesy amount.

| | PAPRICA-3, 64PE's, 33 MHz | | Pentium 100 MHz | | Sparc 10 |
|---|---|---|---|---|---|
| *Image preprocessor* | Worst case ($\mu$s/line) | Worst case (ms/check) | morphol. ($\mu$s/line) | ad-hoc ($\mu$s/line) | morphol. ($\mu$s/line) |
| WINDOW EXTRACTOR + FILTER | 2.76 | 5.52 | 1,700 | 705 | 1,370 |
| BRIGHTNESS | 5.90 | 11.8 | 1,970 | 320 | 1,660 |
| THRESHOLD | 1.82 | 3.64 | 830 | 255 | 570 |
| THINNING | 16.7 | 33.4 | 7,390 | – | 7,850 |
| BASELINE | 8.48 | 16.6 | 4,320 | – | 3,890 |
| FEATURES | 6.10 | 12.2 | 9,490 | – | 10,430 |
| ZOOM | 4.48 | 8.96 | 820 | 160 | 760 |
| COMPRESS[a] | 61.6 | 123.2 | 21,350 | – | 24,950 |
| OTHER (VARIOUS) | 6.50 | 13.0 | 3,330 | – | 5,020 |
| TOTAL PREPROCESSING | 114.4 | 228 | 51,200 | – | 56,500 |

| | PAPRICA-3 | | Pentium 100 MHz | |
|---|---|---|---|---|
| Neural subsystem | Worst case (ms/char) | Worst case (ms/check) | morphol. (ms/check) | ad-hoc (ms/check) |
| CENTERING (FEATURES) | 0.80 | 38.4 | 1,440 | – |
| RECOGNIZER (FEATURES)[b] | 7.20 | 345.6 | 12,960 | – |
| RECOGNIZER (NEURAL)[b] | 3.36 | 61.4 | – | 13,440 |
| Total Recognizer | 11.4 | 545.4 | 27,840 | |

[a] COMPRESS acts on an image zoomed by an average factor 3.2, therefore processing times are scaled accordingly.
[b] CHARACTER RECOGNIZER acts a few times per each character, namely once every 15 lines on average.

First of all, features extracted by the FEATURES block are used to identify all easy-to-recognize characters. For instance, most "0", "6", "9" digits (but not only these) are written well enough that a straightforward and fast analysis of the main features and strokes is sufficient to recognize those characters with high accuracy.

Other characters are more difficult to recognize using only features; for instance, digits "4", "7" and some types of "1" can be recognized more easily using neural techniques. All characters which have not been recognized using features are isolated and passed to a neural network (an ad-hoc 2-layers WRBF [15]) trained by an appropriate training set.

Therefore the CHARACTER RECOGNIZER is made of two blocks, namely a *feature-based recognizer* and a *neural recognizer*, each one optimized to recognize a particular subset of the whole alphabet.

The CHARACTER RECOGNIZER is "triggered" for each character center detected by the CENTERING DETECTOR. As shown in Table 1, the CHARACTER RECOGNIZER is the slowest piece of code. Fortunately it is run at a relatively low rate, namely every 15 lines, in the average, therefore its effects on computing time are limited.

## 7. Performance

Table 1 lists the execution times of the various processing blocks for the example presented in Section 2; figures are given for a system with 64 PE's (namely, 2 chips), running at 33 MHz. All the programs were also tested on both a Pentium at 100 MHz and a Sparc Station 10, using the same algorithms based on mathematical morphology, which are well suited to the specific problems of bitmap processing and character recognition.

Some programs (FILTER, BRIGHTNESS, THRESHOLD, ZOOM, CENTERING DETECTOR, CHARACTER RECOGNIZER) could be implemented more efficiently on a sequential computer using more traditional methods (ad-hoc programs). These were also implemented on

*Table 2.* Performance of the BEATRIX system, measured on 80 checks.

| Module | Parameter | Courtesy | Legal |
|---|---|---|---|
| `CENTERING DETECTOR` | RMS centering error (distance between detected center co-ordinate $x_k$ and actual (geometrical) of character) | 2.5 pels | 3.0 pels |
| `CHARACTER RECOGNIZER` | Average accuracy rate (correct, if the character is in the first five positions in the list $\hat{\mathcal{L}}(x_k)$) | 74% | 52% |
| *Clustering subsystem* | Ratio between number of correct clusters and number of hypothesized clusters | 87% | 74% |
| `CHARACTER RECONGNIZER` | Average accuracy rate (on all centers of the correct clusters) | 88% | 65% |
| *Context analysis subsystem* | Average accuracy rate (correct, if the correct amount is in the first 4 positions of the list of final amounts) | 53% | |

*Table 3.* Performance of the BEATRIX system, in single-chip configuration (namely, 32 PE's), running at 100 MHz, with either internal weights (max. 60 bits/neuron) or external weights (no size limitation).

| | Internal weights | | External weights | |
|---|---|---|---|---|
| Neural Network Paradigm | MCPS | MCUPS | MCPS | MCUPS |
| MLP, $32 \times n$ inputs, 1 bit/input, 8 bits/weight | 130 | 60 | 90 | 50 |
| MLP, $32 \times n$ inputs, 4 bit/input, 16 bits/weight, adaptive learning rate | 33 | 19 | 30 | 16 |
| Kohonen $1 \times 30$ neurons, 8 inputs, 8 bits/input, $1 \times 5$ neighborhood | 110 | 60 | | |
| Kohonen, $30 \times 30$ neurons, 8 inputs, 8 bits/input, $5 \times 5$ neighborhood | | | 90 | 49 |

MCPS and MCUPS stand for, respectively, *mega connections per second* and *mega connection updates per second*.

the Pentium and their performance listed in Table 1 for comparison.

It can be seen that the performance of PAPRICA-3 is 100 to 1000 times faster than that of Pentium 100 MHz and Sparc Station, for nearly all the programs considered. This improvement factor reduces by at most five folds when a Pentium 500 MHz is used.

Table 2 lists the recognition performance of the major processing blocks. Performance is given in terms of accuracy rates, for the `CHARACTER RECOGNIZER` and the *context analysis subsystem*, and in terms of RMS centering error, for the `CENTERING DETECTOR`.

Figure 13 shows the percentage of checks correctly recognized (in the first 4 positions) versus the position of the correct amount in the list of hypothetical courtesy amounts.

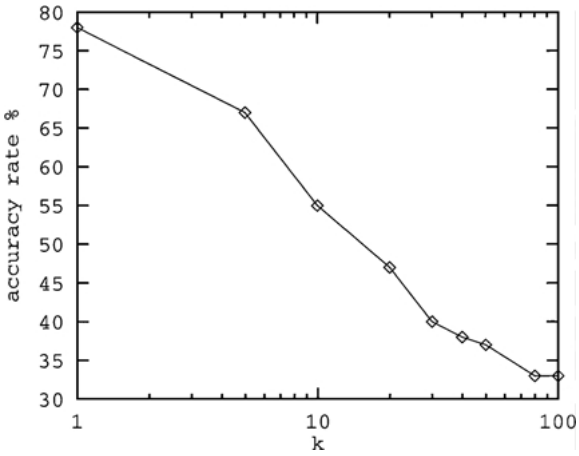


*Figure 13.* Accuracy rate versus the position $k$ in which the correct courtesy amount is found.

The CENTERING and the RECOGNIZER average execution times per character are calculated as the ratio between the average execution time of the amount and the average number of characters of the amount. As the average numbers of characters are 6.9 and 27.4, for the courtesy and the legal amounts, respectively, we obtain the data in Table 1.

Note that the recognition performance of BEATRIX is comparable with other existing handwriting recognition systems [16] but its speed is about 100 times faster than that of Pentium and Sparc implementation. A recognition speed comparable with our system, with the same recognition accuracy, can only be achieved with much more expensive array of processors (e.g., CNAPS [17], Transputers, DSP's).

For further comparison,Table 3 lists the execution times of a single-chip PAPRICA-3 system running at 100 MHz, for other well-known neural algorithms such as Perceptrons (MLP) and Kohonen maps [5]. As all mathematical operations are implemented in a bit-serial fashion, system performance depends heavily on input and weight resolution. Furthermore, the best performance can be obtained when the number of either neurons or inputs matches the number of PE's.

## 8. Conclusion

From the hardware implementation point of view not all the original goals have been reached. In particular all the main full custom blocks (memory, PA) have been designed and verified by simulation to operate within a 10 ns clock cycle in the worst case, but the whole chip is fully functional up to a maximum frequency of 50 MHz. This is due to strict deadlines on the availability of funds for chip fabrication which have reduced the time available for the optimization of the CU layout on the basis of back-annotated simulation.

Moreover, as clearly visible in the microphotograph of Fig. 4, a large area of approximately 10 $mm^2$ is wasted. This is mainly due to the limitations of the tools employed in the placing and routing phase of the CU which has been synthesized into a standard cell library form a HDL description, This has led to a large increase of the ratio between the CU area and the PA area and Preliminary tests with new tools have shown that the current layout size could be reduced by approximately 15%. This would allow to place on board of a single chip system an integrated version of the Status Register File [18] which has been designed in order to minimize the components of a single chip system.

With the current technological evolution of VLSI circuits preliminary evaluations made with a $0.35\mu$m technology have shown that a single chip system could integrate 64 PE's, the SRF and 64Kbit of image memory, making it possible a really fully integrated system for handwritten character recognition.

For what system performance concerns, the performance of the BEATRIX system have shown that the proposed PAPRICA-3 architecture, even in a medium-size configuration, outperforms the Pentium processor by much more than a factor ten. In addition, the recognition accuracy of BEATRIX is comparable with other much more expensive systems [16].

In addition the development environment and the image processing language (not described here), which have been developed explicitly for PAPRICA-3 allow a straightforward design of new mathematical morphology and image processing algorithms, reducing the design time of new algorithms.

## Acknowledgment

This work has been partially supported by the EEC MEPI initiative DIM-103 *Handwritten Character Recognition for Banking Documents.*

## References

1. R.M. Bozinovic and S.N. Srihari, "Off-line Cursive Script Word Recognition," *IEEE Trans. on PAMI*, vol. 11, 1989, pp. 68–83.
2. H.L.Teulings,"Invariant Handwriting Features Useful in Cursive-Script Recognition," in *Fundamentals in Handwriting Recognition*, S. Impedovo (Ed.), Berlin: Springer-Verlag, 1993, pp. 179–198.
3. B. Lazzerini, L.M. Reyneri, F. Gregoretti, and A. Mariani, "High-Speed Recognition of Handwritten Amounts on Italian Checks," *Knowledge-Based Intelligent Techniques in Character Recognition*, L.C. Jain and B. Lazzerini (Eds.), Florida: CRC Press, 1999, pp. 167–189.
4. J. Serra, *Image Analysis and Mathematical Morphology*, Academic Press, London: 1992.
5. S. Haykin, *Neural Networks: A Comprehensive Foundation*, New York: Mc Millan College Publishing Company, 1994.
6. F. Gregoretti, L.M. Reyneri, C. Sansoé, and L. Rigazio, "A Chip Set Implementation of a Parallel Cellular Architecture," *Microprocessing and Microprogramming*, vol. 35, Amsterdam (NL): North-Holland, 1992, pp. 417–425.
7. A. Broggi, G. Conte, F. Gregoretti, C. Sansoé, and L.M. Reyneri, "The Evolution of the PAPRICA System," *Integrated Computer-Aided Engineering*, vol. 4, no. 2, New York (NY): John Wiley & Sons, 1997, pp. 114–136.
8. A. Broggi, G. Conte, F. Gregoretti, C. Sansoé, R. Passerone, and L.M. Reyneri, "Design and Implementation of the PAPRICA

Parallel Architecture," *Journal of VLSI Signal Processing*, vol. 19, 1998, Amsterdam (NL): Kluwer Academic, pp. 5–18.
9. N. Yamashita, T. Kimura, Y. Fujita, Y. Aimoto, T. Manbe, S. Okazaki, K. Nakamura, and M. Yamashina, "A 3.84 GIPS Integrated Memory Array Processor with 64 Processing Elements and a 2-MB SRAM," *IEEE Journal of Solid-State Circuits*, vol. 29, no. 11, 1994, pp. 1336–1342.
10. H.M. Chang, M.H. Sunwoo, and T. Cho, "Implementation of a Slim Array Processor," in *Proceedings of the 10th International Parallel Processing Symposium*, Honolulu, HI, April 15–19, 1996.
11. H.M. Chang, S. Ong, and M.H. Sunwoo, "A Linear Array Parallel Image Processor: Slim-ii," in *Proceedings of the International Conference on Applications-Specific Systems, Architecture and Processors*, Zurich, Switzerland, July 14–16, 1997.
12. D. Helman and J. JáJá. "Efficient Image Processing Algorithms on the Scan Line Array Processor," *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 17, no. 1, 1995, pp. 17–56.
13. R.B. Yates, N.A. Thacker, S.J. Evans, S.N. Walker, and P.A. Ivey, "An Array Processor for General Purpose Digital Image Compression, " *IEEE Journal of Solid-State Circuits*, vol. 30, no. 3, 1995, pp. 244–249.
14. M. Schenkel, H. Weissman, I. Guyon, C. Nohl, D. Henderson, "Recognition-based Segmentation of On-line Hand-printed Words," *Advances in Neural Information Processing System 5*, Morgan Kaufmann, 1993, pp. 723–730, Denver.
15. L.M. Reyneri, "Unification of Neural and Wavelet Networks and Fuzzy Systems," *IEEE Transactions on Neural Networks*, vol. 10, no. 6, July 1999, pp. 801–818.
16. L.C. Jain and B. Lazzerini (Eds.), "High-Speed Recognition of Handwritten Amounts on Italian Checks," *Knowledge-Based Intelligent Techniques in Character Recognition*, Florida: CRC Press, 1999.
17. Adaptive Solutions, "CNAPS" data sheet and manuals.
18. F. Gregoretti, L.M. Reyneri, and C. Sansoé, "An Intelligent Register File for Neural Processing Applications," in *Proceedings of WDTA '98, Workshop on Design, Test and Applications*, Dubrovnik, Croatia, June 1998, pp. 41–44.

**Francesco Gregoretti** received in 1975 the Dr. Eng. degree in Electronic Engineering from Politecnico di Torino where he started his academic activity as Assistant Professor in 1978 and where now is Associate Professor of Microelectronics. From 1976 to 1977 he was at the Laboratoire de Calculatrices Digitales at the Swiss Federal Institute of Tecnology in Lausanne (Switzerland) and from 1983 to 1984 Visiting Scientist at the Department of Computer Science of Carnegie Mellon University, Pittsburgh (USA). His main research interest are in digital electronics and in particular in MOS VLSI circuits, in massively parallel multimicroprocessor systems for VLSI CAD tools, in low level image processing architectures and in cellular neural networks. Prof. Gregoretti is a member of IEEE and has authored or coauthored more than 90 papers and 3 books.
gregor@polito.it

**Roberto Passerone** is a Senior Member of Technical Staff at Cadence Design Systems where he researches topics in system level design, including system description languages, synthesis and formal methods. He is also involved in the design of hardware and software for wireless protocols and of high performance computer architectures. Roberto received his Laurea degree from Politecnico di Torino in 1994, his MS degree from the University of California at Berkeley in 1997 and his doctorate degree in electrical engineering from Politecnico di Torino in 1998.

**Leonardo Maria Reyneri** received the Laurea (M.Sc.) and the Dottorato (Ph.D.) degrees from Politecnico di Torino, Italy, in 1984 and 1992, respectively. He is currently an associate professor of Electronics at the Politecnico di Torino. He teaches applied electronics and neuro-fuzzy systems, and carries out research on applications of neural networks to intelligent control and pattern recognition. He is also involved in the design of VLSI implementations of high-performance neural networks and parallel SIMD architectures. Recent activities also involve design of low-power and low-EME digital circuits, including adiabatic logic. His fields of personal research interest include the design of low-power mixed analog/digital integrated circuits for robotics, and the development of dedicated architectures for neural networks, massively parallel systems and image processing. L.M. Reyneri is a member of IEEE, has published more than 120 papers on scientific journals and conference proceedings, and holds five patents.

**Claudio Sansoé** received the Dr. Eng. degree in Electronic Engineering from Politecnico di Torino, Italy, in 1984. Since 1987 he is with Politecnico di Torino, Department of Electronics, where he currently teaches applied electronics and digital systems. His main research interests are in the design of high speed architectures and systems dedicated to real-time image analysis, in the development and testing of hardware-software codesign methodologies for embedded systems and in the experimentation and introduction of new multimedia technologies in university level Electronics courses.