An FPGA-based Topographic Computer for Binary Image Processing

Alejandro Nieto, Víctor M. Brea and David L. Vilariño
Department of Electronic and Computer Science
University of Santiago de Compostela
Spain

1. Introduction

Initial processing steps in a computer vision system usually consist of low complex operations replicated on all the pixels of the image under processing. These operations are often very time consuming, particularly when they are executed on conventional serial computers. Every pixel-level processing represents millions of operations when moderate image sizes are considered. These processing stages provide the set of features of interest (object edges, movement, area, shape,…) with a considerable reduction of the data flow to subsequent processing stages. Higher level processing will benefit from such data to produce an abstract description of the scene and, if needed, to make a decision.

In visual processing, a great deal of computation is made on B/W images. This is particularly relevant for intermediate level image processing. Pattern recognition, texture detection and classification, or prediction and estimation of motion, are some examples of operations managing binary data. This fact has motivated an intense research focused on the design and implementation of efficient algorithms dealing with 1-bit data flows.

Traditionally, most of the algorithms in computer vision are executed on general-purpose computers using data from imagers as inputs. Recently, new bioinspired systems integrate together both sensing and processing stages, exploiting the massively parallelism inherent to early vision. These vision systems are usually analog or mixed-signal implementations allowing high density of integration as well as fast computation, with low/moderate accuracy. Often, these systems are not optimized to compute B/W images. Nevertheless, in some practical applications, (e.g., tracking with cellular active contours), the set of operations involving binary images are the main bottleneck from the time performance point of view. Keeping in mind the main constraints featuring integrated circuits: speed, area and reliability; the design of a special unit for binary image processing would be advantageous to be part of a complete computer vision system or even like an independent module.

As for the suitable hardware platform, the ever larger availability of hardware resources and faster clock signals on today FPGAs permit to do image processing with similar time-efficiency to that of custom ASICs from the recent past. Also, the time to market of FPGA-based systems clearly outperforms that of ASICs. Altogether makes FPGAs a good candidate to host proof-of-concept systems or even ended solutions.

Source: Image Processing, Book edited by: Yung-Sheng Chen,
ISBN 978-953-307-026-1, pp. 572, December 2009, INTECH, Croatia, downloaded from SCIYO.COM

In this chapter, a pixel-parallel binary image computer is approached. The architecture consists of a 2D array of processing elements based on the simple instruction and multiple data paradigm. The spatial arrangement allows to associate one processor to one pixel or to a reduced number of pixels of the image and consequently to exploit the inherent parallelism in visual computing. A 48x48 processor array has been implemented and tested on a Xilinx Virtex II FPGA. Several examples of practical applications are included to show the efficiency of the proposed system. In addition, based on the International Technology Roadmap for Semiconductors an estimate of how the increasing integration density will affect both FPGA and ASIC solutions is discussed. From this study, some conclusions about the capabilities of state-of-the-art or near-future generations of FPGA to host processor arrays with practical size are made.

2. Why a binary image computer?

The development of a computer vision application is divided into three stages. In the first, *low level vision*, acquisition and pre-processing are performed in order to highlight the features of interest to the specific application. The second, known as *mid-level vision*, has as goal to process information from the previous stage to extract a set of characteristics that define the problem to be solved: objects features, textures, edges, etc. The last stage, *high-level vision*, uses these results to obtain a more abstract description of the scene and makes decisions based on their content (Eklundh & Christensen, 2001).
The mid-level processing steps uses a lower knowledge of the domain of the problem and a significantly lower level of abstraction than the high level vision. In the same way, the data type used also differs. Low and mid-level processing images are represented by arrays, e.g., intensity values, while high level vision only uses the relevant data, usually in a symbolic fashion, greatly reducing the information used. These data represent some kind of knowledge, like the size of an object, its shape or its relationship with other objects. In the mid-level steps, the original image is processed in order to extract these features: segmentation, extraction of the number of objects, their shape, area, etc.
The amount of data to handle is very high, the operations that are performed have low complexity and they are massively parallel in a natural way, unlike what happens during the high level processing. The high computational effort, especially when dealing with high-resolution images or with applications with very demanding time requirements, involves processing millions of data per second. One strategy to tackle this computational power is the parallelization since the processing of each pixel is independent of each other, being this one of the main characteristics of mid-level image processing.
Another strategy to increase the time performance is to have an adequate representation of the data contained in the image arrays. In other words, the fewer the number of bits, the faster the processing. Thus, many algorithms employ gray-scale images instead of a color representation. A more aggressive optimization is to process binary images, i.e., 1 bit per pixel. Binary images are the simplest type of image. They have many other advantages, including easy acquisition, lower storage and transmission requirements and greater simplicity of the algorithms so they are widely used in many industrial and medical applications. It is true that in certain cases, such as the image segmentation in certain environments (Cheng et al., 2001) or the detection and treatment of shadows (Prati et al., 2003), using color images eases the resolution of the problem and can not be used representations in gray-scale or purely binary. However, there are problems that can be

solved using only binary images (Szolgay & Tomordi, 1996) (Spiliotis & Mertzios, 1997) (Rekeczky, 1999) (Tseng et al., 2002). Moreover, many algorithms, including those processing color images have modules that use binary images (Garcia & Apostolidis, 2000) (Hsu et al., 2002) (Vilariño & Rekeczky, 2005) (Park et al. 2007).

A binary image can contain two types of information. On the one hand, a scene, i.e., partial information on the intensity of radiation can be stored. This information can come from a binary sensing or a transformation of an image, e.g. a thresholding or a halftoning of a gray-scale image. Furthermore, information contained in a binary image goes beyond shrinking the range to a 1-bit representation. The *active pixels* store information like edges, motion, occurrences of patterns, etc. This information enhances the original scene and speeds up the computation. Thus an algorithm can work with multiple binary images, representing the position of an object, the edges, and so on.

The retinal vessel-tree extraction algorithm (Alonso-Montes et al., 2008) is a good example of all the facts mentioned above. This approach is based on active contours, fitting the vessels from the outside. It consists of a controlled region growing started from seeds placed out of the vessels. The first step is a pre-segmentation of the vessels to obtain the guiding information for the active contour and its initial position. Next, the evolution is done. This algorithm was designed to operate in a pixel-parallel fashion. Some of the images, as the retinal image or the guiding information, are gray-scale images. Nevertheless, the contour evolution is performed using binary images because regions, contours, collision points, etc, only need 1 bit for its representation. In addition, as it can be extracted from (Alonso-Montes et al., 2008), the pre-segmentation step consumes the minor part of the computation. This algorithm was implemented in a vision chip (see Section 3) and this step consumes only the 3% of the time of the global computation. As a result, to speedup the computation is needed not only to pay attention to the more complex gray operations as additions or convolutions, but also to the simpler ones because in many algorithms they will limit the performance seriously.

3. Hardware for image processing

The most straightforward platform to implement a computer vision application is the personal computer. The main advantage of the *Microprocessors* is their versatility, the ability to perform many different tasks with a low cost. This permits any type of processing, although its performance is not always adequate. Even in this case it allows to simulate and study a wide range of applications, although the final implementation is done on another platform. The current trend is the growing number of processor cores due to the limitations in frequency scaling and power dissipation due to the transistor scaling (Creeger, 2005) (Schauer, 2008). They include parallel computation units to accelerate the computation, known as SIMD extensions (Gepner & Kowalik, 2006). Among other components in a personal computer, we must highlight the *Graphics Processing Units*, GPUs, because they reduce the workload of the main processor. Because of their specialization and because they include several hundred processing units, they have higher computing power. It is now possible to use them as generic accelerators using the concepts of GPGPU, *General Purpose GPU*. Nevertheless, they have certain limitations because of their fixed architecture, although the new families of graphics processors substantially improve its functionality (Owens et al., 2008) (Shen et al. 2005) (Soos et al., 2008).

A *Digital Signal Processor*, DSP, is a system based on a processor with an instruction set, hardware and software optimized for highly demanding applications with numerical operations at high speed. They are especially useful for processing analog signals in real time. They are systems that incorporate digital converter AD/DA for communication with the outside world (Tan & Heinzelman, 2003). Its flexibility comes from the ability to work with multiple data in parallel, its accuracy and expertise to the digital processing, and where the limitations of power and range are significant. However, it is not the most widely used platform for medium level operations by not being able to exploit the characteristics of such operations. However, for early vision tasks or higher-level, such as decoding or video compression, are a serious alternative to other platforms (Akiyama et al., 1994) (Hinrichs et al., 2000). A DSP reduces the hardware requirements by eliminating the no necessary hardware present in personal computers.

To meet the high computational cost of computer vision applications it might be necessary to perform the processing steps in the focal plane. A *Vision Chip* or a *Focal Plane Processor Array* is, in a generic way, an array of simple processors with an associated image sensor, with a one to one correspondence between processors and pixels of the image. These systems have a high performance and, if sensors are available, the bottleneck of uploading the image diminishes. Vision Chips work usually in SIMD mode. SIMD, *Single Instruction Multiple Data,* is a technique used to achieve a high degree of parallelism in data processing. This technique, originally used in supercomputers, is present now in all personal computers. The SIMD refers to an instruction set that are applied the same operation on a broad set of data. A single control unit manages the process and each processor operates on different data sets. They are executed synchronously in all the processors. This paradigm concerns only the way that the control over the operations is done, not the internal structure of the processing units or their spatial organization. Including local bus and distributed memory it evolves into Processors Arrays. The main limitation of Vision Chips is the reduced area available for each processor, cutting the resolution. However, the performance and power consumption achieved is difficult to obtain with other platforms (Linan et al., 2001) (Dudek, 2005) (Anafocus, 2007).

A *Field Programmable Gate Array*, FPGA, is a device that contains programmable logic components by the user. These components are formed by a set of logical blocks with a network of interconnections also programmable. In this way, the blocks are programmed to emulate complex combinational functions that are then chained together using interconnects as buses. The major advantage of FPGAs is their internal interconnect capability, with a complex hierarchy of connections optimized for specific functions. Moreover, they allow to develop a highly parallel hardware with a very low cost compared to an integrated circuit. As the description is done using *hardware description language*, HDL (Maginot, 1992), the designer does not have to worry about the low level electronics problems, reducing the time-to-market. Hence they are widely used in many areas as accelerators. The inclusion of specific hardware as dedicated multipliers or memory blocks, DSP or even microprocessor units, makes it possible to consider an FPGA a System on Chip with a reduced cost. Recent advances in their development lead to a similar performance to recent specific integrated circuits, but with a lower cost. For this, FPGAs are widely used in image processing tasks (MacLean, 2005). However, they have certain limitations, as it is necessary to design a new hardware, as well as control units, for each application, the designer must be an expert in hardware, software and the algorithm to be implemented. Besides, the performance will

depend on the particular implementation done. It should be noted that when compared to the platforms described above, microprocessors, vision chips or other programmable systems, an FPGA implementation not only obliges to do an efficient adaptation of the algorithm, but also a new hardware design, so the required skill level is higher.
An *integrated circuit*, IC, is the miniaturization of an electronic circuit composed of semiconductor devices and passive components fabricated on a semiconductor surface. Its biggest advantage is the miniaturization, allowing building large systems with processing low power and area requirements and high throughput. An ASIC, *Application-specific integrated circuit*, is an IC designed for a particular use, but concepts like SoC (System on Chip), which integrate the whole system on a single chip, or SiP (System in Package), where multiple chips are piled up, are increasingly extended. They include, in general, a microprocessor or microcontroller, memory blocks, digital interfaces such as USB or Ethernet, analog modules, such as DACs, etc. They are designed from scratch and a large number of parameters can be modified. On some occasions, the CMOS technology provide component libraries which can greatly cut down the design time. Anyway, the design is costly in terms of time and manufacturing, and except when the number of units produced is high or when the application is very specific or it has very tight specifications, solutions as FPGAs become a serious alternative (Courtoy, 1998).

4. SIMD array for binary image processing

Low and medium image processing levels have an inherent high degree of parallelism at algorithmic level. This fact is exploited during the design and implementation of new hardware. The increase in parallelism is now in the leading-edge commercial microprocessors. The inclusion of multiple processor cores can greatly speed up computation. However, this is not enough when we talk about early vision. Moreover, the hardware specialization in floating point units is not fully suitable for image processing.
In a natural way, Single Instruction Multiple Data architecture is able to exploit such a massive parallelism. The creation of small processing units that only have to perform simple calculations in a matrix disposition, where each element of the matrix is associated with a pixel of the image, and with local connections matches the nature of early vision algorithms.
The relationship between the size and the computing power of the processing elements is critical. Small processing units allow bigger arrays at the expense of not being able to perform complex operations. Operations have to be segmented, increasing the number of required clock cycles, but with a higher level of parallelism. However, low complex Processing Elements, PEs, shorten the critical path of the design, decreasing the cycle time, so performance will not be heavily penalized with highly complex PE. In the latter, the level of parallelism is limited but complex operations can be performed easier. The complexity of the hardware and the computational power must be balanced. In the proposed architecture, an effort was made to use the smallest possible processor in order to increase parallelism and to build a large array. Being binary images, the complexity of the processing elements does not need to be high to perform an efficient computation.

4.1 The processing array

Fig. 1 shows schematically the processing array. The array is composed of a single type of processing elements, named as PE, as described below. The data flow in and out of a PE to the four nearest neighbors along the NEWS distribution network, a local communication system in four directions, north, east, west and south. It allows to interact through direct physical connections to the four cardinal points of each PE. To access to a farther PE, a strategy for moving data between PEs is needed.

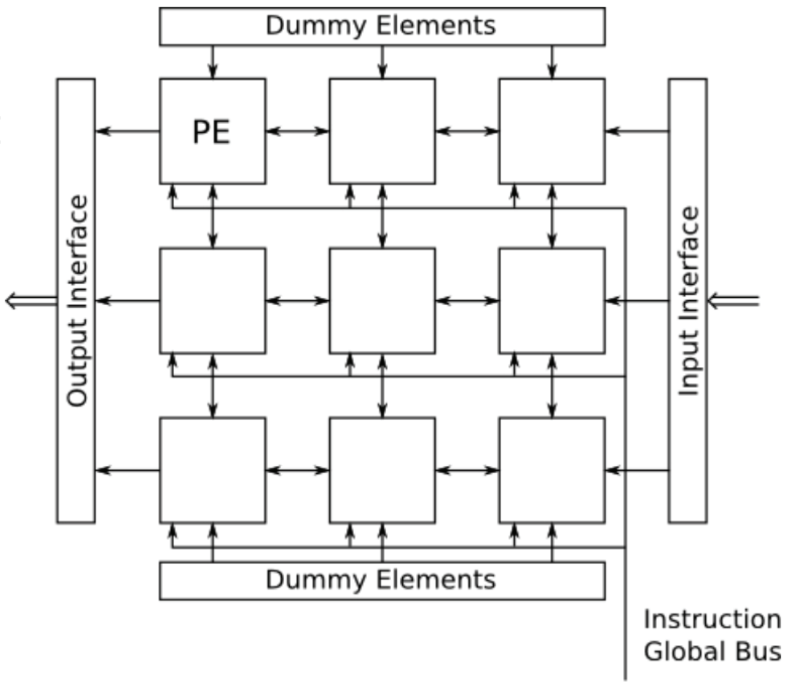


Fig. 1. The Processing Array schematic. Simple PE disposed regularly with local NEWS communication and a ring of dummy cells as boundary condition.

To obtain a full functionality, a cell ring of dummy cells that provides a configurable contour condition for the array was added. These cells are formed only by a memory element that takes values 0 or 1 depending on the algorithm being executed. If a not constant condition is necessary, i.e. varies pixel by pixel, the outer ring of PEs of the array can be used, the latter, at the cost of image resolution. Nevertheless, constant and homogeneous boundary conditions are usually the case.

The schematic of Fig. 1 shows that dedicated buses to load and download data are not present. The following approach is proposed: to use the existing local connections to perform shifts to upload and to download the data contained in the local memories of each PE. Thus, the global data buses are eliminated, being only necessary to design an interface between the array and the external memories, which stores the images, and to permit loading and unloading at one column per cycle. The interface is made by a series of multiplexers connected to the adequate input port of each PE which form the first column and allow to select between the value stored in the dummy ring (during processing) or the external memory (loading and unloading data). The output interface does not need additional hardware, since it is only necessary to connect the output of the PEs to the output bus. In this way, for an NxN array, N cycles are needed to upload or download an image.

Although this setting seems limited in terms of speed, the inclusion of the two interfaces gives the same speed as implementations which include rows and columns decoders. Both cases require as many accesses to the array as rows or columns. However, the input and output interfaces improve loading times allowing to download and to upload the current

and the next image simultaneously, reducing the time by a factor 2. In contrast, the array random access is not permitted. In those cases where it is necessary to extract a concrete column of the array, shifts must be performed as many times as needed. However, despite this drawback, we achieve a significant improvement in the area consumed by the array and the processing speed.
Due to the simplicity of the PE, it is expected a great regularity in the design of the array. Even in programmable architectures such as FPGAs, an automatic *place & route* provides a good approximation to a topographic implementation without requiring manual intervention. With respect to a full custom implementation, it is also expected a high integration.

4.2 The processing element

The processing element, PE, is shown in Fig. 2. There are three main modules: the Logical Unit, the Memory Bank and the Routing Module. It can be checked that there is not an associated control module. The control is done globally and it is common to all of them, greatly decreasing the area. In this way, all the PEs run the same instruction synchronously. Using an appropriate set of instructions, the control unit can be implemented with very reduced hardware resources.

The Routing Module

The Routing Module acts as the interconnection between the PE and the rest of processing elements of the array. It consists of a multiplexer of four 1 bit inputs to the PEs located in the NEWS positions. A 2 bits signal controls the direction of communication, providing the value of one of the neighbors within the PE. The output provided by the PE always comes from a position of the Memory Banks, as it will be detailed later, so no additional logic is necessary for control.

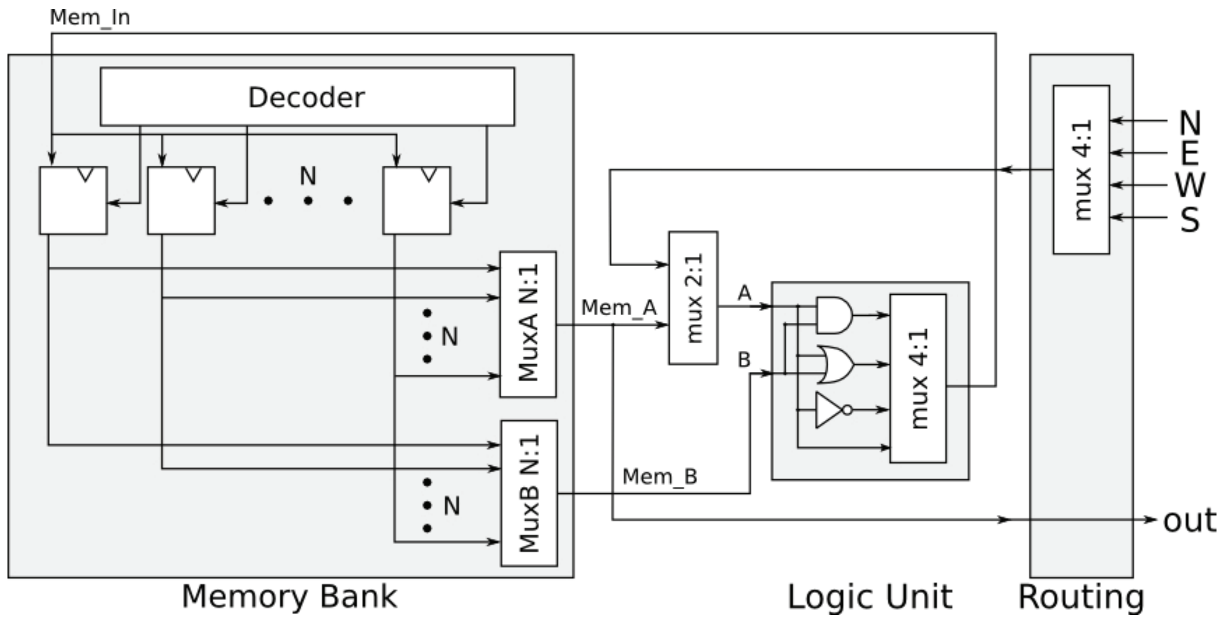


Fig. 2. Processing Element schematic. It is composed by three modules: The Memory Bank, the Logic Unit and the Routing Module.

This communication system is suitable for most of the low and mid-level operations as it can be extracted from the study of different algorithms in the literature or from the configuration of the coefficients in the templates listed on the Cellular Wave Computing Library (Kék et al., 2007). This library collects some of the most used templates in image processing on CNNs. Although the proposed architecture has a different operating mode than a CNN, its templates are fully representative of low-level image processing algorithms (Fernandez Garcia et al., 2008).

The Logic Unit

The Logic Unit performs only the most basic Boolean operations: *AND*, *OR* and *NOT*. In order to increase flexibility, the *Identity* operator is also included. With this set of operations, we can build any kind of binary operation instructions because it forms a functionally complete set.

The Logic Unit has two inputs for the 1-bit operands, labeled *A* and *B*, and one output, *R*, also 1-bit. The *A* operator drives the four operands, while the *B*, only drives the *AND* and *OR* gates. The multiplexer, controlled by a 2 bits signal, is responsible for providing the proper result in *R*.

Operands *A* and *B* have different origins. *B* is always a memory value, internal to the PE. However, *A* can be either an internal value or a value from one of the neighboring PEs obtained through the Routing Module. To select between these two options is necessary to add a 2:1 multiplexer (see Fig. 2) that selects the correct source of *A*. Thus, *A* is present in all operations while *B* is only required when the operation involving two operands.

It is interesting to highlight some of the functions of the *Identity* operator. Its main task is to act as data bus between the elements within the array and inside the PE. In this case, it serves to duplicate the value of a memory. It also stores the value of a neighbor in the PE local Memory Bank. Thus, we can move data across the array at a rate of one pixel per cycle. Thus, we are able to access elements that are not directly connected.

The architecture of the Logic Unit allows the addition of new operations easily. For example, an algorithm may need to perform a more complex operation many times. In that case it might be worthy to add an extra operator to boost speed. In return, the size of the multiplexer increases, as the size of the global control lines does. It is also possible to replace the Logic Unit with a *Look-Up Table* or a *ROM* that stores the appropriate outputs for certain inputs, thus emulating the overall operations of the Logic Unit. Our implementation has the potential to address complex operations with a reduced set of logic gates.

The Memory Bank

The Memory Bank consists of a set of 1-bit registers. It has just an entry, labeled *Mem_In,* of 1 bit and provides two simultaneous outputs, *Mem_A* and *Mem_B*, also 1-bit each.

The data input by *Mem_In* port is controlled by a decoder which activates the write signal in the register selected by an S-bit control signal, where *S* means 2^S=N, where *N* is the number of memory elements of the bank. An additional flag activates the writing on the Memory Bank. Thus the array can be blocked, holding its status, so interruptions in processing can be managed.

The two simultaneous outputs allow a flexible management of resources of the PE. On the one hand, they allow for internal operations to provide two operands to the Logic Unit. On the other hand, when the operation requires access to the neighborhood of the PE, an operand is provided *Mem_B* while the second comes from the Routing Module through the external 2:1 multiplexer. In this case it is necessary to provide an external output, determined now by *Mem_A*. Two N:1 multiplexers with S-bit control signals, *MuxA* and *MuxB,* provide this functionality.

The configuration of the Memory Banks acts as a dual-port memory allowing to simultaneous reads and writes on the same memory position. Thus, operations such *Register(2) = Register(1) OR Register(2)* are possible, reducing the amount of memory needed to implement the algorithms and the area of each PE.
Altogether, the necessary number of control bits is *3S+1*. This module is the most critical, since it consumes the greatest percentage of area of the PE. Its size might have to be shrunk to comply with the specific algorithm implementation. In this way, it also reduces the *fan-out* of the global interconnections, each buffering a smaller number of memory elements, increasing speed. The design of the decoder and the multiplexer will depend on the platform that is implemented, aiming at the greatest efficiency.

Functionality summary

In summary, the processing elements shown in Fig. 2 can:

- Select a neighbor
- Select two memory values
- Choose between two operation modes:
    - Two operands, one from the memory and other either from memory or an external element (AND, OR)
    - One operand from memory or from an external element (NOT, Identity)
- Write the result in the memory bank
- Provide a stable value as external output

The PE control

In order to increase the size of the array and as all PEs perform the same operation, it was decided to implement a global control system. It translates the high-level instruction into the signals arriving at each individual module of the PEs. This control can be implemented very easily if the instruction set is selected correctly.

	OP	**A**		**B**	**R**	**OUT**
		A_from	A_address	Address	address	address
n^o bits	2	1	S	S	S	S

Table 1. Encoded instruction format for the Processing Element.

Table 1 outlines the encoding of the instruction format for the PE. The meaning of each segment is as follows

- *OP* segment encodes the operation performed by the Logic Unit. If there are specific operations its size must be adapted.
- *A* segment encodes the address of the first operand. It consists of two fields, *A_from* and *A_address*. The first is a 1-bit flag indicating if it comes from the local Memory Bank or from the Routing Module. The second, S-bit wide, has a double meaning according to the value of the flag above. If it comes from memory, it encodes the memory address. Otherwise, the two least significant bits encode the neighbor from which the data come.
- *B* encodes the address of the second operand, which always comes from the Memory Bank.
- *R* segment indicates the memory address where the result of the current operation is stored.
- *OUT* segment indicates the address inside the Memory Bank from where the data are to be provided to the array.

The total size of an instruction is 4S+3 bits. It should be noted that a 1 bit flag is needed to control the interruptions of the array, as specified above.
The instruction decoding of the control signals for the particular elements within each module is undertaken by the Global Control Module. From Table 1 it can be extracted that each field directly controls the elements of the PE, without needing decoding. Therefore, this module can be implemented only by distributing the coded instructions to the PEs. For example, the *OP* field encodes directly the multiplexer of the Logic Unit.

5. SIMD array programming

By the nature of the instruction set, a Boolean equation that means how each PE changes its value according to a given neighborhood must be found. This equation can be obtained in various ways: from the original image and the expected image, observing the differences between them; translating the effects of a mask or a filter; directly designing the Boolean equation, etc. Next, a selection of algorithms to test and show as the proposed system works is listed. The examples were extracted from the Cellular Wave Computing Library, CWCL, which collects the most used templates in image processing on CNNs. This selection is not a limitation because it is a widely used platform for these tasks, being a representative set of low-level image processing operators.
In terms of notation, the following variables are used to refer to the neighborhood of the central pixel, *c*: *n*, *nw*, *se*, etc. refer to *north*, *northwest*, *southwest*, etc. We will refer to instructions on the type *neighbor{address}*, for example *north{R2}*, meaning *access to the #2 memory address of the northern neighbor*. If a neighbor is not specified, it is understood that it is the local memory of the PE under study.

5.1 Binary edge detector

The well known edge detector is a good model to show how the architecture works. This operator corresponds to the binarized version of the edge detector present in the CWCL and follows the outline of Fig. 3, where the mask T is defined in Eq. (1).

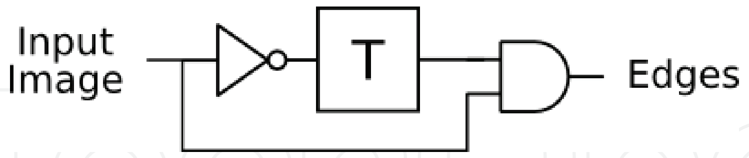


Fig. 3. Algorithm for the binary edge detection.

$$T : A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, I = -0,5 \qquad (1)$$

The interpretation of this template is simple. Only when one or more neighbours *n*, *s*, *e* or *w*, are active, the central pixel is activated. This is equivalent to an *OR* between these four values. As the Logic Unit is not capable of operating simultaneously on the four neighbours, it is necessary to divide *T* into four sub-operations. Thus, in the first clock cycle the image is inverted. Then *T* is applied, requiring 4 clock cycles, to finally perform the *AND* between the previous results. Altogether, it takes 6 clock cycles. The pseudo-code is listed below:

```
• R0 ← input image
• inversion
    • R1 ← NOT R0
• template
    • R2 ← north{R1}
    • R2 ← R2 OR east{R1}
    • R2 ← R2 OR west{R1}
    • R1 ← R2 OR south{R1}
• and
    • R1 ← R0 AND R1
```

In this case, a four input *OR* may be implemented to enhance the speed in applications that require intensive use of edge detection. It clearly illustrates the possibilities of expanding the proposed Logic Unit for this architecture. Fig. 4 shows the application of this operator over a test image.

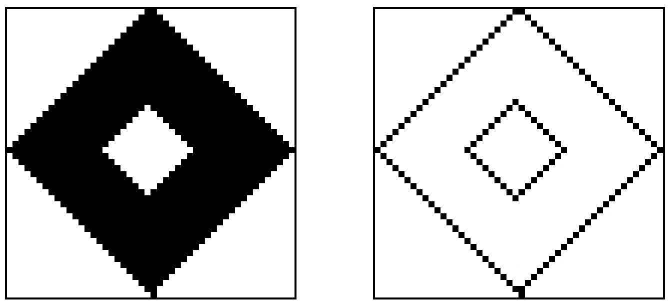

(a) Input image (b) Edge detection

Fig. 4. Binary edge detection example.

5.2 Hole filling

The Hole Filling is an iterative operation. It is used to fill the holes in all the objects that are part of an image. In a synchronous architecture, as the proposed here, it is executed iteratively a number of times that can be fixed beforehand or determined during the execution. The first case is the most common and it is the considered here. The algorithm used is described in (Brea et al., 2006) and shown in Fig. 5, where *T* is the same template described above in Eq. (1).

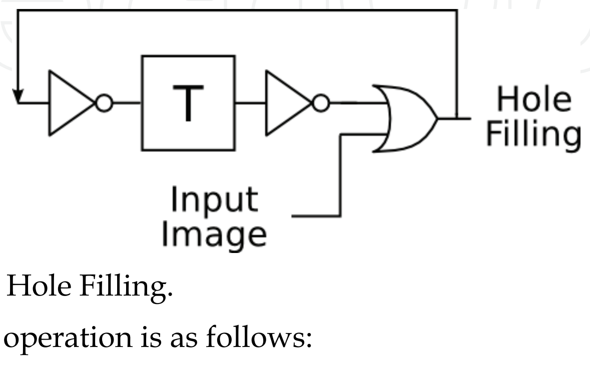


Fig. 5. Algorithm for the Hole Filling.

The pseudo-code of this operation is as follows:

```
• R0 ← input image
• 1st inversion
    • R1 ← NOT R0
```

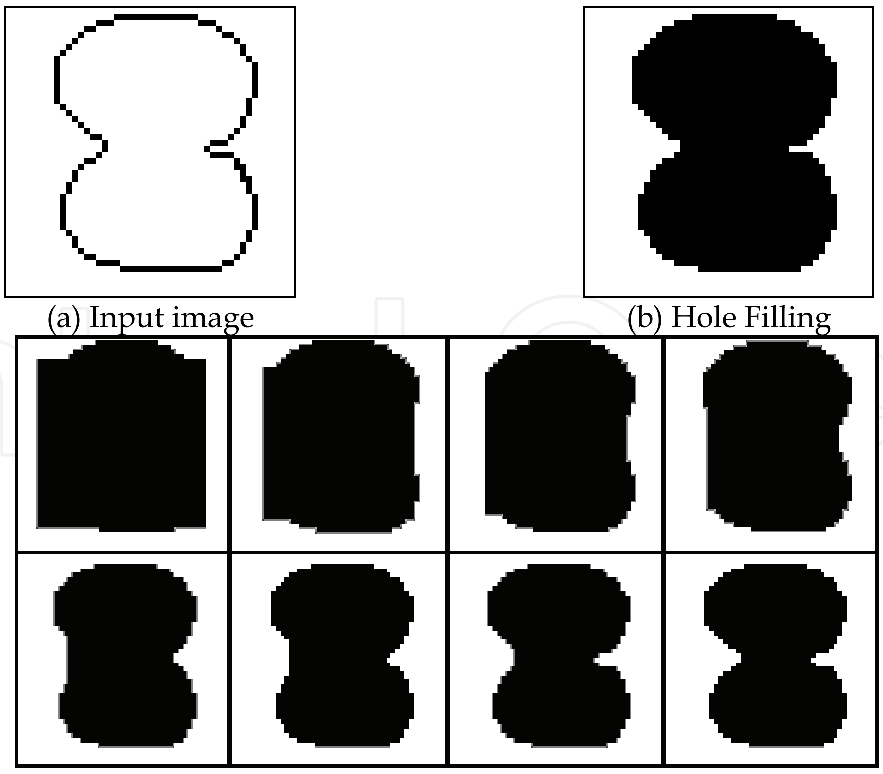
- template
  - R2 ← north{R1}
  - R2 ← R2 OR east{R1}
  - R2 ← R2 OR west{R1}
  - R1 ← R2 OR south{R1}
- 2nd inversion
  - R1 ← NOT R1
- OR
  - R1 ← R0 OR R1

A complete iteration of the algorithm requires 7 clock cycles. The number of iterations needed depends on the shape of the image and its size. The result of applying it on a test image is displayed on Fig. 6.

5.3 Shortest path problem

Finally, an implementation of the algorithm that solves the problem of the minimum path was done. The application is significantly more complex than the other examples outlined previously and it illustrates the capability of the SIMD array.

The aim is to determine the shortest path between two points, avoiding a series of obstacles. It is based on the implementation discussed in (Rekeczky, 1999), which proposed a new approach to solve this problem by using CNN computing. In line with this strategy, a wave front with constant speed explores the labyrinth from the starting point. At each branching of the labyrinth, the wave front is divided. When two wave fronts are at an intersection, the first to reach will continue evolving while the rest remains static, avoiding the collision.


(a) Input image (b) Hole Filling

(c) Evolution steps

Fig. 6. Hole Filling example.

Then, a prune of all paths is done, maintaining fixed the start and end points, which are external parameters of the system, so only the shortest path between those points remains.
The algorithm has two stages, both to carry out iteratively. The first stage, the exploration, is shown in Fig. 7. The templates T1 and T2 defined in Eqs. (2) and (3) along with its translation into Boolean equations.

$$T1 : A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, I = -0{,}5 \qquad \Rightarrow \qquad T1 = n + e + w + s \tag{2}$$

$$T2 : A = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & -1 \\ 0 & -1 & 0 \end{pmatrix}, I = -1 \quad \Rightarrow \quad T2 = \overline{n\cdot(w + s + e) + e\cdot(w + s) + w\cdot s} \tag{3}$$

The second stage, the pruning, is done executing iteratively T3, defined in Eq. (4). This template is equivalent to an AND between the labyrinth and explored the result of invert the application of T2 on the explored labyrinth, so the above equations will be used again.

$$T3 : A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 3 & 1 \\ 0 & 1 & 0 \end{pmatrix}, I = -2 \tag{4}$$

During the exploration phase, T1 requires 4 clock cycles, T2 9 cycles and the additional logic operations, 3 cycles. All in all, each iteration requires 16 cycles. The pruning phase is executed in 9 cycles, 8 for T2 and one for the additional operations. The number of necessary iterations for each stage depends on the labyrinth. Fig. 8 shows the different stages of each phase on a test labyrinth.

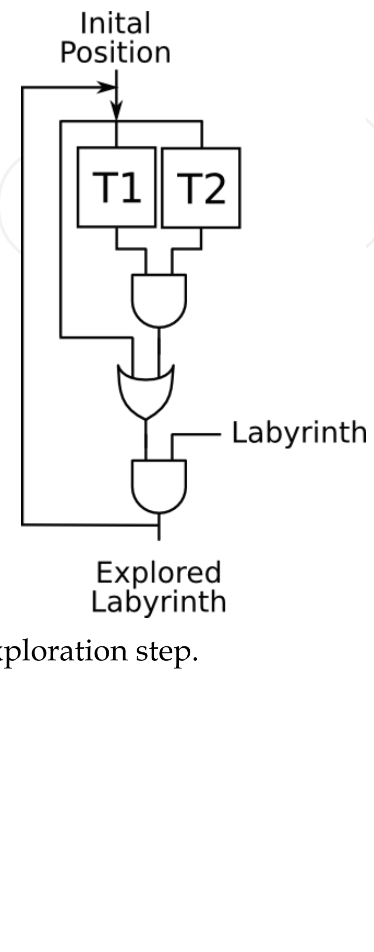


Fig. 7. Shortest path problem: exploration step.

The pseudo-code for both stages is shown bellow.

```
Exploration step:
•   R0 ← input image
•   R1 ← start point
•   Template T1
    •   R2 ← north{R0}
    •   R2 ← R2 OR east{R1}
    •   R2 ← R2 OR west{R1}
    •   R2 ← R2 OR south{R1}
•   Template T2
    •   R3 ← west{R1}
    •   R4 ← R3 AND south{R1}
    •   R5 ← R3 OR south{R1}
    •   R3 ← R5 AND east{R1}
    •   R3 ← R3 OR R4
    •   R4 ← R5 OR east{R1}
    •   R4 ← R4 AND north{R1}
    •   R3 ← R3 OR R4
    •   R3 ← NOT R3
•   Final operations
    •   R2 ← R2 AND R3
    •   R2 ← R2 OR R1
    •   R1 ← R0 AND R2
```

```
Pruning step:
•   R7 ← end point
•   Join start and end point images
    •   R1 ← R1 OR R7
•   Template T3
    •   R3 ← west{R1}
    •   R4 ← R3 AND south{R1}
    •   R5 ← R3 OR south{R1}
    •   R3 ← R5 AND east{R1}
    •   R3 ← R3 OR R4
    •   R4 ← R5 OR east{R1}
    •   R4 ← R4 AND north{R1}
    •   R3 ← R3 OR R4
    •   R1 ← R1 AND R3
```

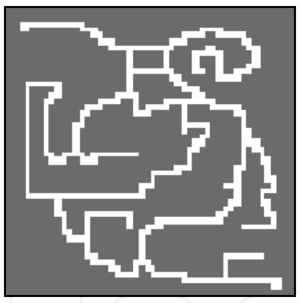
6. Proof of concept: FPGA synthesis

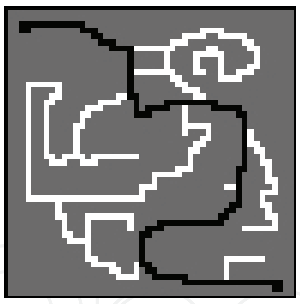
In order to test and verify the functionality of the proposed architecture, a description in VHDL was made for a subsequent implementation on a programmable architecture, specifically on an FPGA.

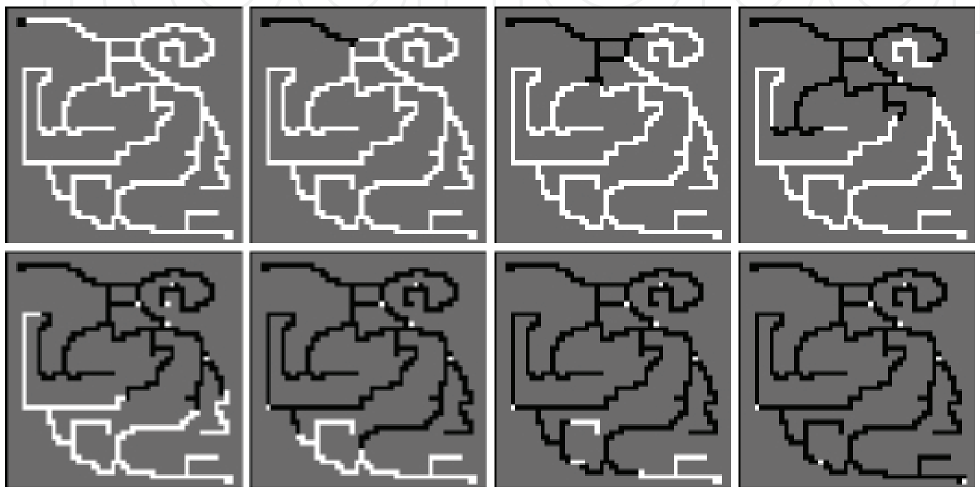
6.1 Global system configuration

Fig. 9 displays the schematic view of the global configuration of the proposed architecture. As our emphasis has been put on the array, the global system architecture addressed here is not optimized and it is only intended as an operation tester of the processing array. It is made up of the following elements:

- Input and Output Memory Banks: to store images to be processed and the processed images. Both of them provide a column per clock cycle.
- Instruction Storage: it stores the decoded instructions, the value of the dummy ring for each instruction and the interrupt signal, *process*.
- Global Control: it synchronizes all existing modules. It consists of a finite state machine and the Instruction Decoder described above.
- Processing Array: has an input column data of the stored image in the memory bank and as output, a column of the processed image. The control is done through the current instruction from the Instruction Memory. The *process* signal controls interruptions of the array acting on the decoder of the Memory Bank of each PE.


(a) Labyrinth


(b) Shortest path


(c) Exploration phase

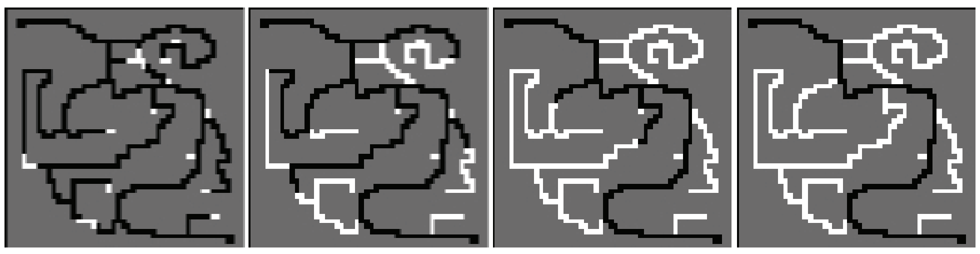

(d) Prune phase

Fig. 8. Shortest path problem. Example

- Address Generator: it calculates the address of the next instruction to execute. It allows loops in order to iteratively execute blocks of instructions, reducing memory requirements for storing the instructions.

6.2 Implementation results

The selected platform was an RC2000 card from Celoxica, which comprises a card with a PMC RC2000 PCI controller, which allows to insert the card into the PCI bus of a PC, and an ADM-XRC-II board by Alpha Data, with the following features:

- Xilinx Virtex-II xc2v6000-4 FPGA
- 6 banks x 2MB RAM ZBT
- 2 banks x 4MB RAM ZBT
- External clocks: [40KHz - 100MHz] and [25MHz – 66MHz]

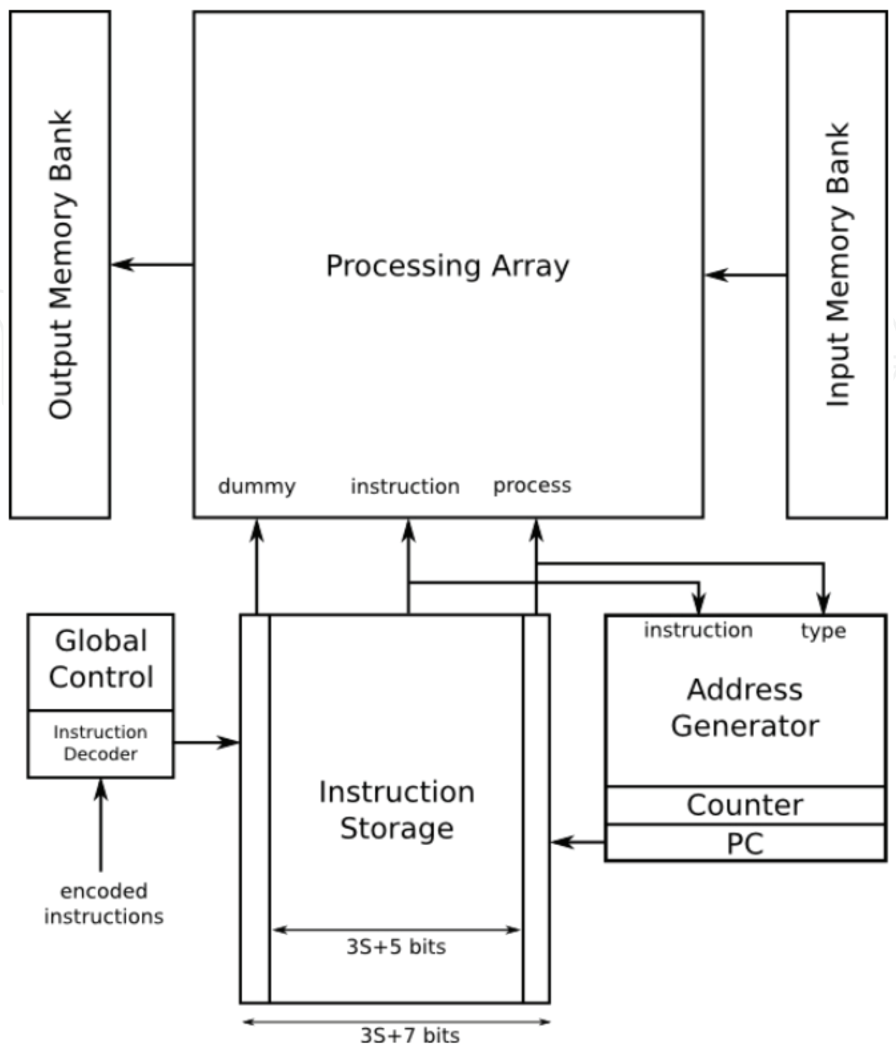


Fig. 9. Global system architecture.

The Xilinx Virtex-II xc2v6000 main characteristics are:

- 6M equivalent logic gates
- 33792 slices
- 1056Kbits of distributed RAM
- 144 dedicated multipliers
- 144 dedicated RAM blocks

The design and mapping process was done using the ISE 9.2i, from Xilinx.

The selected array for the synthesis has a size of 48x48 processing elements, with a Memory Bank of 8 bits per PE (S=3). The external memory banks were implemented using the dedicated RAM blocks of the FPGA. The main results are summarized in Table 2.

Occupied slices	26,428 (78%)
Slices Flip-Flops	20,165 (29%)
4-input LUTs	50,649 (74%)
Number of equivalent gates	527,716
Maximum frequency achieved	67,3MHz

Table 2. Main results of the FPGA implementation.

There are available resources on the FPGA which can increase the resolution of the array, the size of the PE memory, or the instruction storage. With the same configuration, the resolution can be increased to reach an array of 56x56 elements.

6.3 Tested algorithms: Summary of results

Table 3 gives a summary of processing times for each algorithm, considering only one iteration. The maximum working frequency for the Virtex-II FPGA xc2v6000 is 67.3MHz. It also includes the number of required memories for its implementation, including always the memory where the original image is stored and is not changed during processing, although this would not be necessary in all cases. It also includes results for other tested algorithms that have not been detailed in this chapter, but intended to illustrate more clearly the performance of the platform.

Algorithm	#cycles	Time (μs)	#memories
Edge detector	6	0.089	3
Hit and Miss	10	0.149	3
Hole Filling	7	0.104	3
Skeletonization	84	1.248	5
Shortest Path: exploration	16	0.238	6
Shortest Path: prune	9	0.138	4

Table 3. Processing times for the proposed sample algorithms. One iteration at 67,3MHz.

For the iterative algorithms, Table 4 shows the total execution times. The test images have the same size as the matrix, i.e., 48x48 pixels

Algorithm	#cycles	#iterations	Time (μs)
Array load/Donwload	1	48	0.71
Hole Filling	7	45	4.68
Skeletonization	84	40	49.93
Shortest Path: exploration	16	125	29.72
Shortest Path: prune	9	75	10.03

Table 4. Processing times for the iterative algorithms. One iteration at 67,3MHz.

7. Technology evolution: FPGA vs. ASIC

The impressive progress of new technologies, marked by Moore's Law (ITRS, 2007) allows increasingly integration density. This leads to more hardware resources with higher clock frequencies, increasing performance considerably. In this way, programmable systems such as FPGAs are able to get the same performance than recent past specific integrated circuits, IC, keeping the development times and the time-to-market at low(Nagy et al., 2006). In this section we examine the scaling possibilities of the SIMD architecture due to the technology evolution, both for FPGAs and ICs.

7.1 FPGA

The ever-increasing density of the CMOS technology market by the Moore’s Law means that the designers can choose FPGAs as a proof of concept system (Lopich & Dudek, 2005) or even as an end system (Diaz et al., 2006). The time to market is another major reason why FPGAs are becoming increasingly competitive. A study based on the International Technology Roadmap for Semiconductors (ITRS) roadmap which estimates the increased logic capacity and speed of commercial FPGAs is shown below. Area reduction allows to integrate more logic. In this way, the parallelism that can be obtained with the proposed SIMD architecture will increase considerably, allowing large array sizes, so that the final solution could be based on an FPGA and not on an integrated circuit, cutting costs and design time. These results are shown in Fig. 10.

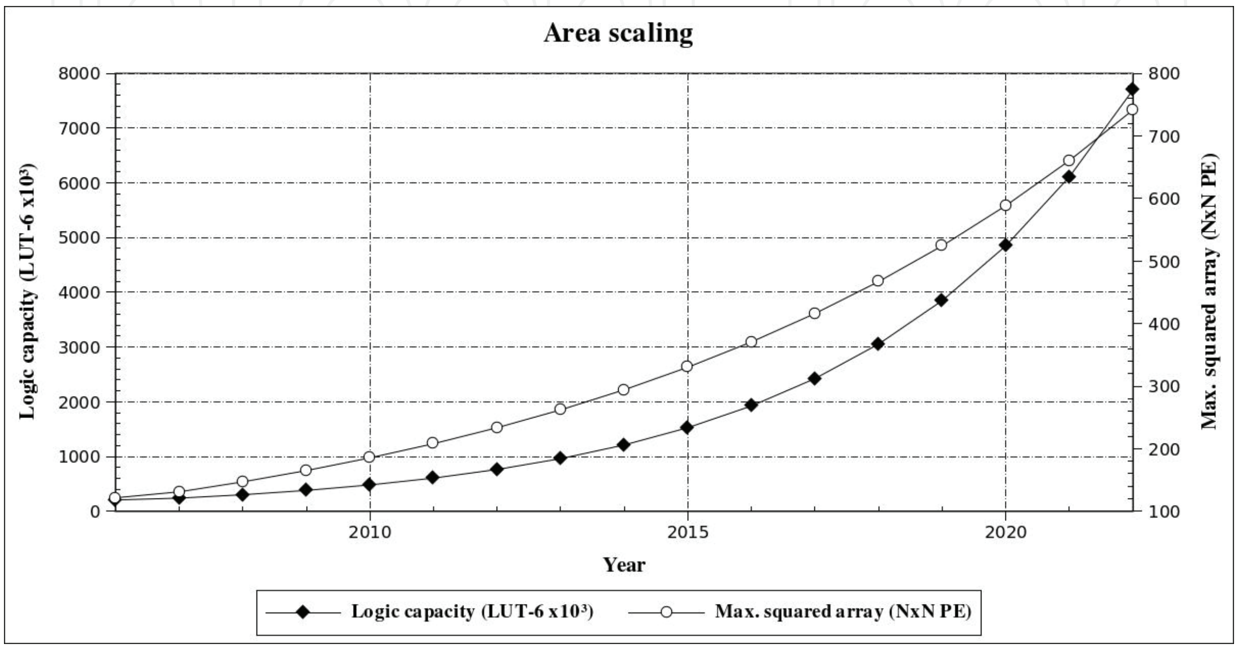


Fig. 10. Scaling of the FPGA logic capability. Logic capacity in thousands of 6-input LUTs. Maximum resolution of a square matrix, in number of PEs per row.

The method used for this estimate is as follows. Given the ability of the FPGA logic for a given year, the number of logic cells, and the prediction of the ITRS, the capacity of the FPGA logic cells for the target year is:

$$\text{capacity}_B = \text{capacity}_A \cdot \frac{Density_B}{Density_A} \cdot \frac{Die_B}{Die_A}$$

where the subscript A refers to the known year and B to the estimate year. As the ITRS shows, the chip size remains constant over the next years, thus simplifying the above equation. This study is based on the one detailed in (Varghese & Rabaey, 2001) and is a review of the last ITRS update (ITRS, 2007). This formula is valid only if the internal architecture of the FPGA does not change.
The selected FPGA to start this study with is one of the most advanced from Xilinx, a Virtex-5 XC5VLX330. The most salient characteristics are gathered in Table 5. This FPGA has more than 50,000 logic cells, CLBs, each composed of 2 slices, each one with 4 LUTs with 6 inputs and 4 Flip-Flops per slice. The logic contained in each CLB is summarized in Table 6. The technology used in the manufacturing process is 65nm at 1.0V with triple-oxide. We must emphasize that the Virtex 5 family changed its internal architecture from 4 input LUTs to 6 input LUTs, with respect to previous family (Cosoroaba & Rivoallon, 2006) (Percey, 2007).

CLBs array	LUT-6	Flip-Flops	Distributed RAM	Shift registers
240x108	207360	207360	3420 Kb	1710 Kb

Table 5. Virtex-5 XC5VLX330. Main hardware details.

Slices	LUT-6	Flip-Flops	Distributed RAM	Shift registers
2	8	8	256 bits	128 Kb

Table 6. Virtex 5 family. Hardware on each CLB.

With these data, whereas the number of basic elements of the selected FPGA is 207,000 in 2006, year that Xilinx introduced the Virtex 5 family, and assuming that the chip size remains constant, in 2010 it is possible to reach 481,000 elements. Fig. 10 shows the scaling of the current Virtex 5 family. The same figure includes what would be the maximum size of the SIMD processing array. This result should be taken as an upper bound and not as a precise value because they are not included the elements employed by control, memory instructions or even new features that may include newer FPGAs.
Concerning the Virtex 5 family, each Processing Element requires 8 Flip-Flops and 14 6-LUT, keeping the same characteristics as the detailed in the previous implementation. Therefore, the limiting factor is the number of available LUTs. Thus, it appears that using the 2009 technology node it would be possible to implement a QCIF array (144x176) and that before 2013 and 2019 it is feasible to process images of 256x256 and 512x512 pixels, respectively, without dividing them into sub-windows, with a PE per pixel.

7.2 ASIC

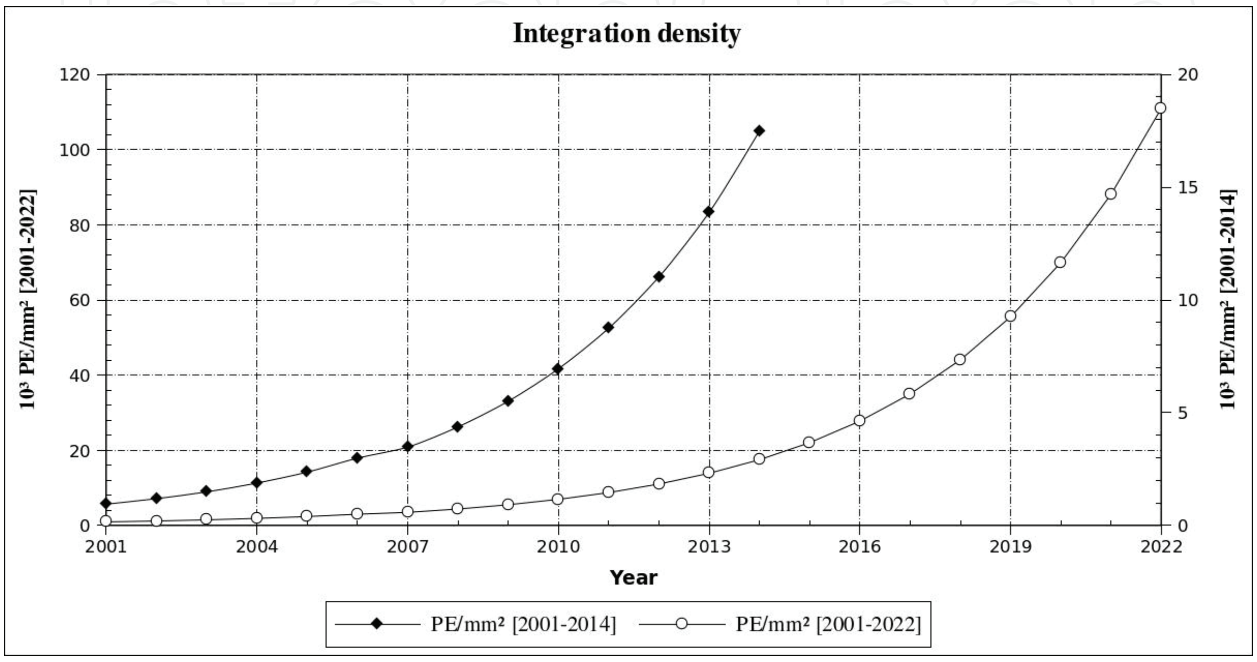
For certain applications, an FPGA-based solution can not meet the appropriate requirements such as higher performance, lower consumption and higher integration. One of its uses is as coprocessor in focal plane vision systems, accelerating the binary computation, where the integration with gray image processing units is necessary. Thus, it is interesting to discuss the features of a full custom implementation. An ASIC implementation for the Processing Element is proposed in this section in order to explore the possibilities of scaling along the ITRS predictions.
CMOS complementary logic with minimum size transistors will be used. This implementation is not optimal because using other CMOS logic families and specific design techniques, such as pseudo-nMOS or dynamic CMOS, a higher yield in both area and speed can be obtained. The different modules of the PE are conceived in the most straightforward manner. For example, a 2:1 multiplexer is formed by two 2-AND gates, an OR-2 and an inverter for the control signal. The features of the PE are the same as those detailed above: an 8-bit Memory Bank. A summary of the number of transistors per PE is given in Table 7.

	Routing	Memory Bank	Logic Unit	Other
No. of transistors	46	902	60	20

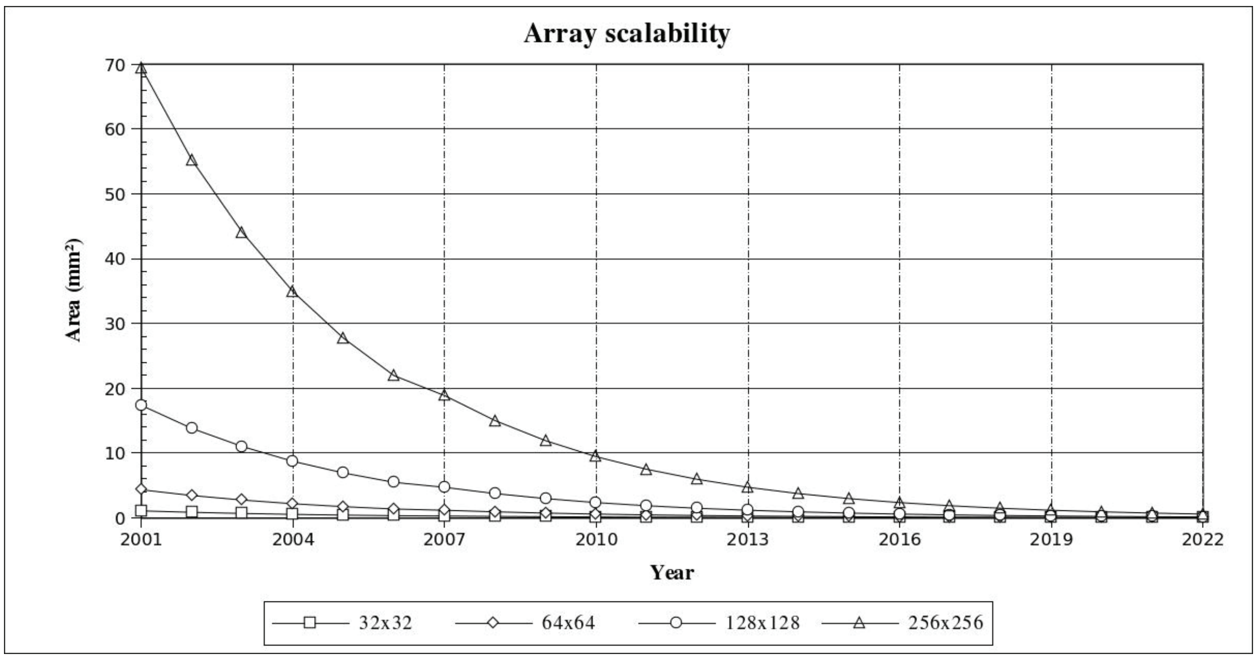
Table 7. Number of transistors for each Processing Element using CMOS logic.

It should be noted that the area occupied by the memory banks amounts to the 87% of the total area. Each memory element is formed by a D master/slave Flip-Flop with reset and enable signals. Clearly the selected implementation is not the best. Using RAM-based elements will considerably reduce the number of transistors. This module will require a

greater effort during the design to optimize the area requirements. The adequate selection of the size of the Memory Bank for a particular implementation is critical.
To determine the area occupied by each PE is assumed, similarly as for programmable architectures, that the integration density predicted by the ITRS includes the parameters that limit it. To estimate the size of the processing array it will also be assumed that the buffering is negligible compared to the processing hardware. This will not affect the precision with which the results are expected.



(a) Integration density scalability



(b) Processing Array scalability

Fig. 11. Scalability of the ASIC implementation.

The results are summarized in Figs. 11.a and 11.b. The first one shows the maximum amount of PEs that a given technology is able to integrate by mm^2. To ease the visualization the figure also shows the Near Term Years expanded. The second figure displays, for some array sizes, the total area that would be occupied according to the technological process selected. The planned integration density will allow use a processing array as accelerator in general purpose visual processors with a very low cost in terms of area, even in high resolution. We emphasize that with the 2001 technology, an array of 128x128 elements needs 18mm2 and that, with the same area, it is possible to integrate an array of 256x256 elements in 2007. It should be remembered that this is only an estimation and that the design is not optimized. Just by using transmission gates for multiplexors and SRAM memories the area requirements are reduced by a factor of 5. With a more aggressive design strategy is feasible to reduce an order of magnitude in the occupied area.

8. Conclusions

The range of applications where binary images are used is very wide due to its particular characteristics, such as low storage and processing requirements or the simplicity of the algorithms. Many of early vision operations can be addressed using a small computer unit. Thus, the path to explore is to increase the parallelism. SIMD architectures, with simple processing elements arranged in a matrix topology with local connectivity between them, are able to efficiently exploit the characteristics of low and medium level operations.

For the development of the processing element, special attention was paid to the reduction of its size while maintaining a generic functionality. Furthermore, it is easy to extend its functionality for those critical operations of a given algorithm. A limited set of instructions allows to easily translate a mask or filter or to deduce the Boolean equation that governs the change of status of each pixel of the image according to its neighborhood.

Functionality and performance were verified using an FPGA, obtaining performance data that indicate that even on a programmable architecture, the efficiency is high. Furthermore, short term future FPGAs will allow to implement bigger arrays, achieving a higher degree of parallelism. In the same way, full-custom implementations allow to use the proposed Processing Array as co-processor on generic vision systems with low area requirements.

9. Acknowledgements

This work is funded by Xunta de Galicia under the project PGIDT06TIC10502PR

10. References

Alonso-Montes, C.; Ortega, M.; Penedo, M. & Vilarino, D. (2008). Pixel parallel vessel tree extraction for a personal authentication system. *IEEE International Symposium on Circuits and Systems. ISCAS 2008*. pp. 1596–1599.

Anafocus (2007). Eye-Ris v1.0/v2.0 datasheet. *Anafocus White Papers*. http://anafocus.com

Akiyama, T.; Aono, H.; Aoki, K.; Ler, K.W.; Wilson, B.; Araki, T.; Morishige, T.; Takeno, H.; Sato, A.; Nakatani, S. & Senoh, T. (1994). MPEG2 video codec using image compression DSP. *IEEE Transactions on Consumer Electronics,* pp. 466-472, Vol. 40, No. 3, August 1994

Brea, V.; Laiho, M.; Vilarino, D.; Paasio, A. & Cabello, D. (2006). A binary-based on-chip CNN solution for Pixel-Level Snakes. I*nternational Journal of Circuit Theory and Applications*, pp. 383-407 Vol. 34(4), 2006

Cheng, H.; Jiang, X.; Sun, Y. & Wang, J. (2001). Color image segmentation: advances and prospects. *Pattern Recognition Letters*. pp. 2259–2281, Vol. 34, No. 12

Cosoroaba, Adrian & Rivoallon, Frédéric (2006). Achieving Higher System Performance with the Virtex-5 Family of FPGAs. *Xilinx White Papers*, July 2006

Creeger, M. (2005). Multicore CPUs for the masses. *Queue,* ACM New York, NY, USA

Diaz, J.; Ros, E.; Pelayo, F.; Ortigosa, E.M. & Mota, S (2006). FPGA-based real-time optical-flow system. *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 274-279, Vol. 16, No. 2, February 2006

Eklundh, J.-O. and Christensen, H. I. (2001). Computer Vision: Past and Future. *Springer Berlin - Heidelberg*.

Fernandez Garcia, N.A.; Suarez, M.; Brea, V.M. & Cabello, D. (2008). Template-oriented hardware design based on shape analysis of 2D CNN operators in CNN template libraries and applications. *Proceedings of 11th International Workshop on Cellular Neural Networks and Their Applications,* pp. 63-68, July 2008

Garcia, C. & Apostolidis, X. (2000). Text detection and segmentation in complex color images. *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*. pp. 2326–2329, Vol. 4, No. 6.

Gepner, P. & Kowalik, M.F. (2006). Multi-Core Processors: New Way to Achieve High System Performance. *International Symposium on Parallel Computing in Electrical Engineering,* pp. 9-13, 2006

Courtoy, M. (1998). Rapid system prototyping for real-time design validation. I*n Proceedings of 9th International Workshop on Rapid System Prototyping*. pp. 108–112.

Dudek, P. (2005). A general-purpose processor-per-pixel analog SIMD vision chip. *IEEE Transactions on Circuits and Systems I: Regular Papers [IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications]*, pp. 13–20, Vol. 52, No. 1:

Hinrichs, W.; Wittenburg, J.P.; Lieske, H.; Kloos, H.; Ohmacht, M.; Pirsch, P. (2000). A 1.3-GOPS parallel DSP for high-performance image-processing applications. *IEEE Journal of Solid-State Circuits,* pp. 946-952, Vol. 35, No. 7, July 2000

Hsu, R.-L.; Abdel-Mottaleb, M. & Jain, A. (2002). Face detection in color images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. pp. 696–706, Vol. 24, No. 5.

ITRS (2007). International Technology Roadmap for Semiconductors. *http://public.itrs.net*

Kék, László; Karacs, Kristóf & Roska, Tamás (2007). Cellular wave computing library (templates, algorithms, and programs). Version 2.1. *Research report of Cellular Sensory and Wave Computing Laboratory CSW-1-2007,* (Budapest, 2007)

Linan, G.; Dominguez-Castro, R.; Espejo, S. & Rodriguez-Vazquez, A. (2001). Ace16k: An advanced focal-plane analog programmable array processor. *Proceedings of the 27th European Solid-State Circuits Conference. ESSCIRC 2001.* pp. 201–204.

Lopich, A. & Dudek, P., R. (2001). Architecture of asynchronous cellular processor array for image skeletonization. *Proceedings of European Conference on Circuit Theory and Design,* pp. III/81-III/84 , Vol. 3, August 2005

MacLean, W. (2005). An evaluation of the suitability of FPGAs for embedded vision systems. *International Conference on Computer Vision and Pattern Recognition*, pp. 131–138.

Maginot, S. (1992). Evaluation criteria of hdls: Vhdl compared to verilog, udl/i and m. *European Design Automation Conference*, pp. 746–751.

Nagy, Zoltán; Vörösházi, Zsolt & Szolgay, Péter. (2006). Emulated digital CNN-UM solution of partial differential equations. *International Journal on Circuit Theory and Applications*, pp. 445-470, Vol. 34, No. 4

Owens, J.D.; Houston, M.; Luebke, D.; Green, S.; Stone, J.E. & Phillips, J.C. (2008). GPU Computing. *Proceedings of the IEEE,* pp. 879-899, Vol. 96, No. 5, May 2008

Park, H. J.; Kim, K. B.; Kim, J. H. & Kim, S. (2007). A novel motion detection pointing device using a binary cmos image sensor. *IEEE International Symposium on Circuits and Systems. ISCAS 2007.* pp. 837–840.

Percey, Andrew (2007). Advantages of the Virtex-5 FPGA 6-Input LUT Architecture. *Xilinx White Papers*, December 2007

Prati, A.; Mikic, I.; Trivedi, M. & Cucchiara, R. (2003). Detecting moving shadows: algorithms and evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. pp. 918–923, Vol. 25, No. 7

Rekeczky, Cs. (1999). Skeletonization and the Shortest Path Problem - Theoretical Investigation and Algorithms for CNN Universal Chips. *7th International Symposium on Nonlinear Theory and its Applications* (December 1999)

Schauer, B. (2008). Multicore Processors--A Necessity. *ProQuest*, September 2008

Shen, G.; Gao, G.P.; Li , S.; Shum, H.G. & Zhang, Y.Q. (2005). Accelerate video decoding with generic GPU. *IEEE Transactions on Circuits and Systems for Video Technology,* pp. 685-693, Vol. 15, No. 5, May 2005

Soos, B.G.; Rak, A.;Veres, J. & Cserey, G. (2005). GPU powered CNN simulator (SIMCNN) with graphical flow based programmability. *11th International Workshop on Cellular Neural Networks and Their Applications,* pp. 163-168, July 2008

Spiliotis, I. & Mertzios, B. (1997). A fast skeleton algorithm on block represented binary images. *Proceeding of the 13th International Conference on Digital Signal Processing*. pp. 675–678, Vol. 2.

Szolgay, P. & Tomordi, K. (1996). Optical detection of breaks and short circuits on the layouts of printed circuit boards using CNN. *Proceedings of the 4th IEEE International Workshop on Cellular Neural Networks and their Applications. CNNA-96.* pp. 87–92.

Tan, Edwin J. & Heinzelman, Wendi B. (2003). DSP architectures: past, present and futures. *SIGARCH Computer Architecture News,* pp. 6-19, Vol. 31, No. 3, New York, NY, USA

Tseng, Y.-C.; Chen, Y.-Y. & Pan, H.-K. (2002). A secure data hiding scheme for binary images. *IEEE Transactions on Communications*, pp. 1227–1231, Vol. 50, No. 8.

Varghese, George & Rabaey, Jan M. Name of paper (2001). *Low-Energy FPGAs, Architecture and Design,* Kluwer Academic Publishers

Vilariño, D. L. & Rekeczky, C. (2005). Pixel-level snakes on the CNNUM: algorithm design, on-chip implementation and applications. International Journal on Circuit Theory and Applications. pp. 17–51, Vol. 33, No. 1.

Image Processing
Edited by Yung-Sheng Chen

ISBN 978-953-307-026-1
Hard cover, 516 pages
Publisher InTech
Published online 01, December, 2009
Published in print edition December, 2009

There are six sections in this book. The first section presents basic image processing techniques, such as image acquisition, storage, retrieval, transformation, filtering, and parallel computing. Then, some applications, such as road sign recognition, air quality monitoring, remote sensed image analysis, and diagnosis of industrial parts are considered. Subsequently, the application of image processing for the special eye examination and a newly three-dimensional digital camera are introduced. On the other hand, the section of medical imaging will show the applications of nuclear imaging, ultrasound imaging, and biology. The section of neural fuzzy presents the topics of image recognition, self-learning, image restoration, as well as evolutionary. The final section will show how to implement the hardware design based on the SoC or FPGA to accelerate image processing.

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:

Alejandro Nieto, Víctor M. Brea and David L. Vilariño (2009). An FPGA-based Topographic Computer for Binary Image Processing, Image Processing, Yung-Sheng Chen (Ed.), ISBN: 978-953-307-026-1, InTech, Available from: http://www.intechopen.com/books/image-processing/an-fpga-based-topographic-computer-for-binary-image-processing