Available online at www.sciencedirect.com
SCIENCE DIRECT®
Microprocessors and Microsystems 28 (2004) 303–316
MICROPROCESSORS AND MICROSYSTEMS
www.elsevier.com/locate/micpro

Codesign methodology for computer vision applications

J. Albaladejo[a,*], D. de Andrés[a], L. Lemus[a], J. Salvi[b]

[a]*Fault Tolerant Systems Research Group, Department de Informatica de Sistemas y Computadores (DISCA), ETS Informática Aplicada, Universidad Politécnica de Valencia, Campus de Vera #14, Valencia 46021, Spain*
[b]*Computer Vision and Robotics, Institut d'Informàtica i Aplicacions, Universitat de Girona, Campus Montilivi, E-17071, Girona, Spain*

Received 20 October 2003; revised 19 February 2004; accepted 3 March 2004

Abstract

The work presented in this paper has been performed under a Spanish research project. The main aim of the tasks, we were responsible of, was the development of a vision subsystem for 2D image preprocessing. These algorithms are the first step of a 3D reconstruction algorithm. These algorithms have been improved by the addition of fault tolerance capabilities. To achieve this goal, the classical codesign methodology has also been improved to allow for the fault tolerant system codesign. This paper presents the results obtained from this work.

Keywords: Codesign methodology; Field programmable gate arrays; Fault tolerance

1. Introduction

According to the evaluation of the United Nations Economic Commission for Europe (UNECE) and the International Federation of Robotics (IFR) [1], the international market of robots will have an increase of the 70% in the period ranging from 2002 to 2005. Technological advances will be more notorious in those application domains that involve the use of robots. Among these application fields are all the services areas. In particular, surveillance services are of special interest since they will suffer an increase greater than 500% (see Fig. 1).

Computer vision is one of the research areas that is very closely linked to this increase in the robots production. The different technological advances that have been developed in 2D images and 3D structures recognition will be applied to robotics [2]. Nowadays, there already exist several applications that make use of 2D techniques. However, the results obtained from the research in 3D techniques are expected to be implemented on robotics by 2005.

Taking it into account, mobile robotics with 3D recognition capabilities will be one of the most promising research areas and most of the new developed application will belong to this domain.

Several different research domains usually take part in robotic applications, such as computer architecture, electronics and system control.

This paper presents the final results from a Spanish CICYT project (Interministerial Commission of Science and Technology). Several Spanish research groups have taken part in this project: Computer Vision and Robotics (VICOROB) from the University of Girona (UdG), and three groups from the Technical University of Valencia, Radio-Fibre Group (FRG), Electronics Engineering Department (DIE) and Fault Tolerant Systems research Group (GSTF).

The main aim of this project was 'the development of a reliable mobile robot for indoor surveillance tasks'.

One of the most important factors of this project was the addition of fault-tolerant mechanisms in the different subsystems the robot consists of in order to improve the reliability and security of the system.

The GSTF was assigned the following tasks:

- The implementation of the fault tolerant distributed control architecture of the mobile robot.
- The implementation of the vision subsystem on reconfigurable hardware.

The analysis of this second task is presented in this paper. It describes all the experiences that have been extracted

* Corresponding author. Tel.: +34-963-87-7575; fax: +34-963-87-7579.
E-mail addresses: jalba@disca.upv.es (J. Albaladejo); ddandres@disca.upv.es (D. de Andrés); lemus@disca.upv.es (L. Lemus); qsalvi@eia.udg.es (J. Salvi).

0141-9331/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.micpro.2004.03.010

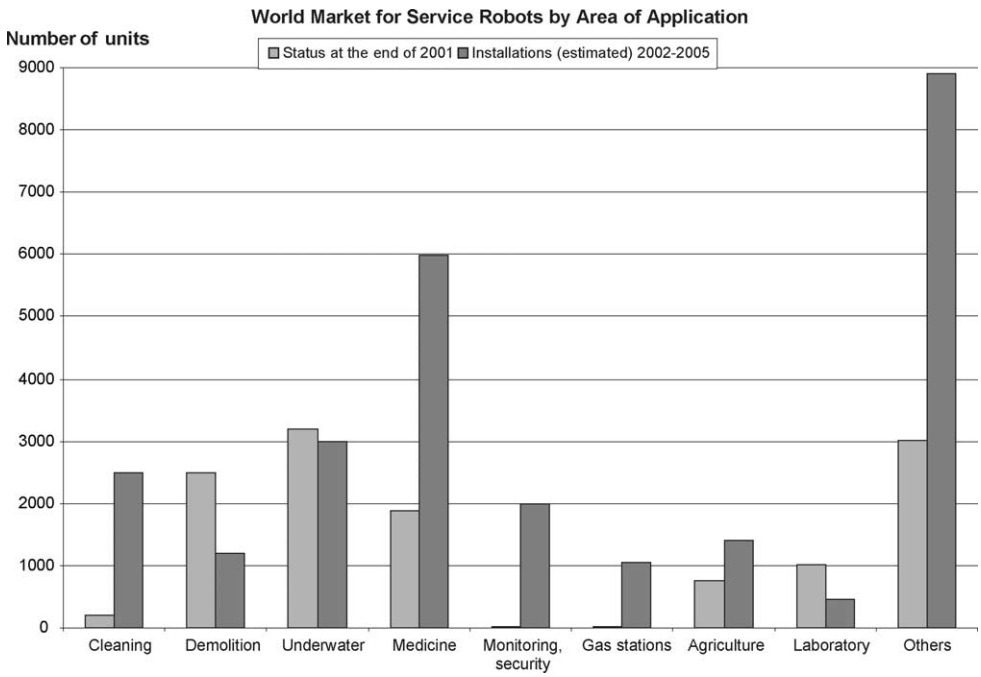

Fig. 1. Robots market evolution depending on the target applications [1].

from the development of the project and that have led to a PhD thesis [3]. This paper has been divided as follows. A brief introduction to the whole project will be presented in Section 2. The methodology that has been applied to the development of the image preprocessing subsystem is described in Section 3. Section 4 presents the implementation of the subsystem on reconfigurable devices. This section describes further in detail the development of the VHDL libraries of floating point operators and the library of fault tolerant operators. Finally, Sections 5 and 6 present the results, conclusions and future work.

2. The project tasks description

The different tasks of the project were divided into the research groups according to their affinities. The computer vision tasks that were leaded by the VICOROB [4] consisted in:

- Obtaining a 3D map of the environment.
- Scene characterisation.
- Controlling some preselected objects in the scene.

Our work, under the supervision of the computer vision group, has consisted in implementing on reconfigurable hardware some image preprocessing algorithms:

- Image equalisation.
- Corner detection in 2D images.

The result obtained from the corner detection algorithm is a list of coordinates (x, y) that locates the corners in the image. This list feeds the 3D map building algorithm that has been developed by the VICOROB group [5]. Fig. 2 shows the control flow of the 3D map building algorithm and the 2D image preprocessing algorithm that must be implemented on reconfigurable hardware.

Apart from this one, there are two secondary aims:

- Improving the preprocessing algorithms performance that are usually implemented on microprocessors.
- Designing an image processing architecture based on reconfigurable devices such as Field Programmable Gate Arrays (FPGAs).
- Obtaining a fault tolerant architecture by adding fault tolerant capabilities to the operators implemented on reconfigurable hardware.

The methodology that has been followed to achieve these aims is presented in Section 3.

3. General methodology vs. proposed methodology

The classical design methodology is based on vertical partitioning (see Fig. 3a). This design methodology makes use of both hardware and software commercial components (*Commercial of the Shell* or COTS). The starting point of this methodology is a hardware platform that meets the performance requirements for the system under design. The operating system and the software application are built on this platform. This methodology offers a very rigid solution, since the performance of the system is usually constraint to that the hardware platform can obtain. Improving the performance of the system usually involves replacing

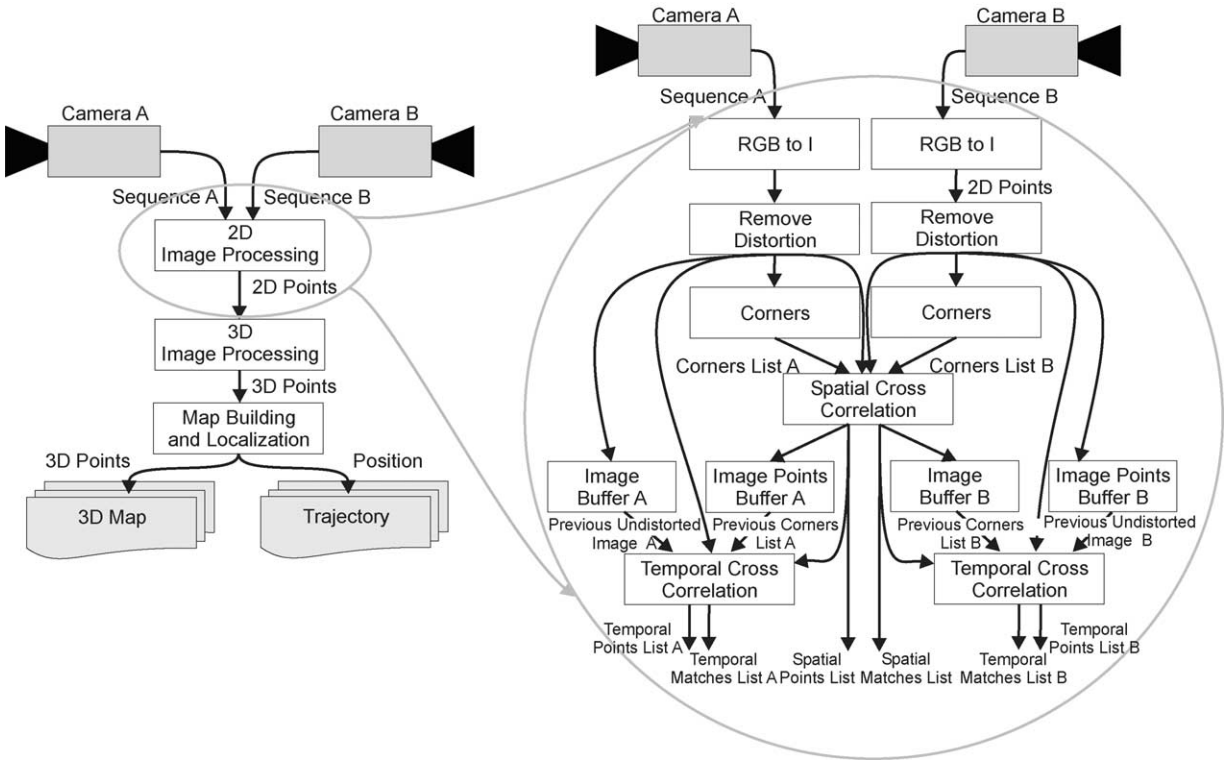

Fig. 2. Control flow of the 3D algorithm and 2D image preprocessing algorithm [5].

the hardware platform (microprocessor or microcontroller) by a more competitive component.

The appearance of reconfigurable hardware components in the market has led to the development of a horizontal codesign methodology. This design methodology, like the classical one, takes performance requirements and system specifications as the starting point of the design flow. In this case, the design flow is based on a horizontal partitioning and not a vertical one (see Fig. 3). This means that, although the hardware and software parts of the system make use of COTS components, they also make use of reconfigurable hardware devices [6]. In this way, a hardware platform with horizontal structure is established. As in the classical methodology, the software is implemented on this hardware platform. In case the performance of the system must be improved, it is possible to program the functions that must be sped up on reconfigurable hardware [7]. The coupling level between the microprocessor (microcontroller) and the reconfigurable hardware device can be any of those proposed by K. Compton and S. Hauk [8].

This section presents a horizontal codesign based methodology that can be applied to the development of fault tolerant microprocessor (microcontroller) based systems. The proposed methodology consists of including fault

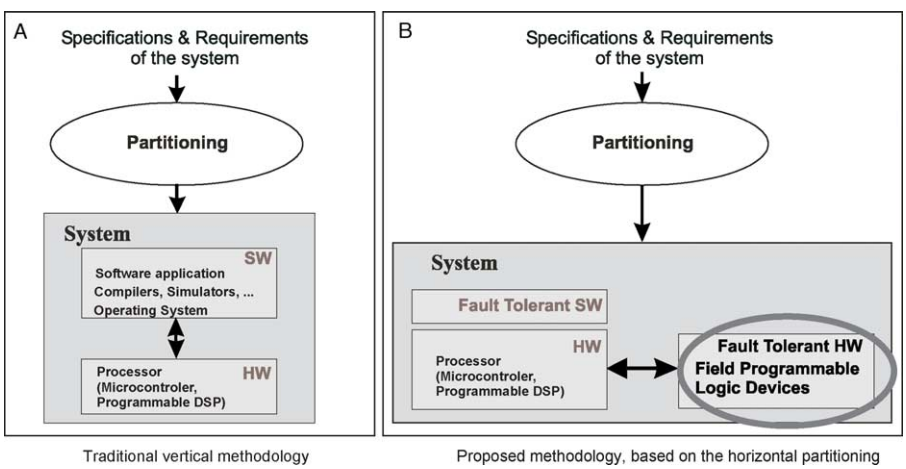

Fig. 3. General codesign methodology (left), proposed methodology (right).

tolerant capabilities to the system when performing its horizontal partition.

To achieve this goal, some fault tolerance software techniques will be included in the application software [24] and will run on the processor. In the same way, some fault tolerance mechanisms will be developed for the reconfigurable hardware. Once all the fault tolerance mechanisms have been included, it is necessary to validate them and analyse the performance of the system.

3.1. Specifications and requirements of the system

In the first step of the methodology, the requirements and high level specifications of the system are established. Different domain related specification languages are used to introduce the specifications [9]. These languages are based on: system specification models, such as states oriented models, finite states machines (FSM), Petri nets (PN) or hierarchical and concurrent FSM (CFSM); activity oriented models, such as data flow graphs (DFG) or control flow graphs (CFG); structure oriented models, like the diagrams of connectivity among components; data oriented models, such as the diagrams of relationship among entities and Jackson diagrams, and hybrid models, like control and data flow graphs (CDFG) [10].

3.1.1. Specifications and requirements

The work our group has to face is: 'the design of an embedded vision subsystem that must be coupled with a microprocessor for a mobile robot. The vision subsystem will be in charge of accelerating the required preprocessing for the 3D reconstruction. The final 3D map will be built by means of the microprocessor.'

The specifications and requirements of this subsystem reflect: the performance, results rate, functionality, power consumed and kind of inputs and outputs of the system.

The functions that must be implemented are:

- A 3D map building algorithm that will run on the robot's embedded microprocessor.
- A corner detection algorithm for 2D images that will run on reconfigurable hardware. To achieve this aim:
  Some classical corner detection algorithms, such as Harris–Stephens [11] and Förstner [12] algorithms, will be used.
  The outputs of these algorithms will be inputs for the 3D vision algorithm.

The inputs and outputs of the vision subsystem are:

- Inputs: the digital (or analogue) video signal.
- Images are 256×256 pixels wide and have a resolution of 256 grey levels (8 bits per pixel).
- Outputs: the results from the image preprocessing algorithm are the coordinates (x, y) of the detected corners. They will be sent to the microprocessor via USB or Ethernet, or to an external SRAM memory. It will also be possible to visualise the processed image on a visual display unit. All these outputs must be available depending on the user requirements.

The general requirements show that the main goal of the robot is surveillance tasks and, therefore, it must be a reliable platform. Thus, the vision subsystem design specifications must include the dependability of the system. This specification leads to the addition of fault tolerant mechanisms to the design.

Dependability of the vision subsystem:

- The vision subsystem must be designed in order to increase the dependability of the global system. Thus, the vision subsystem will implement some fault tolerance mechanisms.

3.1.2. Validation of the specifications and requirements

The second stage of the general methodology consists validating the specifications using any of the formal languages that are available in the market. Specification models make use of formal languages with concrete semantic rules. These rules are used to specify the definitions, causes, actions and/or relationships that can be established in a specific design area, like digital integrated circuits. Some examples of application domains are telecommunications, automation, domestic services, and the industries related to transformation process, military equipment and medicine. During the recent years, the most important inversion has been performed in these application fields, both in marketing and research and development [13].

This second stage has been adopted in the following way. First, the algorithms must be verified and validated by some tool that allows for the simulation of the system behaviour in an easy and quick way. This step must be done before implementing the algorithm, both the hardware and software modules. Therefore, the image preprocessing algorithms are written by means of some high level language. In our case, Matlab™ [14] tool has been used to simulate and validate the image preprocessing algorithms because it offers several capabilities for image processing. In particular, the results obtained from the simulation using integer numbers have been found to be invalid. It is necessary to obtain valid results before implementing the corner detection algorithms on FPGAs. Therefore, all the arithmetic operations have been redesigned to use a floating-point format derived from the standard IEEE-754 [15]. These new operators are validated again using Matlab™. These results are then compared with the ones obtained from the simulation using the *double* format from the standard IEEE-754. In case the requirements were not met, it would be necessary to modify the floating-point format that is being developed (size of exponent and mantissa). This new format needs to be verified again until

the requirements are met. The validation using Matlab™ has been very useful to check data format and type and to determine the correctness of the results.

3.2. Partitioning

Hardware (HW)–software (SW) horizontal partition is performed in this stage of the methodology. The different tasks of the simulated algorithms are usually divided into two modules: those that need a faster execution time (hardware modules) and those that do not need high performance or their timing requirements are already met (software modules).

Hardware components are synthesised on reconfigurable hardware and software components are compiled to run on a processor. Hardware modules are usually written on a high-level hardware description language (HDL), such as Verilog or VHDL. The hardware devices where they are usually synthesised are FPGAs, CPLDs and FPSoCs, although they can be synthesised on ASICs when they need high timing performance. Software modules (functions), which do not need high performance, are compiled to run on an auxiliary microprocessor or microcontroller.

3.2.1. Synthesis

The horizontal partition of the system is performed once the data format and type has been verified. Taking into account the proposed system, hardware oriented in this case, the algorithms are translated into hardware modules by means of a HDL that could be synthesised.

Following this methodology, the image preprocessing algorithms have been manually translated from Matlab™ to VHDL in order to implement them into a Xilinx FPGA.

In case of implementing a fault tolerant design, this paper proposes the addition of a new stage in the traditional horizontal codesign methodology. Software and hardware fault tolerance mechanisms must be added in this new stage (see Fig. 3b). If the design does not have any fault tolerant capabilities, the designer will proceed to the cosimulation phase.

3.2.2. Cosimulation and validation

In the traditional horizontal codesign methodology, the design requirements are verified by means of the hardware and software modules integration. Cosimulation tools are used to perform this verification. In case the requirements are not met, it is necessary to go back to optimise the HW and SW partition. This process is known as refining. Hardware optimisation usually consists in obtaining a higher level of parallelism while software optimisation tries to implement more efficient functions.

This stage follows the classical methodology but includes the coverage analysis. The analysis of the system coverage is related to the expected fault rate. The system coverage must reach a required value. This requirement can be satisfied by those systems with a high level of intrinsic redundancy. If this is not the case, it is necessary to go back to the synthesis phase and add some fault tolerance capability. The new system will be more reliable than the previous one.

The global system must be simulated and analysed to obtain its coverage after implementing the fault tolerant vision subsystem.

3.3. Prototyping and manufacturing

The last stage of the methodology involves the system prototype assembly (hardware, software and their interface) once the requirements and specifications are met.

After the prototype has been validated, it is all prepared for the serial fabrication of the prototype.

4. Implementation

This section presents the different studies that have been performed in order to implement the proposed vision subsystem. It is divided into three subsections. Section 4.1 explains the corner detection algorithms. Section 4.2 details the implementation and simulation of these algorithms. Section 4.3 presents the VHDL design of the algorithms once the data types were validated. This last section also analyses the fault tolerance technique that has been implemented.

4.1. Corner detection algorithms

The theoretical study of the search of maxima and minima points on surfaces can be applied to the image analysis. It makes use of the properties of spatial derivatives on images (analysis of the partial derivatives matrix of a surface or Hessian) to find the corner of the objects that appear on the image. These characteristics are considered as points of interest [16]. The points of interest of an image are defined as those points that present a variation of the light intensity in any direction.

Computer vision researchers have developed several corner detection algorithms [17]. Among them appear two classical algorithms: Harris–Stephens and Förstner algorithms.

The VICOROB group has proposed these two algorithms to be implemented on hardware.

Both algorithms are based on the study of the autocorrelation matrix. It is calculated by the first order partial derivatives in the **x** and **y** directions for an image **I**. This matrix **A** is defined by:

$$A = \begin{bmatrix} < I_x \cdot * I_x > & < I_x \cdot * I_y > \\ < I_x \cdot * I_y > & < I_y \cdot * I_y > \end{bmatrix} \quad (1)$$

where I_i represents the partial derivative in i direction. Symbol ‘$< \ldots >$’ means that this operator has been

smoothed by using a spatial Gauss mask. '.*' operations represent *point to point* operations between derivatives and not the classical matrix product [18].

The study of the diagonalisation of matrix **A** shows that its eigenvectors are the directions of the maxima and minima when light intensity changes.

The eigenvalues of matrix **A**: $\lambda_{max}\mathbf{y}\lambda_{min}$ shows that:

- It is a uniform point when: $\lambda_{max} \times \lambda_{min} < 0$.
- It is a corner point when: $\lambda_{max} \times \lambda_{min} > 0$.
- It is an edge point when: $\lambda_{max} \times \lambda_{min} = 0$.

Harris–Stephens [11] proposed a function named *cornerness* that is represented by **R** :

$$\mathbf{R} = \det(A) - 0.04[\mathrm{Trace}(A)]^2 \quad (2)$$

where **R** measures the gradient variation in each point of an image and the local maxima are reported as corners.

Another corner detection algorithm that has been implemented is the Förstner algorithm [12]. It extracts the corners of an image from the local maxima of another function named $\mathbf{C_1}$:

$$\mathbf{C}_1 = \det(A)/[\mathrm{Trace}(A)] \quad (3)$$

4.2. Data types implementation and validation by means of simulation tools

The previously presented algorithms have been simulated by using the MatLab™ tool. Their execution flow is the following:

Step 1: Derivative in direction $X(I_x)$.
Step 2: Derivative in direction $Y(I_y)$.
Step 3: Point to point X and Y derivatives product $(I_x \cdot * I_y)$.
Step 4: Point to point X and X derivatives product $(I_x \cdot * I_x)$.
Step 5: Point to point Y and Y derivatives product $(I_y \cdot * I_y)$.
Step 6: Gaussian smoothing $(< I_x \cdot * I_y >)$ of [*Step 3*] results.
Step 7: Gaussian smoothing $(< I_x \cdot * I_x >)$ of [*Step 4*] results.
Step 8: Gaussian smoothing $(< I_y \cdot * I_y >)$ of [*Step 5*] results.
Step 9: Corners operator $(C_1)^{-1}$.
Paso 10: for $Y = 1$ to dim(IMG Y)
  for $X = 1$ to dim(IMG X)
    Obtain the minimum local neighbour of (C_1)
    Min(X, Y) = Minimum Neighbour (C_1)
  end for X
  end for Y
Step 11: for $Y = 1$ to dim(IMG Y)
  for $X = 1$ to dim(IMG X)
    if (Min$(X, Y) = C_1(X, Y)$ then
    Store $(X, Y, C_1(X, Y))$
    end if
  end for X
  end for Y
Step 12: Descending ordering of C_1.
Step 13: Obtaining the best qualified n points (corners coordinates).

The different steps of a simulation are shown in Fig. 4.

A first study was performed in order to validate the data types that must be used in the implementation of the algorithms in FPGAs. This work consisted in the simulation of the algorithms using MatLab™ and integer data type.

The previously presented program flow has been simulated using both integer data type (signed) and double data type (64 bits). The different results have been compared to determine the number of bits necessary to perform the operations using integer data type. To perform the operations with integer operands, it is necessary to scale them to achieve a good precision in the results. Taking into account the range of the numbers and coefficients implied in the operations, it is possible to determine that the number of bits required to represent the operands with a good resolution is more than 64. The required size of the operands is so large that it is not advisable to implement them in FPGAs.

The following study consisted in the simulation of the corner detection algorithms using different floating point data types (modifying the exponent and mantissa length). In this way, it was possible to determine the best data format to fit in the FPGA taking into account its resources limitation.

Floating point operators have been implemented on reconfigurable devices in some works related to sound and image processing algorithms. One of the first studies [19] described the implementation of floating point vision algorithms. A parametrizable floating point modules both in exponent and mantissa length were implemented in Ref. [20]. The design of floating point modules for computer vision using images taken by a satellite was presented in Ref. [21].

In all these papers, the format of the floating point numbers is analogue to the standard IEEE-754 format [15] (see Table 1). From these studies, it is possible to determine that there are two common floating-point formats for FPGA implementation [22]: a 16 bits format and a longer (more bits) format (but less bits than the 32 bits simple precision standard). Therefore, the format of the floating point numbers is established according to the internal resources of the FPGA.

The reconfigurable device that has been used to implement the vision subsystem is a Virtex FPGA from Xilinx. Thus, it is necessary to take into account the internal resources of this circuit [23]. Among the internal

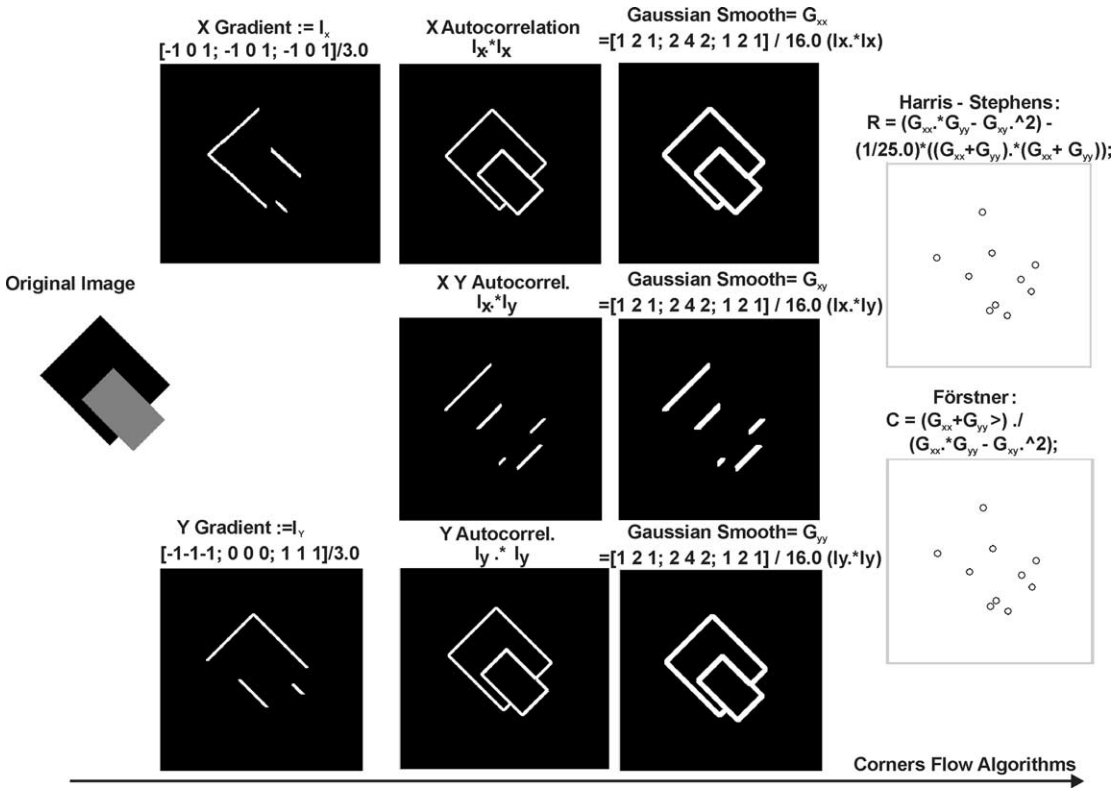

Fig. 4. Corner detection algorithm simulation with Matlab™.

components that are needed to implement the arithmetic operators (such as the configurable logic blocks or *SLICES*) appear several RAM blocks (*SRAMB4_Sm_Sn*). The configuration of these RAM blocks is determined by balancing the number of bits required by the floating-point numbers and the optimal use of these memories. Finally, the optimal solution is obtained by configuring the internal RAM in order to use a word size of 16 bits.

Taking into account this restriction, a 16 bits format for floating point numbers has been implemented. This format is parametrizable: the number of mantissa and exponent bits.

The validation of this data format has been performed by simulating the algorithms using floating point format. The results obtained have been compared to the ones obtained from the simulation using the MatLab™ 64 bits *double* format (this format has been called *ideal* in this work).

MatLab™ allows for the selection of the data format to work in an application domain by using the Simulink™ tool. This tool allows the user to choose among integer format and different fixed point formats, but it is restricted to applications in the system control domain, i.e. applications that work in the frequency domain, such as discrete transformations like FFTs, DCTs, etc. and their inverse transformations.

The problem arises because the operations involved in the corner detection algorithms belong to the spatial domain, like Prewitt spatial derivates convolution and Gauss filtering (see Eqs. (2) and (3)). To solve this problem, it is necessary to develop some new classes to allow the simulation of the operations with this 16 bits floating point format using MatLab™.

A new MatLab™ floating point arithmetic operators library (@$float_{16}$ class) has been designed to allow this simulation. The results from these new simulations are compared to the *ideal* results. All the operations involved in the corner detection algorithms have been implemented: addition, subtraction, product and division.

After this study, the size of the exponent and mantissa of the new floating point format (@$float_{16}$) can be seen in Table 2: 1 bit for the sign, 6 bits for the exponent and 9 bits for the mantissa. The greatest value that can be obtained as intermediate result in the involved operations is around

Table 1
Floating point numbers standard format IEEE-754 [IEEE-754]

Precision	Sign (bit)	Exponent (bits)	Mantissa (bits)	Bias
Double	1 (63)	11 (62:52)	52 (b51:0)	1023
Simple	1 (31)	8 (30:23)	23 (b22:0)	127

Table 2
Structure of the designed class for floating point numbers: @$float_{16}$

Precision	Sign (bit)	Exponent (bits)	Mantissa (bits)	Bias
@$float_{16}$	1 (15)	6 (14:9)	9 (b8:0)	31

4×10^9. Taking into account that any floating point number can be described by Eq. (4), the maximum value that can be represented using 6 bits for the exponent is 8581545984.

$$\text{value} = (-1)^s \times 2^{E-31} \times (1\ M) \tag{4}$$

As it can be seen, it is enough to represent the maximum and minimum values for the intermediate results. Matlab™ has been used to perform several simulations to determine whether a mantissa of 9 bits offers a good precision for the related operations. Several simulations have been performed and the comparison among the results obtained using the IEEE-754 double precision format and the floating point format presented here shows that the algorithms find exactly the same corners in the 99% of the cases. Therefore, we have chosen this floating point format since it has less bits than the IEEE-754 format and obtains nearly the same results.

4.3. VHDL implementation of the floating point operators (@$float_{16}$ class)

Once the format of data and the floating point operators have been validated by MatLab™ simulation, the next step in the design is the implementation of this components in hardware. This is usually done by using some HDL. VHDL has been selected in this case. The translation of the @$float_{16}$ operators into VHDL modules has been manually performed. The design process involved several refinements. The design has followed a *Down–Top* methodology, implementing each operator in first place and, after that, assembling the required operators to implement the proposed algorithms.

The VHDL floating point operators have been implemented by using the arithmetic operators already existing in the standard IEEE-1164 libraries: *IEEE_std_logic_arith.ALL* and *IEEE_std_logic_unsigned.ALL*. The implementation of the operations of addition (plus_float), subtraction (minus_float), product (multiplic_float) and division (division_float) and the convolution operator (2D_convol) has made use of the operands the development tool can synthesise. The basic arithmetic operations implemented follow the IEEE-754 standard [15] adapted to the @$float_{16}$ data type.

The design of the basic floating point operators in VHDL is based on the IEEE algorithms. According to this idea, the addition has been implemented following to this algorithm assuming that the operands are already in the floating-point format:

Result $= X + Y = (Xm \times 2^{Xe}) + (Ym \times 2^{Ye})$ involves the following steps:

1. Align binary point:
   Initial result exponent: the larger of Xe, Ye
   Compute exponent difference: Ye − Xe
   If $Ye > Xe$ Right shift Xm 'exponent difference' positions to form $Xm\ 2^{Xe-Ye}$
   If $Xe > Ye$ Right shift Ym 'exponent difference' positions to form $Ym\ 2^{Ye-Xe}$
2. Compute the addition of aligned *mantissa*s:
   i.e. $Xm2^{Xe-Ye} + Ym$ (or) $Xm + Ym2^{Ye-Xe}$
3. If the result must be normalised, then a normalisation step follows:
   Left shift the result and decrement its exponent (e.g. if result is 0.001xx…) or
   Right shift the result and increment its exponent (e.g. if result is 10.1xx…)
   Continue until MSB of data is 1 (hidden bit in IEEE-754 Standard)
4. Check result exponent:
   If larger than maximum exponent allowed return exponent overflow
   If smaller than minimum exponent allowed return exponent underflow
5. If result mantissa is 0, it will be necessary to set the exponent to zero by a special step to return a proper zero.

The floating point subtraction operation has been implemented using the addition operator and the two complement of the second operand.

The floating point product operation performs several shiftings and additions to obtain the right result.

Finally, the floating point division follows a restoration algorithm that makes use of the previously implemented floating point subtracter.

The convolution operation has been implemented using the previously developed product, addition and subtraction operators as building blocks. It has been designed following the MatLab™ algorithm named *valid*. This algorithm consists in multiplying the elements of a mask ($n \times m$) by the pixels of an image [16]. The result is an image with int(n/2) less rows and int(m/2) less columns than the original image.

The data path has been designed according to the VHDL modules previously implemented. In this case, the data path follows a predefined strategy to access external memories where data are stored. Both images and results are stored on these external (left and right) SRAMs (see Fig. 5).

Data (images) in the floating point format are initially stored into the external SRAMs. The 'mask_3 × 3' operators are in charge of performing convolution operations to obtain the gradients (Ix, Iy) in the **x** and **y** directions. After that, the autocorrelations are carried out ($Ix{\cdot}*Ix$, $Iy{\cdot}*Iy$, $Ix{\cdot}*Iy$). A Gaussian smoothing is applied to the autocorrelation results (Gxx, Gyy, Gxy). At last, corners are extracted by means of Harris–Stephens or Förstner algorithms (see Fig. 5). The detailed analysis of the data path is presented in Ref. [3].

The next step is the validation of the algorithms implemented on the FPGA. The results obtained from

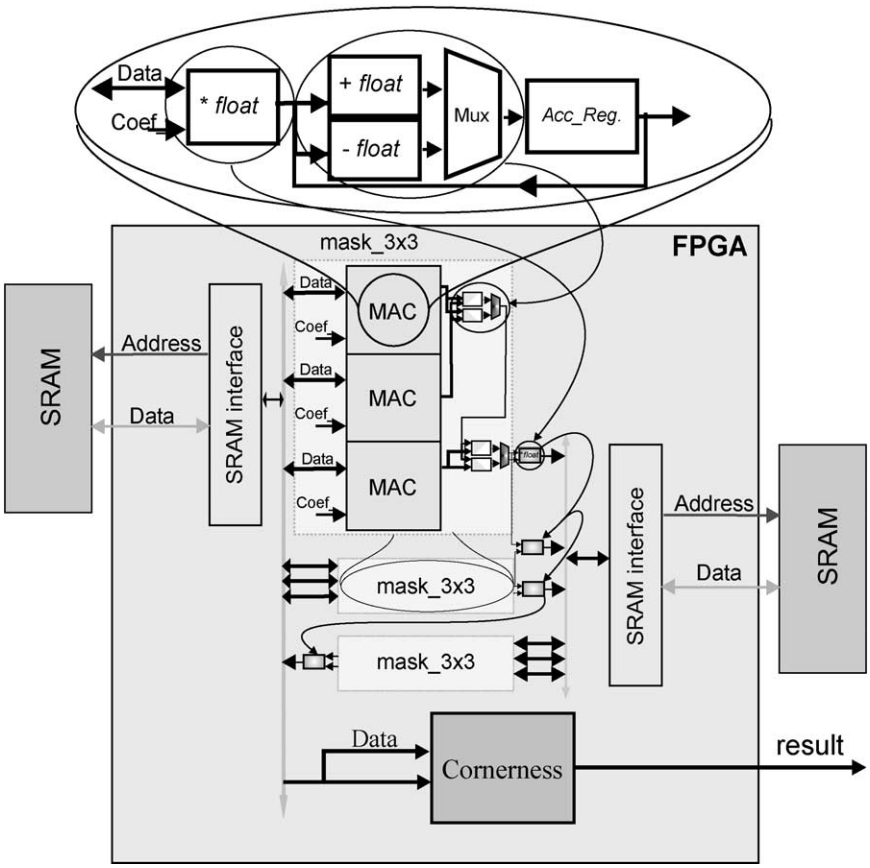

Fig. 5. Data paths of the corner detection algorithms.

these algorithms are compared to the results of the simulation.

Once the results have been validated, it is necessary to add fault tolerant capabilities to the system if needed.

In this case, the requirements of the project specify that it must be a reliable system. Therefore, the image preprocessing subsystem must present a lower failure rate than the simple one.

The fault tolerant vision subsystem has been implemented following a classical software fault tolerance technique known as 'N-Version Programming' [24]. The arithmetic operators that are used in the corner detection algorithms have been replicated by using different implementations of the same algorithm.

There exist two different versions for each arithmetic operator: a simple version and a 3-versions. In the simple version each operator has been replicated three times (same functionality and same structure). In the three-versions, the replicated modules performed the same function but do not have the same structure.

- The three-versions floating point adder implements three different adders: Carry-Look-Ahead, Carry-Select and Ripple-Carry adders.
- The three-versions floating point subtracter implements three different subtracters by using the previously presented adders (data in two-complement format).
- The three-versions floating point multiplier also implements three different versions by using the adders components (shifting and addition).
- The three-versions floating point divisor (restoring algorithm) implements three different versions based on the previously presented subtracters.

All these operators implement the treatment of the special cases such as infinity, zero and not-a-number (NaN).

The simple arithmetic operators are substituted by their fault tolerant version in order to increment the dependability of the system. The fault tolerant ones have a structure known as *Shift-Out Redundancy* (SoR) that allows for the verification of the results. This structure consists of a comparator, a detector and a collector (see Fig. 6). This module checks the result of each operator. The result is valid when the majority of the result is equal. The module detects which operators are not properly working and they are removed.

Fig. 6 shows the block diagram of the mixed SoR floating point adder that implements three different components: *Ripple-Carry*, *Carry-Select* and *Carry-Look_Ahead*.

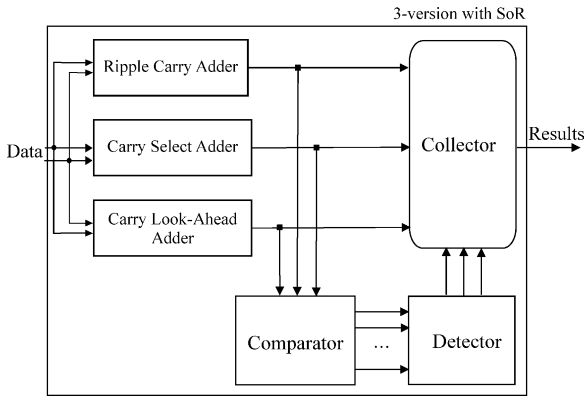

Fig. 6. Block diagram of a hybrid TMR floating point adder.

The coverage analysis begins with the specification of the theoretical model of the system. This model is usually described by means of Markov chains, PN or any other systems modelling techniques. Experimental results are obtained by means of fault injection campaigns both in the model or the prototype of the system.

The Markov chain model of the SoR components is shown in Fig. 7. It is to note that each replicated module is implemented in a different way but with the same functionality and, therefore, the fault tolerant module implements three-versions with SoR. The failure rates (λ_1, λ_2, λ_3) of the components that appear in the model are usually different. In this case, the logic resources (SLICES) that each implemented component uses are nearly the same ($\text{Look-Ahead}_{\text{Slices}} = 208$, $\text{Ripple-Carry}_{\text{Slices}} = 207$, $\text{Carry-Select}_{\text{Slices}} = 210$). Thus, the failure rate of each module ($\lambda = \lambda_1 = \lambda_2 = \lambda_3$) has been considered to be the same (see Fig. 7).

The initial state of this model (3) represents that all the three modules of the system are fault free. There are two possible alternatives in case a module fails: it is possible to reach states (2D) and (2ND). The state (2D) represents that the system has detected the error and continues working properly. The state (2ND) represents that the error has not been detected and the system continues working. The final states of the system represent a safe failure (SF) and a non-safe failure (NSF).

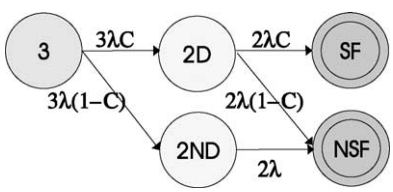

Fig. 7. Markov model of the TMR components with fault rate λ and coverage C.

The SF state is reached when two of the modules have failed and the system has detected both errors. The NSF state is reached when one of the errors has not been detected and is represented by the transitions $2\lambda(1 - C)$ y $3\lambda(1 - C)$. The system is also working in the state (2ND) but the error has not been detected. Then, a failure in any of the other two modules will lead to the NSF state through the transition 2λ.

The differential equations that can be deduced from the three-version with SoR model and the probability of the different states are shown in the following equation [25]:

$$\begin{bmatrix} \dot{p}_3(t) \\ \dot{p}_{2D}(t) \\ \dot{p}_{2ND}(t) \\ \dot{p}_{SF}(t) \\ \dot{p}_{NSF}(t) \end{bmatrix} = \begin{bmatrix} -3\lambda & 0 & 0 & 0 & 0 \\ 3\lambda C & -2\lambda & 0 & 0 & 0 \\ 3\lambda(1-C) & 0 & -2\lambda & 0 & 0 \\ 0 & 2\lambda C & 0 & 0 & 0 \\ 0 & 2\lambda(1-C) & 2\lambda & 0 & 0 \end{bmatrix} \times \begin{bmatrix} p_3(t) \\ p_{2D}(t) \\ p_{2ND}(t) \\ p_{SF}(t) \\ p_{NSF}(t) \end{bmatrix}; \begin{cases} p_3(t) = e^{-3\lambda t} \\ p_{2D}(t) = 3Ce^{-2\lambda t} - 3Ce^{-3\lambda t} \\ p_{2ND}(t) = 3(1-C)e^{-2\lambda t} - 3(1-C)e^{-3\lambda t} \\ p_{SF}(t) = C^2 + 2C^2e^{-3\lambda t} - 3C^2e^{-2\lambda t} \\ p_{NSF}(t) = (1-C^2) + 2(1-C^2)e^{-3\lambda t} \\ -3(1-C^2)e^{-2\lambda t} \end{cases} \quad (5)$$

The analytical solution for the reliability of the system appears in:

$$R(t) = p_3(t) + p_{2D}(t) + p_{2ND}(t) = 3e^{-2\lambda t} - 2Ce^{-3\lambda t} \quad (6)$$

The security of the system is represented by:

$$S(t) = R(t) + p_{SF}(t) = C^2 + 2(1-C^2)e^{-3\lambda t} - 3(1-C^2)e^{-2\lambda t} \quad (7)$$

Eqs. (6) and (7) are dependent on λ, C and t (time). The failure rate (λ) is provided by the manufacturer. Then, it is necessary to obtain the coverage (C) of the fault tolerance mechanisms of the system.

The coverage of the system can be obtained by the injection of faults into the system. There exist different fault injection techniques [26,27]. A simulation-based fault injection technique has been used in this work. In particular, faults will be injected in the VHDL model of the system. Three different fault models have been used:

- *Bit-Flip*. It represents a change in the logic level of memory cells (flip-flops, registers, memory).

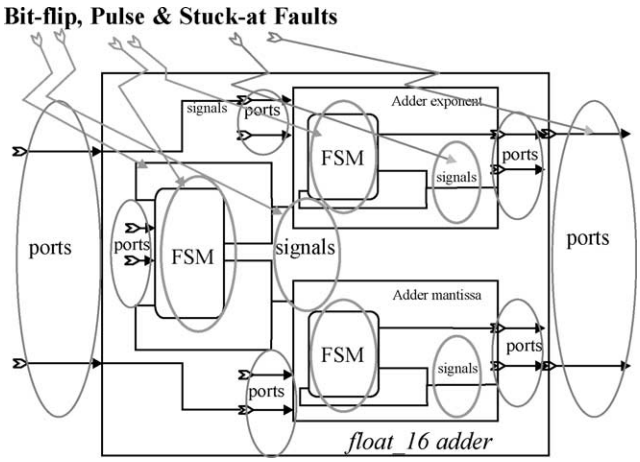

Fig. 8. Fault injection levels for the synthesised VHDL modules.

- *Pulse*. It represents a change in the logic level in combinational circuits (signals).
- *Stuck-at*. It represents a permanent change in the logic level of both sequential and combinational circuits.

Fault injection campaigns have consisted in the injection of 3000 faults following a uniform distribution on the VHDL modules that have been synthesised. Fig. 8 shows the levels where faults have been injected:

- At component ports.
- At component signals.
- At component FSM.

The campaigns have been performed both on the fault tolerant and non-fault tolerant components.

The Section 5 presents the results obtained from this work.

5. Results

This section details the results obtained from the simulations, the VHDL implementation and the reliability study of the image preprocessing subsystem that has been designed.

The main results that can be extracted form MatLab™ simulations are:

- The operators must use more than 64 bits to work with integer numbers in order to avoid any loss in the precision of the results [28].
- Having in mind the size of these operands, a new floating point format was developed (@$float_{16}$). This class allows for the simulation of the algorithms in MatLab™. This new format takes into account the internal resources of the reconfigurable circuit where the algorithms will be implemented (Virtex FPGA from Xilinx).
- These algorithms have been simulated varying the size of the exponent and mantissa. The results from these experiments are compared to the results obtained from the simulation of the algorithms with *double* data type (64 bits).

This comparison shows that the implementation of the Harris–Stephens and Förstner corner detection algorithms using *double* and @$float_{16}$ data types detect nearly the same amount of corners. An example of this comparison can be seen in Fig. 9.

The algorithms execution time depends on the operating system and the processor where they are running.

This study has used images of 256 × 256 pixels and 256 grey levels. The algorithms were written in C/C++. They were executed on different microprocessors running under Windows and Linux. The VHDL implementation was simulated with ModelSim tool [29].

Table 3 shows that the execution time of the algorithms running on an FPGA at 35.7 MHz was 50 times faster than the result obtained from the PowerPC 823 at 50 MHz.

It must be taken into account that:

- The PowerPC is a fixed point microprocessor and, therefore, most of the time is used in the emulation of floating point operations. This microprocessor is in

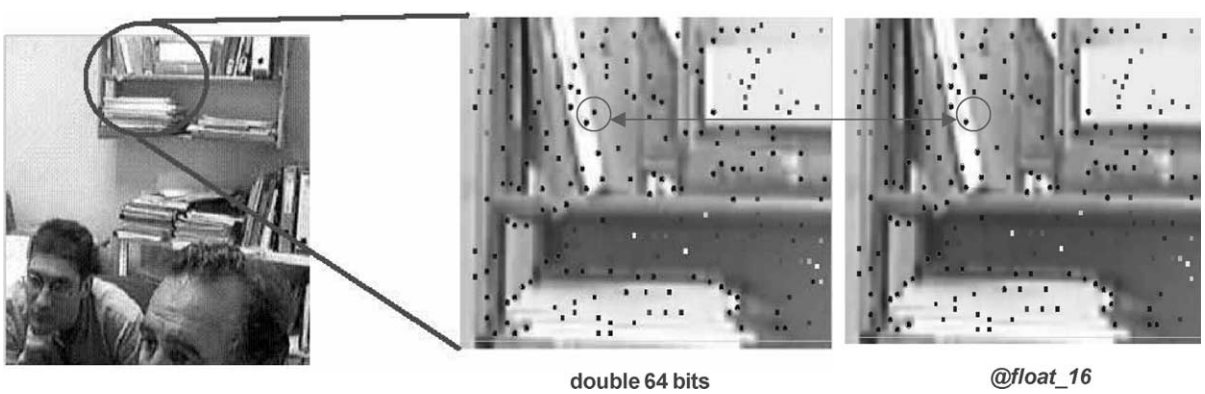

Fig. 9. Corner detection results from: (A) Förstner algorithm, (B) Harris–Stephens algorithm.

Table 3
Execution times for the Förstner ($time_{(F)}$) algorithm and Harris–Stephens ($time_{(H-S)}$) algorithm using the $@float_{16}$ class for the processing of images (256 × 256 pixels) on different platforms

Platform	MHz	$time_{(F)}$ ns	$time_{(H-S)}$ ns
Athlon™	1333	40	41
Athlon™	1000	52	52
Pentium™ Pro	400	68	43
PowerPC MPC823E	50	29244	30691
XCV3200EFG1156	35.7	660	633

Table 4
Failure rate and coverage of simple module replicated three times with SoR and three-versions module with SoR

Component	% Non detected errors (a)	% Detected errors (b)	% Coverage (c = b/(a + b))
Carry-select	5	16	76.2
Look-ahead	7	20	74.1
Ripple-carry	6	18	75.0
SoR	5	19	79.2

charge of processing the 3D-map building algorithm and programming the FPGA. The FPGA is coupled to this processor in order to accelerate the images preprocessing (corner detection algorithms).

- The Pentium™ Pro has a floating point unit and therefore, it obtains very low execution times. There is a difference of 25 ms of execution time between the Förstner (division operation) and Harris–Stephens (only additions, subtractions and products) algorithms. This microprocessor has been used because it is widely used in applications with very intensive computing.
- Athlon™ processors from AMD obtained the lowest execution times. It is a very common platform for standard computing.
- The execution of the algorithms on the FPGA is 20 times slower than the execution on the Athlon™ processors. However, it is to note that the processing frequency of these processors is 35 times faster than the frequency of the FPGA.
- Although, it could seem that the use of a high frequency commercial processor would be the best solution it is not advisable in our case. The implementation of software fault tolerant techniques, such as N-Version Programming, in this kind of processors will produce a great decreasing of the processor performance. The implementation of these techniques on reconfigurable hardware has no real impact on the performance of the system and, therefore, these devices are well suited for that purpose.

Several fault injection campaigns were performed to determine the coverage of the system. The following results (see Table 4) can be extracted from these campaigns:

- The fault tolerant VHDL modules are obviously more reliable than the non-fault tolerant ones (without

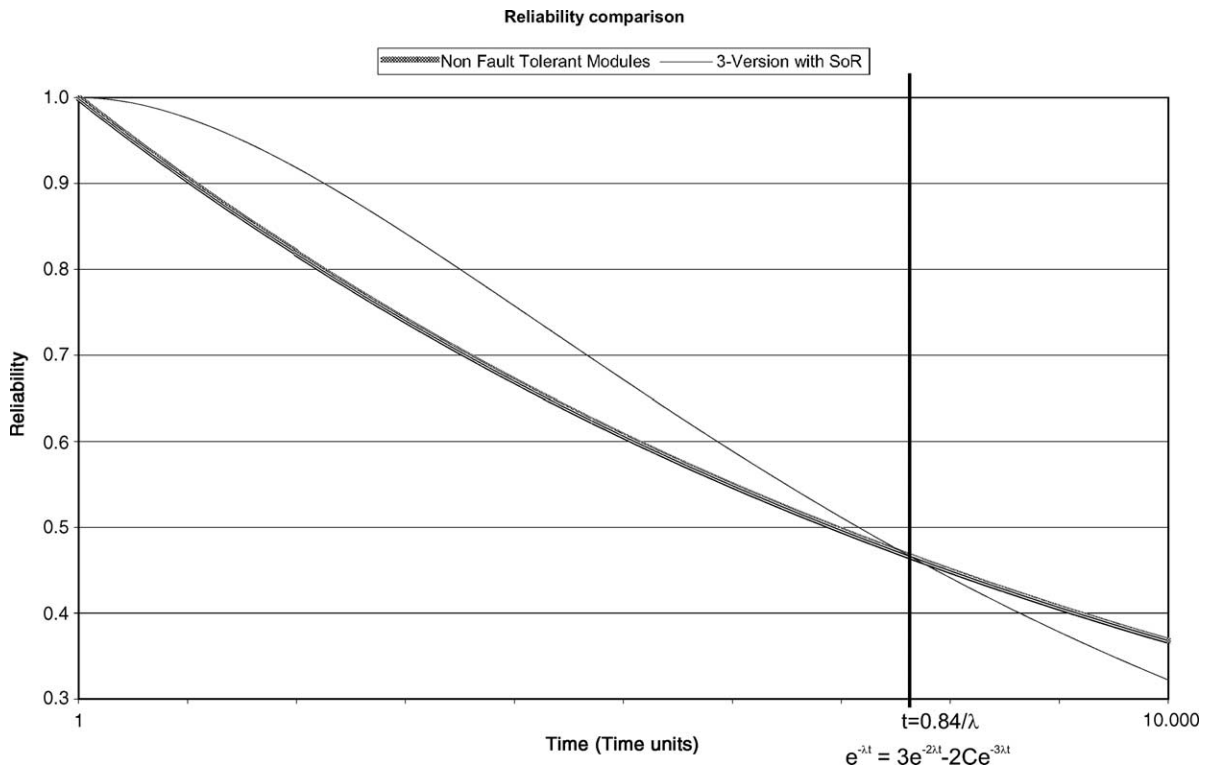

Fig. 10. Reliability comparison between the fault tolerant and non-fault tolerant components.

coverage). These non-fault tolerant components have an average failure rate of 34%.
- The fault tolerant modules behave in the following way in the presence of faults:
  - A simple component replicated three times with SoR: the average failure rate is of 6% (column a), the average error detection is of 18% (column b) and the average coverage for these components is of 75.1% (column c).
  - A three-versions with SoR component: the average failure rate is of 5% (column a), the average error detection is of 19% (column b) and the average coverage for these components is of 79.2% (column c).
- Fig. 10 shows the results obtained from the reliability equation (see Eq. (5)) when the variable **C** is substituted by the previously calculated coverage. The fault rate (λ) clairvoyance of the components has been supposed to be of 0.0001 faults per time unit.

6. Conclusions and future work

From the presented work, it is possible to conclude that:

- It is necessary to follow a proper codesign methodology (like the one presented here) to obtain the desired results. It allows, for example, for the validation of the data types of the operands before implementing the design into a FPGA. In this way, it can obtain an optimal use of the internal resource of the device.
- It is advisable to use a reconfigurable device (FPGA-like) as coupled coprocessor to the main processor. The tasks that need a faster execution time can be implemented on this device.
- Simulations allow for the validation of hardware components before their implementation on reconfigurable hardware.
- The library of arithmetic synthesisable floating point operators allows for the reusability of components.
- In case of dependable system requirements, it will be necessary to add fault tolerance mechanisms to the components.

It is to note that this work has used a *software* fault injection technique (*N-Version Programming*) to develop fault tolerant components that are implemented on real *hardware*. This fact provides better performance than the software-implemented technique. It supports both software faults (in design of the modules) and hardware faults (those that appear in the hardware where the algorithms are running). In particular:

- A fault tolerant floating point arithmetic operators library has been implemented.
- These components have a SoR structure:
  Simple components replicated three times have an average failure rate of 6% and an average coverage of 75.1%.
  three-versions components have an average failure rate of 6% and an average coverage of 79.2%.
- These components can be reused in any design to improve its dependability.
- Obviously, the fault tolerant system is more reliable than the non-fault tolerant one.
- The three-versions with SoR components have a lower failure rate than the other SoR components. Each component implementation takes different area and resources from the FPGA. A fault that affects a particular component with great probability will be very dangerous for the version with three identical simple components. The three-versions components will be more robust in the presence of this kind of fault (the other two modules will work fine).
- However, these fault tolerant implementations use a huge number of FPGA resources. It will be neglected as more logic reconfigurable blocks could be integrated on reconfigurable circuits.

The future work will consist of:

- The fault tolerant components library will be expanded by the addition of more floating point operators such as exponentiation, trigonometric functions, etc. This will allow for the implementation of new fault tolerant computer vision algorithms.
- A new fault injection technique will allow for the injection of faults into FPGAs [30].
- It could also be possible to use the reconfiguration capabilities of the FPGA to implement a faulty module on a fault free area of the device.
- New FPGA are also including microprocessor cores into the reconfigurable logic. Thus, it will be very interesting the improvement of the codesign methodology to use these capabilities to implement both the fault tolerant hardware and software partitions inside the same Field Programmable System on Chip (*FPSoCs*).

References

[1] United Nations Economic Commission for Europe (UNECE) and International Federation of Robotics (IFR), http://www.unece.org/stats/robotics/.
[2] G. Hirzinger, Why robots are just around the corner, Pictures of the Future Magazine, Siemens, 2002, 62–65.
[3] J. Albaladejo, Codesign of a real time image processor, PhD thesis (in Spanish), Computer Engineering Department, Technical University of Valencia, 2003.
[4] Computer Vision and Robotics, University of Girona, http://iiia.udg.es/English/VICOROB_english.html.
[5] X. Armagué, Modelling stereoscopic vision systems for robotic applications, PhD thesis, Universitat de Girona, Spain, 2003.

[6] G. de Micheli, M. Sami, Hardware–software co-design, Kluwer Academic Pub., Dordrecht, The Netherlands, 1996.
[7] COSYnthesis for eMbedded Architectures, http://www.ida.ing.tu-bs.de/general/start.e.shtml, 2003.
[8] K. Compton, S. Hauck, Reconfigurable computing: a survey of systems and software, ACM Computing Surveys, ACM Press, USA, 2002, p. 171–210.
[9] R. Lauwereins, M. Engels, Specification languages, PhD course, K.U. Leuven: ESAT/ACCA, and IMEC-DESICS, 2001.
[10] ESAT-K.U. Leuven, http://lesbos.esat.kuleuven.ac.be/courses.
[11] C. Harris, M. Stephens, A combined corner and edge detector, Alvey Vision Conference, Rovaniemi, Finland (1988) 189–192.
[12] W. Förstner, A feature based correspondence algorithm for image matching, International Archives of Photogrammetry and Remote Sensing, Rovaniemi, Finland XXVI-3/3 (1986) 150–166.
[13] F. Balarin, M. Chiodo, P. Giusto, H. Hsich, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, B. Tabarra, Hardware–software co-design of embedded systems: the POLIS approach, Kluwer Academic Pub, Dordrecht, The Netherlands, 1997.
[14] Matlab™, MathWorks, http://www.mathworks.com/.
[15] IEEE Standard for Binary Floating-Point Arithmetic (#754), Piscataway NJ, 1985.
[16] R.C. Gonzalez, P. Wintz, Digital image processing, Addison-Wesley Pub, Boston, MA, 1987.
[17] L. Kitchen, A. Rosenfeld, Gray-level corner detection, Pattern Recognition Letters 1 (9) (1982) 95–102.
[18] R. García, A proposal to estimate the motion of an underwater vehicle through visual mosaicking, PhD thesis, Universitat de Girona (Spain), 2001.
[19] N. Shirazi, A. Walters, P. Athanas, Quantitative analysis of floating point arithmetic on FPGA based custom computing machines, Workshop on FPGAs for Custom Computing Machines.Napa, USA (1995) 155–163.
[20] A. Jaenicke, W. Luk, Parameterised floating-point arithmetic on FPGAs, IEEE International Conference on Acoustic, Speech, and Signal Processing (2001).
[21] P. Belanovic, Library of parameterised hardware modules for floating point arithmetic with an example application, Ms thesis, Northeastern University, Boston, USA, 2002.
[22] A.A. Gaffar, W. Luk, P.Y.K. Cheung, N. Shirazi, J. Hwang, Automating customisation of floating-point designs, field-programmable logic and applications, Reconfigurable Computing Is Going Mainstream, Springer-Verlag (2002) 523–533.
[23] Virtex 2.5 V field programmable gate arrays, Xilinx DS003-2, v 2.6, 2001.
[24] B. Randel, Systems structures for software fault tolerance, IEEE, Transactions on Software Engineering 1 (2) (1975) 220–232.
[25] D.P. Siewiorek, R.S. Swarz, Reliable computer systems: design and evaluation, A K Peters Ltd, 1998.
[26] P. Gil Fault tolerant system with watchdog processor: validation with physic fault injection. PhD thesis (in Spanish), Technical University of Valencia, 1992.
[27] D. Gil, Validation of fault tolerant systems with fault injection in VHDL models, PhD thesis (in Spanish), Technical University of Valencia, 1999.
[28] J. Albaladejo, D. de Andrés, L. Lemus, P. Gil, Reconfigurable fault tolerant image processor (in Spanish), Workshop on Reconfigurable Computing and Applications, Alicante, Spain, 2001.
[29] ModelSim SE PLUS 5.5e, Mentor Graphics, 2001.
[30] D. de Andrés, J. Albaladejo, L. Lemus, P. Gil, FPGA dynamic reconfiguration for speeding up the simulation based fault injection (in Spanish), Workshop on Reconfigurable Computing and Applications, Madrid, Spain (2003) 39–46.

J. Albaladejo. José Albaladejo graduated in Physics in the University of Valencia in 1984. He joined the Fault Tolerant Systems Group in 1996 and obtained the PhD in Computer Engineering in December 2003. At the present, he is an assistant professor in the Computer Engineering Department of the Technical University of Valencia. His current interests are related to the field of codesign, reconfigurable systems, and fault tolerant systems design.

D. de Andrés. David de Andrés obtained the degree in Computer Science in the Technical University of Valencia in 1998. He joined the Fault Tolerant Systems research Group in the Technical University of Valencia, where he received the MS degree in Computer Engineering in 2000. At present, he is an assistant professor in the Computer Engineering Department of the Technical University of Valencia and is working on his PhD. His main research interests are fault tolerant systems and digital systems design.

L. Lemus. Lenin Lemus obtained the degree of Electrical Engineer in 1987 by the National Polytechnical Institute of Mexico (IPN). In 1991, he received the MS degree on Computer Engineering by the Advanced Studies and Research Institute of the National Polytechnic Institute of Mexico (CINVESTAV). In 2001 He obtained the PhD in Computer Engineering at the Technical University of Valencia (UPV) Spain. His research interests are computer architecture and dependability benchmarking.

J. Salvi. Joaquim Salvi graduated in Computer Science in the Polytechnical University of Catalunya in 1993. He joined the Computer Vision and Robotics Group in the University of Girona, where he received the MS degree in Computer Science in July 1996 and the PhD in Industrial Engineering in January 1998. He received the best thesis award in Industrial Engineering of the University of Girona. At present, he is an associate professor in the Electronics, Computer Engineering and Automation Department of the University of Girona. His current interests are in the field of computer vision and mobile robotics, focusing on structured light, stereovision and camera calibration.