We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

19

Embedded Intelligence on Chip: Some FPGA-based design experiences

Félix Moreno[1], Ignacio López[2], Ricardo Sanz[3], Ruben Salvador[4] and Jaime Alarcon[5]
([1]CEI, [2]TAII , [3]ASLab) Universidad Politécnica de Madrid, Spain
([4]CEI) ETSI Industriales - UPM, Spain
[5]Tec. de Monterrey Campus Toluca, Departamento de Mecatronica, Mexico

1. Introduction

The following FPGA-based (*Field Programmable Gate Array*) experiences stem from a broader line of research investigating methodologies for scaling cognitive/intelligent architectures into limited-resource implementations. Some of these architectures were designed for solving problems involving abstract concepts and uncertain environments, without regarding resource requirements: their original implementations are cluster-based, PC-based or other.

Commercial, embedded applications have afforded being independent from them for years. However, today there exists a drive towards new functionalities which put conventional technologies to their limit. From the point of view of the user, these functionalities stand, for instance, for HMI capable of interacting with the user in natural language, perception systems capable of detecting objects in the environment and reacting to their presence and automatic control systems capable of replacing human operation in complex scenarios.

A representative field of application of these functionalities is the automotive industry. Active research is being carried out to develop highly dependable, low-cost real-time systems that may predict forthcoming accidents, may assess risk situations when driving or that may take control of the vehicle when collision is detected as inevitable (for example some PSSIS: Primary and Secondary Interaction Systems).

From the point of view of the engineer, many of these functionalities involve complex tasks as object recognition, decision making and action planning, supported by enriched models of the environment and the system itself, capable of representing concepts, abstract properties and uncertainty. All of these tasks are carried out as part of the operation of existing cognitive architecture-based systems.

2. What are cognitive or intelligent architectures?

Although there exists certain debate regarding the meaning of cognitive architectures, we can accept two main senses of the term:

Architectures which base their operation in exploiting knowledge.

Architectures that are designed to operate depending on their recognition of the environment.

What is done with that recognition that the system gathers from its environment? The representation of information fluxes and processes from sensors to actuators is the definition of a cognitive architecture. Given the broadness of these definitions, we may revise techniques and grossly categorize cognitive architectures as follows:

- Control architectures: An elementary type of architecture is the classical feedback control loop, from the basic PID implementation, including the wide varieties of topologies derived from it. The basic idea is that the controller tries to maintain the difference between a required variable value and the actual measured one null.
- Reactive and behaviour-based architectures: Control architectures are adequate to operate executing simple, well defined tasks in environments with limited uncertainty and limited sources of perturbance, as the controlled environments in industrial settings. However, other systems such as those designed to move autonomously in an unknown, dynamic environment require more complex control schemes. Reactive architectures are designed to make systems act in response to their environment in such a way that the action of the system appears as a direct reaction to a certain combination of input values. Entire sequences of tasks may be executed as a reaction to a certain input pattern. When the inputs change, ongoing tasks may be interrupted and replaced by new ones, according to a task allocation hierarchy which is also pre-designed in the system architecture as a function of the system inputs. This combination of task sequences and allocation hierarchies allow reactive architectures to operate within a range of conditions suitable for some basic functionalities (explorer robots, automatic hoovers). A system out of this range, due to inadequate design or to unexpected environmental events would lead to undetermined or undesired actions.
- Goal-driven architectures: This type of architectures are designed to operate in highly uncertain environments, typically where some kind of analysis is required to assess the conditions of operation, prior to the system executing any action. Uncertainty emerges from a mismatch between the system and its scenario of operation: the system not being configured for a particular environment, so that it ignores what may happen next and therefore the consequences of a potential action. Goal–driven architectures are designed to achieve their objectives in these circumstances by first analyzing the situation, then building a model, and finally designing and executing the appropriate actions. The major difference from the other types of architecture is that environment models and possible actions, which are static in control and in reactive architectures, are built in run time and changed by the system itself. In general, goal driven architectures operate following a common, basic sequence of processes executed in cycles: a) Build an objective, b) Analyze the environment, c) Design a task to achieve the objective within the given environment. If this cannot be done, build a sequence of lower level targets aimed at progressing toward the higher level objective.

  These processes may imply highly developed perceptive, deliberative and actuation functions. In parallel to them, these architectures may implement learning algorithms that help optimize the system for future or eventual conditions of operation. As a result, these architectures may achieve high levels of autonomy. However, some limitations have been met when implementing them in actual systems, relative to resource consumption. Some deliberative, learning and

perceptive processes would require extremely large memory and computational resources for real time operation, especially in fast-evolving environments, or when dealing with highly abstract tasks.

There exist hybrid approaches which combine and integrate devices and elements of the three categories above. In particular, PID controllers are used by the majority of implementations dealing with mechanical systems, although they may be reconfigured in real time by complex goal-driven architectures.

The ultimate goal of the research line which will be illustrated by the FPGA experiences described in subsequent sections is to build systematic methodologies in order to engineer all aspects of scaling high level goal architectures to low cost, real time devices.

3. Scaling high level architectures to low-cost FPGAs

We may realize that the process of designing a low cost, embedded cognitive architecture implies a wide range of problems motivated by two major properties of the triad system-environment-desired functionality:

- **Complexity** of the desired functionality, the given system and especially the environment in which it will be operating.
- **Uncertainty**, derived from the vast range of possible scenarios of operation that may emerge.

3.1 Managing complexity

There exist two main techniques for reducing model complexity:

- Excluding or ignoring variables: Not measuring or evaluating them.
- Coarsening measurement: Reducing resolution enough to be representative but avoiding unnecessary detail.

These two techniques must be applied repeatedly until a reasonable degree of representativeness is reached.

In practice, eliminating uncertainty completely from an environmental model is either impossible or would make the system useless. We have to bear in mind that many of the functionalities that we shall be trying to implement have to do with making the system predict events or analyze scenarios, in which it is difficult to know precisely what is going to happen or how intense it will be. In other words: uncertainty will be intrinsic to our own application, so our model should be able to represent it in some way. In conclusion, when a certain optimal is achieved in complexity, uncertainty may and must not be completely eliminated from the model of the environment. The problem now is managing uncertainty.

3.2 Managing uncertainty

As it has been pointed out, uncertainty may appear in two ways: not knowing what is going to happen or not knowing how intense it will be. More formally, we can say that there are two types of uncertainty:

- Qualitative uncertainty: Ignoring the nature of the actual event that may occur or that is already taking place. This type of uncertainty is the specialty of goal driven architectures.

- Quantitative/intensive uncertainty: Given a certain event, ignoring its intensity. A typical example of this is ignoring the actual value that will be measured the next instant.

There exist well known, established techniques which allow dealing with both types of uncertainty. Knowing which variable to measure and how to act upon it, it is only a matter of estimating how intense the action must be. This problem can be successfully managed by classical control. Enhancements of the classical PID loop such as some non-linear controllers or model reference adaptive control (MRAC) even achieve a limited range of reaction to qualitative uncertainty.

In general, managing qualitative uncertainty is a much deeper and broader problem. Classical artificial intelligence techniques as neural networks, fuzzy logic and expert systems offer basic tools. Expert systems and fuzzy logic in combination may analyze uncertain scenarios provided that their dynamics fall within the range covered by the rule databases, variables and member functions are well designed. Neural networks may analyze sensor readings in raw and extract patterns within the limitations of their own type (ie. perceptron, cognitron), number of neurons, etc.

However, some new functionalities require wider ranges of qualitative uncertainty management than those provided by these techniques. Complex environmental analysis, complex action selection processes and advanced learning mechanisms, exceeding typical neural network ones, are some examples. This is the use of cognitive architectures.

The role of a cognitive architecture is to define a sequence of operation and to assign roles to the resources of the system, these resources and processes being implemented by expert systems, neural networks, PIDs or whatever other technique, hardware or software. Scaling cognitive architectures equals to selecting which parts of their original specification are really needed and how could they be simplified, which stages of the operating cycles are indispensable and how to integrate both while preserving functionality.

3.3 Perception, deliberation, action

The operation of any system, can, in general, be explained in terms of perceptive, deliberative and actuating processes. Sometimes the processes are not designed thinking of any of those three roles, though inevitably they end up performing one when functioning. Complex realizations of any of them may rely on the use of extensive memory resources, which may also be classified in roles: long-term memory and short-term memory. While the first contains information that may stay in the system during long periods of activity, short term memory contains transiting variable values or sensory measurements as such. It can be observed that, as a general rule, goal-driven architectures tend to exploit long term memory (which stores knowledge) as intensely as short term memory, while reactive architectures, with simpler design, may even have none.

For scaling cognitive architectures (Albus, 1995), it is useful to have a clear idea of the character of the system to be designed: whether it will be mainly perceptive, deliberative or actuating. This will allow a gross idea of the necessary resources. In general, systems centered in action processes will demand little or none deliberative processing (action selection, planning, decision making), while problem solving systems will demand little action processes.

Of course, it is not only the type of processing we are interested, but mainly the type and quantity of resources each type of process has associated. Sometimes, the designer is able to

choose the nature of the system, or it is given by the application itself. For example, in the case of ADAS (Advanced Driver Assistance System), the target is to warn or inform the driver about a variety of risks, options for driving, etc. This case is expected to require little or none actuating functions except, perhaps, some kind of HMI. On the contrary, perceptive functions are expected to become quite developed, if analyzing risk for example. On the other hand, an autonomous driving vehicle may rely on extensive perceptive and deliberation processes for analyzing the environment, possible action and tradeoffs.

4. The process of scaling an architecture

Scaling a cognitive architecture adds up to finding a match between functionality, architecture and implementation. The process is simple: given a certain functionality, a certain architecture must be designed, by iteration, coupling capacities and resources with the implementation design. During the process, the engineer must continuously work for optimization of resource use. There is no general formula as to how to optimize. The two ways of dealing with complexity which were mentioned above apply, however: first, to eliminate any parts, elements or functions of the architecture that are not strictly needed to achieve the desired functionality. Second, to eliminate any excess of resolution in measurements, calculations and precision.

Exaggerated examples of these rules help to understand their meaning. If you have the necessary information to make your application work by only adding and subtracting, do not build a cognitive architecture to do the same job. It will consume more resources and take more time, and may even do it wrong sometimes. If your artificial vision system must detect spots between 1 and 2 cm^2 at 1 m distance, do not spend money in 1920x1200 resolution cameras and image processing algorithms. A low resolution camera will do the filtering for free, quicker and cheaper. If your system will always operate with objects A, B and C, you may spare designing learning algorithms that will enable it recognizing any new object around.

Naturally, there might not always exist a solution to a scaling problem, due either to an exceedingly complex functionality or to insufficient resources. Logically, the more strict the functionality specifications or the higher the system flexibility they demand, the more difficult it will be to simplify any architecture into low cost hardware. Any scaling process should begin with dividing functionality specifications, to ensure that parts may be achieved progressively in spite of possible overall failure.

When a solution is possible, however, a good approach is to start assuming the best of cases, when resources are unlimited, designing the basic architecture for it, and proceed simplifying the design in each iteration. The process ends when a certain match is achieved between the three. That is, when the desired functionality has yielded a simplified architecture that can be implemented in the available resources. Over-optimizing the architecture is, nevertheless, good for two reasons. First, it may allow implementing the system in lower cost hardware, or may leave free resources in your board for other purposes. Second, it reduces the probability of errors. In general, the bigger the system –and therefore, its architecture- the higher its probability of failure; so keep it as small as possible.

While resource optimization derives from minimizing architectural complexity, there are overall factors to be taken into account during the design stage. A basic collection of criteria could be:

- The automatic process admits drastic optimization in time and resources. When possible, make things automatic.
- The higher the uncertainty in the environment, the more flexible the system, the less automatic it can be, the more resources necessary. The better the modeling of the environment, the lesser the uncertainty.
- There is a tradeoff between memory and computational power: in general designing memory-based processes may simplify run time computation and vice versa. If you have enough memory and information to store, use it. It makes things more automatic.
- In general, there is also a tradeoff between deliberation and perception: the more developed the perceptive processes, the simpler the deliberative processes may be and vice versa. But here there is no general rule, for perceptive processes may be as complex and demanding as purely deliberative ones.

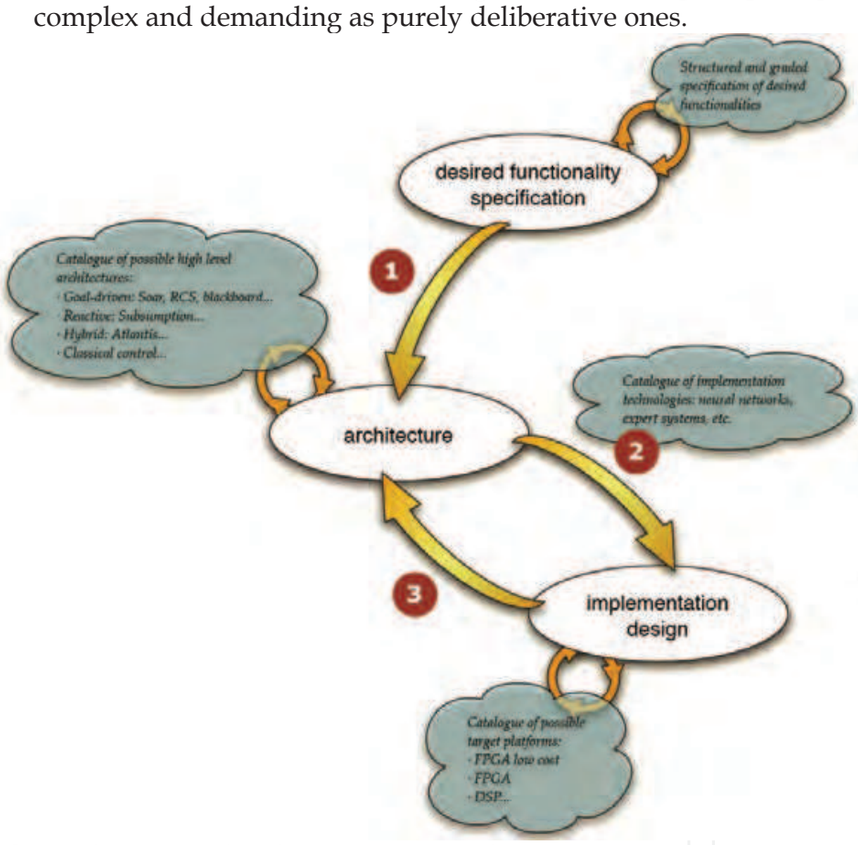


Fig. 0. General process of scaling cognitive architectures

5. A low-cost Real-Time FPGA solution for driver drowsiness detection

In this work some of the most recent advances in digital image processing techniques have been used to make vehicle drivers face analysis by detecting symptoms of tiredness and distraction in order to prevent sudden risk situations.

The results of the experiments show (Moreno et al., 2003) that a large number of car or trucks accidents can be avoided by detecting real-time physical and psychological states of the drivers in normal driving conditions. There are three main objectives in this design: To

detect the driver eyelid movements, to detect the number of frames the driver has his eyes closed and to detect when the driver turns right or left (or bows) his head for a long time. Thus, several well known algorithms have been used and optimized for this field of application, such as spatial and temporal filtering, motion detection, optical flow analysis, etc.

Digital signal and image processing techniques have been used together. Furthermore, a low-cost Real-Time solution based upon both FPGA (ALTERA FLEX 10K30 and ALTERA Cyclone Device EP1C3) have been achieved.

Figure 1 shows the flexibility of the system, because it can be used for driver drowsiness detection or road lane markers, both in real-time.

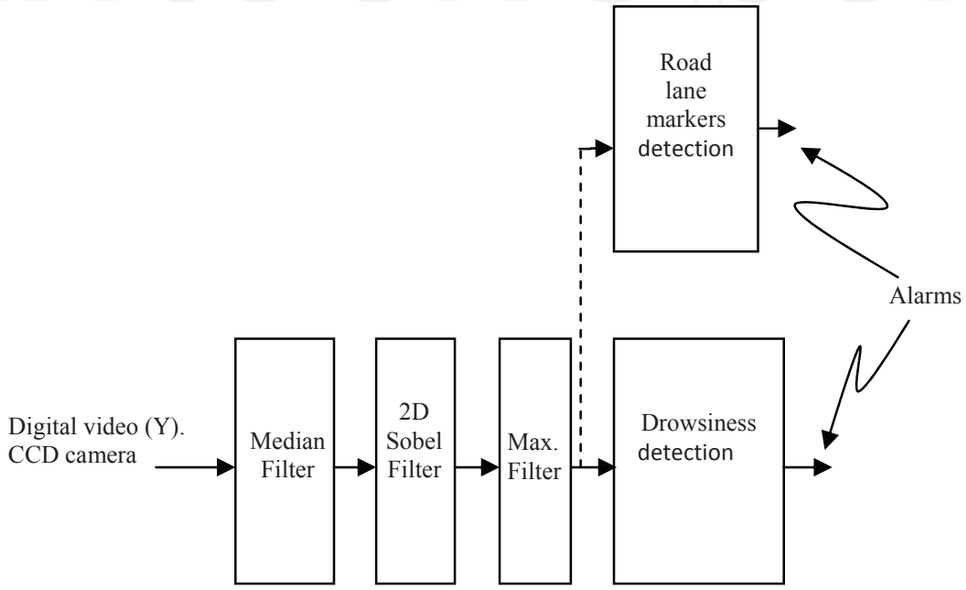


Fig. 1. FPGA architecture

Median filtering is used to minimize the effect of the Gaussian noise. It is very well suited for this type of application; however, the trade-off between the number of gates (FPGA logic elements, LE) used and the benefit obtained is very poor. For this reason, and in order to get a very low-cost solution, removal from the final design must be considered; at the same time a high quality digital video signal coming from a CCD camera must be employed containing an infrared system suitable for low-light conditions (0 lux) must be employed. Some experiments were carried out with very low-cost CMOS cameras; but, unfortunately they do not work properly in low-light situations.

Next, the 2D Sobel filtering uses two matrices applied sequentially over 9x9 pels (picture element) blocks:

$$\begin{Bmatrix} -1,-2,-1 \\ +0,+0,+0 \\ +1,+2,+1 \end{Bmatrix} \quad \begin{Bmatrix} -1,+0,+1 \\ -2,+0,+2 \\ -1,+0,+1 \end{Bmatrix} \tag{1}$$

Then, the final result is calculated by the equation:

$$|Ysob| = \sqrt{|\, y_{1d}^2 - y_{2d}^2 \,|} \tag{2}$$

Finally, Ysob is compared to a configurable threshold value (UMB_SBL):

```
If Ysob > UMB_SBL then
        Ypel = White_value (235d);
else
        Ypel = Black_value(16d);
end if;
```

This pseudo-code is not exactly a Sobel algorithm because we assign the White_value or Black_value instead the |Ysob| (Sobel parameter/luminance_pel). This allows us to simplify the calculations just suitable for the case of the driver drowsiness detection or road lane markers detection, without introducing a noticeable error. It also allows us to save a large amount of FPGA logic elements (making the fitter process much easier) and as a result of that we get a significant speed up (in terms of logic depth reduction).

Once the image has been filtered (median and 2D Sobel filtering), Maximum filtering is used usually to enlarge the edges of the objects detected by the Sobel filtering. However, this is not really necessary, because the algorithm for driver drowsiness detection and the algorithm for road lane markers detection are both based upon differential optical flow analysis, instead of traditional motion estimation algorithms.

The algorithm designed is able to detect: driver eyelid movements, number of frames the driver has his eyes closed and when the driver turns right or left (or bows) his head for a long time (this is a configurable parameter). In our system, a long time means 12 frames because at 25 frames/sec (PAL video rate) the elapsed time is 480msec. So, if the car speed were 120km/h, the distance covered would be 16 meters.

The implemented algorithm consists in comparing the number of white pels in the current frame with two parameters (maximum and minimum white pels values) previously calculated. When the current number of white pels is greater than the maximum value or smaller than the minimum one, the algorithm automatically calculates those new parameters over the next 12 frames, and the process would start again. This is very useful in order to adapt system sensitivity to light conditions.

According to ITU-R 601 Recommendation for PAL systems, each frame is formed by 720x576pels, which produces a very large amount of luminance information to be processed (414,720 pels/frame). The key factor of our system is to process only the area where the driver eyes would be located. We have tested algorithms for eyes detection but the trade-off between cost and performance is very poor. We propose to adjust the camera to the drivers'head in a comfortable driving position previously and to set the image area to be processed (configurable parameter) at the beginning (Figure 2 shows the result when processing the whole frame, no real-time and Figure 3 shows the result when processing only a specific frame area, in real-time).

(2)

Fig. 2. Drowsiness detection

Fig. 3. Real-time drowsiness detection

The reduction of the number of pels to be processed is very significant: 20,400pels for 100 lines x 204 pels area (see Figure 3). Nevertheless, Median and Sobel filtering work over 9x9pels blocks in order to yield a processed pel; so only by means of a pipeline architecture would real-time processing be achieved.

In the case of road lane markers detection algorithm only the 24 lines of the bottom of each frame are processed, because only those lines are really relevant to detect if a car is leaving its tracks due to a driver distraction. On the other hand, in our system only one video camera is employed, so, no 3D image analysis algorithm can be used at all. We have used a very simple approach that consists on fitting the camera zoom to the car width. In this way, while driving along, the lane markers "disappear" just through the right side and/or the left side of the image. In case of the car is out of the track, the lane markers would "disappear" through the bottom side of the image and the system would warn the driver (see Figure 4). Some experiments have been carried out on real roads and we have obtained very satisfactory results.

The algorithm consists in obtaining the difference between the present frame and the frame immediately before. The difference is calculated over all the pels belonging to the last 24 bottom lines of each frame. If the result is zero and the pel processed is a White_pel, this means that a lane marker probably starts in this frame. The process continues over the next frames. The algorithm is able to detect if the object detected is really a lane marker or just a shadow or other disturbance on the road.

Fig. 5. Road line tracks detection

6. A new Real-Time Hardware Architecture for Road Line Tracking Using a Particle Filter

In this work a new real-time hardware architecture based on real time image processing and the use of a Particle Filter, as the fundamental element for tracking lines of a road, is presented. To this end a hardware system has been designed based on the use of low-cost high-reliability FPGA integrated circuits (ALTERA-Cyclone and ALTERA-Cyclone II). For this purpose, a multilevel pipeline architecture (at pixel block - 3x3- and pixel level), which aims to guarantee the processing of the 8-bit digitized images obtained from a single video camera (SONY in PAL format, ITU-R 601, ITU-R 656), has been developed. The entire processing and prediction system has been developed in VHDL-93, simulated and synthesized with ModelSim and Quartus-II respectively (Alarcon et al., 2006).

Although many systems have been proposed for detecting involuntary lane departure of motor vehicles, based on vision systems (or based on other technologies), they have not had the expected success. System reliability is limited by weather conditions and visibility, as well as those imposed by the state of the highway. In general, it can be said that these systems basically perform three functions: 1) image feature extraction, 2) matching and 3) taking decisions. Moreover, all of them should be considered deterministic functions, except in some approaches, real time in most cases and, to a lesser extent, implemented in application specific hardware.

A new model to represent lane lines and the relative position of the vehicle with respect to the lane boundaries is proposed. The position of the lane lines is tracked on the successive images obtained from the camera (25 fps, PAL frame rate), by projecting the model. Model parameters are updated by superimposing the image, with the projection of the model on the following image. In the model, a search area of the lane boundaries is defined on the image, which allows processing time to be reduced, and the problems caused by false lane detections resulting from the inherent noise in the images to be reduced or eliminated. The parameters of the model are processed and updated by means of the Particle Filter.

A robust, artificial vision based system for detecting involuntary lane departure which depends only on the processing speed and the frame rate of the camera has been developed. This system allows road line position to be predicted with a bounded error.

The Particle Filter, in Artificial Vision applications, is used for tracking objects in an image sequence. The Particle Filter models the probability function a posteriori of a stochastic

process, by means of an N particle distribution and their associated probabilities in State Space. This type of filter needs a dispersed search to properly track the features sought in the image sequence. It is a model used to calculate the state of a time-variant system or, in other words, a sequential estimation algorithm.
In this study, artificial vision techniques have been applied with no restriction on the incoming images. Hence no type of marker is used on the highway. So the system, after the acquisition of the monochrome images of a camera, is able to process the information and to track the lane lines in a reliable and robust manner.
The VHDL implementation of the proposed system, shown in Figure 6, required an image preprocessing stage. Median filtering, a non linear filter, is used to minimize the effect of Gaussian noise, and Sobel filter is used to detect edges in the image.

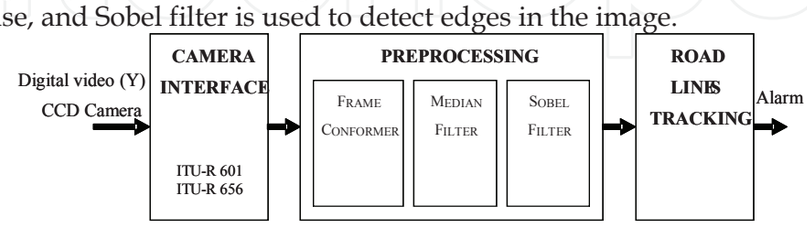


Fig. 6. System architecture

For the system to work optimally, several morphological operations must be carried out on binary images to obtain the necessary information in filter processing. The reason for this is based on the computational load reduction necessary for the hardware implementation, since not all the information in the image is pertinent for the reference application. As it has been mentioned, only the bottom part of the image is relevant, as shown in Figure 7; and, inside that part, the image has been segmented into three regions, left, center and right, of 37x102 (rows x columns) pixels, corresponding to the Regions of Interest (RoI), where the corresponding lanes are located.

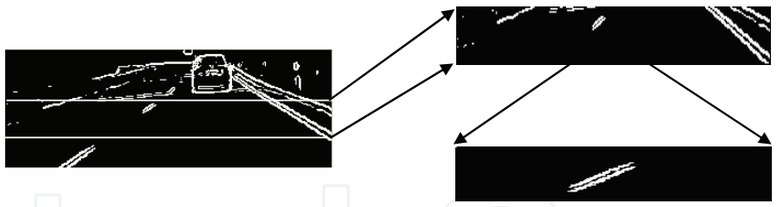


Fig. 7. Image Segmentation

From an algorithmic point of view, the Particle Filter (Arulampalam et al., 2002) is detailed in Figure 8. The Line Model Detection (L_M_D) carries out line detection in the image by means of a morphological line model. Next, particle weights computation is done in the Survival Model (S_M), checking if the error made in the prediction of the Center of Mass done in the previous image, is delimited as explained in Hardware Architecture section. At the same time, in Particle Displacement (P_D), the displacement speed of the particles is computed. By means of the Movement Model (explained in Hardware Architecture section) in Prediction Update (P_U), the position of the new particles is predicted and updated.

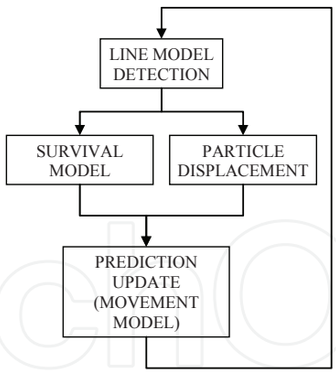


Fig. 8. Diagram of the Particle Filter designed

Some of the results obtained with the proposed architecture are shown in Figure 9. It can be observed that the 12 initially generated particles (Figure 9.a) predict the position of the lane line in accordance with the established deviation parameters (deviation = 0.1) (see Figure 9.b), and how they converge in the last 2 images (Figure 9.c and 9.d) toward the real position of the Center of Mass of the line.

One of the essential objectives of the hardware implementation of the algorithms was to achieve a high processing speed to get a real-time low-cost system. The hardware system implemented in the FPGA, Figure 6, corresponds to a linear multilevel pipeline architecture. These hardware modules implement the first pipeline level of the architecture adapting perfectly to the requirements of the image prefiltering algorithms:

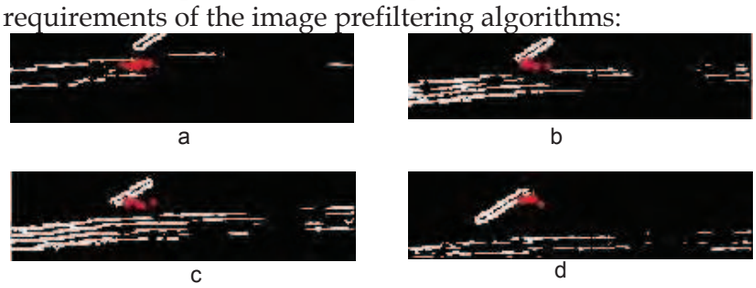


Fig. 9. Prediction of the lane line using 12 particles

a) Median Filter, to suppress the Gaussian noise and b) Sobel Filter, to extract the boundaries. Furthermore, all the submodules that constitute the complete system are also based on a pipeline architecture and are capable of delivering a valid result in each clock cycle during their stable operating cycle. The hardware design methodology employed in the second module, Preprocessing, is directed to minimizing both the use of large memory banks for storage of complete frames and the complexity of the control system. Figure 10 shows a more detailed design of the 3 submodules that make up this Preprocessing Block. The use of FIFO buffers allows a relaxation in memory restrictions, by not having to store a whole image. In short, it is only necessary to store two lines of an image in the corresponding FIFO. The Filter Control, implemented in a distributed manner, together with the adaptation of the information obtained from the camera via the Camera Interface and Frame Conformer blocks, Figure 6, guarantees a minimum use of hardware resources, at the same time as total independence of the camera and of the Particle Filter model used respectively. This is possible by adding a header (sync embedded bits) to each of the pixels

coming from the camera, which indicates the type of pixel being handled at all times: SoFrm (Start of Frame), SoLn (Start of Line), PoLn (Pixel of Line), EoFrm (End of Frame) and NoPix (No Pixel). At the same time some discrete synchronization signals are created in the system. This results in a Datapath width of 11 bits.

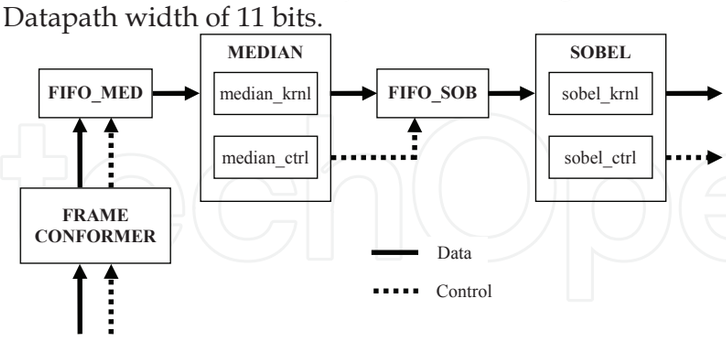


Fig. 10. Preprocessing Module

The data frame so formed has allowed each of the filters to be structured in two major submodules: the filter Kernel and the filter Control, respectively, both in the Median and in the Sobel filter. The Kernel responds to a linear pipeline structure at pixel level and 3x3 pixel block whilst the Distributed Control in each of the blocks guarantees a maximum throughput for a minimum consumption of logic.

As for the most important module in the system, the Particle Filter, Figure 11 shows the hardware implementation of the architecture which, written in VHDL, operates over the RoI, Figures 6 and 7.

In each image the presence of lane lines is established and their Center of Mass is calculated, thereafter predicting their position by application of the Movement Model which can be seen before. Until all terms present in this equation are available, three consecutive images (frame0, frame1 and frame2) are needed for prediction and full tracking.

To functionally explain the hardware implementation of the algorithm, it must be assumed that the Centers of Mass of the lane lines for the initial and successive frames have been found, and that the predictions are correct. In such a case, the algorithm is processed as indicated previously. The procedure, in other case, will be explained later.

$$\begin{aligned} t &= 0 & X_1 &= MC + RND_{init} \\ t &= 1 & X_2 &= X_1 + \left[RND_\sigma\right] \\ t &= 2 & X_3 &= X_2 + \left[\left(X_2 - X_1\right) + RND_\sigma\right] \\ \forall t &\geq 2 & X_{t+1} &= X_t + \left[\left(X_t - X_{t-1}\right) + RND_\sigma\right] \end{aligned} \tag{3}$$

where $X = \{x_0, x_1, ..., x_{n-1}\}$ is the set of all the particles, MC the centers of mass detected, RNDinit the initialization random function, and RNDσ the random function that represents the variance term of the Particle Filter. Each increment of t is assumed to be in whole multiples of TFrame (TF, 40 ms, PAL rate). The design of the architecture has been carried out so that each particle is evaluated and appropriately updated, independently of the rest. This allows each of them to be found in one of the three states described before, and which are explained below.

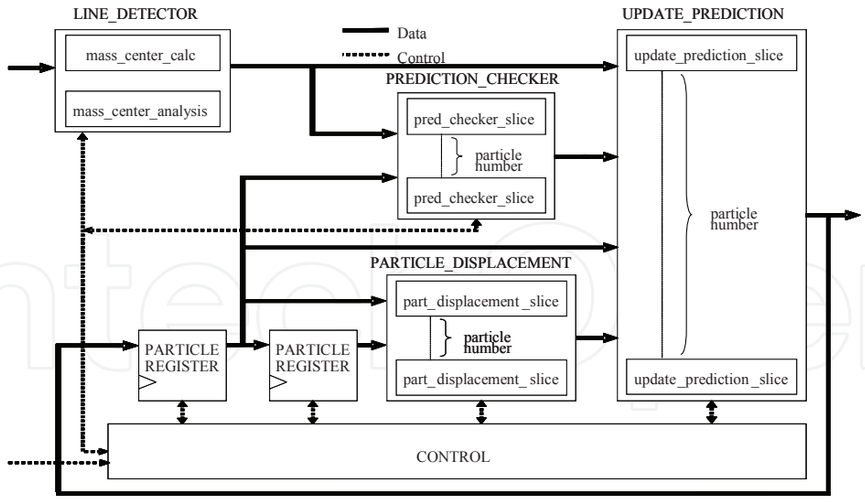


Fig. 11. Functional Description of the Particle Filter Hardware Implementation

The equation shown below, which represents the so called Movement Model, governs the overall behavior of the filter:

$$X_{t+1} = X_t + \left[\left(X_t - X_{t-1}\right) + RND_\sigma\right]$$

1) init_state → t = 0. frame0 is analyzed, so there is no initial information at all. After finding the Centers of Mass of the line, the particles are distributed around the same with a dispersion given by RNDinit, the latter constituting the prediction of the position of the Center of Mass in the following image made from that detected in the current image.
2) transitory_state → t = 1. The prediction consists on applying the variance term to the prediction made in the previous image.
3) steady_state → t ≥ 2. All data is now available to apply the equation of the filter completely, that is, the previous prediction Xt, the particle displacement term (Xt – Xt-1), which represents the speed at which the lines are moving in the real scenario, and the variance RNDσ.

This routine is followed for the case in which the error committed in the prediction, measured with respect to the Center of Mass of the current image, is delimited within a margin of error defined by the maximum value of RNDinit. At this point, steady_state, two situations can arise:

1. A new line is detected. Depending on the error made, two further situations can arise.
   - The error is bounded. This is the general case, steady_state, which operation process has been explained above, (1).
   - The error is not bounded. The calculation sequence begins again for the three following frames, considering this as init_state, t = 0.
2. It is in the case where no line is detected that the prediction makes most sense, being the prediction validated when a new line is found in the following frames. Thus, when the line appears again, its position will have been predicted, and it will be possible to evaluate the correctness thereof, proceeding again as in point 1.

The Particle Filter subsystem implemented uses 4 particles for each video line analyzed, resulting a total of 12 particles, operating only on the horizontal axis of the image. The

dynamic margin (range) of the pseudo random numbers generated, RNDinit (4-bit LFSR) and RNDσ (3-bit LFSR), as well as the number of particles and the number of bits used for their encoding, are closely related to the number of video lines used for detecting the lane lines, the width of the RoI and the number of frames necessary to establish a good prediction in real time (PAL frame rate = 25 fps). The impact in dimensioning the DataPath of the pipeline architecture and its control, have constituted one of the main challenges in implementing the total system. The number of bits used to encode the particles is 10, representing integers with sign, whereby this coding could be employed for RoIs of up to 512 pixels in width.

Synthesis results for different low-cost FPGAs were obtained, trying to compare the different architectures. This Module is composed of two FIFO memories that store 2 video lines for the implementation of the Median and Sobel Filters respectively. The Distributed Control of this Module has the responsibility of ensuring that the information at pixel level reaches the Particle Filter Module, Figure 11, in optimum conditions. A Cyclone EP1C20F400C7 FPGA has been taken as the reference low-cost development system.

The FIFO memories were implemented making use of internal RAM resources available in the FPGA (294912 bits distributed in 64 blocks of 4608 bits, known as M4Ks). The results obtained with Quartus II "MegaWizard Plug-in Manager", an assistant that helps in the instantiation of architecture-specific resources, have been compared with the direct instantiation of the *scfifo lybrary* component. This has been the design decision made since errors were detected in the assistant that made the creation of the desired FIFO memory impossible.

The Median and Sobel Filters FIFOs -11 bits per word- (Figure 10) differ in two positions. It must be pointed out that, as far as synthesis is concerned, this has a relative importance. After several synthesis tests, it was observed that the design of the LPM component of Altera *scfifo* (single-clock fifo) and the mapping process of the memory in HW, always produce a FIFO with a power of two number of positions. Thus, a FIFO of 126 positions would occupy the same resources (memory and LEs) as one of 128. On the other hand, one of 129 positions will occupy the same as one of 256. The difference probably rest in the generation of the FIFO control signals. Besides, it also must be pointed out that the proposed architecture for both the Sobel and the Median filter, as regards the number of FPGA resources employed, is completely independent of the size of the image to be processed. The key point in the architecture that makes such synthesis results possible is the series communication protocol implemented at pixel level. This protocol makes that the Submodules that implement the filters do not need to know the RoI size. This information is managed by the Frame Conformer block, being the insignificant increase in the size of the Preprocessing Module due to this submodule, as well as to the FIFOs.The synthesis of the Particle Filter, as well as its fitting in different types of FPGA with a different architecture, number of logic elements (LE), availability of internal RAM memory, etc., have allowed a comparison to be carried out among all of them and some final conclusions to be reached. This Filter is the module in the architecture of the system developed that consumes most logic element (LE) resources.

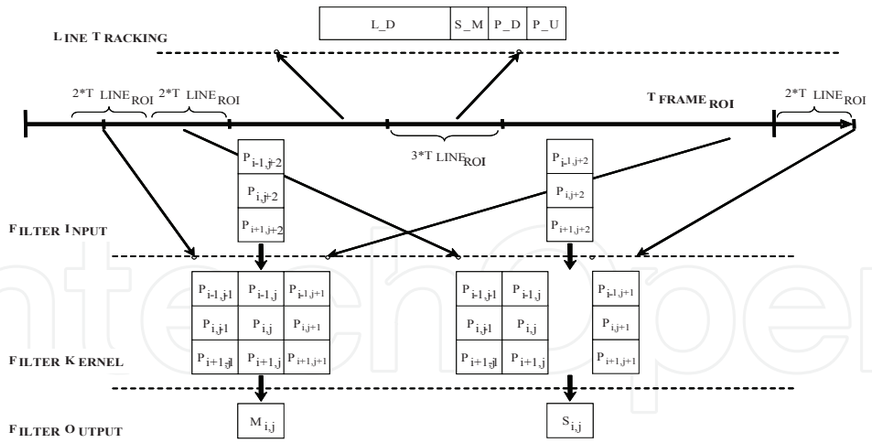


Fig. 12. System Pipeline Processing

7. FPGA Implementation of an Image Recognition System based on Tiny Neural Networks and on-line Reconfiguration

Neural networks are widely used in pattern recognition, security applications and robot control. We propose a hardware architecture system; using Tiny Neural Networks (TNN) specialized in image recognition. The generic TNN architecture allows expandability by means of mapping several Basic units (layers) and dynamic reconfiguration; depending on the application specific demands. One of the most important features of Tiny Neural Networks (TNN) is their learning ability. Weight modification and architecture reconfiguration can be carried out in run time. Our system performs shape identification by the interpretation of their singularities. This is achieved by interconnecting several specialized TNN (López et al., 2007).

There are several levels of parallelism in the neural network recognition system that we are proposing: Parallelism among networks, among the layers of a network, among neurons and among connections. All of them are shown on the General Architecture of the system (Figure 13).

We can classify the ANN hardware implementation in two main categories: that based on microprocessors by using Digital Signal Processors (DSP) or general purpose processors, and that using an Application Specific Integrated Circuit (ASIC) or Field Programmable Gate Array (FPGA).

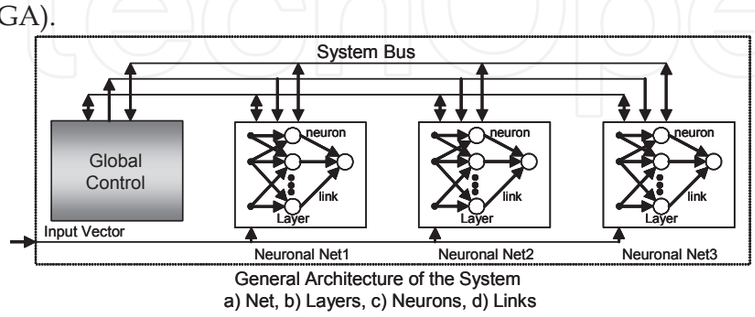


Fig. 13. System Architecture

The first one based on microprocessors, is more flexible and relatively easy to implement. However, when the network becomes larger (for example, in a fully connected network), it is not the best option: general purpose computers are required, and the usage and application depend on power and area characteristics available.
The number of "synapses" and multipliers included in a fully interconnected network is proportional to the squared total number of neurons. The speed slows down due to the increase in the number of multipliers, and the chip size or chip area increases significantly, which becomes one of the critical points in ANN design. In order to solve this problem, the use of hardware multipliers seems to be an option to resolve the chip size problem; as well as the design of neural networks without multipliers or reusable ones. Our work explores multiplier re-usability based on an internal bus structure. Taking into account the parallelism of the neural network model, it is possible to map the architecture on array processors, obtaining a linear growth in the number of multipliers. Figure 14 shows a network interconnected by mean of an array processor model.

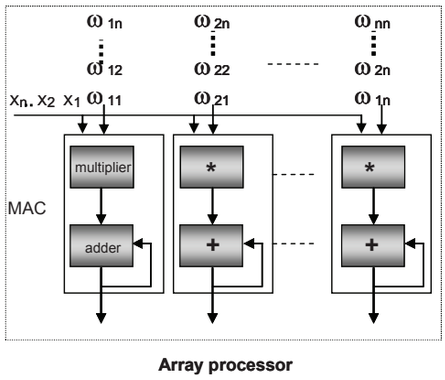


Fig. 14. Network Example

The main objective of this research is the design of a reconfigurable, efficient, low cost architecture for shape recognition. Robust methods for the analysis of images, and the implementation of a system based on specialized TNN have been developed for shape recognition by means of the analysis of some characteristics of the image (singularities). Traffic signal recognition and/or pedestrian recognition are two of the most relevant applications. These networks work cooperatively to obtain the classification of the image.
The main restriction comes with the complexity of the information contained in the image data, because they are sensible to changes of the environment. It is then necessary to have a recognition system that allows dynamic reconfiguration. It is necessary to develop an architecture that allows optimum usage of hardware resources, due to the limitations in power and available area. The suggested system is formed by small Perceptron multilevel networks, and it was implemented in an Altera Cyclone II FPGA.
The requirements of recurrent learning processes can be satisfied by the reconfiguration and flexibility of FPGAs. Weight modification and architecture reconfiguration can be carried out during run time.
When talking about ANN implementation, the following considerations should be taken into account: frequency, precision, configuration issues, and ANN parallelism. In order to improve general design characteristics, there are two units: Basic units and Control units.

Basic units (specialized neural networks) are in charge of signal processing and weight and bias data storage, including the multiplication of the weights by the inputs, the accumulation and the nonlinear function activation. Control units work on the basic of signal transmission including parallel processing and the algorithm work. By considering those units, the proposed design (as shown in Figure 15) has an efficient architecture based on specialized neural networks by recognition, to be implemented in FPGAs.

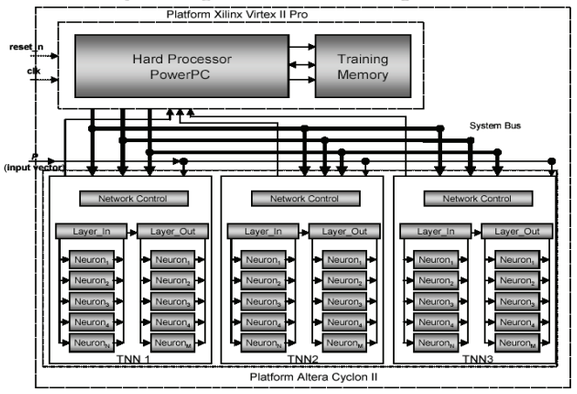


Fig. 15. System Architecture

For achieving the learning operation the algorithm is divided in three phases, known as: feed-forward, back-propagation and up-date. In the feed-forward phase the input signals propagate through the network layer by layer, eventually producing some response at the output of the network. This response is compared with the desired (target) response, generating error signals that are propagated in backward direction through the network. In this backward phase of operation, the free parameters of the network are adjusted so as to minimize the sum of square error. Finally, weights and biases are updated using the data obtained in the previous phase. The process is repeated as many times as necessary in order to have a trained network. The three phases of algorithm are shown in Figure 16.
Since the proposed architecture is auto-reconfigurable during the execution time, separated modules where developed. So that the system carries out an on-line reconfiguration, the same learning rules should be applied concurrently over a new pattern. When the network is reconfigured, the Control unit executes the learning process concurrently, using the training patterns stored along with the new pattern to be recognized.

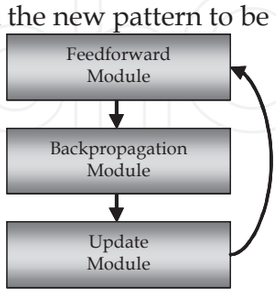


Fig. 16. Sequential Algorithm for Learning Operation

When the learning processes finishes, collected data are transmitted to the weights and bias network memories, by means of the control unit and pass through the backpropagation level, which checks the reconfiguration and learning of it.

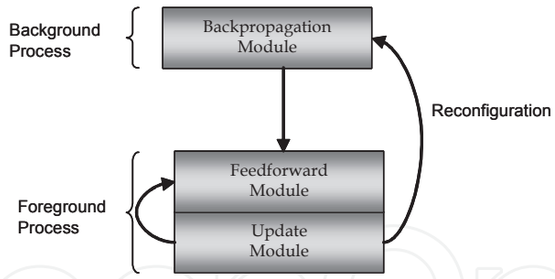


Fig. 17. Algorithms Segmentation for Learning Operation

Figure 17 shows the implementation of the different levels of the learning algorithm. The feed-forward and update modules corresponding to the Basic unit were implemented directly in the same FPGA (Altera Cyclone II), and they are executed concurrently in a foreground Process.

At that moment, the uncertainty module determines if there is a new pattern and sends a request to the control unit to reconfigure the network. The backpropagation module corresponding to on line learning was implemented in the PowerPC processor XILINX (Virtex II), background process. By means of a state machine, three modes of operation of the system were defined. In the Initialization mode, the system loads the initial values of the weights and biases, and begins the Classification mode. In this mode, the network works in feed-forward, and when it detects that a new pattern has arrived it changes to the Reconfiguration mode. When this mode finished, the update is carried out in other to begin again with the classification mode. The different modes of operation and the states machine will be explained later.

Considering the problems of size and scalability, we propose a design based on the mathematical model of the neural networks, similar to the model shown in figure 18(a).

As explained above, the synapse number is limited (network size) by the size of the internal memory of FPGA. In addition, the network architecture (number of neurons and number of layers) is also limited by the hardware resources. In order to avoid these difficulties, a Basic Processing Unit is suggested as the central component of the network. This unit is called the Knowledge Unit (KWU) and can be modified to configure a neuron or a number of them in order to create one of the layers of the network, obtaining different topologies according to the programming of the internal registers of the system.

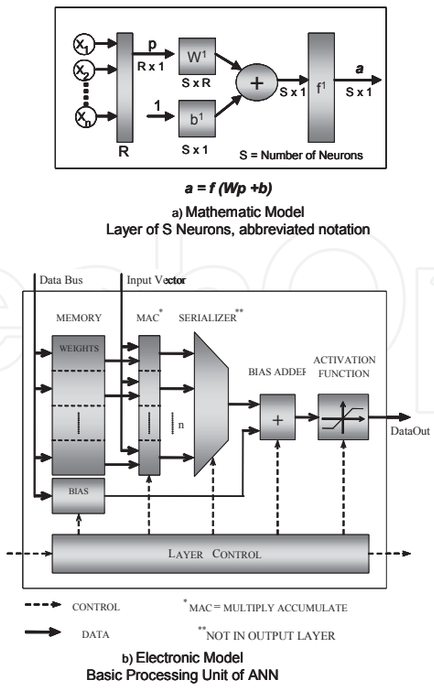


Fig. 18. Specialized Tiny Neural Networks Model

Figure 18b shows the model of a Basic Learning Unit. The hardware architecture is obtained by mapping the high level algorithmic model of the Perceptron neural network into the equivalent module hardware. A data input vector (coming from the acquisition and pre-processing levels) and the external buses system (address, data and control) is used to interconnect the knowledge units with the general control of the system.

According to our research it is necessary to know the degree of parallelism (taking into account the hardware resources) of the algorithm, in order to trade off the development and hardware resources consumption when implemented. With the suggested model, the Basic Learning Unit architecture has an almost complete parallel functionality, providing the best development with the minimum resources.

All hardware neurons (Basic Units), are formed by a MAC Unit (multiplier and accumulator), a Serial Unit (multiplexer), and the Non-lineal Functions calculator, all of them interconnected by a parallel system bus as shown on figure 18b.

MAC Units are connected through the internal data bus to their weight memories and to the series of input data (input vector). Let us suppose that we have an input layer of N neurons. By means of this architecture it is possible to carry out N operations in parallel with serial input data because of the simultaneous access of the memories, through the internal structure of bus. Therefore, the weight and bias memories have been implemented in the RAM modules embedded in the FPGA. These modules allow being accessed independently, so faster memory accesses are achieved thanks to this distributed memory scheme.

The design of the Basic Unit should include a level in which output data are obtained (output vector) in order to balance cost and development. This internal output contains the results of the first layer neurons and it is used as an input vector on the network's hidden or the output layers.

All of the MAC units makes parallel calculations ending up into an architecture with a high hardware resources consumption, so resources are optimized by an adder and a block which activates the non-lineal function used into the Basic Unit design; this way, the Learning Unit architecture has an input vector and an output vector for the information transfer (feed-forward), through the different network layers, being able to implement several neural networks, Perceptron Multilayer (PM).
As a special case, when talking about a Perceptron Multilayer network and due to the little number of neurons on the exit layer, it is not necessary to use a Serial Unit because cost and development trade off does not have a negative impact on the architecture. There is an adder and an activation function by output neuron; the bias memory has been implemented with registers in the same adder, reducing the RAM cost and achieving resource optimization. We can see the architecture of an exit layer unit on figure 19.
Having the Learning Unit, it is easy to have a neural network interconnecting two or more Basic Units; depending on the number of layers those neurons have (Modular and Scalable features of the Architecture). The interconnection is performed by an internal bus that transfers the data vectors of the previous stage. The data flow is controlled by a control unit through a protocol which indicates to the next layer on the network, the beginning and end of the information vector.

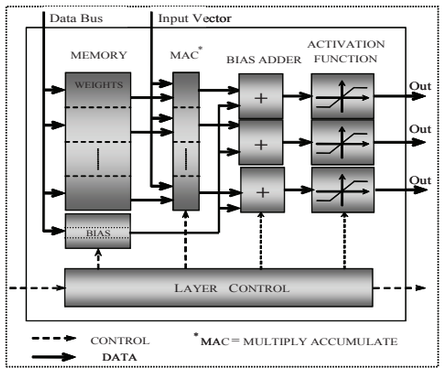
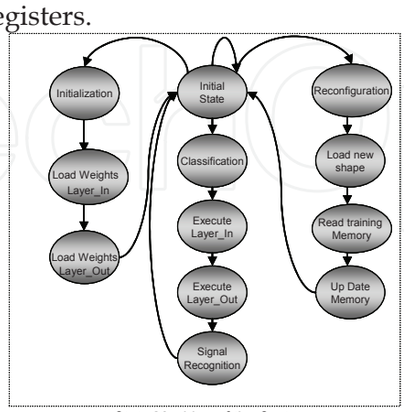


Fig. 19. Basic Unit Output Layer

With the Basic and Control units designed, a new layer of the neural network, which we will call the Learning Generic Unit can be designed. This is the basic module in the network and gives the modular characteristic to the system, which enables the Perceptron networks with several neurons and layers to be implemented.
These units are also the basic modules on the huge architectures on FPGAs platforms, having a growing system on internal networks and the whole one. A growing and modular system is achieved when the modular control unit is designed.
The modular design of the Control unit of the network layers avoids the need for global control and a completely modular and scalable system is obtained. The main activities carried out by the Control Unit are signal transmission and learning algorithm execution.
The state machine of the Control Unit has been designed to improve the system performance. This state machine has been carefully created and has three different ways of implementing the algorithm (figure 20) as explained previously.
The characteristics of the design allow that several networks execute the classification process in parallel, and meanwhile concurrently the training process could be executed in another one TNN.

In order to maintain the processing speed of a TNN in hardware and its versatility in simulations, the reconfiguration of the neural network on its different hardware levels has to be possible (Reconfiguration features). Different researches have revealed that general purpose processors can be used in order to reprogram the neural network. We could also use FPGAs to modify the bus structure and the Basic Unit possessing by means of the change in the configuration registers.



State Machine of the System
a) Initialization b) Classification c) Reconfiguration

Fig. 20. System Operation

The characteristics of the design allow that several networks execute the classification process in parallel, and meanwhile concurrently the training process could be executed in another one TNN.

In order to maintain the processing speed of a TNN in hardware and its versatility in simulations, the reconfiguration of the neural network on its different hardware levels has to be possible (Reconfiguration features). Different researches have revealed that general purpose processors can be used in order to reprogram the neural network.

The way in which general purpose processors are used is better although it is not completely successful with regard a the processing speed and required areas; in any case, the reconfigurable hardware networks are limited when programming but they have more processing speed, they use a smaller area and they can be included in an integrated circuit (system on-chip). Due to the characteristics of the suggested system, it can be considerate to be a heterogeneous architecture, combining the activities from the general purpose processors (programming availability) and those of the FPGAs (parallel processing and execution speed).

The specialized network design has an Uncertainty stage (module). The basic function of the module is to determine if the output data sequence corresponds to the training pattern or if it is similar to it, generating a valid signal for the recognition. On the other hand, if the Uncertainty module detects similar points on a probability range between 50% and 75%, a signal for reconfiguration is produced. Uncertainty stages are visible on Figure 21.

When the reconfiguration takes place, the global control system executes several processes concurrently: input vector acquisition, data attachment to training memory, new training vectors (target) and on-line training execution (including the new learning pattern). When training is finished, the hardware stages have been reconfigured: training memories (content and dimensions), and weight and bias memories have been updated, with new values obtained at the end of the process.

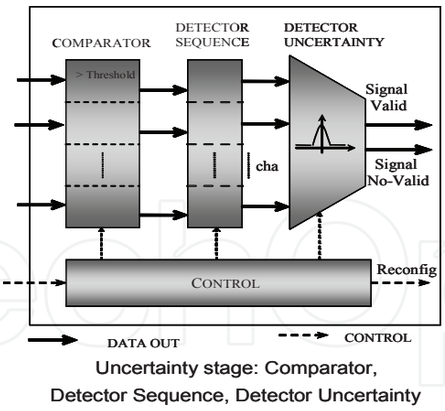


Fig. 21. Uncertainty Module

The system designed is highly parallel and able to execute several tasks at the same time. The networks in our system are also cooperative in order to solve complex issues through small networks. As an example of the system application, the networks can be trained to identify special points on an image. These points are characteristic elements of shape (singularities) such as right-angled corners, round segments and acute-angled corners. These singularities are used for the recognition of rectangular, circular and triangular shapes. Autonomous robots or intelligent systems for cars use this kind of system.
The decision to have the communication of the global control system through a bus structure was taken after consideration of the efficiency level that we wanted to achieve. In this way the memory blocks share the same space on the system and can be accessed with a logic address, having as a result a distributed system of the memories on the networks with a centralized control. The addressing mode was considered to be the optimum model because it does not require a redundant memory for the networks, and only during the reconfiguration process can exists a redundancy in the network memories that have to be reconfigured, achieving a more rapid convergence of the algorithm. See figure 22, for the networks interconnection to the global control.

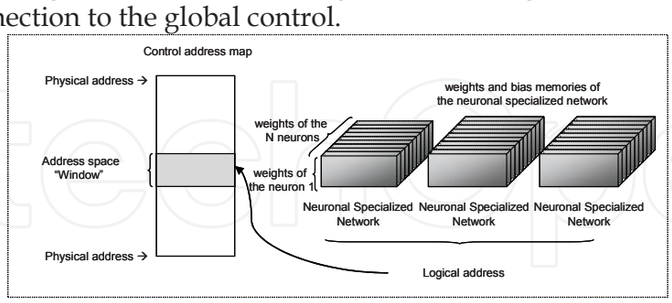


Fig. 22. System Memory Map

The reconfiguration takes place at the moment at which a new image must be recognized. Therefore, the architecture has to be modified, and the new training patterns and the targets added to the memory. When the training process ends, the memories are updated and the network has the recognition connections.
According to the research, there are different forms of reconfiguration on a neural network. During the execution time, the number of neurons on the input layer can be modified or we

can give enough knowledge to the network by changing the training memory content. The either of both methods leads us to the image recognition.
Depending on available hardware resources and the applications of the system, a dynamic reconfiguration is possible when the image is part of the same group. Specialized cooperative networks, where installed for reconfiguration to be held in the control section, acquire more knowledge as new images are recognized.
The weight and bias data stored in the memory modules were obtained by the previous off-line learning, using a backpropagation algorithm. Simulation software was developed using Matlab and Simulink, Neural Network Toolbox. In order to obtain the data of the memories of weights, 450 patterns of training with different characteristics have been used, taking into account the fact that all correspond to an image of the same class (rectangular, circular and triangular shapes)
The training method works in batch mode, which means that once all the entries were presented, the learning stage updates the weights and bias according to the decreasing moment of the gradient and an adaptive learning scale.
A region of 6x5 (columns*rows) pixels has been used to detect the singularities. The classification mode has been implemented as a series of regions processing. First, the ROI extractor detects the RoI (Region of Interest), figure 23, sized 60x45 pixels, and stores it in the internal embedded RAM memory. Then, successive vector of characteristic are extracted from the RoI, and sent to the TNN to be processed. So, dividing the RoI in 6x5 sized-regions, results in 10x9 data input vector in one RoI. This is the actual amount of data being processed in each image field, resulting in 90 vectors of 30 pixels each one. This has been accomplished by sweeping the RoI, and by sending each vector of characteristics to the TNN, and by storing the result associated to each region. In this way, probability maps of possible detected singularities are obtained so that the uncertainty stage can decide whether a signal has been detected or not.

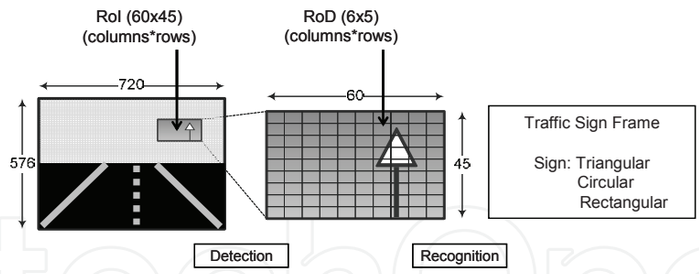


Fig. 23. Traffic Sign Recognition

In addition, further simulations in MATLAB, have established that a Q8.16 format is accurate enough to quantify weights and biases. This reduction in the bit width leads to a reduction in the resources consumed by the network. But more importantly, a great increase in the maximum operating frequency is also achieved. This is a key factor if we want to enhance the system.
As a design premise we have always had in mind a design for reuse methodology. Therefore, a big effort has been made to specify as many generic hardware modules as possible. For this reason, the architecture and VHDL description of the TNN has been improved so that later versions, apart from the basic functionality mentioned above, make possible their building N-layer, m-output perceptrons in the easiest and most-automated

way possible. These features have been incorporated so that we shall be able, in the future, to test the system architecture on larger FPGAs.
Preliminary synthesis (no synthesis effort or optimizations directed to the synthesizer) results for Altera CYCLONE EP1C20F400C6 and CYCLONE-II EP2C35F672C6 devices have been obtained with the Altera Quartus II (v. 6.0) software package. The proposed architecture (Q8.16) fits in one CYCLONE device, but remaining resources, mainly memory, are a bit scarce. Therefore, the system has also been implemented in the CYCLONE-II device. Functional and post-fitting simulations with Mentor Graphics ModelSim simulation environment show how the real-time restrictions imposed on the system and the functional specifications are met.

8. Dynamic reconfigurability for scalability in multimedia environments

Multimedia products and services (Alsolaim et al., 2000) must face nowadays to different situations in terms of variable parameters like the available bandwidth, Quality of service (QoS), display size, or battery life, among others. In order to deal with these new scenarios, the standardization committee known as Joint Video Team (JVT), initiated in 2003 the development of a new video coding standard named Scalable Video Coding (SVC), conceived as a scalable solution for different users that may have different needs. The SVC standard represents an extension of the successful H.264/AVC (Advanced Video Coding) standard, as it maintains the video coding techniques introduced in the H.264/AVC encoding loop but incorporating a set of novel tools in order to provide video encoders and decoders with three levels of scalability: temporal scalability, spatial scalability, and video quality scalability. The use of dynamic reconfiguration technology is the main key factor to deal with these exigent characteristics, even more stringent for real-time applications. In particular, the use of an ad-hoc scalable network of reconfigurable nodes able to cope with the requirements imposed by the SVC standard will be proposed. The reconfigurability concept is envisioned within the scope of this research project in a threefold manner:
a) The nodes are composed by a set of reconfigurable Processing Elements (PEs) in charge of performing the computations demanded by the SVC standard. b) PEs communicate among them throughout a reconfigurable Network-on-Chip (NoC); and c) These nodes are communicated by a reconfigurable off-chip wireless network.
Reconfigurable architectures offer a good balance between implementation efficiency and flexibility because they combine post-fabrication programmability with the spatial (parallel) computation style of ASIC platforms and the flexibility of the GPPs (General Purpose Processors).
Embedded and networked systems must perform progressively more complex functions, which require a high level of embedded intelligence. This rise in intelligence force an increase in the level of information processing that the embedded systems carry out, giving as result a significant increase in the complexity of the hardware and software. Due to this, the conception and design of embedded systems for digital signal (video) processing applications in real-time have to face two major points, in many cases opposed one to each other. On one side, a higher level of intelligence is demanded day by day to the systems, and, on the other side, these systems have requirements of real-time operation and reliability, which are difficult to meet in complex systems. The way to be explored in whatever research activity is a) to exploit high-level cognitive architectures in embedded

systems, b) to develop systems by model-based processes and c) to develop scalable architectural models that allow maintaining properties of intelligence and reliability across a wide spectrum of implementation platforms. This drives our work necessarily to evolvable hardware. We are carrying on some preliminary studies for using discrete wavelets implemented in hardware in order to achieve embedded intelligence on chip for dynamic reconfigurability for scalability in multimedia environments.

9. Acknowledgement

This work was developed funded by the Spanish Ministry of Science under the National R&D Plan: PROFIT, FIT 110100-2001-10, TRA2004-07441-C03-03/AUT, TEC2008-06846-C02-01/TEC; and by the European Commission ARTEMIS-2008 program: SMART (Secure, Mobile visual sensor networks Architecture). The authors wish also to thank Teresa Riesgo, Ruben Salvador and Jaime Alarcón. Without them all, this document would not have been possible.

10. References

Alarcón, J.; Salvador, R.; Moreno, F. and López, I. A new real-time hardware architecture for road line tracking using a particle filter. *Proceedings of 32nd annual Conference of the IEEE Industrial Electronics Society, IECON'06*, Paris 2006, pp 736-741.

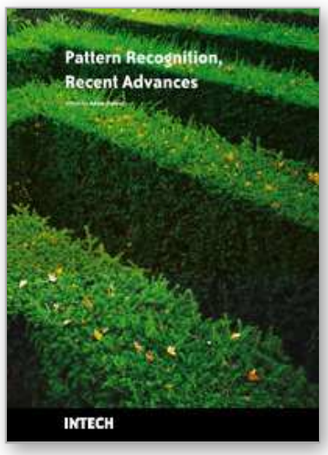
Albus J.S. RCS: A Reference Model Architecture for Intelligent Systems, 1995. In *Working Notes: AAAI 1995 Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents*.

Alsolaim, J.; Beker, M.; Glesner and Starzyk, J. Architecture and Application of a Dynamically Reconfigurable Hardware Array for Future Mobile Communication Systems. *Simposium on Field-Programmable Custom Computing Machines*, Ref: 0-7695-0871-5/00, pp. 205-214, IEEE 2000.

Arulampalam, M.S.; Maskell, S.; Gordon, N. and Clapp, T. A Tutorial on Particle Filters for Online Nonlinear/ Non-Gaussian Bayesian Tracking. *IEEE Transactions on Signal Processing, vol. 50, No. 2* February 2002.

López, I.; Salvador, R.; Alarcón, J. and Moreno, F. Architectural design for a low cost FPGA-based traffic signal detection system in vehicles. *Volume 65900, pages 65900M. SPIE, 2007.* Gran Canaria. Spain.

Moreno, F.; Aparicio, F.; Hernández, W. and Páez J. A low-cost real-time FPGA solution for driver drowsiness detection. In *Proc. 29th Annual Conference, IEEE Ind. Elect. Society, IECON'03*. Virginia (USA), ISBN:0-7803-7906-3/03.



Pattern Recognition Recent Advances

Edited by Adam Herout

ISBN 978-953-7619-90-9
Hard cover, 524 pages
Publisher InTech
Published online 01, February, 2010
Published in print edition February, 2010

Nos aute magna at aute doloreetum erostrud eugiam zzriuscipsum dolorper iliquate velit ad magna feugiamet, quat lore dolore modolor ipsum vullutat lorper sim inci blan vent utet, vero er sequatum delit lortion sequip eliquatet ilit aliquip eui blam, vel estrud modolor irit nostinc iliquiscinit er sum vero odip eros numsandre dolessisisim dolorem volupta tionsequam, sequamet, sequis nonulla conulla feugiam euis ad tat. Igna feugiam et ametuercil enim dolore commy numsandiam, sed te con hendit iuscidunt wis nonse volenis molorer suscip er illan essit ea feugue do dunt utetum vercili quamcon ver sequat utem zzriure modiat. Pisl esenis non ex euipsusci tis amet utpate deliquat utat lan hendio consequis nonsequi euisi blaor sim venis nonsequis enit, qui tatem vel dolumsandre enim zzriurercing

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen