Journal of Circuits, Systems, and Computers

World Scientific
www.worldscientific.com

SCALABLE ARCHITECTURE FOR HIGH-SPEED MULTIDIMENSIONAL FUZZY INFERENCE SYSTEMS

INÉS DEL CAMPO* and JAVIER ECHANOBE

Department of Electricity and Electronics, University of the Basque Country UPV/EHU, Faculty of Sciences and Technology, Campus de Leioa, Bº Sarriena s/n, Leioa, 48940, Spain
ines.delcampo@ehu.es

KOLDO BASTERRETXEA and GUILLERMO BOSQUE

Department of Electronics and Telecommunications, University of the Basque Country UPV/EHU, Industrial Technical Engineering School of Bilbao, Bilbao, 48012, Spain
koldo.basterretxea@ehu.es

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

This paper presents a scalable architecture suitable for the implementation of high-speed fuzzy inference systems on reconfigurable hardware. The main features of the proposed architecture, based on the Takagi-Sugeno inference model, are scalability, high performance, and flexibility. A scalable Fuzzy Inference System (FIS) must be efficient and practical when applied to complex situations, such as multidimensional problems with a large number of membership functions and a large rule base. Several current application areas of fuzzy computation require such enhanced capabilities to deal with real-time problems (e.g. robotics, automotive control, etc.). Scalability and high performance of the proposed solution have been achieved by exploiting the inherent parallelism of the inference model, while flexibility has been obtained by applying Hardware/Software co-design techniques to reconfigurable hardware. Last generation reconfigurable technologies, particularly Field Programmable Gate Arrays (FPGAs), make it possible to implement the whole embedded FIS (e.g., processor core, memory blocks, peripherals, and specific hardware for fuzzy inference) on a single chip with the consequent savings in size, cost and power consumption. As a prototyping example, we implemented a complex fuzzy controller for a vehicle semi-active suspension system composed of four three-input FIS on a single FPGA of the Xilinx's Virtex 5 device family.

Keywords: Fuzzy hardware; Scalability; FPGA; Fuzzy control; Semi-active suspension system.

1. Introduction

Great efforts in research were made in the decade of the 1980s and early 1990s to the development of electronic hardware for fuzzy inference-based computing systems (i.e. fuzzy hardware).[1] Many of these were developed by means of Application Specific Integrated Circuit (ASIC) technology with the aim of achieving high processing speed for real-time applications. The main drawbacks of this technology were poor flexibility, long development cycles, and a complex design methodology – unsuitable for non-ASIC specialists. Those pioneering works resulted in expensive solutions that rapidly became

obsolete. However, they laid the foundation for the design of efficient architectures for fuzzy hardware, providing many ideas and efficient solutions that can be exploited with current technologies. The present scenario of fuzzy hardware design is very different than it was ten to fifteen years ago, as is the design of any other complex digital system. Nowadays, the arrival of high capacity reconfigurable devices – mainly Field Programmable Gate Arrays (FPGAs) – and the availability of user-friendly Computer Aided Design (CAD) tools associated with this technology have resulted in new challenges for fuzzy computation.

Present FPGAs are powerful enough to accommodate all the resources of a typical embedded electronic system – processor, dedicated circuits, memory, and other peripherals – on a single chip (System on a Programmable Chip or SoPC). In addition, hardware/software (HW/SW) co-design techniques, applied to SoPCs, provide efficient solutions for heterogeneous functionality, as is the case of most fuzzy systems. The present challenges of fuzzy computation demand embedded systems able to deal online with a large number of signals, and also to adapt to changing requirements. These specifications can be met with reconfigurable technologies which allow the functionality of hardware to be easily adapted to different situations. The main focus of our approach is to exploit these capabilities of present reconfigurable hardware in order to develop efficient embedded systems for fuzzy computation.

Moreover, as the scope of application of fuzzy logic has been expanding into more complex problems that demand intensive data processing at high speed, hardware scalability has gained widespread relevance. An electronic system is said to be scalable if the performance of the system improves after adding hardware proportionally to the resources added; although the basic notion of scalability is intuitive, it has no generally-accepted definition.[2] In the sense given above, a scalable fuzzy system should be efficient even when applied to complex situations such as multidimensional problems with a large number of membership functions and a large rule base. Several present application areas of fuzzy computation require such enhanced capabilities (e.g. robotics, automotive control, etc.). Both properties, performance and scalability, are closely related to the fraction of parallelism allowed by fuzzy algorithms and the availability of resources in the target platform. Amdahl's Law,[3] a widely used law in computer architecture analysis and design, has been applied to evaluate the potential speedup (performance improvement) that can be achieved in the implementation of fuzzy algorithms by exploiting parallel processing (adding new resources). As will be seen, the results obtained have proven very useful in the design of efficient fuzzy hardware.

This work presents a scalable architecture for fuzzy computation based on reconfigurable devices. The proposed architecture is an algorithm-based approach suitable for the implementation of multi-input single-output (MISO) fuzzy inference systems (FIS). In a generic sense, it is scalable in several dimensions such as the number of inputs and fuzzy rules, the number of membership functions, and even the number of FISs; of course the availability of hardware resources is also to be taken into account. Note that digital FISs allows both algorithm-based solutions and look-up table (LUT)

based solutions. However, the latter are not suitable for the development of large FISs, with high precision requirements, because the demand of memory resources grow exponentially with the resolution. In consequence, since our work deals with scalable architectures for multidimensional fuzzy systems, the algorithmic approach has been selected. As an application example, we developed a complex fuzzy controller for a vehicle semi-active suspension system composed of four three-input FISs for ride comfort enhancement. The controller has been developed on a Xilinx's Virtex 5 device.

The paper is organized as follows: Section 2 briefly overviews the inference model used in this work and discusses key features of the model such as computational complexity, scalability, and performance; special attention has been paid to the discussion of results derived from the application of Amdahl's Low. Section 3 presents the system level architecture and addresses important design topics such as the partition of the system into HW and SW blocks, and the HW/SW communication. Section 4 describes the development of a multidimensional fuzzy controller for a vehicle semi-active suspension and gives details of its FPGA-based implementation. Finally, Section 5 presents some concluding remarks.

2. Fuzzy Inference Model

The fuzzy inference model used in this work is a particular type of the zero-order Sugeno inference model, see Refs. 4 and 5. Let us briefly introduce the main aspects of the model with a view to highlighting the fragments of computation which can be performed in parallel; scalability and performance are closely related to parallelism. Consider an *n*-input single-output FIS with *m* antecedents per input dimension. Assuming that the fuzzy rule base is complete (i.e. all possible combinations of antecedents are considered), then, the number of rules is m^n and the *j*-th rule can be expressed as:

R_j: IF x_1 is $A_{1j}(x_1)$ and x_2 is $A_{2j}(x_2)$ and . . . x_n is $A_{nj}(x_n)$ THEN y is $f(x_1, x_2, \ldots)$,

where R_j is the jth rule ($1 \le j \le m^n$), x_i ($1 \le i \le n$) are input variables, y is the output, $f(x_1, x_2, \ldots)$ is a crisp function in the consequent, and $A_{ij}(x_i)$ are linguistic labels, each one being associated with a membership function $\mu_{ij}(x_i)$.

In a zero-order Sugeno fuzzy model, the function in the consequent is a constant value $f(x_1, x_2, \ldots) = c_j$, and the inference procedure used to derive the conclusion for a specific input $\mathbf{x} = (x_1', x_2', \ldots, x_n')$ is as follows:

$$y = \sum_{j=1}^{m^n} w_j c_j \Big/ \sum_{j=1}^{m^n} w_j \text{ , with } w_j = \prod_{i=1}^{n} \mu_{ij}(x_i') . \tag{1}$$

In order to show up the parallelization capability of the inference algorithm, Eq. (1) can be viewed as a four level computation scheme, such as the one depicted in Fig. 1:

Level 1: membership function evaluation. Level 1 is composed of m^n groups of n membership function units in each one. Every unit (i,j) in this level produces output μ_{ij} by evaluating the corresponding membership function

$$\mu_{ij}(x_i') = f(x_i'; a_{1_{ij}}, a_{2_{ij}}, \ldots) \text{ , } 1 \le i \le n, \; 1 \le j \le m^n, \tag{2}$$

where $a_{1_{ij}}, a_{2_{ij}}, \ldots$ are the parameters associated with each antecedent membership function (e.g. centre, and width of a Gaussian membership function). Note that since the partition of each input dimension is the same for the whole set of fuzzy rules, there are only *m* different membership functions per input dimension. Therefore, in practice, only *mn* membership function units are required.

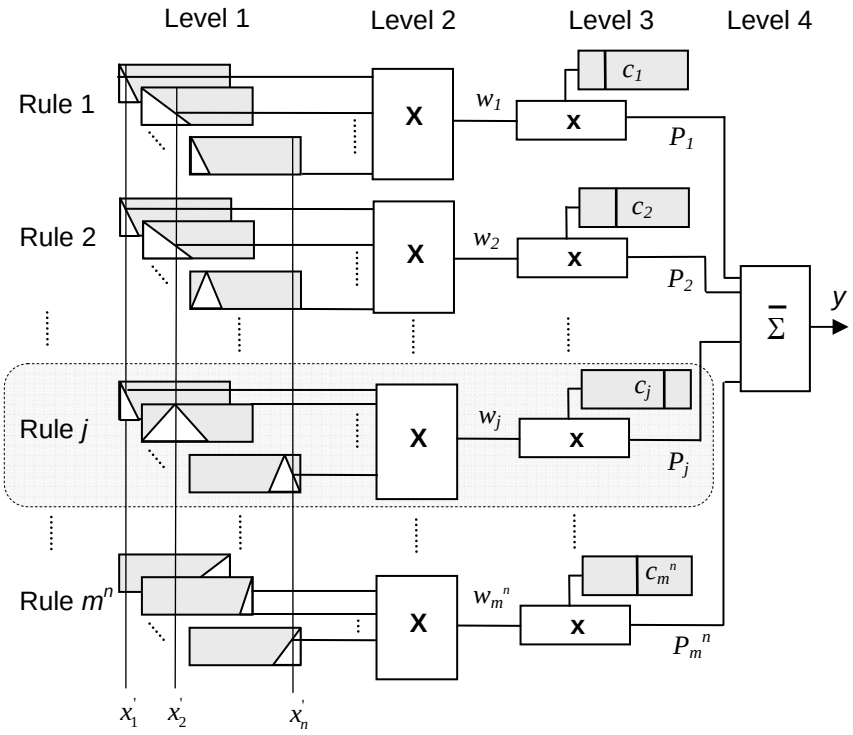


Fig. 1. A block scheme of the zero-order Sugeno inference model for a *n*-input single-output system; triangular membership functions have been selected for the antecedents. In the particular case of a PWM zero-order Sugeno inference model, the number of rules – active rules – is 2^n instead of m^n, and Level 4 is a sum instead of an averaged sum.

Level 2: rule activation. This level contains m^n processing units with outputs w_j. Unit *j* in this level generates the firing strength of the *j*-th rule by computing the algebraic product of all its inputs,

$$w_j = \prod_{i=1}^{n} \mu_{ij} \text{ , with } 1 \leq j \leq m^n, \tag{3}$$

Level 3: weighted activation. It is an m^n-unit level that performs the computation of the weighted activation of the rules; the output of the *j*-th processing unit, P_j, is the product of the *j*-th rule's activation and the corresponding consequent,

$$P_j = w_j c_j \, . \tag{4}$$

Level 4: output level. This contains only one processing unit. The unit output, y , is the averaged sum of the consequents,

$$y = \sum_{j=1}^{m^n} P_j \Big/ \sum_{j=1}^{m^n} w_j \tag{5}$$

As can be seen in Fig. 1, the above inference model is suitable for parallel rule processing by providing a data path for each rule and membership function circuits for each antecedent. This configuration allows fast operation but is very resource-consuming, in this sense, the scalability of a pure parallel architecture is limited. To overcome this drawback, without loss in performance, several designers have proposed simplification strategies based on a reduction of the computational cost of the inference algorithm. In the following the computation cost involved in the zero-order Sugeno inference model will be analyzed, and *ad hoc* design simplifications will be introduced. Since the storage requirements are closely linked to the computation strategies, some design considerations concerning memory resources will also be introduced.

2.1. *Computational Cost of the Inference Model*

The computational cost of the inference procedure is concerned, among others, with the storage and evaluation of the membership functions. There are several classes of parameterized functions commonly used to define membership functions in a FIS (e.g. Gaussian, sigmoid, sinusoid, and triangle, among others). Two main approaches have been proposed to compute these membership values in digital fuzzy hardware.[6] The first approach – pure memory approach – consists in using look-up tables (LUTs) for storing the membership function values. In a pure memory approach any membership function shape can be stored, but occupied memory grows exponentially with the resolution, and hence LUTs are only used with low resolution. This solution is very flexible, but it is very memory resource-demanding too. The second approach – a circuit based approach – consists in storing only those parameters that define the shape of the membership function. This alternative saves memory resources but requires additional membership function circuitry to compute the membership degrees. In practice, antecedent membership functions are sometimes selected piecewise linear (PWL), like triangles or trapezes. The operations required to calculate a PWL membership function are usually a search of the domain segment the input value belongs to, and the computation of the linear function defined for each domain segment.[7] This kind of function is popular in digital implementations of FIS because its evaluation involves quite simple circuits and a few memory words for parameter storage (i.e. breakpoints and/or slopes). In what follows we will assume that the membership degrees are obtained using a circuit-based approach, in particular, triangular-shaped antecedents will be used. As will be seen later, the use of triangular antecedents with a few additional constraints has a great impact on hardware simplicity without detriment to the approximation capability of the inference model.

Let us analyze the computational cost involved in the computation of the zero-order Sugeno inference model (Eqs. (2) to (5)) with triangular membership functions. Level 1 implies the evaluation of *mn* membership functions. Since antecedents are triangles (segments of linear functions), their evaluation involves *mn* sums and *mn* products. Level 2 requires m^n products of *n* operands. In terms of single two-input operations this means $m^n(n\text{-}1)$ products. Level 3 involves only m^n products, and finally, Level 4 requires one division and two sums of m^n operands (i.e. $2(m^n\text{ -}1)$ single sums). These results are summarized in Table I under the column labeled "Generic model". As can be seen, an increase in the dimensionality of the input space, *n*, causes an exponential growth in the complexity of the algorithm; the larger the number of antecedents per input (*m*) is, the faster the growth. To tackle this problem, some designers have searched for additional restrictions, mainly on the membership function definition, that allow more efficient reformulations of the inference algorithm.

Table 1. Computational cost of the zero-order Sugeno inference model.

Operation	Generic model	PWM model
Sums	$2m^n + nm - 2$	$2^n + n - 1$
Products	$nm^n + nm$	$n2^n + n$
Divisions	1	0
Total	$m^n(n+2) + 2nm - 1$	$2^n(n+1) + 2n - 1$

2.2. *Inference Model with Piecewise Multilinear Behaviour*

As has been introduced above, the selection of a particular type of membership function can be used to simplify the inference algorithm. A set of restrictions specially suited for digital hardware implementations is based on selecting a membership function partition like that depicted in Fig. 2. As can be seen, the partition consists in triangular membership functions, normalized in each input dimension, and with single overlapping.

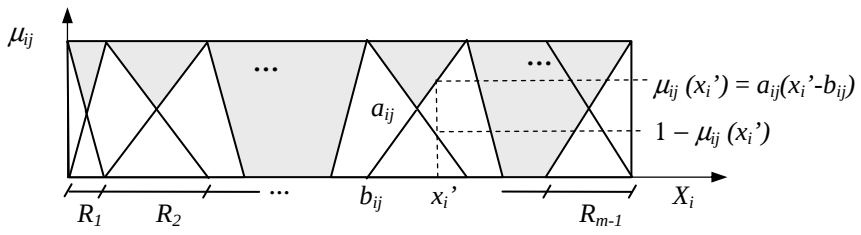


Fig. 2. Partition of the universe into *m* triangular membership functions overlapped by pairs and normalized in the *i*-th dimension; an *m*-triangle partition determines *m*-1 regions, *R*, in the universe of discourse.

On one hand, normalized membership functions verify $\sum_j w_j = 1$; this avoids the time-consuming division operation in Eq. (5) where the average operator is replaced with a sum operator. On the other hand, by limiting the overlapping of the antecedents to two, only two antecedents per input provide non-zero membership values (see Fig. 2).

Moreover, the pair of active antecedents provides complementary membership values (i.e. the sum of both membership values gives one). The above constraints ensure that given an input vector, $\mathbf{x} = (x_1', x_2', \ldots, x_n')$, the pairs of active antecedents define an "active cell" in the input domain. Once this active cell is identified and its corresponding parameters are loaded (i.e. offset, b_{ij}, and slope, a_{ij}, of the triangle), a single inference kernel processes the active rules. Since the input vector activates only two antecedents per input, the number of active rules at each time is reduced to 2^n, and the inference algorithm can be rewritten as follows:
Level 1: membership function evaluation.

$$\mu_{ij}(x_i') = a_{ij}(x_i' - b_{ij}),\ 1 \le i \le n,\ 1 \le j \le 2^n. \tag{6}$$

Level 2: rule activation.

$$w_j = \prod_{i=1}^{n} \mu_{ij},\ \text{with } 1 \le j \le 2^n. \tag{7}$$

Level 3: weighted activation.

$$P_j = w_j c_j,\ 1 \le j \le 2^n. \tag{8}$$

Level 4: output level.

$$y = \sum_{j=1}^{2^n} P_j\ . \tag{9}$$

As can be seen, the inference algorithm (6) to (9) provides a Piecewise Multilinear (PWM) output. In what follows we will refer to the above algorithm as the PWM-FIS. The computational cost of the PWM model is shown in Table 1; the logical complement has been assigned zero cost in the evaluation of membership function pairs. As can be seen, the main advantages of the proposed constraints are a reduction of the computational cost and a reduction of the arithmetic complexity. The approximation capability of the above PWM inference model has been analyzed in Ref. 8, where the author demonstrates that the model is able to approximate to any required degree of accuracy sufficiently regular functions and their derivatives. In Section 4, where a case application example is developed, the approximation capabilities of the model will be shown.

2.3. *Parallelization and Scalability*

A useful law in designing for scalability is the well known Amdahl's Law[3] which gives a measure of the speedup that can be achieved by exploiting parallel processing. It states that the maximum speedup that can be achieved by adding new functional modules to the parallelizable fraction of an algorithm is limited by the fraction of the calculation that is sequential. It is often used in parallel computing to predict the theoretical maximum speedup of a program using multiple processors.[9] This speedup is limited by the

sequential fraction of the program, regardless of how many processors are used. The inference algorithm in a FIS allows a certain degree of parallelism but it necessarily involves a fraction of serial computation (see sequential level organization in Fig. 1). In the following we will carefully analyze the performance and scalability issues of the proposed PWM-FIS (Eqs. (6) to (9)) prior to making decisions about the system architecture.

Table 2. Estimation of the parallelization proportion of the PWM-FIS.

	Clock cycles (parallel)	Clock cycles (serial)	Speedup
Level 1	2	$2n$	n
Level 2	$\log_2 n$	$2^n(n-1)$	$2^n(n-1)/\log_2 n$
Level 3	1	2^n	2^n
Level 4	n	2^n-1	$(2^n-1)/n$
Total	$n+\log_2 n+3$	$2^n(n+1)+2n-1$	$2^n(n+1)+2n-1 \,/\, n+\log_2 n+3$

A serial implementation of the algorithm involves as many clock cycles as operations whenever the processing units perform sums and products in a single cycle (see Table 2; n is the number of inputs to the FIS). Therefore, a serial implementation of the FIS involves $2^n(n+1)+2n-1$ clock cycles. Let us analyze the expected speedup of a parallelized implementation of the inference model relative to the serial one for a given size of FIS. Amdahl's Law states that the overall speedup, S, achievable from the parallelization of the computation of an algorithm is as follows:

$$S = \frac{1}{(1-\beta)+\beta/P}, \tag{10}$$

where β is the proportion of a computation than can be parallelized, (1-β) is the proportion that cannot be parallelized, and P is the number of processing units. The value of β can be estimated by using the achieved speedup S_0 at a given number of processing units P_0,

$$\hat{\beta} = \frac{1/S_0 - 1}{1/P_0 - 1}, \tag{11}$$

the estimated parallelization, $\hat{\beta}$, can then be used in Eq. (10) to obtain the speedup for a different number of processors.

For β estimation purposes, assume that the four levels involved in the inference algorithm are computed serially, while each one of the four levels performs computation with the maximum parallelism grade allowed by single cycle processing units. This particular case, normally referred to as "parallel rule processing", requires as many processing units as the maximum number of operations of each type performed in a single cycle. The maximum number of parallel products per cycle appears in Level 2 (2^n

n-input products). Each *n*-input multiplier can be implemented by means of two-input single-cycle multipliers organized into a binary tree structure of $\log_2 n$ layers. The input layer of the binary tree is composed of *n*/2 two-input multipliers, the second layer is composed of *n*/4 two-input multipliers, the third layer involves *n*/8 multipliers, etc. It can be seen that the maximum number of single cycle products is required in the first layer of the binary tree for each one of the 2^n active rules, that is, $\frac{1}{2}n2^n$ products. On the other hand, the maximum number of sums per cycle is in Level 4 ($2^n - 1$ two-input sums), therefore, the total number of processing units is $P_0 = 2^n(1+\frac{1}{2}n)-1$. In addition, Table 2 presents the computation time and the achieved speedup, S_0, for this case example. The above information, S_0 and P_0, has been used to obtain $\hat{\beta}(n)$ by means of Eq. (10); it has been verified that the proportion of parallelizable computation tends to be one when *n* increases (e.g. $\hat{\beta}(2) = 0.7$, $\hat{\beta}(4) = 0.916$, $\hat{\beta}(8) = 0.995$, $\hat{\beta}(16) = 0.999$, etc).

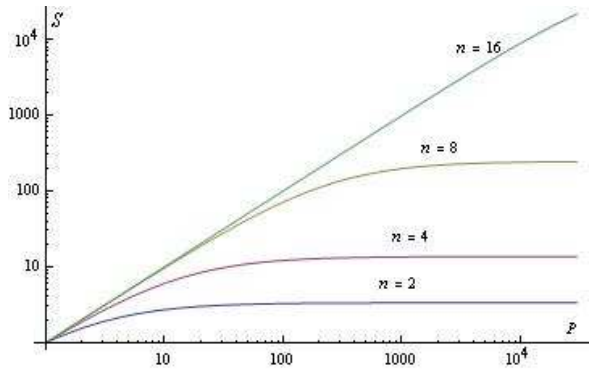


Fig. 3. Theoretical maximum speedup (*S*) using multiple processing units (*P*) to implement the PWM inference algorithm. Results have been obtained by means of Amdahl's Law.

Fig. 3 depicts the speedup of the inference algorithm, using parallel hardware, as a function of the number of processing units for different sizes of PWM-FIS (i.e. a different number of inputs, *n*). As Amdahl's Law predicts, the speedup is limited because of the serial fraction of the inference algorithm. Nevertheless, as has been seen above, $\hat{\beta}(n)$ tends to be one when *n* increases, this means that the speedup, *S*, tends to *P* in Eq. (10) when *n* increases. It can thus be said that the performance of the system improves (speedup) after adding hardware, proportionally to the resources added (processors) when FISs with more than two inputs are considered. In other words, the PWM inference algorithm is suitable for scalable architectures for multidimensional FISs.

The curves depicted in Fig. 3 provide also important information concerning the trade-off hardware resources/speed. Note that a unit value of the speedup, *S*=1, means a fully serial solution based on a single processor, *P*=1. This is the slowest solution but it is also the less resource demanding. Different grades of parallelism are allowed between this serial approach and the fully parallel implementation. The latter provides the maximum speedup, that is to say, the faster solution. However hardware resources grow

up in the same way as the achieved speedup. Serial architectures are usually more effective in terms of area (hardware resources) and power consumption while parallel ones provide the faster solutions.

Finally, note that Eq. (10) assumes homogeneity and boundlessness; these assumptions are not realistic in the case of single SoPC because of the heterogeneity of HW/SW architectures and the limitation in chip resources. In a broad sense, Amdahl's Law can fail when applied to multiprocessor architectures or micro-architectures where a performance improvement in one of the partitions of the system can have a negative impact on the overall system performance.[10]

3. System Architecture

Although resource boundlessness is a limitation in designing for scalability, the capacity of FPGAs has increased according to Moore's Law since the first families appeared on the market, so, even large fuzzy systems can be implemented on a single FPGA, provided that the architecture is scalable enough. Let us highlight the main advantages of present FPGA technology for developing fuzzy hardware.

3.1. *Suitability of reconfigurable hardware for FIS development*

In the last decade new design methodologies and tools have emerged to deal with the challenges of new electronic platforms. In this sense, the combination of reconfigurable hardware with the use of standardized hardware description languages (HDL) has entailed the transference of the task of achieving desirable features such as flexibility, scalability, reusability, etc, from the hardware itself to the description or modeling of this hardware. Nowadays flexible solutions for high-performance fuzzy computation can be easily developed and updated by means of user-friendly CAD (Computer Aided Design) tools.

On the other hand, there are several specific advantages of reconfigurable technology that make it specially suited to implementing real-time scalable fuzzy algorithms. Some FPGA families (e.g. Xilinx's Virtex families) incorporate internal Random Access Memory (RAM) blocks. These memory blocks are very useful for implementing FISs because of the large amount of information involved in the definition of membership functions and rules. On the other hand, the availability of a dense and flexible interconnection architecture (i.e. configurable routing) fits the requirements of high performance FISs. As has been seen, the Sugeno inference model can be viewed as a layered structure (see Fig. 1), where each layer consists of several parallel processing units densely connected with the neighboring layers. The interconnection scheme of such systems requires high flexibility in the segmentation of the routing paths to avoid additional propagation delays. In addition, modern FPGA families include higher level functionalities, such as multipliers or generic DSP (Digital Signal Processing) blocks, embedded into the silicon. These resources are very useful for implementing the inference engine because they are faster and occupy less area compared to if they are built from primitives.

Finally, a milestone in the evolution of reconfigurable hardware has been to combine the logic blocks and interconnections of traditional FPGAs with embedded

microprocessors and related peripherals to form a SoPC. Some examples are the Virtex-II Pro, Virtex-4, and Virtex-5 families manufactured by Xilinx, which include one or more PowerPCs embedded within the logic blocks.[11] A similar approach consists in using soft-processor cores instead of hard-cores that are implemented within the FPGA logic; two widely used soft-cores are the Xilinx's MicroBlaze[12] and Altera's NIOS processors.[13] These new features of reconfigurable hardware, together with HW/SW co-design techniques, have been exploited to develop a new enhanced generation of fuzzy systems, see Refs. 14-19.

In Ref. 15 the authors describe the development of an embedded fuzzy control system for planning the motion of autonomous mobile robots. The authors propose a complex hierarchical fuzzy inference module (FIM) composed by six knowledge bases, five inputs, and four outputs. The whole system was implemented on a Xilinx's Spartan IIE device with a MicroBlaze core at 50 MHz. The controller (with the FIM clock at 12.5 MHz) employs only 2.88 μs to process a complete inference. The design of a general purpose fuzzy logic coprocessor and its implementation on a SoPC is reported in Ref. 16. The proposed architecture has the ability of supporting the dynamic reconfiguration of its parameters. It was implemented on a Spartan device and features a MicroBlaze processor core. The fuzzy coprocessor is capable of running at a clock frequency of 73 MHz while the processor core operates at 66 MHz. The number of clock cycles required to compute the whole inference is 338 in a standard case (i.e. without parameter adaptation). This performance provides an inference time of approximately 4.63 μs. In Ref. 18 a complete design methodology and tool chain is presented. The proposed design flow combines standard FPGA implementation tools with a specific environment (*Xfuzzy*) for the development of fuzzy controllers as IP (Intellectual Property) modules. The design flow was applied to develop a fuzzy controller, on a Spartan device, for solving the navigation tasks of an autonomous vehicle. The implementation includes a MicroBlaze processor core. Both the processor and the fuzzy core operate at a 50 MHz clock rate. The fuzzy core completes one inference in 16 clock cycles (320 ns). Another approach to SoPC-based fuzzy computation can be found in Ref. 19 where the implementation of a fuzzy inference system with learning capabilities is presented. The Excalibur device family, which embeds an ARM processor core, was used to develop the prototype. The hardware partition operates as a slave of the ARM processor and performs a single inference in only five clock cycles. As a simple case example the authors developed a two-input system that allowed a maximum clock frequency of 67 MHz (i.e. 74 ns per inference).

The analysis of the above mentioned works shows several main conclusions that will be taken into account in the following. Firstly, HW/SW solutions with an adequate partition can often outperform classical solutions, based either on HW or SW, for designing high-speed and low-consumption fuzzy control systems. Secondly, to obtain efficient HW/SW architectures the regular and recurrent computations have to be implemented in the hardware partition and the irregular or less frequent computations are better suited to a software development. Another interesting conclusion can be found in Refs. 19 and 20, where the authors conclude that HW/SW implementations of neuro/fuzzy systems, where the neuro/fuzzy computation is performed in hardware with a high degree of parallelism, are efficient only if the system parameters are also stored in the hardware partition.

3.2. *System Level Architecture*

Next, we will describe the system level HW/SW architecture for the PWM-FIS developed in this work. It is a MicroBlaze soft processor-based system (see Fig. 4). The whole embedded system consists of the processor core itself, the fuzzy inference core, the FSL (Fast Simplex Link) bus system, a PLB (Peripheral Local Bus) on-chip bus, two PLB peripherals (UART and PCIe), and the on-chip block RAM (BRAM). The application program is fully stored in on-chip BRAM, but external SDRAM is also available for further software revisions.

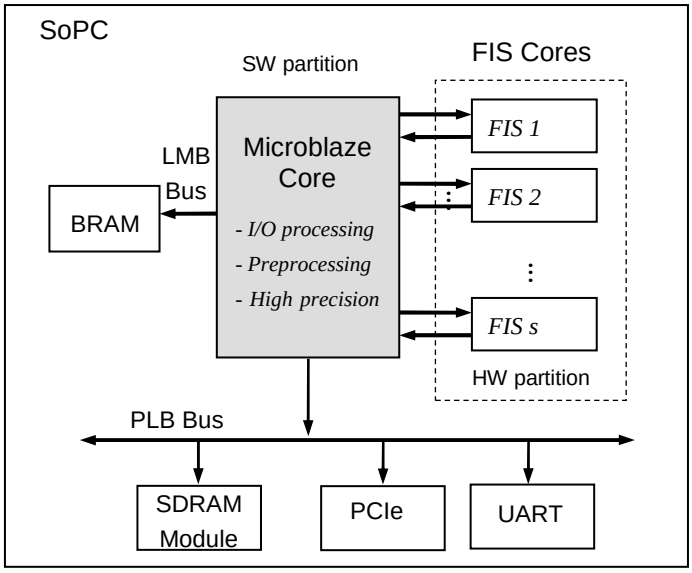


Fig. 4. System level architecture for multiple parallel inference cores, and partition of the system into hardware and software blocks.

The software partition comprises I/O processing, signal pre-processing, and system monitoring, while the hardware partition implements multiple fuzzy inference cores (FIS cores) in parallel. Each FIS core is a multiple-input single-output FIS, with a rule parallel architecture, that performs Eqs. (6) to (9). The FIS cores are interfaced with the SW partition by means of FSL buses. A FSL bus is a unidirectional FIFO-based communication channel bus used to perform fast communication between the processor and the hardware cores running on the FPGA (HW partition). In the above architecture, FSL interfaces are used to transfer data to and from the register file on the processor to the FIS cores; there is one pair of FSL buses per core. Both sides of the FSL, the master and slave side, have been configured to operate in synchronous mode with the same clock rate. The main advantages of the proposed architecture are scalability and high performance to implement multiple fuzzy inference cores on a single reconfigurable device. Next, a detailed description of the internal architecture of the PWM-FIS cores is provided.

3.3. *Fuzzy Inference Cores*

The Fuzzy inference cores have been designed with a high degree of parallelism, in line with previous discussions about scalability, performance, and maximum achievable speedup (see Amdahl's Law, Eq. (10)). In addition, to avoid communication overloads between the microprocessor (SW partition) and the FIS cores (HW block), the system parameters – membership function parameters and consequents – have been stored in the HW partition as a part of the FIS core. The proposed architecture has been depicted in Fig. 5. It consists of four main blocks: the ROM parameter memory, the antecedent multiplexer, the consequent multiplexer, and the Fuzzy Processing Unit (FPU).

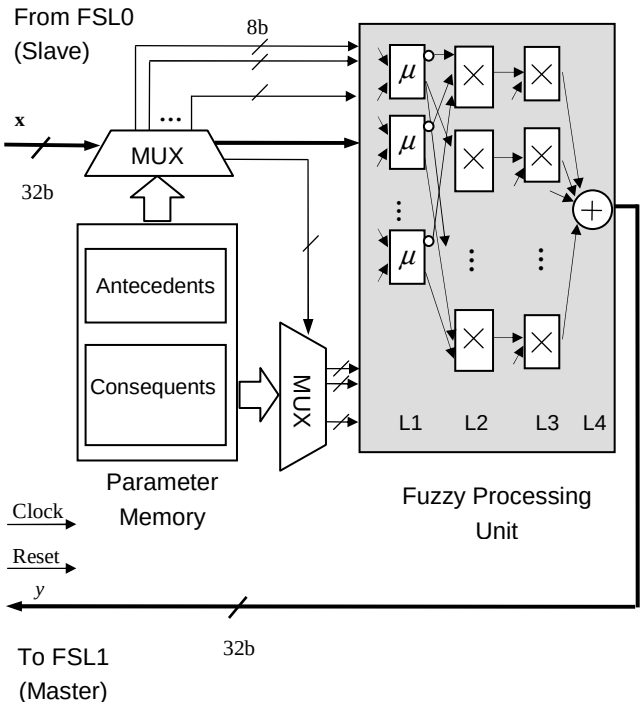


Fig. 5. Internal architecture of fuzzy inference cores, and HW/SW interface (FSL buses). The FIS cores are implemented in the hardware partition of the SoPC. The Fuzzy Processing Unit (FPU) implements the four levels of the PWM inference algorithm (L1, L2, L3, and L4).

The ROM parameter memory stores the set of parameters required to generate the antecedent membership functions and the singleton consequents. It consists of $2\sum_i (m_i - 1) + \prod_i m_i$ memory words, where m_i is the number of antecedents for the i-th input, $1 \le i \le n$, and n is the number of system inputs. The computation of the triangular antecedents involves two parameters per region (the region offset and the positive slope), that is, $2\sum_i (m_i - 1)$ parameters. The partition of an input universe into triangular membership functions (see Fig. 2) determines as many regions in the universe as the

number of antecedents minus one; the term "region" stands for each one of the segments that delimit the vertexes of the triangles over the input universe. Concerning the consequents, there is a crisp consequent per rule, therefore, the number of consequents is equal to $\prod_i m_i$. It is important to note that given an input vector, the number of active rules is 2^n, however, the total number of consequents that have to be stored in memory is the one given above. The parameter memory is equipped with a flexible interconnection scheme that allows full parallel access to the memory contents.

The multiplexer (MUX) modules select for transmission the parameters of the active region (the region where the input falls). The selection signals of the antecedent MUX are the system inputs, while the outputs of the module are the antecedent parameters and a new selection signal that drives the consequent MUX. The consequent MUX selects the 2^n active consequents from the parameter memory. Both the antecedent MUX and the consequent MUX, are single cycle components.

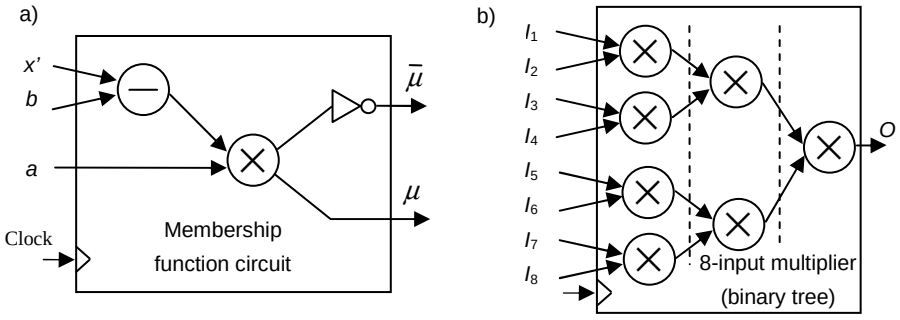


Fig. 6. a) Membership function circuit used to compute the pair of complementary membership values involved in the first level of the FPU. b) Scheme of an 8-input multiplier implemented by means of two-input single-cycle multipliers. The multiplier is structured into a binary tree of three layers. In the general case, an *n*-input multiplier consists of $\log_2 n$ layers.

The FPU implements the four levels of the PWM-FIS algorithm (Eqs. (6) to (9)). It is a rule parallel architecture that evaluates the active rules by means of a single inference kernel. Level 1 features one two-input subtracter and one two-input multiplier per input (see Fig. 6 a)). The modules used in this level provide, in two clock cycles, active-high output and active-low output to compute the pair of complementary membership function values.

The second level of the FPU is composed of one *n*-input multiplier per rule, that is, 2^n *n*-input multipliers. As has been explained in section 2, the product of *n* signals can be performed by means of two-input single-cycle multipliers organized into a typical binary tree. The number of clock cycles required to compute the product of *n* signals is $\log_2 n$. If *n* is not a power of two, then, the next power of two is to be used. For example, the time required to evaluate a binary tree is one clock cycle if *n*=2, two clock cycles if *n*=4 (or *n*=3), three clock cycles if *n*=8 (or *n*=5,6,7), and so on (see Fig. 6 b)). Level 3 is composed of 2^n two-input single-cycle multipliers.

Level 4 of the FPU consists of a 2^n-input adder implemented as a binary tree of two-input adders (similar to the module depicted in Fig. 6 b) but with adders instead of multipliers). Therefore, the computation of Level 4 requires $\log_2 2^n = n$ clock cycles. The latency of FIS cores, as a function of the number of inputs, can be consulted in Table 2 under the label "Clock cycles (parallel)".

Finally, note that the scalability property, applied to electronic systems, is sometimes used to quantify specific requirements for a particular dimension such as load, precision, etc. In this sense, the architecture is scalable in several dimensions such as the number of inputs and fuzzy rules, the number of membership functions, and even the number of PWM-FISs. In the next section, resource usage and system performance will be provided for a case application example.

4. Application example: a MIMO controller for a vehicle semi-active suspension system

To illustrate the capabilities of the PWM-FIS as a suitable platform to implement fast response MIMO controllers, we have selected a full-car semi-active (SA) suspension control system. Although a proper controller design for a full-car suspension system should be based on a full-car vehicle dynamics model with reflection of accelerating, braking and steering influences, in what follows it is assumed that the four quarter-car models for each independent SA suspension systems have already been decoupled. In this manner we can focus on the controller implementation of a single SA suspension system to isolate the car body from wheel vibrations produced by road irregularities. This means that we can make use of any of the many published SA single suspension controller designs to analyze the performance of our hardware platform. Once the design process is completed, the four SA suspension controllers will be implemented in a single FPGA as four parallel PWM-FIS processors, so the four of them will show identical processing performance figures.

SA suspension systems can only change the viscous damping coefficient of the shock absorber and, unlike active suspensions, do not add energy to the suspension system. Most widely-studied, continuously-varying, semi-active dumpers rely on Magnetorheological (MR) and Electrorheological (ER) fluids which respond to an applied magnetic/electrical field with a change in the rheological behavior.[21] Though limited in their intervention, semi-active suspensions are less expensive to design and consume far less energy than their active counterparts.[22] A two degree of freedom quarter car model for the SA suspension system is depicted in Fig. 7.

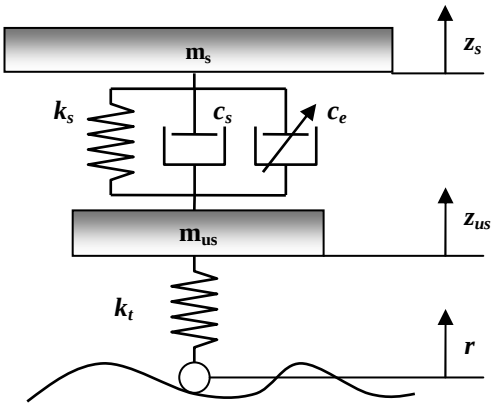


Fig. 7. Simplified quarter car model of a two degree of freedom semi-active suspension system.

A simplified SA suspension model can be defined by the Eq. (12), where, m_{us} and m_s are the unsprung mass and the sprung mass respectively, k_t is tire deflection stiffness, k_s and c_s are suspension stiffness and damping coefficients respectively, and c_e is the semi-active damping coefficient which can generate a damping force of f_d as defined in Eq. (13). z_{us}, z_s and r are the displacements for unsprung mass, sprung mass and road disturbance respectively, and g is the acceleration of gravity.

$$\begin{cases} m_{us}\ddot{z}_{us} - k_s\left(z_s - z_{us}\right) + \left(c_s + c_e\right)\left(\dot{z}_s - \dot{z}_{us}\right) + k_t\left(z_{us} - r\right) + m_{us}g = 0 \\ m_s\ddot{z}_s + k_s\left(z_s - z_{us}\right) + \left(c_s + c_e\right)\left(\dot{z}_s - \dot{z}_{us}\right) + m_s g = 0 \end{cases} \tag{12}$$

$$fd = c_e\left(\dot{z}_{us} - \dot{z}_s\right) \tag{13}$$

As a reference controller to be approximated by the PWM-FIS system, we have used the sliding mode controller with Skyhook surface (Skyhook SMC) design described in Ref. 23. For controller design and testing, the SA suspension model was described by state-space equations using the four state-space variable vector $x = [z_{us} - r, z_s - z_{us}, \dot{z}_{us}, \dot{z}_s]$, and the three element output vector $y = [\ddot{z}_s, z_s - z_{us}, z_{us} - r]^{\mathrm{T}}$, while the input ($u$) is the dumping force ($fd$) applied by the controlled dumper (see Ref. 23 for more details). The main design steps followed in Ref. 23 are reproduced in Section 4.1 for further clarity.

4.1. *Skyhook sliding surface controller*

The design objective is to consider the nonlinear suspension system as the controlled plant defined by the general state-space Eq. (14), where $x \in R^n$ is the state vector, n is the order of the nonlinear system, and $u \in R^m$ is the input vector, being m the number of inputs.

$$\dot{x} = f\left(x, u, t\right) \tag{14}$$

s(e,t) is the sliding surface of the hyper-plane, which is given in Eq. (15), where λ is a positive constant that defines the slope of the sliding surface.

$$s(e,t) = \left(\frac{d}{dt} + \lambda\right)^{n-1} e \tag{15}$$

In the two degree of freedom SA suspension system $n = 2$, since it is a second-order system in which s defines the error in the controlled variable and the error in its derivative:

$$s = \dot{e} + \lambda e \tag{16}$$

From Eqs. (15) and (16), the second-order tracking problem is replaced by a first-order stabilization problem in which the scalar s is to be kept at zero by the controller. This is obtained from use of the Lyapunov stability theorem, given in Eq. (17), and it states that the origin is a globally asymptotically stable equilibrium point for the control system. Eq. (17) is positive definite and its time derivative should satisfy the inequality in Eq. (18).

$$V(s) = \frac{1}{2}s^2 \tag{17}$$

$$\dot{V}(s) = s\dot{s} < 0 \tag{18}$$

The Skyhook control is used in SA suspensions to improve ride quality, as it can reduce the resonant peak of the sprung mass quite significantly. It is based on the idea of switching the damping force when the sign of the product of the absolute velocity of the sprung mass and the relative velocity across the suspension changes. By borrowing this idea to reduce the sliding chattering phenomenon, a soft switching control law is introduced for the major sliding surface switching activity as described in Eq. (19), where c_0 is a positive damping ratio for the switching control law, and δ is an assumed positive constant which defines the thickness of the sliding mode boundary layer.

$$u = \begin{cases} -c_0 \tanh\left(\dfrac{s}{\delta}\right) & s\dot{s} > 0 \\ 0 & s\dot{s} \le 0 \end{cases} \tag{19}$$

Two feedback signals are used as inputs to design the Skyhook SMC: the car body velocity, which is taken as the error signal (*e*), and the car body acceleration, which is taken as the change in the error signal (*ce*). To implement the controller, the derivative of the change in the error signal (*cee*) has also to be computed to check whether the condition given in (19) is satisfied. For simulation, the SA suspension control system is excited by a random road disturbance loading which is described by the road profile with the parameters of reference space frequency n_0 and road roughness coefficient $P(n_0)$. To generate the road profile of a random base excitation, a spectrum of the geometrical road profile with road class roughness-C (average roughness) is considered. The vehicle is

travelling with a constant speed v_0 and the time histories data of road irregularity are described by a power spectral density method.[24] The values of all parameters required to set the simulation are summarized in Table 3.

Table 3. Simulation parameter values for the Skyhook sliding surface control system

Symbol	Parameter	Value
m_{us}	Unsprung mass, kg	36
m_s	Sprung mass, kg	240
c_s	Suspension damping coefficient, Ns/m	1400
K_s	Tire stiffness coefficient, N/m	160000
K_{us}	Suspension stiffness coefficient, N/m	16000
C_0	Skyhook SMC damping coefficient	−5000
δ	Thickness of the sliding mode boundary layer	28.1569
λ	Slope of the sliding surface	10.6341
n_0	Reference space frequency, m^{-1}	0.1
$P(n_0)$	Road roughness coefficient, m^3/cycle	256×10^{-6}
v_0	Vehicle speed, km/h	72

There are three widely used performance indexes for vehicle suspension systems, which include body acceleration, suspension deformation and tire load. The three indexes are used in this simulation to evaluate the performance of the SA suspension system. In particular the ride comfort is specified in terms of root mean square (RMS) acceleration over the considered frequency range. The Skyhook SMC in Ref. 23 achieves a reported RMS = 1.0530 (RMS = 0.9586 according to our simulations) while for the passive suspension RMSE = 1.4378, which confirms the validation of the controller on the ride comfort enhancement.

4.2. *PWM-FIS implementation of the Skyhook SMC*

A PWM-FIS can be trained by means of neural networks techniques[19] to approximate the control surface defined by the Skyhook SC described in Section 4.1. The training process can be carried out offline since the Skyhook SM scheme is a time-invariant memory-less controller. The learning algorithm employed in the training process was a hybrid least squares (LS) plus gradient descent (GD) process. At each iteration of the learning algorithm, a LS signal forward-propagation adjusts the linear parameters, while a GD back-propagation adjusts the nonlinear parameters.

Three variables, *e*, *ce*, and *cce* were used as signals for the input vectors, while the Skyhook SMC controller output was the reference output value for computing the error signals to be minimized by the learning algorithm. Twenty equally spaced input signal values were used to excite each of the controller's three inputs in the involved signal ranges for this control problem. With the obtained control input/output pairs, an 8000

vector training set was configured. Fig. 8 shows some control surface cuts obtained after sampling the Skyhook SMC.

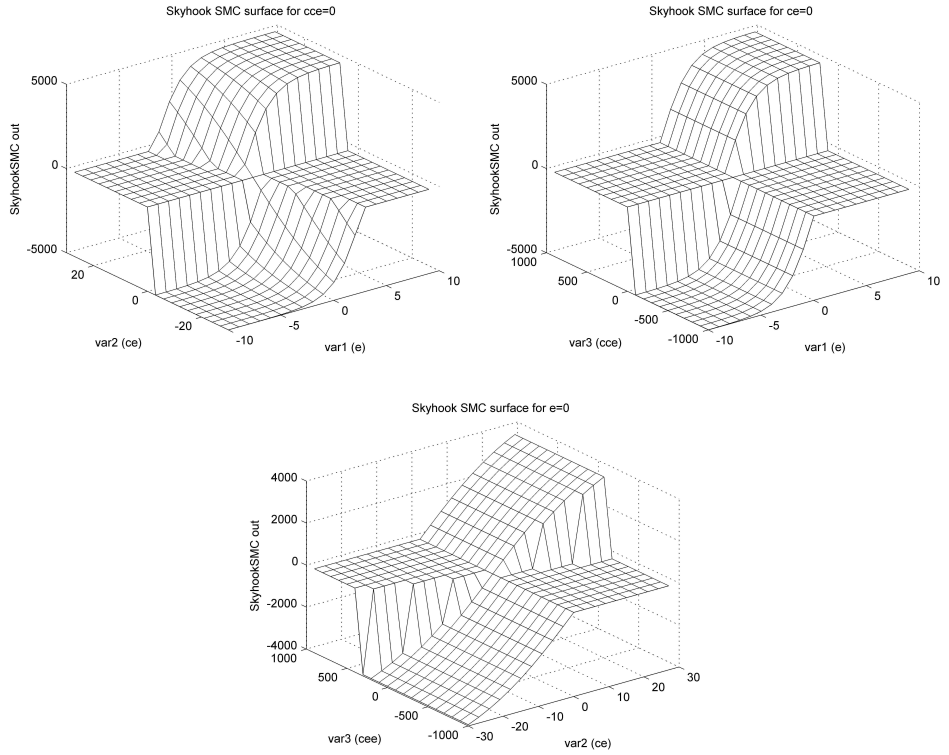


Fig. 8. Three control surface cuts for the Skyhook sliding mode controller used as a reference to train the PWM-FIS controller.

The initial setting of the PWM-FIS parameters (before training) was an equally spaced seven membership function partition for the three input universes (grid partition). This architecture gives rise to a 343-rule zero-order Sugeno-type PWM inference model with 373 adjustable parameters. The membership functions were overlapped by pairs with crossing points at 0.5 (see Fig. 9 on the left), while all the consequents were set to one. All signals, both inputs and outputs, were normalized to the unitary range [0,1]. Normalization improves the performance of training algorithms and, in any case, the FIS cores in the SoPC must be fed with normalized 8-bit input signals. After completing the training process, both antecedent parameters and consequents were adjusted to minimize the training error as much as possible (see Fig. 9 on the right for antecedents). In this case the achieved sum of squared error (SSE) for the complete training set (8000 points) was $SSE = 1.706 \times 10^{-3}$. Some cuts of the obtained PWM-FIS approximated control surfaces are depicted in Fig. 10 (same views as in Fig. 8).

.

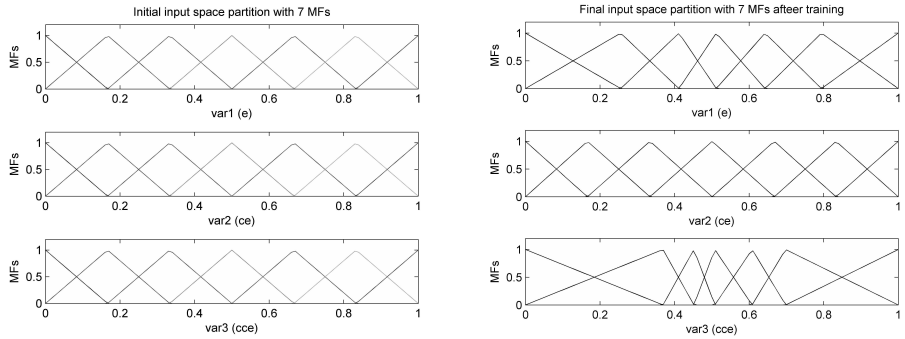
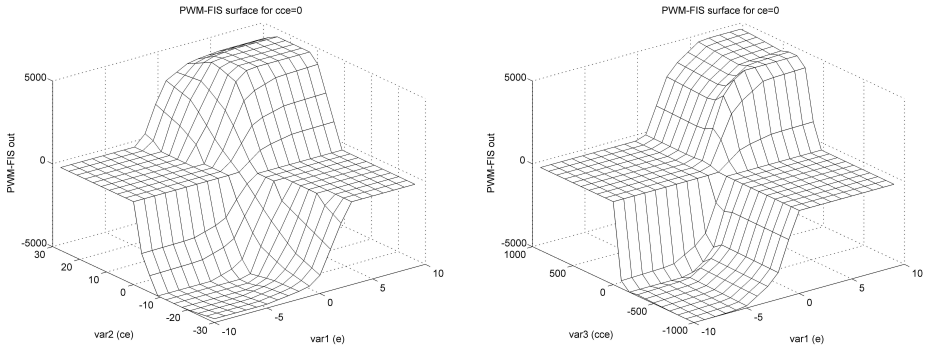


Fig. 9. Seven membership function partitions of the input space for the PWM FIS system. Left: before training. Right: after training.

Since when running the PWM-FIS controller the output signal has to be denormalized to cover the whole range of the output dumping force, this may give rise to an amplified error in the control signal that can be problematic at the more sensitive signal ranges of the control system, mainly for those inputs that should produce a zero output. To overcome this problem, the near-zero outputs are forced to zero by a dead zone function in the PWM-FIS output. In a similar manner, a saturation function has been added to assure that minimum and maximum dumping force values are not exceeded.



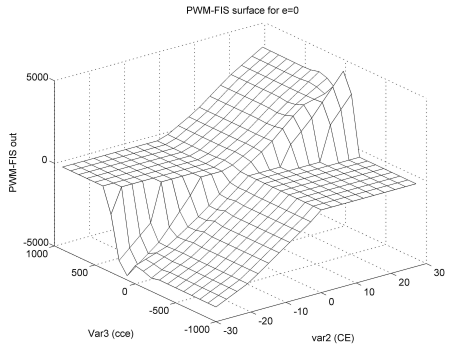


Fig. 10. Three control surface cuts for the PWM-FIS controller trained as approximator system for the Skyhook SMC.

The simulation performed in Ref. 23 was reproduced and later modified by substituting the original Skyhook SMC by the trained PWM-FIS controller simulator. A sampling frequency of 1 KHz maximum, i.e. a 1ms sampling period (T_s), was considered since higher operation rates are hardly achievable by SA suspension systems.[22] At this rate, if the delay introduced by the control signal computation (τ_s) is negligible in the controller design (as it was in this case), such delay must be approximately two orders of magnitude smaller than the sampling period ($\tau_s << T_s$). In this manner we can be sure that the sampling and the actuation are performed at the same sampling time, so the so-called *causality rule* is fulfilled as long as the analog to digital (AD) and digital to analog (DA) devices and device I/O pads are fast enough. The fulfillment of this condition implies that the control output should be computed in approximately τ_s = 10μs. As will be seen, our PWM-FIS FPGA implementation is able to compute the control outputs in just nine clock cycles at a 100MHz clock-rate, i.e. it has a τ_s = 0.09 μs input-output control signal latency (I/O pad delays aside). This is true for the computation of the four suspension control signals since they are computed in parallel, as explained at the beginning of this section and depicted in Fig. 11. Comparing to the average τ_s = 3600 μs that takes the MATLAB software implementation of the PWM-FIS controller or the average τ_s = 76.699 μs of the original Skyhook SMC running on a Pentium D at 3GHz to produce the control signal for one suspension system, the control signal delay of the PWM-FIS hardware seems negligible[*].

The outputs obtained from the simulation for car body acceleration, suspension deflection and tire load, both for the passive suspension and for the PWM-FIS controlled suspension, are shown in Figs. 12, 13 and 14 respectively. The RMSE value of the body acceleration signal for the PWM-FIS controlled SA suspension system is RMS = 1.1194, not as good as that obtained by the original controller but still better than the passive suspension figures.

[*] These time measures were made on a general purpose PC running a Windows OS and not on a dedicated microprocessor with optimized code, so they are only approximate values with comparative purpose.

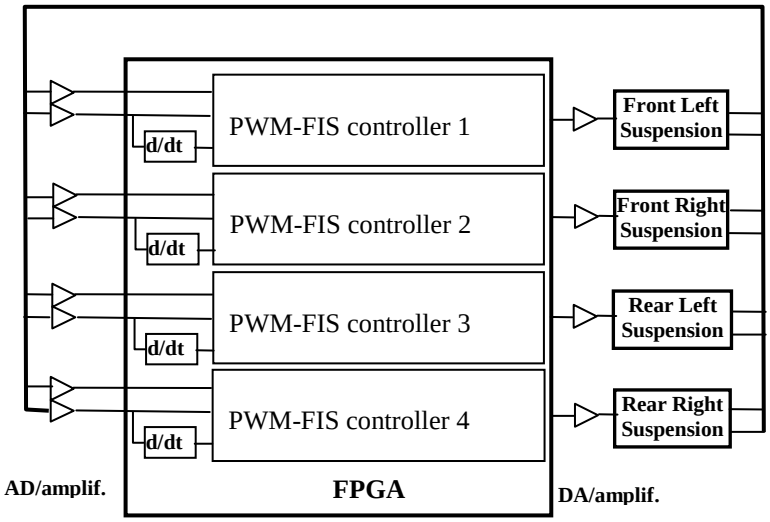


Fig. 11. Simplified scheme of the full car semi-active suspension control system. The full controller is embedded in a single FPGA.

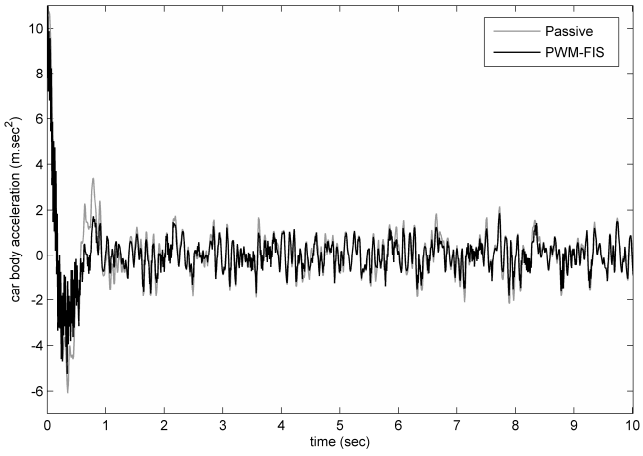


Fig. 12. Semi-active suspension system car body acceleration response with PWM-FIS control vs. passive suspension response

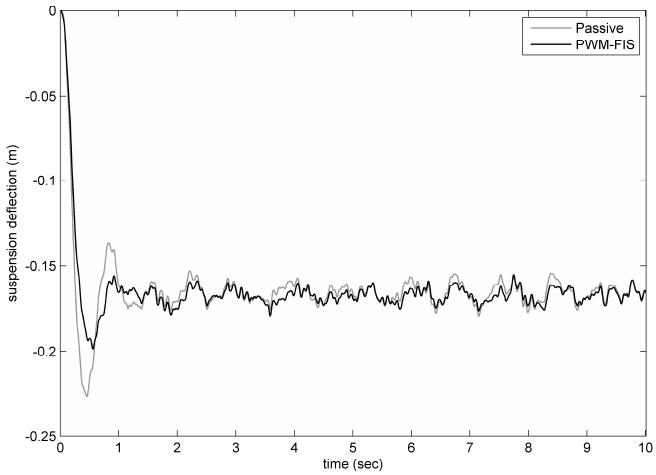


Fig. 13. Semi-active suspension deflection response with PWM-FIS control vs. passive suspension response.

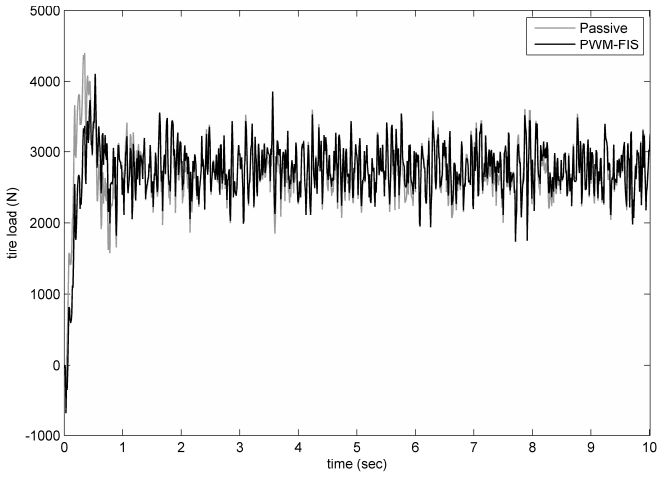


Fig. 14. Semi-active suspension system tire load with PWM-FIS control vs. passive suspension response

4.3. *PWM-FIS hardware characterization*

We developed the architecture depicted in Fig. 4 – with four FIS cores – for full car semi-active suspension control; each core implements one of the four PWM-FIS controllers depicted in Fig. 11. The design has been implemented using the XC5VLX50T device of Xilinx's Virtex 5 family; this family is specially suited for high performance logic. The

selected device has 7200 slices (each Virtex-5 slice contains four LUTs and four flip-flops), 48 DSP slices (each DSP slice consists of a multiplier, an adder, and an accumulator), and 60 RAM blocks of 36 Kbit.

After synthesis, our design uses 39 DSP slices (81%), 20962 Slice LUTs (72%), 22566 Slice LUT-Flip Flop pairs (78%), and 1044 Kbits of total memory (48%), among other resources. The MicroBlaze processor executes the software part of the system operating at 100 MHz, while the hardware partition, implemented in the logic fabric, performs the FIS cores with the same frequency. A three-input core requires nine clock cycles to perform the PWM-FIS computation, that is to say, only 90ns. In view of these results, it can be concluded that the proposed solution is suitable for high-speed fuzzy computation. The achieved performance is better than the one obtained in other hardware/software FPGA-based solutions for fuzzy hardware reported in the bibliography (see Section 3.1).

The hardware partition has been developed in VHDL with the aid of the ISE Design Suite 10.3 and the ModelSim 6.4 environment. The SW partition and the whole system integration have been performed by means of EDK and SDK 10.3 tools.

5. Conclusions

In this work, we have reported the development of a scalable architecture suitable for the implementation of high-speed fuzzy inference systems on reconfigurable hardware. The main advantages of the proposed architecture are scalability, high performance, and flexibility. Therefore, it can be used to develop fuzzy inference systems for real-time multidimensional problems with a large number of membership functions and a large rule base. The proposed solution has been used to implement a complex fuzzy controller for a particular application in the area of automotive control, a fuzzy controller for vehicle semi-active suspension system. The SoPC-based architecture implemented for this example, which features four three-input single-output PWM-FISs arranged in parallel, takes full advantage of the architecture scalability and the FPGA-technology capabilities. Our single chip hardware/software approach is able to compute a whole inference in only 90 ns, while serial processing of the four PWM-FIS would require more than 1.5μs to perform this computation.

On the other hand, with the aim of reducing the computational cost of the zero-order Sugeno inference algorithm, a few restrictions have been introduced in the membership functions definition. It has been shown that the use of triangular antecedents with a few additional constraints has a great impact on hardware simplicity, without detriment to the approximation capability of the inference model. This aspect of the model has been verified by approximating the input/output mapping of a sliding mode controller with Skyhook surface for a vehicle semi-active suspension system. The proposed approach is suitable for developing high-performance implementations for already known application areas of embedded fuzzy systems such as automotive, robotics, consumer electronics, and pervasive computing, among others.

Acknowledgments

The authors would like to thank the Basque Country Government and Spanish Ministry of Science and Innovation (MICINN) for supporting this work under Grants S-PE08UN49 and GIC07/138-IT-353-07, and TEC2009-07415, respectively.

References

1. L. Petters, and, S. Gou, *Fuzzy Hardware. Architectures and Applications*, eds. A. Kandel, and G. Langholz (Kluwer Academic Press, Massachusetts, 1998), pp. 27-42.
2. M. D. Hiller, What is scalability?, *ACM SIGARCH Computer Architecture News* **18** (1990) 18-21.
3. G. M. Amdahl, Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities, *Proc. of the AFIPS spring joint computer conference*, Vol. 30, 1967, pp. 483–485.
4. M. Sugeno, and G. T. Kang, Structure identification of fuzzy model, *Fuzzy Sets Syst.* **28** (1988) 15–33.
5. T. Takagi, and M. Sugeno, Fuzzy identification of systems and its applications to modeling and control, *IEEE Trans. Systems, Man, and Cybernetics* **15** (1985) 116-132.
6. K. Basterretxea, and I. del Campo, Electronic Hardware for Fuzzy Computation, *Scalable Fuzzy Algorithms for Data Management and Analysis*, eds. A. Laurent and M-J. Lesot (IGI Global, Hershey-New York, 2009), pp. 1-30.
7. A. Barriga, S. Sánchez-Solano, P. Brox, A. Cabrera, and I. Baturone, Modelling and implementation of fuzzy systems based on VHDL, *International Journal of Approximate Reasoning*, **41** (2006) 164-178.
8. R. Rovatti, Fuzzy piecewise multilinear and piecewise linear systems as universal approximators in Sobolev norms, *IEEE Trans. Fuzzy Systems* **6** (1998) 235–249.
9. G. Goth, Entering a Parallel Universe, *Communications of the ACM* **52** (2009) 15-17, doi:10.1145/1562164.1562171.
10. J.M. Paul, and B.H. Meyer, Ahmdahl's Law revisited for Single Chip Systems, *International Journal of Parallel Programming* **35** (2007) 101-123, doi:10.1007/S10766-006-0028-8.
11. Xilinx Inc.,Virtex-5 Family Overview (2008), http://www.xilinx.com/support/documentation /data_sheets/ds100.pdf.
12. Xilinx Inc., Microblaze Processor Reference Guide (2008), http://www.xilinx.com/support /documentation/sw_manuals/mb_ref_guide.pdf.
13. Altera Corporation, NIOS II Processor Reference Handbook (2008), http://www.altera.com /literature/lit_nio2.jsp.
14. A. Cabrera, S. Sánchez-Solano, P.S. Brox, A. Barriga, and R. Senhadji, Hardware/software Codesign of Configurable Fuzzy Control System, *Applied Soft Computing* **4** (2004) 271-285.
15. A. Cabrera, S. Sánchez-Solano, I. Baturone, F. J. Moreno-Velo, P. S. Brox, and A. Barriga, Development of Fuzzy Control Systems on Programmable Chips: Application to Mobile Robot Navigation, *Proc. Of the 10th International Symposium on Robotics and Applications*, 2004, pp. 167-172.
16. A. Di Stefano, and C. Giaconia, An FPGA-Based Adaptive Fuzzy Coprocessor, *Lecture Notes in Computer Science* **LNCS 3512** (2005) 590-597.
17. P. Echevarría, M. V. Martínez, J. Echanobe, I. del Campo, and J. M. Tarela, Design and HW/SW Implementation of a Class of Piecewise-Linear Fuzzy System, *Proc. of the XII Seminario Anual de Automática, Electrónica Industrial e Instrumentación*, 2005, pp. 360-364.

18. S. Sánchez-Solano, A. J. Cabrera, I. Baturone, F. J. Moreno-Velo, and M. Brox, FPGA Implementation of Embedded Fuzzy Controllers for Robotic Applications, *IEEE Transactions on Industrial Electronics* **54** (2007) 1937-1945.
19. I. del Campo, J. Echanobe, G. Bosque, and J. M. Tarela, Efficient Hardware/Software Implementation of an Adaptive Neuro-Fuzzy System, *IEEE Transactions on Fuzzy Systems* **16** (2008) 761-778.
20. L. M. Reyneri, Implementation Issues of Neuro-Fuzzy Hardware: Going Toward HW/SW Codesign, *IEEE Transactions on Neural Networks* **14** (2003) 176-194.
21. N. Jalili, A comparative study and analysis of semi-active vibration-control systems, *Journal of Vibration and Acoustics* **124** (2002) 593-605.
22. D. Fisher and R. Isermann, Mechatronic semi-active and active vehicle suspensions, *Control Engineering Practice* **12** (2004) 1353-1367.
23. Y. Chen, Skyhook surface sliding mode control on semi-active vehicle suspension system for ride comfort enhancement, *Engineering* **1** (2009) 23-32, doi:10.4236/eng.2009.11004.
24. E. M. ElBeheiry and D. C. Karnopp, Optimal control of vehicle random vibration with constrained suspension deflection, *Journal of Sound and Vibration* **189** (1996) 547-564.