A SOFT COMPUTING APPROACH TO HARDWARE SOFTWARE CODESIGN

V. Catania(1), N. Fiorito(1), M. Malgeri(1) and M. Russo(1)
(1)Istituto di Informatica e Telecomunicazioni e (2)Centro di Calcolo
Facoltà di Ingegneria - Università di Catania - Viale A. Doria, 6 1-95125 Catania

Abstract
This paper faces the problems connected to hardware software codeign partitioning phase. We propose a tool which novelties are the approach to perform the choice between hardware and software implementation. To achieve this, the tool proposed benefits from the simultaneous use of fuzzy logic and genetic algorithms, which allow the performance of single modules to be evaluated without having to actually implement them. At last, we propose an algorithm to choose a good solution.

1 Introduction
Hardware software Codeign is a design discipline whose aim is the simultaneous design of the hardware and software components of a system with a view to optimizing both the performance and the cost of the system as a whole. Research in this field has been given great incentive by events such as the development of VLSI technology applied to ASICs, which has made it possible to implement complex algorithms in hardware, with excellent performance levels and a moderate cost. By contrast, the development of RISC processor technology has allowed the software implementation of several functions which previously had to be implemented in hardware. Research in the field of Hardware/Software Codeign is therefore oriented towards the definition of systematic design methodologies which will reduce the limits of the classical approach to codeign, in which premature partitioning between hardware and software modules led to drawbacks such as hardware being specified without taking software calculation requirements into account, or software which was unable to influence the development of hardware, and which will allow choices regarding implementation on each single module to be postponed.

In the last two years several methodologies have been presented, along with frameworks to implement them, in which it is possible to distinguish between two approaches: software-oriented[1] and hardware oriented. The former is based on a totally hardware implementation and moves blocks to software according to delay times; the latter implements the system in software and moves the modules which do not allow certain requirements to be met to hardware. Both, however, are based on the knowledge of the designer and relatively simple cost functions. This paper aims to provide a possible solution to the problem mainly in the embedded system area, with a view to drastically reducing design costs and at the same time facilitating implementation choices by faster exploration of the possible implementation alternatives. To achieve this, the tool proposed benefits from the simultaneous use of fuzzy logic and genetic algorithms, which allow the performance of

The fuzzy approach to the development of a system falls into the so-called soft computing category, that is, one based on the lower computational cost inherent in imprecision and uncertainty. The choice of fuzzy logic[2] is due to the fact that it is capable of specifying the magnitudes observed in fuzzy terms, i.e. in an imprecise way: it does not, in fact, use the traditional concept of membership or non-membership in a given set, but refers to the concept of degree of membership. Thanks to this it is possible to define an object as very good, or slow, or cheap, and so on. Unfortunately, a fuzzy approach normally supplies knowledge bases which are easy for a human to interpret but have a low learning capacity. To overcome this limit the genetic technique has been used since they have a high learning capacity.

We have integrated an expert system, based on fuzzy logic, in the framework which interacts with the other elements in each phase of design and assists the designer in optimizing specifications, partitioning and evaluating the system. The latter phase involves two steps: first the software or hardware cost of each module in the system is assessed; so it is possible to make an initial partitioning hypothesis which will be evaluated in the second step.

The cost of assessing partitioning by subsequent refinement is clearly reduced because the assessment being made by the expert system requires no simulation or synthesis.

The paper is organized as follows: Section 2 introduces fuzzy logic and genetic algorithms. Section 3 describes the kernel of the tool proposed in this paper: in the subsection 3.1 it describes the tool; Section 3.3 is an example of application of genetic algorithms to a real case and the section 0 describes the algorithm used by the module which performs the partitioning.

2 Fuzzy Logic and Genetic Algorithms

2.1 A Brief Description of Fuzzy Logic

Fuzzy logic is based on the concepts of linguistic variables and fuzzy sets. A fuzzy set in a universe of discourse (usually called \( \mathcal{U} \)) is characterized by a membership function (called \( m \)) which assumes values in the interval \([0,1]\). A fuzzy set \( F \) is represented as a set of ordered pairs, each made up of a generic element \( u \in \mathcal{U} \) and its degree of membership \( m(u) \). A linguistic variable \( x \) in a universe of discourse \( \mathcal{U} \) is characterized by a set \( \mathcal{W}(x) = (W_{x1}, W_{x2}, \ldots, W_{xn}) \) and a set \( \mathcal{M}(x) = (M_{x1}, M_{x2}, \ldots, M_{xn}) \), where \( \mathcal{W}(x) \) is the term-set, i.e. the set of names the linguistic variable \( x \) can assume, and \( W_{xi} \) is a fuzzy set whose membership function is \( M_{xi} \). If, for instance, \( x \) indicates a temperature, \( \mathcal{W}(x) \) could be \( \mathcal{W}(x) = \{ Low, High \} \).
Medium, High), each element of which is associated with a membership function.

If x and y are taken to be two linguistic variables, fuzzy logic allows these variables to be related by means of fuzzy conditional rules of the type: "if (x is A) then (y is B)" where (x is A) is the premise of the rule, while (y is B) is the conclusion. This rule makes it possible to deduce, using specific inferential methodologies, a fuzzy set for y for each input value of x, whether it is associated with a fuzzy set or assumes a numerical value (crisp). The degree of membership of the premise is calculated and, through application of a fuzzy logic inference method (typically max-dot or max-min) to the conclusion, it allows the output y to be determined. In general in a fuzzy conditional rule if prem is then conclusion, the premise is made up of a statement in which fuzzy predicates \( P_j \) (in the following also called antecedents) of the general form \( (X_j \text{ is } A_j) \) are combined by different operators such as the fuzzy operators and or, in this case \( X_j \) is a linguistic variable defined in the Universe of the Discourse \( U \) and \( A_j \) is one of the names of the term set of \( X_j \).

The following is an example of a fuzzy conditional rule using such operators:

\[
\text{if } P_1 \text{ and } P_2 \text{ or } P_3 \text{ then } P_j \quad \text{where:}
\]

\[
P_1 = (X_1 \text{ is } A_1), \quad P_2 = (X_2 \text{ is } A_2), \quad P_3 = (X_3 \text{ is } A_3), \quad P_j = (Y_j \text{ is } B_j).
\]

To apply an inference method to the conclusion, it is first necessary to assess the degree of membership, \( \theta \), of the premise, through assessment of the degrees of membership \( \alpha_i \) of each predicate \( P_i \) in the premise.

The membership degree \( \alpha_i \) is calculated by assessing the degree of membership of a generic value of \( X_j \) in the fuzzy set \( A_j \). If \( X_j \) is made up of a fuzzy set, its degree of membership \( \alpha_i \) is determined by making an intersection between the fuzzy value of \( X_j \) and the fuzzy set \( A_j \) and choosing the maximum value of membership.

\[ \text{MIN}(\alpha, \beta) \quad \text{MIN}(\alpha, \beta) \quad \text{MIN}(\alpha, \beta) \quad \text{MIN}(\alpha, \beta) \]

\[ \text{MAX}(\alpha, \beta) \quad \text{MAX}(\alpha, \beta) \quad \text{MAX}(\alpha, \beta) \quad \text{MAX}(\alpha, \beta) \]

\[ 1 - \alpha \quad 1 - \alpha \quad 1 - \alpha \quad 1 - \alpha \]

The generic operators are summarized in Figure 2-1.

\[ \alpha \text{ and } \beta \quad \text{MIN}(\alpha, \beta) \quad \text{MIN}(\alpha, \beta) \quad \text{MIN}(\alpha, \beta) \quad \text{MIN}(\alpha, \beta) \]

\[ \alpha \text{ or } \beta \quad \text{MAX}(\alpha, \beta) \quad \text{MAX}(\alpha, \beta) \quad \text{MAX}(\alpha, \beta) \quad \text{MAX}(\alpha, \beta) \]

\[ \text{not } \alpha \quad 1 - \alpha \quad 1 - \alpha \quad 1 - \alpha \quad 1 - \alpha \]

\[ \text{Figure 2-1 Fuzzy logic operators} \]

Once the value of \( \theta \) is known, an inference method can be applied to assess the conclusion. The latter is expressed in the form:

\( Y_j \text{ is } B_j \) and \( Y_2 \text{ is } B_2 \) and \( Y_3 \text{ is } B_3 \)

where: \( Y_j \) are linguistic variables and \( B_2 \) are names belonging to the term set \( W(Y_j) \).

In this case, the fuzzy operators and acting on the output fuzzy predicates have a different meaning than in the premise.

Application of the inference process, in our case center of area[3], means that the output variable is obtained using the following formula:

\[
y = \sum_{i=1}^{k} \theta_i \cdot C_i \cdot A_i = \sum_{i=1}^{k} \theta_i \cdot A_i.
\]

\( \Theta_i \) is the generic degree of membership of the premise belonging to \( i \)-th rule, \( C_i \), and \( A_i \) are, respectively, the center and the area of the output fuzzy set belonging to \( i \)-th rule.

2.2 An Overview of Genetic Algorithms

Genetic algorithms are models which are based on naturally occurring situations. These algorithms are a sort of artificial evolution of virtual individuals, selected by means of a Fitness Function. In practice, genetic algorithms are a robust way to search for the global optimum of a genetic function with several variables which are very flexible and not sensitive to the classical problem of local optimum. Genetic algorithms are essentially based on three operators: reproduction, crossover and mutation. The flow of a classic genetic algorithm can be simply explained as follows:

\[ \text{Given an initial population of } n \text{ elements (where } n \text{ is the cardinality of population):} \]

\[ P(0) = (P_1, \ldots , P_n) \]

\[ \text{While not (End Condition)} \]

\[ \text{Obtain } P(t+1) \text{ from } P(t), \text{ applying reproduction operator} \]

\[ \text{Obtain } P_a(t+1) \text{ from } P(t+1), \text{ applying crossover operator} \]

\[ \text{Obtain } P_b(t+1) \text{ from } P_a(t+1), \text{ applying mutation operator} \]

\[ P(t+1) := P_b(t-1); \]

\[ t := t+1; \]

end while

The generic operators are summarized in Figure 2-2.

\[
\begin{array}{|c|c|}
\hline
\text{Operator} & \text{Operation performed} \\
\hline
\text{Reproduction} & \text{The individuals in the population } P(t+1) \text{ come from the probabilistic selection of the individuals in the population } P(t) \text{ with the greatest fitness.} \\
\text{Crossover} & \text{Applied to two chromosomes (parents) it creates offspring using the genes of both. In most cases a point is chosen and two offspring are created. The first will have a chromosome with the father’s chromosome in the first part and the mother’s in the 2nd. The sibling will have the opposite.} \\
\text{Mutation} & \text{Each single bit of a generic sibling will be inverted with a probability of pm.} \\
\end{array}
\]

\[ \text{Figure 2-2 Main operator applied to population} \]

The basic theorem of genetic algorithms guarantees that the individual with the greatest fitness and shortest chromosomes will increase in each generation. This allows a rapid evolution towards individuals with the best degree of fitness.

2.3 Implementation of the Genetic Algorithm

Several genetic algorithms have recently been developed with different implementations of the genetic operators, thus
allowing the creation of a new class of algorithms which
give better results in a shorter time in terms of convergence
towards an optimal solution. One of these is the algorithm
used to implement the Fuzzy Performance Estimator.

We used a variant of canonical genetic algorithms which
always maintains the best solution into the population be-
cause it is proved [4] that canonical genetic algorithms will
never converge to the global optimum.

2.4 Fuzzy Rules

Each individual in the genetic population is a set of \( R \)
fuzzy rules; in turn, each rule comprises \( I \) inputs
(antecedents) and \( O \) outputs (consequents) represented by
an equal number of fuzzy sets connected by the fuzzy operator
\( \land \). The membership functions are Gaussian in the form:
\[
\mu(x) = e^{-\frac{(x-c)^2}{2\sigma^2}},
\]
which can thus be characterized by the
centre (\( c \)) and the variance (\( \sigma \)).

Every element in the grid is therefore an individual whose
genetic heritage is made up of the sequence of \( I \) inputs and \( O \)
outputs for all the \( R \) rules, giving a total of \((I+O)R\) fuzzy
sets.

Optimization is achieved by calculating a normalized
relative percent error. Given the nature of the genetic
algorithms, the number of rules and antecedents in the fuzzy
program is minimized and the fitness function is naturally
maximized. In order to enhance performance in terms of
execution time and minimization of genetic algorithm error
[5], an equivalent neural network has been associated to
processing of the fuzzy rules. The resulting network is a
three-layered one, in the first state of which are represented
the \( \alpha \) (degrees of truth) resulting from each input fuzzy set
each input pattern and each rule (thus a total of \( IR \) nodes, respectively the number of inputs, rules and patterns).
The second layer represents the \( \theta \) resulting from
the processing of each rule for each input pattern (\( RP \) nodes).
The third layer represents the consequents of the fuzzy rules,
that is, the outputs of our system.

By applying this procedure it is possible to adjust the
centre and variance values of the fuzzy sets so as to improve
the convergence and execution time of the learning
algorithm, which means minimizing the standard deviation
mentioned above.

Therefore the mixture of genetic algorithms and neural
networks generates a fuzzy inference, i.e. fuzzy rules and
tuned membership functions. These results represent the best
description of the cost function in term of fuzzy logic which
is represented by the examples used to learn.

3 The Framework

The framework this paper proposes is a complete tool
which goes from specification of the system to be imple-
mented to actual physical implementation. Where possible,
the tools used were ones available on the market.

The design of an embedded controller is a classical field
in which to adopt a method based on co-design, on account
of the fact that design is not so complex as to introduce an
excessive number of degrees of freedom. Currently, the de-
sign of a controller mainly consists of selecting the best
commercially available components that can be used.

Therefore the tool proposed in this paper, in this early de-
velopment phase, has been built mainly to design embedded
systems. These systems are, usually, dedicated to a specific
application and vary greatly in both size and field of
application, e.g. automotive applications, control of
electrical appliances, industrial plant control. They are often
reactive systems, that is, systems which perform specific
functions in response to stimuli from the surrounding
environment. On account of the great number and variety of
embedded systems, it is extremely important to automatize
design. In development environments it is necessary to be
able to use both very simple processors (8 bits) and highly
complex ones to be able to cover all possible fields of
application.

In defining a system, the designer does not always have
precise parameters by which to judge the quality of
his solution, and so uses approximate parameters mainly based
on experience. He can, in fact, expect a particular function to
give a good performance when implemented in software, or
an interface with the outside world to be sufficiently fast if
hardware-implemented, and so on. His final choice falls on
the solution which represents a trade-off between cost and
efficiency, and checks that this is true by synthesizing the
hardware components and simulating their behaviour, or
even by going so far as to build a prototype of the controller.
This solution is obviously very costly in terms of time and is
not easy to generalize as it mainly depends on the individual
designer developing the system.

In order to provide real assistance in the development of a
design by using the co-design approach, it is necessary to
reduce the search for an optimal solution and provide a
knowledge base comparable to that of an expert designer. If
this knowledge base can then be updated by learning each
time the system is used, even further improvement is
achieved. To design a controller using this method it is
therefore necessary to imitate the behaviour of a human
being, in particular his capacity to reason by approximation,
i.e. to define and use imprecise assessments such as quite
good, very fast, inexpensive, etc. In addition, a designer
often has to deal with requirements expressed in linguistic,
and therefore imprecise, terms: a device which is "not too
slow and not too large in size" instead of a device with "t
seconds" delay and a maximum size of x centimetres".

The first step towards solving the problem is to find a tool
capable of describing the system as a whole before it is split
into hardware and software (partitioning). The tool thus has
to be capable of representing the system without establishing
that a subset has to be hardware- or software-implemented.
The description of the system also has to be modular and
able to be simulated to provide the designer with parameters
he can use to assess his choices. But the choice of which
modules to implement in hardware and software and on the
basis of which data this choice is to be made is the key
problems to solve. The idea proposed in our framework is
that of using an expert system based on fuzzy logic and
genetic algorithms.

3.1 The Structure of the Framework

To obtain a versatile, efficient tool. various tools available
on the market, whose reliability was already known, were
used.
To insert the high-level specifications of the system the SpeedChart EFSM formalism was used, although originally conceived for hardware, it proved suitable for specification of the software parts needed in a system of the kind being studied. SpeedChart allows a specification to be made in the form of deterministic extended finite state machines (EFSM) as a code block can be associated with each state. SpeedChart can also provide specification implementations in VHDL. As the SpeedChart tool does not provide an output in a programming language, a translator was developed from the internal representation into C. The choice of VHDL outputs for the hardware and C for the software allow any kind of synthesizer and CPU to be used for the controller.

The framework comprises the following elements (see Figure 3-1):
- insertion of the specifications of the system to be developed and subdivision into components;
- pre-partitioning of the system, i.e. identification of the components which have to be in hardware (HW components in Figure 3-1), in software (SW components in Figure 3-1), and those whose implementation has not yet been established (codesign components);
- characterization of codesign modules. It is performed by FPE basing its results on the knowledge database created in a learning phase which must be performed on the technology;
- partitioning of the system by a module called a Decision Maker (DM). It puts out a possible implementation;
- synthesis of the components of the partitioning solution found, simulation and assessment. This phase allows to update the technology database and, by the use of FPE off-line, also the updating of knowledge database.

In the light of the results obtained, it is possible to see whether the partitioning meets the design requirements. If not, another partitioning hypothesis is made by the DM, taking the results of the co-synthesis into account. If the requirements are met, the final implementation of the system can take place. As can be seen, the great advantage of the methodology proposed is that the co-synthesis phase is only repeated if the real co-synthesis results greatly differ from those estimated by the expert system. In practice design starts with insertion of the specification according to the requirements, which is currently done using SpeedChart. When the description of the whole system has been inserted its accuracy is checked, where possible. The designer then divides the specification into functional modules. It should be pointed out that at this stage no choice has been made as regards the type of implementation (hardware or software) for each single module. The next step consists of translating the specification for each module into both VHDL and C. At this point the partitioning, or rather what we call pre-partitioning, of the system begins: the modules are divided into three sets: software, hardware and codesign. The first and second sets comprise modules which necessarily have to be implemented in software and hardware respectively for the requirements to be met, while the third set includes all those modules for which no precise decision has been made. It is on the modules belonging to this latter set that Decision Maker, using the knowledge previously created by the Fuzzy Performance Estimator, works.

3.2 The Fuzzy Performance Estimator

Fuzzy Performance Estimator, exploiting genetic algorithms, estimates some fundamental parameters for choice of the modules, on the basis of the semantics of the description in VHDL (hardware) or C (software). The results of Fuzzy Performance estimator are the value needed by the Decision Maker to choose a partitioning.

It is a fact[6] that the VHDL constructs used to describe a hardware module affect the characteristics of its silicon implementation, especially as regards latency times and the area occupied. Unfortunately it is not possible to quantify the final silicon characteristics, nor is it possible to state with any accuracy which parameters are of importance at each stage: it is only possible to state that the structure of the VHDL code contains information about the cost (not only in term of money) of the final circuit. By the same reasoning the structure written in C includes information about the cost of the software module.

In the tool proposed in this paper it was therefore decided to consider almost all the parameters which could be extracted from the codes - either VHDL or C - because genetic algorithms by themselves does not take into account the parameters which are not relevant. The advantage of this approach is that it is independent of the kind of technology since, as mentioned previously, it takes all the significant parameters into consideration and so their weight can vary according to the technology used.

The parameters taken into consideration during our work to assess the hardware modules are the following: the total number of code lines, the width of the bus, the number of processes and blocks accessing the bus, the number of internal signals, the number of ports, the size of the registers, the number of parallel processes, the number of local variables, the number of loops, the maximum length of the loops, the number of conditional assignments, the number of elementary logical instructions, the number of multiplication or division instructions in a loop, the number of addition instructions in a loop, the number of if constructs, the number of case constructs, the number of synchronization instructions, the number of assignments to variables, and the number of assignments to signals. For software the following characteristics were taken into consideration: the number of synchronization instructions, the number of assignments to variables, the number of type conversions, and the number of multiplication or division instructions outside a loop. Up to now the number of parameters used to evaluate the software are a subset of the final number since we are still working on
them. But we feel that could be easy to extend their number in order to cover other aspects or different technologies.

At this point the problems to solve are:

- expressing the relationship between the parameters listed and those needed to assess the cost of hardware or software implementation (cost and maximum delay, for example)
- establishing the weight each parameter has in this relationship.

Fuzzy logic, which by its nature tolerates imprecision, is highly suitable to solve this kind of problem. Through fuzzy conditional rules, in fact, it allows the formalization of procedures that can only be imprecisely expressed. At the same time, using the concept of fuzzy sets, it allows us to handle imprecision regarding the values of the parameters, weighting them through the definition of appropriate membership functions.

In our approach the fuzzy sets needed to obtain the fuzzy relationships are generated using the genetic algorithms, which allow construction of the fuzzy relationships needed to interpret the parameters chosen for a given technology. They also allow us to eliminate those parameters which, on the basis of the examples chosen, are not significant.

The system obviously depends on the accuracy of the set of examples used to learn the technology, i.e. a preliminary phase is required to gather a set of significant examples the expert system can use to create its database; therefore this database could be updated automatically as the system develops so as to take design variations into account.

3.3 An Example

The Fuzzy Performance Estimator (FPE) is the block which estimates the characteristics of the modules belonging to the co-design set, using exclusively parameters which can be extracted from their high-level specification or implementation in VHDL or C.

In order to check that the FPE is functioning properly, some real cases were chosen, more specifically a certain number of combinatory modules including multi-input adders, subtractors and multipliers, totalling 200 examples which are part of a fuzzy processor currently being designed. These modules were implemented using a 0.7 µm CMOS technology. For the sake of simplicity each module featured the following parameters: width of external bus (ebd), number of VHDL code lines (cln), I/O ports (top), number of internal operations (nio), number of internal variables (niv), number of combinatory operations (nco). These parameters were chosen as they seemed to be the most significant for the examples chosen. An analysis performed on a large number of parameters including, for example, the number of divisors and the number of synchronization instructions showed, in fact, that only those chosen had significant variations in their values. Of course, in assessing further examples it will be necessary to take all the other significant parameters into account.

To allow the FPE to learn this technology, the 200 patterns were subdivided into 160 learning patterns and 40 testing patterns. After about 80000 iterations the confidence parameters reached highly satisfactory values, the standard deviation from the testing patterns being about 10%, while that from the learning patterns was about 5%. The FPE also showed that the number of conditional constructs is not as significant as it was thought to be before the learning program was executed on the basis of analysis of the variations in the values of the parameter.

<table>
<thead>
<tr>
<th>rule</th>
<th>ebd</th>
<th>cln</th>
<th>top</th>
<th>nio</th>
<th>niv</th>
<th>ncc</th>
<th>nco</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Figure 3-2: output membership set](image)

In practice the complete study of a technology requires identification of a number of significant examples and the parameters which appear to affect the values to be estimated. The data collected is supplied to the FPE, which will only be able to learn what it requires, eliminating any insignificant input parameters.

The fuzzy program represented in Figure 3-2 is the result given by the assessor and allows us to estimate the characteristics of the two outputs we have chosen: area and delay.

<table>
<thead>
<tr>
<th>rule 2: if (ebd is very high) then (area is very high) and (delay is very high)</th>
<th>ebd is very high because it is at the upper end of the membership set, so</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule 4: if (ebd is high) and (nio is (high or very high)) and (niv is medium) then (delay is very high)</td>
<td>ebd is high because the value it assumes is in the upper part of the membership set, even though it is not on the upper bound nio is high or very high because the value assumed is in the upper part of the membership set, the curve representing this set is not very steep niv is medium because the value falls in the centre of the membership interval, with a very slight curve.</td>
</tr>
</tbody>
</table>

3.4 The Decision Maker

The DM closes the partitioning phase which starts with division of the modules into the three classes - hardware, software and codeign.

It is evident that the choice of the possible partitioning is fundamental. For the process to be used it cannot be confined to proposing just any one of the various possible
partitionings, but has to supply the one that appears to be the best. To make this choice, the DM has to be able to analyze the possible solutions as quickly as possible, assessing each single module but without having to make simulations which are usually long and wasteful.

The novelty of our approach is that these choices are made using fuzzy logic which, on the basis of the knowledge obtained by the Fuzzy Performance Estimator data calculated by the assessment module, can extract the partitioning which minimizes a chosen cost function (e.g. the cost of the system, latency times, etc.). Use of a fuzzy technique obviously does not ensure that the result is the very best but it is a good partitioning.

The problem the Decision Maker has to solve is that of finding the best partitioning possible without, however, carrying out an exhaustive search. For this purpose we propose an algorithm using the information obtained from the previous modules, specifically the cost and delay parameters estimated by the assessor. In order to clarify the functioning of the algorithm we have to point out that the cost of partitioning mainly depends on three factors: the cost of the single module (hardware or software); the cost of implementing the communication primitives independent of the other modules; the cost of communication between two modules. Below we will indicate the cost functions as follows:

\[ CR_i(l) \text{ where } l \in \{ \text{hardware, software} \} \text{ the cost of implementing module } i \]

\[ CC_i(l) \text{ where } l \in \{ \text{hardware, software} \} \text{ the cost of independent communication} \]

\[ CM_{ij}(l) \text{ where } l \in \{ hwh, ssh, hsw, shw \} \text{ the cost of communication due to interaction between module } i \text{ and } j \]

The function CR is estimated on the basis of fuzzy rules and fuzzy sets given by Fuzzy Performance Estimator, for each module in the codesign set, for both the hardware and the software versions; likewise, it is possible to estimate the CC function on the basis of the technique chosen. The CR function depends on the way in which the modules are actually implemented, so it can assume four separate values.

The cost function associated with a partitioning is given by the following formula:

\[ C_p = \sum_{l=1}^{m} (CR_i(l) + CC_i(l)) \]

\[ C_p = \sum_{l=1}^{m} (CR_i(s) + CC_i(s)) \]

\[ C_p = C_{loc} + C_{tc} + \sum_{j} CR_{ij} \]

The task of the Decision Maker is therefore none other than to minimize the function \( C_p \).

The approach we propose here is an algorithmic one: it starts from the consideration that the values of the functions CR and CC are known, or rather estimated, for all the modules in the codesign set.

1. Assessment of the cost of all the modules belonging to the codesign set, both the software and hardware configuration, \( CM_{ij} = CR_i + CC_i \).
2. Verification that the system requirements are met, both in terms of costs and also with respect to other factors.
3. Elimination of any module not meeting the codesign set requirements.

4. Selection for each module of the implementation whose (estimated) overall cost is lowest. This set of modules represents a possible partitioning, henceforward called Ph, where h is the number of cycles already performed in the search for a good partitioning.

5. Descriptive simulation of the system based on the estimated values, i.e. currently using SpeedChart: this simulation is made possible by knowledge of the modules which have to be implemented in hardware and software (according to partitioning h) and the characteristics estimated by the Fuzzy Performance Estimator (e.g. delay, cycle time, maximum delay, etc.). If this simulation is satisfactory the algorithm goes to step 7.

6. Search for the module k whose interaction communication cost is maximum and exchange of implementation. The new partitioning is called h + 1. If the module found was exchanged in the previous cycle the partitioning h - 1 is selected and the exchange is made eliminating k from the search set.

8. Procedure as in 5, but synthesizing and simulating the system.

4 Conclusions

This paper proposes a tool which will allow the designer to (partially) automatize the development of embedded systems by using the codesign approach. The tool takes advantage of both tools which are already available on the market for VLSI CAD and techniques based on soft computing. The main novelty proposed is the use of artificial intelligence techniques to imitate the behaviour of a human in defining the partitioning of the system, choice of the techniques to be used fall on fuzzy logic, which allows the imprecision. We have also shown that the use of genetic algorithms allows cost functions to be obtained from the various modules without excessive computational effort.

Further studies should be devoted to techniques to optimize the genetic algorithm, in both the representation and processing of data. Work is also being done on the use of formal techniques to describe the system.