Until recently, it was possible to distinguish between software- and hardware-oriented approaches to design. In software-oriented design, the designer implements the system in software and moves modules that do not meet certain requirements to hardware. Hardware-oriented design starts with a fully hardware implementation; the designer moves blocks to software according to delay times. Both, however, are based on the designer's knowledge and relatively simple cost functions.

Hardware-software codesign allows simultaneous design of a system's hardware and software components, thus exploiting the advantages of both, optimizing performance and cost, and allowing easier redesign when needed. Incentive for research in this field has increased greatly since the development of hardware devices that can implement complex algorithms with excellent performance levels and moderate cost. In addition, RISC processor technology allows software implementations of several functions that previously required hardware implementations to meet time requirements. Some of the more interesting studies on codesign seek to establish an integrated design methodology supported by appropriate tools.

Codesign includes the allocation of hardware and software, performance evaluation, validation, and synthesis. One of the key items is evaluation of the system's cost and partitioning. We are striving to provide a possible solution for this area of codesign—mainly in the embedded system arena. Our aim is to drastically reduce design costs and at the same time facilitate implementation choices by offering faster exploration of the possible alternatives. To achieve this, we propose a partitioning tool that also allows designers to evaluate the performance of single modules without having to actually implement them. Our tool benefits from the simultaneous use of soft computing techniques: fuzzy logic, genetic algorithms and neural networks.

The fuzzy approach to system development falls into the category known as soft computing, which is based on the lower computational cost inherent in imprecision and uncertainty. In particular, we chose fuzzy logic because it can specify observed magnitudes in fuzzy terms—that is, in an imprecise way. It does not, in fact, use the traditional concept of membership or nonmembership in a given set, but refers to the concept of continuous degree of membership. Thus, we can use fuzzy logic to define an object as very good, slow, inexpensive, and so on.

Unfortunately, a fuzzy approach normally supplies knowledge bases that are easy for a human to interpret but have no learning capacity. To overcome this limit, we combined a genetic algorithm technique with neural networks, since these have a high learning capacity.

The idea underlying our project was to meet two specific requirements. The first was to provide a work environment that would integrate tools already available for hardware and software system design, thus simplifying the designer's work. The second, the central part of our work, was the use of fuzzy techniques to develop the system. In this context, we had essentially two correlated problems to deal with:

- to allow a preliminary cost evaluation of the single modules making up the system being developed, without low-level synthesis and simulation; and
- to partition through fuzzy techniques, providing a solution at an acceptable cost in terms of calculation time.

The approach we propose is a homogeneous one; it uses fuzzy logic both in the preliminary cost evaluation of the various modules and in the partitioning phase. The module evaluation tool uses genetic algorithms and neural networks but generates a set of fuzzy rules. For partitioning, we are developing a tool that uses a fuzzy knowledge base. This tool does not yet learn auto-
matically, but it is already capable of heuristically achieving fast, efficient partitioning. Our technique clearly reduces the cost of assessing partitioning, because the expert system's assessment requires no simulation or synthesis.

In this article, we introduce the basic concepts of fuzzy logic and genetic algorithms (see the Fuzzy logic and genetic algorithms box) and illustrate how we use them in our tools. We also give a detailed description of the framework we used with particular reference to our fuzzy performance evaluator, which estimates the cost of the modules, and our decision maker, which partitions the system.

**What our tools do**

The framework we propose is a complete tool for embedded controller design that covers all aspects of design, from specifying a system to its physical implementation. Where possible, we used products available on the market.

Embedded systems are usually dedicated to a specific application and vary greatly in both size and field of application—the range includes, for example, automotive applications, control of electrical appliances, and industrial plant control. Embedded systems fall into the class of reactive systems—that is, systems that react to the environment by exer-

---

**Fuzzy logic and genetic algorithms**

Together with neural networks, fuzzy logic and genetic algorithms give us soft computing techniques.

**Fuzzy logic**

Fuzzy logic is based on the concepts of linguistic variables and fuzzy sets. A fuzzy set in universe of discourse $U$ is characterized by membership function $m$, which assumes values in the interval $[0,1]$. We represent fuzzy set $F$ as a set of ordered pairs, each made up of a generic element $u \in U$ and its degree of membership $m(u)$.

A linguistic variable $x$ in universe of discourse $U$ is characterized by sets $W(x) = \{W_1, \ldots, W_n\}$ and $M(x) = \{M_1, \ldots, M_n\}$. $W(x)$ is the term set—the set of $n$ names linguistic variable $x$ can assume. $W_i$ is the generic fuzzy set whose membership function is $M_i$. If, for instance, $x$ indicates a temperature, $W(x)$ could be $W(x) = \{\text{low}, \text{medium}, \text{high}\}$, each element is associated with a membership function.

We write the rules governing a fuzzy system using linguistic expressions that formalize the empirical rules with which human operators use their own experience to describe a process. If we take $x$ and $y$ to be two linguistic variables, fuzzy logic allows these variables to be related by the type of connective operators such as the fuzzy operators AND and OR. In this case, $x$ is a linguistic variable defined in universe of discourse $U$, and $A_i$ is one of the names of $x$'s term set.

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

If $P_1 \text{ AND } P_2 \text{ OR } P_3$, then $P_4$.

where $P_1 = (x_1 \text{ is } A_1); P_2 = (x_2 \text{ is } A_2); P_3 = (x_3 \text{ is } A_3); P_4 = (y \text{ is } B_4)$.

To apply an inference method to the conclusion, we must first assess the premise's degree of membership $\theta$. To do this, we assess degrees of membership $\alpha_i$ of each predicate $P_i$ in the premise. See Figure A.

We calculate membership degree $\alpha$ by assessing the degree of membership of a generic value of $x$ in fuzzy set $A_i$.

**Figure A.** Inferential procedure using Min-Max method: If $x$ is low OR $y$ is low then $z$ is medium (a); if $x$ is low AND $y$ is medium, then $z$ is low (b). $Z = (\theta_1 c_1 A_1 + \theta_2 c_2 A_2)/(\theta_1 A_1 + \theta_2 A_2)$. 

---

May/June 1997  63
Fuzzy logic and genetic algorithms (continued)

A, If \( x_i \) is made up of a fuzzy set, we determine its degree of membership \( a_i \) by making an intersection between the fuzzy value of \( x_i \) and fuzzy set \( A \), and choosing the maximum value of membership. If \( x_i \) is a crisp value, its degree of membership in fuzzy set \( A \) consists of the value the membership function of \( A \) assumes corresponding to \( x_i \).

Thus, we calculate the premise’s membership degree \( \theta \) by assessing the fuzzy operations on the predicates. More specifically, fuzzy logic operator AND, when applied to predicates \( P_i \) and \( P_j \), returns the minimum of \( (a_i, a_j) \). In the same situation, fuzzy logic operator OR returns the maximum of \( (a_i, a_j) \). The precedence rules governing the AND and OR operators are similar to those of binary logic operators. Once we know the value of \( \theta \), we can use an inference method to assess the conclusion. We express this conclusion as

\[
(y_i \text{ is } B_i) \text{ AND } (y_j \text{ is } B_j) \text{ AND } (y_k \text{ is } B_k)
\]

Here, \( y_i \) are linguistic variables, and \( B_i \) are names belonging to term set \( W(y) \). In this case, the AND fuzzy operators acting on the fuzzy output predicates (also called consequents) simply express in a single rule several rules with the same premise but different conclusions.

For defuzzification, we use the center-of-area process.  

Genetic algorithms

These models are not based on mathematical considerations but on naturally occurring situations. Genetic algorithms are a sort of artificial evolution of virtual individuals, selected by means of a fitness function.

In practice, genetic algorithms provide a robust way to search for the global optimum of a generic function with several variables. This method is very flexible and is not sensitive to the classical problem of local optimum.

Genetic algorithms are essentially based on three operators: reproduction, crossover, and mutation. The following simple algorithms show the flow of a classic genetic algorithm:

Given an initial population of \( n \) elements, where \( n \) is the cardinality of population,

\[
\begin{align*}
1 & : t := 0 \\
2 & : P(0) := (P_1, ..., P_n); \\
3 & : \text{While not (End Condition) do} \\
3.1 & : \text{Generate } P_a \text{ from } P(0), \text{ applying reproduction operator; } \\
3.2 & : \text{Generate } P_b \text{ from } P_a, \text{ applying crossover operator; } \\
3.3 & : \text{Generate } P_c \text{ from } P_b, \text{ applying mutation operator; } \\
3.4 & : P(t + 1) := P_c; \\
3.5 & : t := t + 1. \\
4 & : \text{end while}
\end{align*}
\]

Table A summarizes a genetic algorithm’s generic operators. The basic theorem of genetic algorithms guarantees that the virtual individual with the greatest fitness and shortest chromosomes will increase in each generation. This allows a rapid evolution toward individuals with the best degree of fitness.

Implementing genetic algorithms

Researchers have recently developed several genetic algorithms with different implementations of the genetic opera-
delay and a maximum size of x square centimeters.

The first step toward solving the problem is to find a tool capable of describing the system as a whole before partitioning. The tool thus must be capable of representing the system without establishing that a subset must be implemented in hardware or software. The system description also must be modular and simulatable to give the designer parameters to use in assessing choices. But the choice of which modules to modular and simulatable to give the designer parameters to use in assessing choices. But the choice of which modules to implement in hardware or software—and which data will be the basis for this choice—is the key problem to solve.

The idea our framework embodies is that of using an expert system based on fuzzy logic to estimate the parameters needed to characterize the modules and to allocate each module to hardware or software.

**Framework structure**

To obtain a versatile, efficient tool, we used various commercially available tools, whose reliability we already knew.

To insert the system’s high-level specifications, we used SpeedChart Extended Finite-State Machines (EFSM). Although originally conceived for hardware, this tool proved suitable for specifying the software parts needed in a system of the kind we were studying. SpeedChart allows the designer to make a specification in the form of deterministic EFSM, because a code block can be associated with each state. SpeedChart can also provide specification implementations in VHDL. As the tool does not provide output in a programming language, we developed a translator to translate the output from the internal representation into C. Our choices of VHDL outputs for the hardware and C for the software allow us to use any kind of synthesizer and CPU for the controller.

The framework includes the following phases:

1. inserting system specifications and subdividing the system into components;
2. prepartitioning the system—that is, identifying the components that must be in hardware (HW components in Figure 1), software (SW components in Figure 1), and those whose implementation has not yet been established (codesign components);
3. characterizing codesign modules with the fuzzy performance estimator (FPE), which bases its results on the knowledge database created in a learning phase. This learning phase must be performed on the technology used to implement our fuzzy performance estimator.

Researchers have established that execution of a genetic algorithm leads to genetic homogenization. That is, the individuals in a population end up so alike that no significant improvements between one generation and the next are possible. We therefore made the attempt to find countermoves that allow us to avoid this problem but at the same time prevent the genetic algorithm from degenerating into a purely random search.

Because Rudolph proved that canonical genetic algorithms will never converge to the global optimum, we used a variant of canonical genetic algorithms that always maintains the best solution in the population.

Before we describe the algorithm itself, however, we’ll discuss the representation of the fuzzy rules the tool uses. 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 AND. The membership functions are Gaussian in the form

\[ \mu(x) = e^{-\frac{(x-c)^2}{2\sigma^2}} \]

which can thus be characterized by center c and variance \( \sigma \). Every element in the grid is therefore an individual whose genetic heritage comprises the sequence of I inputs and O outputs for all R rules. This gives a total of \((I + O)R\) fuzzy sets.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reproduction</td>
<td>The individuals in population ( P_A ) come from the probabilistic selection of the individuals in population ( P(t) ) with the greatest fitness.</td>
</tr>
<tr>
<td>Crossover</td>
<td>Applied to two chromosomes (parents), this operator creates offspring using the genes of both. In most cases, we choose a point called Crossover, 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 second. The sibling will have the opposite.</td>
</tr>
<tr>
<td>Mutation</td>
<td>This operator inverts each single bit of a generic sibling with a probability of ( p_m ).</td>
</tr>
</tbody>
</table>

We achieve optimization by calculating a normalized mean-root-square relative-percent error. Given their nature, genetic algorithms naturally minimize the number of rules and antecedents in the fuzzy inference and maximize the fitness function. To enhance performance in terms of execution time and minimization of genetic algorithm error, we have associated an equivalent neural network to the fuzzy inference. For a more detailed description of the learning tool, see Russo. We can minimize the standard deviation mentioned earlier.
that will be used in system implementation;
4. partitioning the system with the decision maker (DM),
a tool that produces a possible implementation; and
5. synthesizing, simulating, and assessing the components of
the partitioning solution, and simulating and assessing
the global system. This phase updates the technology
database and, through offline use of the fuzzy performance estimator, also updates the knowledge database.

Figure 1 shows these steps. Steps 3 and 4 represent the
main novelty of this article. Here, we combine genetic
algorithms, neural networks, and fuzzy logic to achieve good
results in a reasonable time.

With the results obtained from these steps, the designer
can see whether the partition meets design requirements. If it
does, the final system implementation can take place. If not, the decision maker makes another partitioning hypothesis,
taking the cosynthesis results into account. Sometimes
low-level synthesis results differ from those estimated by
the expert system; then the designer must undertake a new partitioning. Of course, during the partitioning phase, the expert
system takes the low-level synthesis results into account.

In practice, design starts with insertion of the specifications
according to the requirements; our system accomplishes
this using SpeedChart. When the designer has inserted
the description of the whole system, the system checks its
accuracy where possible. The designer then divides the specifi-
cation into functional modules. Note that at this stage, the
designer has made no choice regarding the type of imple-
mentation (hardware or software) for each single module.

The next step consists of translating the specification for each
module into both VHDL (the hardware language) and C (the
software language). At this point, the system’s partitioning—or
rather what we call prepartitioning—begins. Supported by the
tool, the designer divides modules into three sets: software,
hardware, and codesign. The first and second sets comprise
modules that necessarily must be implemented in software and
hardware for the requirements to be met. The third set includes
those modules for which no precise decision has been made.
It is on the modules belonging to this latter set that the deci-
sion maker, using the knowledge previously created by the
fuzzy performance estimator, works.

**Fuzzy performance estimator**

Exploiting the genetic algorithms, the fuzzy performance estimator estimates
fundamental parameters for module choice, on the basis of the semantics of
the description in VHDL or C. The fuzzy performance estimator’s results are the
values the decision maker requires to choose a partitioning.

Carlson has demonstrated that the
VHDL constructs used to describe a
hardware module affect the characteristics of its silicon implementation,
especially vis-à-vis latency times and area occupied. Unfortunately, it is not possi-
ble to quantify the final silicon characteristics, nor is it possible to state with
any accuracy which parameters are of importance at each
stage. We can state, however, that the structure of the VHDL
code contains information about the cost (area, delay, and
so on) of the final circuit. By the same reasoning, the structure
written in C includes information about the cost of the software module.

The fuzzy performance estimator takes into account the
parameters that the designer considers most important for
evaluation of the code. An important feature of genetic algo-
rithms by themselves is that they do not consider irrelevant
parameters. Therefore, we must choose the set of examples
for the learning phase in such a way that they represent a general
situation. The advantage of this approach is that it is inde-
pendent of the kind of technology used. This is because, as
mentioned previously, it considers all the significant parameters,
and so their weight can vary according to the technolo-
gy used. In practice, we can take into consideration a set of
parameters that are common to various hardware technologies
(the number of signals, for example, does not depend on the
technology size). On the basis of the example data, the tool
eliminates insignificant parameters.

Table 1 lists the parameters taken into account during our
assessment work. Until now, the parameters used to evalu-
ate the software have been a subset of the final number
because we are still working on them. It would be easy, how-
ever, to extend their number to cover other aspects or dif-
ferent technologies.

At this point the main 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); and
- establishing the weight each parameter has in this relationship.

Fuzzy logic, which by its nature tolerates imprecision, is
highly suitable for solving 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 parameter, weighting them
through the definition of appropriate membership functions.
In our approach, we use the genetic algorithms to generate the fuzzy sets needed to obtain the fuzzy relationships, which allow interpretation of the parameters chosen for a given technology.

The system obviously depends on the accuracy of the set of examples used to teach the technology. That is, we need a preliminary phase 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 to take design variations into account.

**Example**

The fuzzy performance estimator is the block that estimates the characteristics of the modules in the codesign set. To do this, it uses exclusively parameters that can be extracted from their high-level specification or implementation in VHDL or C.

To check that the fuzzy performance estimator is functioning properly, we examined some real cases. Specifically, we chose several combinatory modules including multi-input adders, subtractors, and multipliers; these 200 examples are the components of the fuzzy processor we designed. We implemented these modules using 0.7-micron CMOS technology. For the sake of simplicity, each module featured the following parameters: bw, cl, pt, is, lv, if, if, no. We chose these parameters because 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, we will have to take all the other significant parameters into account.

To allow the fuzzy performance estimator to learn this technology, we subdivided the 200 patterns into 160 learning patterns and 40 testing patterns. After about 80,000 iterations, the confidence parameters reached satisfactory values. The error obtained from the testing patterns was about 10%, while that from the learning patterns was about 5%. The fuzzy performance estimator also showed that the number of conditional constructs is not as significant as we had thought after analyzing the variations in the parameter values before executing the learning program.

As Figure 2 shows, bw 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. Parameter is 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.

### Table 1. Parameters taken into account during our work.

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>cl</td>
<td>Total number of code lines</td>
</tr>
<tr>
<td>bw</td>
<td>Bus width</td>
</tr>
<tr>
<td>pb</td>
<td>Number of processes and blocks accessing the bus</td>
</tr>
<tr>
<td>is</td>
<td>Number of internal signals</td>
</tr>
<tr>
<td>pt</td>
<td>Number of ports</td>
</tr>
<tr>
<td>rs</td>
<td>Register size</td>
</tr>
<tr>
<td>pp</td>
<td>Number of parallel processes</td>
</tr>
<tr>
<td>lv</td>
<td>Number of local variables</td>
</tr>
<tr>
<td>lp</td>
<td>Number of loops</td>
</tr>
<tr>
<td>ll</td>
<td>Maximum loop length</td>
</tr>
<tr>
<td>ca</td>
<td>Number of conditional assignments</td>
</tr>
<tr>
<td>li</td>
<td>Number of elementary logical instructions</td>
</tr>
<tr>
<td>md</td>
<td>Number of multiplication or division instructions in a loop</td>
</tr>
<tr>
<td>ad</td>
<td>Number of addition instructions in a loop</td>
</tr>
<tr>
<td>if</td>
<td>Number of if constructs</td>
</tr>
<tr>
<td>nc</td>
<td>Number of combinatory operations</td>
</tr>
<tr>
<td>cs</td>
<td>Number of case constructs</td>
</tr>
<tr>
<td>si</td>
<td>Number of synchronization instructions</td>
</tr>
<tr>
<td>sa</td>
<td>Number of assignments to signals</td>
</tr>
<tr>
<td>tc</td>
<td>Number of type conversions</td>
</tr>
<tr>
<td>mdol</td>
<td>Number of multiplication or division instructions outside a loop</td>
</tr>
</tbody>
</table>

If (bw is very high),
then (area is very high) AND (delay is very high).

As can be seen in Figure 2b, bw is very high because it is at the upper end of the membership set.

We can express rule 4 by the following function:

If (bw is high) AND (is is (high or very high))
then (delay is very high).

As Figure 2d shows, bw 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. Parameter is 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. Parameter lv is medium because the value falls in the center of the membership interval, with a very slight curve.

The other functions can be obtained in the same way from the other parts of Figure 2.

**Decision maker**

On the basis of the module cost in terms of area and delay, this block allocates the modules to hardware or software. The module cost can be the true cost, the cost estimated by the fuzzy performance estimator, or that given by the designer. (The literature contains various proposals on this
It is evident that the choice of the possible partitioning is fundamental. For the process to be effective, it cannot be confined to proposing just any one of various possible partitionings; rather, it must supply the one that appears to be the best. To make this choice, the decision maker must analyze the possible solutions as quickly as possible, assessing each single module without making simulations, which are usually long and wasteful.

The novelty of our approach is that the decision maker uses fuzzy logic (based on the knowledge obtained by the fuzzy performance estimator) to extract a possible partitioning. The problem the decision maker must solve is that of finding a partitioning without, however, carrying out an exhaustive search. (In this case, the total number of possibilities is \(2^n\), where \(n\) is the total number of modules.) As mentioned previously, before the modules are mapped onto the hardware or software part of the system, a target architecture must be identified.

Here, we refer to the typical architecture of embedded reactive systems (see Figure 3). There is a precise limit on the number of modules that can be implemented in hardware; this is determined by the size and number of FPGAs (or other similar components) used. For software, this constraint depends on the amount of memory available, and so is much less limiting. In this phase, the system also develops the algorithm for scheduling the modules (whether hardware or software).

In our framework, the cost of each module is represented in fuzzy terms, that is, a fuzzy set is associated with each module. The decision maker uses these estimated values to obtain a possible partitioning.

The decision maker is essentially based on a fuzzy program that gives the best allocation for any module. Depending on the input values, it returns a quality index measuring whether the module could or must be done in hardware (or software) to meet requirements or software to meet the implementation cost requirement. If a module cannot be mapped either in hardware or software, the decision maker traces back to find a different allocation for already placed modules, moving them from hardware to software or vice versa.

Table 2 defines the parameters the fuzzy program uses.

Values \(C_{\text{max}}\) and \(D_{\text{max}}\) are threshold values above which the cost and delay are unacceptable. The designer must provide these values. The following rules give us an evaluation of the best allocation for the modules.

**Relative (monetary) cost of a module.** For this parameter, we introduce

\[
\text{RCost}_i = \frac{C_{i,\text{hw}}}{N} - \frac{C_{i,\text{sw}}}{N} \sum_{i=1}^{N} C_{i,\text{hw}} - \sum_{i=1}^{N} C_{i,\text{sw}}
\]

the value of which is in the range of \([-1,1]\). This allows us to assess the resources needed to allocate the module to software or hardware. The associated fuzzy rule is the following:

If (RCost is positive)
then (\(M_i\) should be sw)
else (\(M_i\) should be hw).

If the value of RCost tends toward 1, we know that module \(M_i\) requires a greater amount of the hardware resources available than of those for software. Thus, (in fuzzy terms) it is advisable to implement it in software. The opposite happens if the value tends toward \(-1\).
Relative cost of communication. This parameter allows us to assess to what extent the choice to implement a module in hardware or software affects communication cost. We define the following relations:

\[
\text{HwCost}_i = \sum_{j=1}^{N} CC_{ij}^{hw, tj}
\]

HwCost measures the communication cost involved if the \( i \)th module were implemented in hardware.

\[
\text{SwCost}_i = \sum_{j=1}^{N} CC_{ij}^{sw, tj}
\]

SwCost, on the other hand, measures the communication cost involved if the module were implemented in software.

\[
\text{MaxCost} = \max(CC_{ij}) \quad \forall i \in [1, N], j \in [1, N]
\]

MaxCost represents the maximum communication cost for all the modules. The following formula, therefore, gives the next fuzzy rule input:

\[
\text{RelCommCost}_i = \frac{\text{HwCost}_i - \text{SwCost}_i}{\text{MaxCost} + N}
\]

We use the denominator factor to normalize the values of RelCommCost in such a way that they belong to \([-1, 1]\). The fuzzy rule associated with this input is

If (RelCommCost) is positive
then (\( M_i \) should be sw)
else (\( M_i \) must be hw).

This measures the impact on communication of the choice to implement module \( i \) in hardware or software. Here again, the considerations made earlier hold. If, in fact, the function tends toward a value of 1, a software implementation is preferable; if it tends toward \(-1\), we should implement the module in hardware.

Implementation cost. For this parameter, we define

\[
\text{ImplCost}_i = \frac{\text{CR}_{\text{tot}} + \text{CR}_{\text{hw}} - \text{CR}_{\text{max}}}{\text{CR}_{\text{max}}}
\]

This value allows us to assess the cost of hardware implementation. The output value is again in the range \([-1, 1]\) and allows us to assess whether it is possible, according to the monetary cost, to implement the module in hardware. The fuzzy rule associated with this input is

If (ImplCost is negative)
then (\( M_i \) should be hw)
else (\( M_i \) must be sw).

If the value is negative, we can implement the module in hardware, because there are still resources available. If, however, it tends toward 0, the module would use up all the remaining resources, and so we must implement it in software. If even this is impossible, the decision maker discards the partitioning and traces back to look for an alternative.

Delay introduced by module. As our next input rule, we introduce

\[
\text{DelayCost}_i = \frac{\text{CR}_{\text{tot}} + \text{CR}_{\text{sw}} - \text{CR}_{\text{max}}}{\text{CR}_{\text{max}}}
\]

This input evaluates the delay introduced by implementing the module in software. The associated fuzzy rule is as follows:

If (DelayCost is negative)
then (\( M_i \) should be sw)
else (\( M_i \) must be hw).

Similar considerations to those made for implementation costs apply to this rule.

Communication cost. The last input of rules identified is

\[
\text{CommCost}_i = \frac{\sum_{j=1}^{n} CC_{ij}^{sw, tj} - \sum_{j=1}^{n} CC_{ij}^{hw, tj}}{n}
\]

This input quantifies the communication cost introduced by the module. Again, it ranges between \([-1, 1]\) and the associ-
which soft computing seems to have great advantages over traditional approaches. The main novelty of our approach is the codesign approach to partially automate the development of the behavior of a human in defining a system’s partitioning. Our use of artificial intelligence techniques to imitate the processing of data. We are also working on the use of formal techniques to describe our system.

The process. The decision maker uses this fuzzy program to search for a possible partitioning. It starts with the partition that uses the least software space, checks to see whether this process meets—in fuzzy terms—the global requirements, and allocates it. If the partition does not meet these requirements, the decision maker searches for a different partitioning until it reaches a solution to the problem. This algorithm is efficient because it makes all the evaluations in fuzzy terms with extremely short execution times. At present, the assessor module is at an experimental stage, but it has already given interesting results, especially as concerns execution times.

We propose a tool that will allow designers using the codesign approach to partially automate the development of embedded systems. The framework takes advantage of tools already available on the market for VLSI CAD as well as soft computing techniques. We focused our work mainly on evaluation of cost and partitioning, because this is the area in which soft computing seems to have great advantages over traditional approaches. The main novelty of our approach is our use of artificial intelligence techniques to imitate the behavior of a human in defining a system’s partitioning.

We hope to devote further studies to techniques to optimize the genetic algorithm, in both the representation and processing of data. We are also working on the use of formal techniques to describe our system.

References

Vincenzo Catania researches computer networks as an associate professor of computer science at the University of Catania. His interests include performance and reliability assessments of parallel and distributed systems, VLSI design, and fuzzy logic. Catania holds an electrical engineering degree from the University of Catania.

Michele Malgeri works as a researcher for Instituto di Informatica e Telecomunicazioni and manager of the local network. His current interests are hardware description languages, networks, and distributed systems. He holds an electrical engineering degree from the University of Catania.

Marco Russo works as a researcher for the Institute of Informatics and Telecommunications at the University of Catania. His current research interests are in fuzzy logic, neural networks, hardware architectures, and fuzzy inference learning algorithms.

Russo received the electronic engineering degree in electronics, informatics, and telecommunications from the University of Catania and the PhD degree in electronics, informatics, and telecommunications at the University of Palermo. He is on the editorial board of the Int’l Journal of Knowledge-Based Intelligent Engineering Systems.

For further information contact Marco Russo, Instituto di Informatica e Telecomunicazioni, Facoltá di Ingegneria—Universitá di Catania, Viale Andrea Doria, 6—1—95127 Catania, Italy; mrusso@k200.unict.it; www.cdc.unict.it/~mrusso/.