A Macroscopic Time and Cost Estimation Model Allowing Task Parallelism and Hardware Sharing for the Codesign Partitioning Process.

J.A. Maestro, D. Mozos, H. Mecha
Dpto. de Arquitectura de Computadores y Automática
Universidad Complutense - 28040 Madrid, Spain
E-mail: maestro@eucmax.sim.ucm.es

Abstract

This paper describes a method to estimate the implementation cost of the hardware part in a mixed hardware/software system, as well as the related performance. These estimations try to avoid the use of many implementation details in order to keep the complexity order of the process under control. The concepts of hardware sharing and parallelism are exploited to make a picture of the whole hardware cost associated to a given partition.

1.- Introduction.

The current demand of embedded systems, formed by a standard processor supporting a software program, and one or several hardware specific circuits (ASIC) grows quickly, making their time-to-market shorten progressively. As the manual design task becomes harder, an automation process is required, aiding to overcome this problem in a reliable way. The embedded systems have the important characteristic of being hybrid, that is to say, of being formed by a hardware and a software component. Thus, it is not advisable to use specific and single design techniques, but to integrate them in order to satisfy the implicit necessities of both parts.

The set of steps leading to perform a simultaneous and automatic design of these hardware and software elements is known as **Codesign** [1]. The main task within this process is **partitioning** [2], which decides whether a module is going to be assigned to hardware or to software. This task depends on the intrinsic necessities of the problem, mainly time and area constraints. So the main goal of the partitioning is to get a design that meets all these constraints while minimizing the area.

Most partitioning algorithms use an iterative process that, starting from an initial state, and moving functionalities through software and hardware, tries to get at a solution accomplishing the design goals. To guide this movement of functionalities, it is necessary to determine if a design is better than others, by means of an **estimation** process. The more realistic the estimations are, the better they will reflect the intrinsic capabilities of the system.

In the partitioning process, the software cost and performance are determined by the chosen architecture and memory hierarchy models, that are usually fixed in a previous stage. So, the main goal during this process is to estimate both parameters of the hardware part. Talking about this, and given a functionality, there exist several features that must be considered for a good estimation:

- It is possible to obtain several valid hardware implementations of this functionality with different values of area and performance by carrying out the inner scheduling and allocation in distinct ways. So, the partitioning algorithm not only has to decide where to place every module, but also which implementation should be used in case it is assigned to hardware.
- The hardware cost does not increase or decrease in a linear way by moving functionalities through the hardware and software partitions, since it depends, at any moment, on the particular distribution of the system. In other words, the cost of two single functionalities implemented in hardware does not correspond with the sum of the individual costs. This consideration is based on two basic concepts, hardware sharing and parallelism. Sharing implies that a module's functionality can be implemented with part of the hardware of a previous one. Parallelism means that two modules may be executed simultaneously and therefore, no hardware could be shared between them in that case.
- Finally, the estimation process has to consume as little time as possible, since this task is to be repeated many times. That is why most of the partitioning algorithms perform the estimation process in a quite unrealistic way, without considering the hardware sharing or the several possible hardware implementations for each module.

This work has been granted by CICYT TIC 96/1071 and HCM CHRX-CT94-0459

There are several Codesign systems that try to accomplish this estimation process. Some of the best known are:

In *Castle* [3], an environment for the Codesign process, the estimations are calculated by analyzing the basic blocks contained in the specification. The possibility of hardware sharing and parallel execution is not considered. Thus, a simplistic view of the whole system is applied to the process.

Cosmos [4] is a Codesign environment destined to an interactive performance of the partitioning task. Internally, it works with a set of interconnected processes, extracted from the initial specification. A certain parallelism degree among them is allowed, but nothing is said about how estimations are carried out. The possibility of sharing hardware is not available in an automatic way. What is more, the manual effort that the designer must do leads to a lack of efficiency in the process.

Cosyma [5], a well known Codesign system, performs the estimations without taking the hardware sharing or parallelism among tasks into account. The partitioning process is carried out by means of the Simulated Annealing algorithm, and then, and outer loop is executed in case that the obtained results do not match with the expected ones. This loop is to be executed too many times, due to the unrealistic estimations, with a considerable time consumption.

In [6], an Integer Programming approach is used to perform the partitioning process. When estimating the functionalities, the sharing possibility among them is allowed. Nevertheless, only modules that are equal can be considered, leading to an excessive simplification of the problem. Besides, nothing is said about the several implementations that a single module can have.

Another reasonable possibility is adapting the estimation processes developed under the HLS systems, in order to work on Codesign environments. In this way, Kurdahi [7] carries out an interesting approach with the SCALE system, which is able to estimate the area of a circuit at the RT-level. Although the predictions of this system are within 5% of the actual layout areas, the execution times for medium-low sized examples are unacceptable (5 to 226 seconds) for an environment of these characteristics.

In this system, as well in other similar ones [8], the greatest drawback arises when performing the area calculation, as this process can only be made on single hardware modules. Therefore, if some of these modules are integrated into a larger system, it must be completely designed to estimate its area, what consumes a great amount of time. Instead, this estimation should be carried out starting from the single modules' area values, that were previously found out, and with the consideration of the possible parallelism and hardware sharing.

Our approach tries to estimate the hardware implementation cost of several functionalities from a macroscopic view, but considering all the relevant aspects of the problem, as sharing and several possible hardware implementations. By macroscopic, we mean that the computation does not consider all the implementation details, but only those aspects most relevant to the problem.

2.- Problem definition.

The main pursued goal is to achieve a time and cost estimation model for a Codesign system, able to provide reliable values to the partitioner, and so fulfill this process in a realistic way. It is necessary to exploit the previous concepts of parallelism and hardware sharing to get at the proposed model, since it is the only possibility of succeeding in this purpose. According to our approach, the Codesign problem can be stated as follows:

Given a Codesign graph G={N,E,S,I}, a sharing matrix K, and a superset of individual estimation pairs P, find the minimum execution time and the related optimum cost.

N stands for the set of nodes, equivalent to the different system tasks.

E is the set of graph edges, which can be classified in several different types, and will be explained in the next section.

S is the function

$$S : \{i \mid i \in N\} \rightarrow Z_i = \{SW, HW_1, HW_2, \ldots, HW_{m(i)}\},$$

which assigns each node *i* to the software partition or to one of the different *m(i)* possible hardware implementations.

P is a superset $\{ P_1, \ldots, P_i, \ldots, P_n \}$ containing a set of implementations $\{ P_i^{SW}, \ldots, P_i^{HW_{m(i)}} \}$ for every node *i*, $1 \leq i \leq n$, $n = \text{Card}(N)$. Each element P_i^j , $j \in Z_i$, is a pair $(ex\_time_i^j, cost_i^j)$, which represents the different possible implementation parameters, execution time and cost, of the node *i*.

I is the function

$$I : \{i \mid i \in N\} \rightarrow P_i^{S(i)}$$

which assigns to each node the suitable implementation parameters, depending on the node implementation, *S(i)*.

K is the sharing matrix $\kappa_{[i1,j1],[i2,j2]}$, in which for every pair of node and hardware implementation, [i,j], its similarity with the rest of the pairs is stated as a real factor between 0 and 1. The matrix *K* is formed by means of a static analysis of the hardware implementations.

With all these considerations and trying to solve the stated problem, a time and cost model is proposed. In section 3 the time model is presented. In section 4, the scheduling process is described. The explanation of the cost model is in section 5. Finally, some experimental results are offered in section 6, and the conclusions and future work appear in section 7.

3.- Time model description.

From the inner structure of the system, a timing graph is extracted, whose description fits the previously defined graph *G*. As it was said, the graph's nodes are equivalent to the functionalities contained into the initial specification. Now, the set of edges must be defined, in order to give the right structure to this graph.

The graph has to be a support to study and calculate the system execution time, taking all the possible parallelism degrees enclosed in the design into consideration, as it was explained in the introduction.

The execution time of a system can be defined as the time interval in which there exists any kind of active information related to that system. By information is understood any of the two kinds of existing dependency flows, namely data and control flows. So, both of them must be reflected in the graph, to study the possible starting and ending time instants for each node. The different types of edges in this graph are:

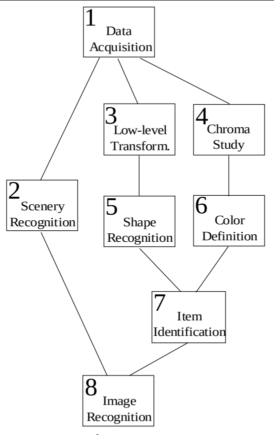
a) *SW Data and Control edges:* There is a predefined sequential order for the software nodes, forced by the impossibility of a parallel execution in the processor. So, there exists an associated control dependency flow, which can also be a data dependency flow, as the order of the nodes must respect the different data needs.

b) *HW Data edges:* Within the hardware part, there is also a set of data dependencies representing the data flow among the different hardware nodes.

c) *HW Control edges:* Apart from the loop and conditional control dependencies, that are not considered in this first approach, it does not exist any *natural* hardware control dependency, since any parallelism degree is acceptable. Nevertheless, there may be some *forced* control dependencies, obliging to the sequential execution of two nodes. This could cause an increase of the hardware sharing factor, and therefore, a reduction in the system cost. As the number and position of these control edges is arbitrary, a certain freedom degree is presented in the process, making the possible design space wider.

d) *HW-SW Data edge:* When two adjacent nodes are assigned to different partitions, the interchange of variables between them is performed by means of the system bus. This communication introduces a data dependency, with an associated time consumption.

With all these sets of edges, the formed timing graph is complete, that is to say, it contains every information flow, without any kind of redundancy. A typical Codesign problem is depicted in Figure 1, which represents an Intelligent Vision System modeled into generic tasks, with a certain bipartition previously given by the partitioner. The equivalent dependency graph is shown in Figure 2. The white nodes represent the software partition and the shaded nodes, the hardware partition. The bold lines are the active communications, with their related time overhead. The timing graph can be seen in Figure 3, together with the four explained set of edges. It is important to notice that in this graph, there is not any kind of control dependencies between hardware and software, since the parallelism between them is always exploited.



$S(1) = SW$	$I(1) = (100, 0)$
$S(2) = HW_2$	$I(2) = (40, 15)$
$S(3) = SW$	$I(3) = (45, 0)$
$S(4) = HW_1$	$I(4) = (25, 65)$
$S(5) = HW_3$	$I(5) = (10, 130)$
$S(6) = HW_1$	$I(6) = (12, 95)$
$S(7) = SW$	$I(7) = (75, 0)$
$S(8) = SW$	$I(8) = (90, 0)$

Table 1
Values of function *S* and *I*:
Status and Parameter definition

Figure 1

Given a timing graph as described before, the minimum time to execute the whole set of operators is defined by the critical path time. In order to execute a node, two different information flows must be taken into consideration: the data and control flows. That is to say, an operation cannot start until all the needed data have been produced by its predecessor, and the control, or permission to be executed, has been received.

As it was said, the timing graph contains in its edges all these information flows. So, taking the maximum path this information has to go through, means the impossibility of facing any piece of information with a higher life-time. So, since the global execution time was defined as the interval in which any data or control is active in the system, it will correspond to the critical path time.

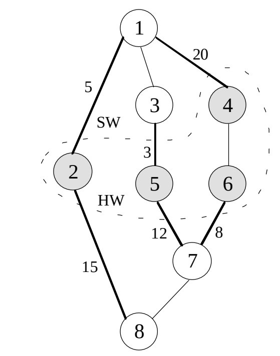


Figure 2
Data-dependency graph

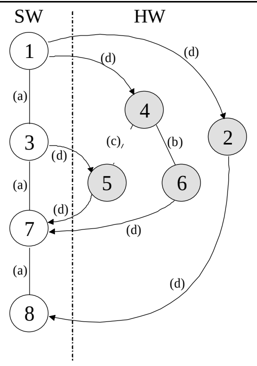


Figure 3
Timing graph

Before performing the timing study, it is necessary to define the topologic order (t.o.) of a graph. The t.o. is an ordered list of nodes, in which all kinds of dependencies are respected, and therefore all the predecessors of a node appear before it on the list, and all their successors are after it. Although the t.o. is not unique, the order of the software nodes given by the compiler is a valid one.

First, two time instants must be defined for each node, a Sooner Start time (SS) and a Later Start time (LS). Notice that these two parameters are similar, but not equal, to the scheduling times in HLS, as in the latter there were several discrete steps to schedule an operation, and in the former there exists a continuos time range. To compute these parameters, it is necessary to calculate the maximum and minimum time in which information can flow. The proposed algorithm to find out the SS values is:

1.- Assign to the first node in t.o., α, the time 0:

$$SS(\alpha) := 0$$

2.- For every node *i*, in t.o., calculate SS(i) as:

$$SS(i) = \max_{j \in Pred(i)} \{SS(j) + ex\_time_j^{S(j)} + \beta \bullet comm(j,i)\}$$

$$\beta = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are in different partitions} \\ 0 & \text{otherwise} \end{cases}$$

comm(i,j) is the time to transfer parameters from node *i* to *j*, using the system bus, and it is only active when both nodes are assigned to different partitions.

In other words, the sooner time in which a task can begin is when it receives all the needed information produced by its predecessors.

The procedure to calculate LS is analogous. First of all, let ω be the last node in t.o.

3.- The first step is

$$LS(\omega) := SS(\omega)$$

4.- For every node *i*, in reverse t.o., find out LS(i) as:

$$LS(i) = \min_{j \in Succ(i)} \{LS(j) - ex\_time_i^{S(i)} - \beta \bullet comm(i,j)\}$$

At the end, there will be a pair (SS, LS) for each node.

Every node *i* in the critical path meets the following condition:

$$SS(i) = LS(i)$$

So, the final execution time of the system would be:

$$T = SS(\omega) + ex\_time_{\omega}^{S(\omega)}$$

The results of the algorithm for the previous example appear in Figures 4 and 5. These data show that the critical path is formed by nodes 1, 3, 5, 7 and 8.

SS
1: **0**
2: 0+100+5=................**105**
3: 0+100=....................**100**
4: 0+100+20=..............**120**
5: max(100+45+3, 120+25)=.............**148**
6: 120+25=..................**145**
7: max(100+45, 148+10+12, 145+12+8)=.........**170**
8: max(170+75, 105+40+15)=.......**245**

Figure 4
SS calculation

LS
8: **245**
7: 245-75=...................**170**
6: 170-12-8=................**150**
5: 170-10-12=..............**148**
4: min(148-25, 150-25)=..............**123**
3: min(148-3-45, 170-45)=..............**100**
2: 245-15-40=..............**190**
1: min(100-100, 123-20-100, 190-5-100)= **0**

Figure 5
LS calculation

Two passes are done through the list of nodes to determine the associated times. For each node, it is necessary to check all its predecessors and successors, but as the node connectivity is much lower than the number of nodes, the latter operation is not relevant respect to the former. Therefore, the complexity of this algorithm is O(n).

For every node *j*, out of the critical path, *SS(j) < LS(j)*. So, it is possible to start its execution at any instant of time in the interval

$$I_1 = [SS(j), LS(j)].$$

In the same way, the moment in which the node finishes its execution is contained in the interval

$$I_2 = [SS(j) + ex\_time_j^{S(j)}, LS(j) + ex\_time_j^{S(j)}].$$

We define the range of a node *i*, *rg(i)*, as the interval

$$rg(i) = I_1 \cup I_2 = [SS(j), LS(j) + ex\_time_j^{S(j)}].$$

This interval corresponds to the period of time in which the node can be active. This definition is also valid for the nodes in the critical path. The ranges obtained in the example appear in Figure 6.

Now, for any node *i*, there is an associated range, and except for the nodes in the critical path, it is necessary to assign them an exact starting point, t_i, within I_1. The selection of this point is quite relevant to obtain the optimum system cost, as it will determine the possible hardware sharing factors.

4.- Node scheduling.

Talking about the concept of hardware sharing, the matrix *K* is the most clearly influent factor, which states the similarity between any pair of nodes, taking any of their possible hardware implementations into account. The meaning of *similarity* refers to the percentage of a node that can be shared to implement part of another one's functionalities. For instance, if two multipliers are used to implement a node *i*, those two functional units can be used by another node *j*, in a later instant of time.

This matrix is used to have available information of the different nodes' inner structures, without handling all the scheduling details, not only for the huge storage space necessary to keep them, but also for the elevated time consumption in their processing.

As it was said, the estimation task must be simple enough to keep the processing time within reasonable limit bounds, and therefore, tackling this problem from the previous orientation would break this principle. So, to meet the proposed goals in a reasonable time, all the inner scheduling information must be compacted into a simpler data, making it easier to handle. Here is where *K* makes sense within the design process.

The details that are considered to calculate *K* may vary, from a simple analysis to a more complex one. The more thoroughly this analysis is performed, the better it will reflect the sharing relationship among the nodes. Up to the moment, only the number and type of the different nodes are studied to perform this calculation, without considering their relative scheduling time. This provides an easy and quick method that can be carried out for many different implementations in a reasonable time.

The question that arises here is whether ignoring this information would lead to a significant error in the estimation process. It is important to notice that once several nodes have been assigned to hardware, the possibility of sharing functional units depends on their relative position and not only on their inner structure.

If there are two very similar nodes, even equal, but whose starting points coincide, no hardware sharing is allowed, as the parallel execution needs the duplication of the functional units. But if those two nodes are executed in different moments, the functional units could be shared, leading to a higher occupation, and therefore, to a reduction of the overall system cost. So, the possible node coincidence, known as *overlapping*, must be considered, as a basis to study the hardware sharing.

The overlapping degree of the node *i* over the node *j*, $\sigma_{i,j}$, represents the percentage of node *i* that may coincide in execution with any part of the node *j*. Basically, it can be defined as:

$$\sigma_{i,j} = \frac{rg(i) \cap rg(j)}{rg(i)}$$

Notice that as the overlapping degree represents a relative size, it is not a symmetrical factor, $\sigma_{i,j} \neq \sigma_{j,i}$. The same consideration can be applied to *K*.

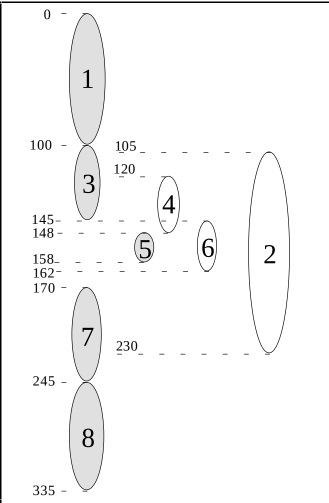


Figure 6
Ranges of the nodes

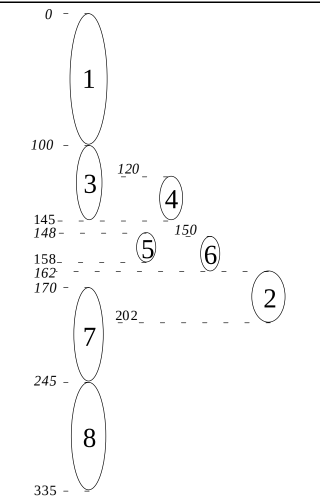


Figure 7
Exact starting points

Basically, the similarity $\kappa_{[i,S(i)],[j,S(j)]}$ between two nodes, *i* and *j*, has a positive impact on their possible sharing degree, and their relative overlapping $\sigma_{i,j}$ introduces a negative effect. However, this assert should be discussed, since it is possible to think of a particular case in which a maximum overlapping between two nodes allows a higher sharing than a partial one. Thus, the inner scheduling of the functional units has a relevant importance in this process, questioning the validity of the proposed approach.

Nevertheless, a closer look at the problem shows that this objection can be left aside within the overall problem environment. Providing that the focus is being set on Codesign typical applications, the nodes forming the graph, as a reflect of the initial system specification, are bound to contain a high number of operators. When the design is complex enough, as it usually is, the granularity chosen in a previous stage tends to be coarse, assigning to each node whole functions with large portions of code.

With this consideration, the previous side effect is not so influent, and the introduced relative error is lower, since the general cost is much higher. Not only that, these errors are supposed to compensate among them through the whole process, as statistically, and looking from a more abstract point of view, the extra cost estimated in a particular moment will be canceled by a negative error in a later one.

So, the macroscopic approach based on the shared matrix *K* is intuitively justified, rather than the microscopic technique dealing with a large quantity of complex data difficult to handle. Then, as a conclusion, to achieve an optimum cost, the overlapping factor among the different nodes should be minimized. This is possible by examining the different ranges of the nodes calculated in the previous stage, which introduce a certain freedom degree in the scheduling process. Thanks to this, the overall cost can be optimized, assigning a suitable starting point, t_i, to every node.

Among all the possibilities, the more similar two nodes are, the more interesting to decrease their overlapping degree, as the possible sharing would be higher. So, the following factor should be minimized:

$$\sum_{i \in HW} \sum_{j \in HW} (\sigma_{i,j} \cdot \kappa_{[i,S(i)],[j,S(j)]} + \sigma_{j,i} \cdot \kappa_{[j,S(j)],[i,S(i)]})$$

So, an algorithm based on *list scheduling* is introduced to achieve this goal in a reasonable amount of time. The main steps followed are:

1.- Schedule all the nodes in the critical path, as there is no possibility of selecting their starting point.

2.- Select the hardware node i, with a higher factor

$$\frac{(\sigma_{i,j} \cdot \kappa_{[i,S(i)],[j,S(j)]} + \sigma_{j,i} \cdot \kappa_{[j,S(j)],[i,S(i)]})}{rg(i)}$$

respect to another hardware node j, already scheduled in hardware. It is important to choose the nodes with smaller ranges in an early stage, because they have less chances to move, and therefore, when the assignation map is too loaded, there would be more problems to avoid the overlapping with the rest of the nodes.

3.- Study the relative position of i and j's ranges, in order to try to avoid their parallel execution as much as possible. Find the intervals I_1, I_2 obtained by dividing the initial range into two, as a result of the operation

$$I_1 \cup I_2 = rg(i) - [rg(i) \cap rg(j)]$$

The meaning of this is taking *rg(i)* out of the possible scheduling space of j, and therefore, avoiding the overlapping as much as possible. This operation is repeated recursively in order to place i in the best possible location. Whenever a node i is scheduled, the ranges of all their predecessors and successors have to be recalculated, in order to keep meeting the control and data dependencies. After this operation, there is the possibility that for some of these nodes, their SS and LS become equal. In this case, they have to be scheduled without any further consideration.

4.- Repeat the whole process, starting from 2.-, until no hardware nodes remain unscheduled.

By this method, the ranges of the nodes are gradually reduced, tending to move toward the areas with lower presence of parallelism, until the nodes become fixed and scheduled.

5.- Finally, all the remaining software nodes are scheduled in their respective SS, as they have no influence in the hardware sharing.

The result obtained for the previous example appears in Figure 7. The complexity of this algorithm, since it is based on *list scheduling*, can be bound by $O(n^2 \cdot \log(n))$.

At this point, when a starting time has been assigned to every node, it is the moment to calculate the overall system cost.

5.- Cost model description.

When a starting point, t_i,, has been assigned to every existing node, i, the related range, *rg(i)* is reduced to the interval $rg(i) = [t_i,\ t_i + ex\_time_i^{S(i)}]$. As it is going to remain without changes until the end of the process, it is possible to calculate the matrix $\Sigma = \sigma_{i,j}$, $\forall i,j \in N$. Basically, it represents the *actual* overlapping degree for every pair of nodes, that will be used to compute the overall system cost. If the scheduling process has been accurately performed, the relative positions of the nodes will lead to a maximum sharing factor, for the minimum allowed execution time.

The basis of this operation lies in sharing among various nodes some functional units implemented in hardware. Thus, the total number of them is reduced, and so is the system cost. As it was said before, the sharing degree between two nodes will be higher when:

a) their similarity, κ, is higher, as the possible number of common functional units will increase.

b) their overlapping factor is lower, as the parallelism degree will decrease, allowing to share more units.

Therefore, given two nodes, i and j, and the mentioned parameters, it is possible to calculate the sharing degree of the node j over the node i, $\rho_{j,i}$.

A simple observation of the previous conditions makes that degree be

$$\rho_{j,i} = (1 - \sigma_{i,j}) \cdot \kappa_{[j,S(j)],[i,S(i)]}$$

It is important to notice the subindex order. In the case of the overlapping degree, it is the projection of i over j, rather than the opposite, the one that should be taken.

Therefore, the total cost of both nodes will be

$$c = cost_i^{S(i)} + (1 - \rho_{j,i}) \cdot cost_j^{S(j)}$$

As it was expected, the resulting cost is lower than the sum of the individual costs. Now, a cost estimation technique for more than two nodes has to be proposed. The idea is the same that the one explained before.

When the cost of adding a node i to the hardware partition has to be calculated, all the sharing degrees with the rest of the nodes are found out, $\{\rho_{i,j}\}$, $\forall j \in N,\ j \neq i$.

K	2	4	5	6
2	-	0.80	0.35	0.30
4	0.95	-	0.80	0.30
5	0.45	0.55	-	0.60
6	0.65	0.20	0.45	-

Σ	2	4	5	6
2	-	0	0	0
4	0	-	0	0
5	0	0	-	0.80
6	0	0	0.67	-

ρ^{act}	2	4	5	6
2	-	0	0	0
4	0.95	-	0	0
5	0.45	0.30	-	0
6	0.50	0.07	0.04	-

$$cost\_actual_2^{HW_2} = 15$$
$$cost\_actual_4^{HW_1} = 4$$
$$cost\_actual_5^{HW_3} = 33$$
$$cost\_actual_6^{HW_1} = 38$$

***Overall cost* = 15 + 4 + 33 + 38 =90**

Figure 8
Cost model results

As it is obvious, no sharing factor can be higher than 1, which would correspond to the case in which all the needed functionalities for a particular node are already implemented and can be shared by it to perform its execution. Therefore, it is not suitable to add all the sharing factors, $\{\rho_{i,j}\}$, to obtain the general one, as the result would probably be higher than 1. So, each $\rho_{i,j}$ has to be weighted by a factor, proportional to the part of the node that still remains without implementation, to finally obtain the actual sharing degree, $\rho_{i,j}^{act}$

Let *i* be the node whose sharing factors are going to be calculated, and let {α, β, γ, ..., ψ, ω}, be the set of nodes implemented in hardware, from which some units are going to be shared.

$$\rho_{i,\alpha}^{act} = \rho_{i,\alpha}$$
$$\rho_{i,\beta}^{act} = (1 - \rho_{i,\alpha}^{act}) \cdot \rho_{i,\beta}$$
$$\rho_{i,\gamma}^{act} = (1 - \rho_{i,\alpha}^{act} - \rho_{i,\beta}^{act}) \cdot \rho_{i,\gamma}$$
$$\ldots$$
$$\rho_{i,\omega}^{act} = (1 - \rho_{i,\alpha}^{act} - \rho_{i,\beta}^{act} - \ldots - \rho_{i,\psi}^{act}) \cdot \rho_{i,\omega}$$

Then, the overall sharing factor of *i*, ρ_i^{tot}, will be

$$\rho_i^{tot} = \rho_{i,\alpha}^{act} + \rho_{i,\beta}^{act} + \rho_{i,\gamma}^{act} + \ldots + \rho_{i,\omega}^{act}$$

Now, every factor is comprised in the interval [0,1]. In short, the previous calculation is formulated:

$$\begin{cases} \rho_{i,\alpha}^{act} = \rho_{i,\alpha} \\ \rho_{i,j}^{act} = (1 - \sum_{r=\alpha}^{j-1} \rho_{i,r}^{act}) \cdot \rho_{i,j}\ , \quad \text{if } j \neq \alpha \end{cases}$$

$$\rho_i^{tot} = \sum_{j=\alpha}^{\omega} \rho_{i,j}^{act}$$

Therefore, the real cost associated to *i* would be

$$cost\_actual_i^{S(i)} = (1 - \rho_i^{tot}) \cdot cost_i^{S(i)}$$

Apparently, with these considerations the cost model would be complete, but there is still a slight detail to think about. Let assume a node *j* that shares part of the functionalities of another one, *i*. This fact produces an undesirable side effect, that is necessary to eliminate from the model. The utilization of this set of functional units makes impossible to every node *k*, whose execution has a certain parallelism degree with *j*, to use them, as a structural conflict would arise. This leads to a decrease of the sharing capability of *k* respect to *i*, that has to be taken into account for later calculations. So, if *j* shares $\rho_{j,i}^{act}$ from *i*, let us define the occupation factor of *i*'s functionalities during the execution of *j*,

$$\omega_i^j = \rho_{j,i}^{act} \cdot \frac{\kappa_{[i,S(i)],[j,S(j)]}}{\kappa_{[j,S(j)],[i,S(i)]}}$$

The quotient of both κ is associated to the relative cost of both nodes. Thanks to that, ω_i^j is a proportional factor to *i* and not to *j*, as $\rho_{j,i}^{act}$ is.

Now, for every node *k* whose execution has a certain parallelism with *j*, the following correction has to be applied to their sharing factors:

$$\kappa_{[k,S(k)],[i,S(i)]}^{act} := \kappa_{[k,S(k)],[i,S(i)]} \cdot (1 - \omega_i^j \cdot \sigma_{k,j})$$

The results of the cost estimation process for the previous example are shown in Figure 8.

Since to calculate the cost of a node it is necessary to examine the rest of them already implemented in hardware, the complexity of this algorithm is O(1 + 2 + ... + (n-1) + n) = O(n·(n+1)/2) = O(n^2). As the number of nodes executed in parallel with another one is much lower than the overall number of nodes, *n*, the complexity of modifying κ is not relevant respect to the previous one.

6.- Experimental work.

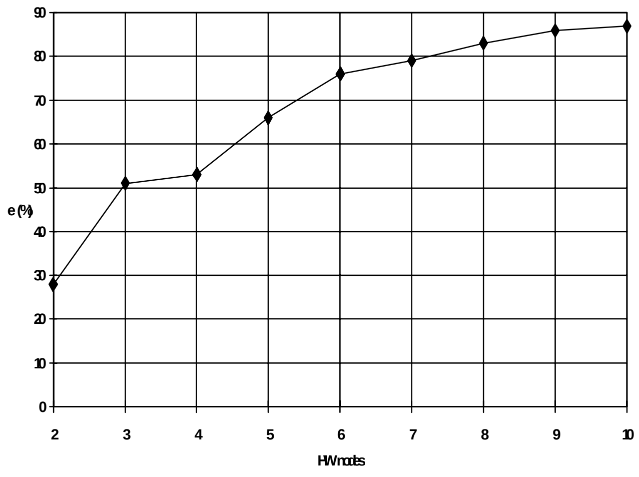
Once the time and cost models are presented, we offer some results obtained by performing the estimation process on several Codesign graphs. The advantages attained by the new model are compared with the time and cost parameters achieved with the standard estimation technique, ignoring the parallelism and sharing factors.

The experiments have been carried out on several sets of graphs, with a number of total nodes between 5 and 20, and a number of hardware nodes between 2 and 10. Each set contains 5 different graphs, which gives an overall number of 80 results.

Every graph has been created following the intrinsic Codesign characteristics, and so, the obtained results are relevant. The sharing parameters among nodes, κ, have been generated by means of a random process, which makes the experiment have a wide value range.

The results, offered in Figure 9, are listed by the number of hardware nodes. For every entry, the average cost without sharing, *c*, with sharing, c_s, and the improvement percentage, $100 \cdot (c - c_s) / c$, are depicted. The latter factor corresponds to the relative error, *e*, produced if the first set of parameters is used rather than the second. Besides, a graphical representation of the results is shown. It is clearly seen that, as the number of hardware nodes is increased, the difference between both models, with and without sharing, becomes greater. This is a reasonable conclusion, since once there is a certain number of functionalities in hardware, the necessary cost to implement a new node is minimal because of the functional unit sharing.

This justifies the use of the time and cost model allowing parallelism and sharing, rather than the classical and inexact estimation approach.



HW nodes	c	c_s	e
2	80	58	27.5%
3	190	92	51.6%
4	305	143	53.1%
5	315	104	66.9%
6	415	99	76.1%
7	490	100	79.6%
8	560	90	83.9%
9	670	94	85.9%
10	740	95	87.2%

Figure 9
Experimental results

7.- Conclusions and future work.

In this paper, a new time and cost model for the Codesign estimations has been proposed. The main novel features introduced are:

- Possibility of working with any degree of parallelism among tasks.
- Multiple hardware implementation for every node.
- Possibility of sharing hardware units among nodes.

A mathematical approach has been offered, explaining with detail the three stages comprised into this technique: *time calculation*, *node scheduling* and *cost computation*.

Respect to the future work, the main lines that will be followed are:

- To apply the present model to the Codesign partitioning process, studying the repercussion on the iterative algorithms. More specifically, the *Fiduccia-Mattheyses* algorithm [9] has been used in our previous approaches to partitioning [10]. From this research, an improvement in the results and a reduction of the time consumed by the process are expected.
- To compare the results given by the model with real measures provided by our High Level Synthesis tool [11] and the *Synopsys Behavioral Compiler*. In this way, a study of the *K* matrix calculation can be made in order to improve this task and get the maximum level of reliability.

8.- References.

[1] W. Wolf, "Hardware-Software Co-Design of Embedded Systems", Proceedings of the IEEE, vol. 82, nº 7, July 1994.

[2] C. J. Alpert, A. B. Kahng, "Recent Directions in Netlist Partitioning: a Survey", Integration, the VLSI journal 19, 1995.

[3] M. Theiβinger et al., "Castle: An Interactive Environment for HW-SW Co-Design", 3rd Intl. Workshop on Hardware/Software Codesign, September'94.

[4] T.-B. Ismail, A. Jerraya, "Synthesis Steps and Design Models for Codesign", Computer Magazine, February 1995.

[5] D. Herrmann et al., "An Approach to the Adaptation of Estimated Cost Parameters in the COSYMA System", 3rd Intl. Workshop on Hardware/Software Codesign, September'94.

[6] R. Niemann, P. Marwedel, "Hardware/Software Partitioning using Integer Programming", ED&TC'96.

[7] F.J. Kurdahi, "Evaluating Layout Area Tradeoffs for High Level Applications", IEEE Trans. On VLSI Systems, vol. 1, #1, 1993.

[8] H. Mecha et al., "A Method for Area Estimation of Data-Path in High Level Synthesis", IEEE Trans. On CAD, vol. 15, #2, 1996.

[9] C.M. Fiduccia, R.M. Mattheyses, "A Linear-time Heuristic for Improving Network Partitions", DAC'82.

[10] J.A. Maestro et al., "Una Técnica de Particionamiento para Reducir la Sobrecarga por Comunicaciones en Codiseño", DCIS'96.

[11] J. Septién et al., "FIIDIAS: an Integral Approach to High Level Synthesis", IEE Proceedings Circuits, Devices and Systems, vol. 142, #4, 1995.