# A New Approach to Execution Time Estimations in a Hardware/Software Codesign Environment

JAVIER RESANO, ELENA PEREZ, DANIEL MOZOS, HORTENSIA MECHA, JULIO SEPTIÉN
Departamento de Arquitectura de Computadores y Automática
Universidad Complutense de Madrid
Ciudad Universitaria CP. 28040
SPAIN

*Abstract:* - This study addresses a new approach to the Hardware/Software partitioning problem focused on the integration of communication scheduling during design space exploration. Frequently in this phase most communication channel features are ignored, since communications are identified with abstract channels. It is conceivable that once the abstract channels have been mapped into an actual communication channel (i.e. a bus) the area and execution time of the solution may change considerably. A novel methodology, that schedules the communications taking into account the physical features of the communication channel, is presented. It has been integrated into a partitioning tool based on Genetic Algorithms. This methodology attempts to accomplish a deep study of the communication impact during the design space exploration without increasing significantly the complexity of the algorithm.

*Key-Words:* - Hardware/Software codesign, Computer-Aided Design, partitioning, communications scheduling, time estimations, FPGAs.

## 1 Introduction and Previous Work

Communications are often a system's performance bottleneck, even more when both hardware and software performance are improving much faster than communication channels do.

Most of the work on Hardware/Software (HW/SW) partitioning considers that abstract channels carry out communications between tasks in different platforms. Once the partitioning has been done, the abstract channels have to be mapped into one or more physical channels. This process is called communication synthesis. For instance in [1] a methodology is presented that, once the partitioning has been finished, tries to choose the best bus bandwidth for mapping the abstract channels. In [2] more complex interconnection topologies are allowed. An algorithm is presented that, starting from a set of channels, (one for every communication), tries to cluster processes in order to share a communication channel. Some features like bit widths, number of ports of the HW resources, probability of access collision, cost of the arbitration logic, are taken into account, attempting to minimize both the delay owing to the access conflicts to busses, and the system cost. Both approaches work with abstract communication channels, (although they consider some physical features) during the communication synthesis process. Once obtained, the abstract interconnection topology generated should be mapped into an actual one. In [3] a useful tool for this mapping is presented, that generates automatically the drivers and a DMA controller for a given communication application. In [4] a design methodology is proposed that separates the communications from the system behavior. This idea proves to be useful for verification and simulation at different abstraction levels, however may be deceitful for automatic system synthesis. Another interesting work on communication synthesis can be found in [5].

All of these approaches to the HW/SW communication problem use abstract channels while they are exploring the design space so physical communication features are ignored. Nevertheless ignoring the communication channels properties may lead to an inefficient design space exploration, since it is not possible to estimate correctly the communication time, or the impact of the access conflicts upon the communication channels.

In [6] simple models for some communication platforms are presented (USB, PCI buses, packing, burst transmission mode). A model is also given for estimating the area needed for the communication drivers. Moreover, these models are integrated within the HW/SW partitioning, so communication synthesis timing and area trade-offs are studied during the design space exploration. However communications are not scheduled during partitioning, so access collisions to the system bus

are not taking into account.
The approach proposed in this paper aims to:.

a) Consider the physical features of the channel in order to accomplish accurate time estimations.

b) Study the possibility of conflicts upon the communication channel

c) Schedule the communications trying to minimize the global execution time.

d) Integrate all these features into a HW/SW partitioning tool without increasing significantly the time needed for the design space search.

## 2 Initial Specification

The initial specification is modelled by an acyclic graph, where each node represents a computational task, and the edges correspond to dependencies between the nodes. Three different dependencies are considered: communication dependencies, internal dependencies, and temporal dependencies. A communication dependency edge (CDE) connects two nodes of different partitions. It represents a data transfer between nodes that will be carried out upon a communication channel. An internal dependency edge (IDE) connects two nodes in the same partition. It also represents a data transfer between the nodes, but in this case there is no need for using a communication channel. A temporal dependency edge (TDE) represents a dependency between two nodes in the same partition that has been imposed by the scheduler.
Each node of the graph contains estimations for its execution time and HW area (if needed).
Each CDE is tagged with the amount of data to be transferred, and an estimation of the communication time needed. A specification graph example is shown in figure 1 (HW area is not included for readability).

## 3 Cost Function Choice

The cost function of a codesign system includes generally the area and the execution time of the solution. One of the more difficult topics for designing a partitioning system is to find how to mix rather different magnitudes into a cost function that should be able to lead the design space exploration in an optimum fashion.
This work has been developed for a HW/SW system where the HW partition is assigned to a FPGA. Since a FPGA has constant area a straightforward cost function may be obtained if the area is removed, so the cost function can be identified with the execution time. The area is now considered as a constraint that solutions should meet. The constraint must take into account that only the 96% of the FPGA area should be used, since routings problems may occur when more area is used.
Once the area has been removed and the cost function has been directly identified with the execution time, the design space exploration will try to find the fastest solution that meets the area constraint.

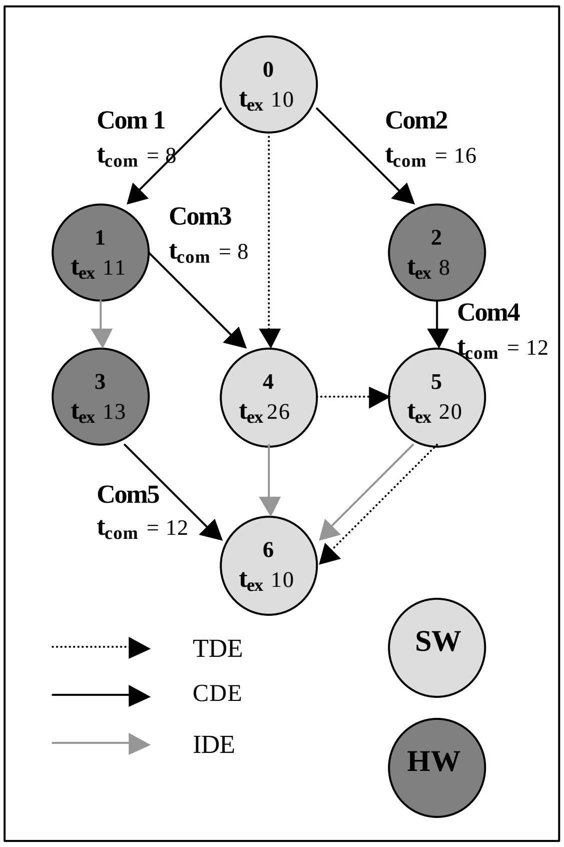


**Fig. 1** Specification graph example.

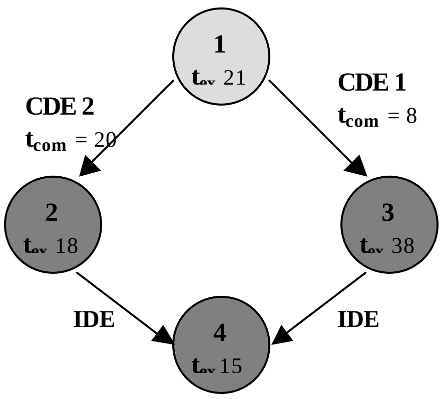
## 4 Execution Time Estimation

Since the execution time leads the design space exploration, an accurate estimation is needed.
The steps followed for estimating the execution time of a solution once a HW/SW partitioning is done are:

**A)** *Assign a weight to each node.*

**B)** *Choose the execution order for the SW assigned nodes.*
**C)** *Recalculate the weights taking into account the new dependencies.*
***D)*** *Schedule those nodes that are not waiting for a communication.*
**E)** *While there is a communication waiting for execution do:*
- **E1)** *Choose one communication and schedule it.*
- **E2)** *Schedule those nodes that are not waiting for a communication*

**Step A:** ***Assign a weight to each node***
In the first step a weight is assigned to each node of the partitioned graph. The node's weight will be used for taking decisions that will try to minimize the global execution time. This will be done by giving priority to the nodes with heavier weights.
For instance, if there is a node that should execute several communications, the order in which these communications are going to be executed should be decided. Most of the time estimation algorithms in literature take these decisions according to local considerations. In [5] the longest communication is scheduled first. Another local considerations for the same question are scheduling first the communication which destination node has more successors, or the shortest one since parallel execution might be improved.



| | Node1 | | Node 2 | | Node 3 | | Node 4 | |
|---|---|---|---|---|---|---|---|---|
| | L | G | L | G | L | G | L | G |
| **G.Weight** | | 82 | | 33 | | 53 | | 15 |
| **$T_{start}$** | 0 | 0 | 41 | 49 | 49 | 29 | 87 | 67 |
| **$T_{end}$** | 21 | 21 | 59 | 67 | 87 | 67 | 102 | 82 |

**Fig. 2** Global weight example

Since local considerations may lead to inefficient decisions in our algorithm, a global consideration has been selected for weighting the nodes. The weight chosen is the maximum distance of a node to the end of the execution in the initial graph. This distance is calculated by carrying out an ALAP scheduling. This scheduling should take into account the HW/SW communication times.
Figure 2 shows an example where both local and global weights are used. The local weight used is the one proposed in [6], i. e. the longest communication is scheduled first.
In this example node 1 must carry out two communications upon the same communication channel. So both communications should be scheduled. If the decision is taken based on the local weight, Com2 (CDE2) would be scheduled first, so Com1 should wait until Com2 ends. Since Com1 is in the critical path of this graph, delaying it affects negatively the system performance. Nevertheless, if the global weight described above is used, nodes in the critical path will be more weighted so Com1 will be scheduled first and no unnecessary delays will be introduced.

**Step B:** ***Choose the execution order for the SW assigned nodes***
The choice of the SW execution order is the next decision that should be taken to estimate the execution time. The specification graph allows parallel execution between their nodes, but those nodes assigned to SW should be executed sequentially.
The order is selected sorting the nodes by their weights, so the heaviest node will be executed first. Then the new dependencies are added as CTRE to the specification graph as it is showed in figure 1.
It is easy to prove that this SW execution order does not allow the new dependencies to create cycles in the graph.

**Step C:** ***Recalculate the weights***
In order to be absolutely strict, the weights should be recalculated each time a decision is taken since this decision might create a new dependence that could change the critical path of the graph. However, since the aim of the algorithm is improving the precision of the estimations without increasing significantly the complexity, the weight is only recalculated once. New weights will be assigned to each node after selecting the SW execution order. These weights will be calc ulated in the same way that in step A, but considering the dependencies derived from the sequential execution of the SW partition. Table 2 shows how SW execution dependencies change the nodes weight of the graph in figure 1.

| *Node* | **0** | **1** | **2** | **3** | **4** | **5** | **6** |
|---|---|---|---|---|---|---|---|
| *Weight A)* | **76** | **55** | 50 | 35 | **36** | 30 | 10 |
| *Weight C)* | **93** | **75** | 50 | 35 | **56** | 30 | 10 |

**Table 2**. Comparison between weights in steps A) and C) for the graph in figure 1.

### Steps D y E: Scheduling

A heuristic has been developed for the scheduling process trying to minimize the global execution time. The heuristic decides when each node and each communication is going to be executed, assigning to it a $\mathbf{t_{start}}$ and a $\mathbf{t_{end}}$ times. The aim of this process is to detect when communication channel access conflicts happen, so the execution time estimation will also consider the delays caused by these conflicts. This heuristic starts after the SW execution has been sorted, and the weights have been recalculated.

The heuristic addressed in this paper has been developed for a platform with just one communication channel. This channel should be modelled at first in order to obtain accurate estimations of the execution time. The algorithm can be easily extended for systems with more than one communication channel if the number of channels is prefixed.

The scheduling starts assigning $\mathbf{t_{start}} = 0$, and $\mathbf{t_{end}}=\mathbf{t_{ex}}$ to the first node, where $\mathbf{t_{ex}}$ is its execution time in the partition where it has been assigned to. Then the algorithm continues scheduling the successors of the first node. A greedy policy is followed to schedule nodes while there is no need for HW/SW communications. When a scheduled node requests a HW/SW communication with another node this request is stored in a list. All requests are tagged with the time when the sender node demanded the communication.

Once all the nodes without need for HW/SW communication have been scheduled, one of the requested communications is selected and scheduled.

There are two criteria for selecting the communication from the requesting list (E1):

- If at a given time **t** the communication channel is not carrying out any communication and there is just one request that has been made before **t** the communication channel is assigned to this request, so no other communication may use it until this one finishes.
- Otherwise if there are more than one request, the one with the greater weight will be selected. The weight of a communication will be computed as the weight of the destination node plus the time needed to execute the communication.

Once the selected communication has been scheduled the graph is examined (E2) and all the nodes that are able to start their execution without waiting for another HW/SW communication are also scheduled. The process is repeated until all the communications are scheduled.

Figure 3, shows in detail how the example graph in figure 1 is scheduled.

The scheduling heuristic has a quadratic complexity.

**Step D)** Node 0 is scheduled (0,10). **Com1** and **Com2** are requested at t= 10.

**Step E) first iteration:**
- **E1) Com1** is schedule (10,18) since Com1 weight > Com2 weight.
- **E2) Node 1** is scheduled (18,29). **Com3** is requested at t=29. **Node 3** is scheduled (29,42). **Com5** is requested at t= 42.

**Step E) second iteration:**
- **E1) Com2** is scheduled (18,34) since at t=18 there are no more communications requested.
- **E2) Node 2** is scheduled (34, 42). **Com4** is requested in t=42.

**Step E) third iteration:**
- **E1**) **Com3** is scheduled (34,42) since at t=34 there are no more communications requested.
- **E2) Node 4** is scheduled (42,68).

**Step E) fourth iteration:**
- **E1) Com4** is scheduled (42, 54) since Com4 weight>Com5 weight.
- **E2) Node 5** is scheduled (68,88).

**Step E) fifth iteration:**
- **E1) Com5** is schedule (54,66) since at t=54 there are no more communications requested.
- **E2) Node 6** is scheduled (88,98), so $\mathbf{T_{ex}=98}$.

**Fig. 3** Scheduling execution example corresponding to the graph in figure 1. (T1,T2) represents starting and ending execution times.

## 5 Area Estimation

The design area is estimated as follow:

(1) $\mathbf{Area = S^{n}{}_{i}\, A_i + A_{driver} + A_{control} + A_{storage}}$

$\mathbf{A_i}$ is the area of the node i. $\mathbf{A_{driver}}$ is the area needed to implement the communication driver. $A_i$ and

$A_{driver}$ should be estimated from a core library. When a new core is added to the library its area can be estimated using an automatic synthesis tool. $\mathbf{A_{control}}$ is the area needed for the control logic that schedules the communications. In this approach the scheduling control is assumed by a state machine, so the area requested is estimated as a function of the number of communications. $\mathbf{A_{storage}}$ is the area needed for storing the data to transfer until a communication is executed. This storage space is computed during the communication scheduling. During this process a record keeps the maximum storage space required.

The SW area is not taken into account since it is supposed that the controller has enough instructions and data memory for all the allocated nodes. If it is possible to exceed the data or the instruction memory, the SW area will be included as a second constraint that every solution should meet.

## 6 Design Space Exploration

The scheduling algorithm has been integrated into a partitioning tool based on genetic algorithms (GA). GAs have been used before for partitioning problems, e.g. in [8] the problem of partitioning a HW specification between a set of FPGAs is addressed.

Our tool creates a random initial population of valid solutions. A solution is said to be valid if meets the area constraint. Invalid solutions are rejected in order to save computing time.

The solutions consist on a HW/SW partition and a communication channel. All the communications must have time estimations for each possible communication channel.

During the design space exploration solutions evolve by reproducing themselves, generating new solution's offspring. Then worst solutions are deleted in order to keep the population constant. Reproduction is carry out by the crossover and the mutation operators. If invalid solutions are created, these solutions are discarded.

Rejecting invalid solutions improves the performance of the partitioning tool, since a quick evaluation of the area of a solution just involves an add operation, so all the time needed for the scheduling is saved. Furthermore, since a genetic algorithm is able to reach all the design space points through crossing and mutating the initial solutions, there is no danger that such decision will make the algorithm to fall into local minima, as it may happen if the exploration is performed with a neighbourhood algorithm.

Experiments have been made comparing the execution times of the partitioning tool with a genetic algorithm that rejects invalid solutions and with a neighborhood algorithm (simulated annealing), showing up that in this case the genetic algorithm obtains better solutions in less computational time.

## 7 Experimental Results

The following experiment illustrates the impact of communication access conflicts over the global execution time.

The experiment has been performed using an application that implements the Hough Transform [9] over a matrix of pixels in order to find some simple geometric figures like a circle or a rectangle. Communications in this design are a critical factor since nodes have to interchange large data structures.

The initial specification of the design contains fifteen nodes. Each node has a source code written down both in C and VHDL.

The target platform for this design is a XILINX 4010 board that contains a FPGA with 400 CLBs, (i. e. 20.000 logic gates), an 8051microcontroller, and a RAM memory accessible to both of them.

The first step of the experiment has been to design a communication protocol between the FPGA and the microcontroller, and to implement the communication driver. Once the protocol was fixed, accurate time estimations were done for each possible HW/SW communication.

The area of each node, and the execution time in HW, were estimated with XILINX Foundation implementation and verification tools. For the SW execution-time estimation, it was used an automatic assembler 8051/C translator.

After completing the initial specification graph with all the needed estimations, it was passed to the partitioning tool. In order to obtain different measures the experiment was repeated with several area constraints (The full design implemented in HW used needs 384 CLBs).

In order to explain our approach the design space has been searched running the partitioning tool in two different ways: first ignoring access conflicts upon the communication channel, and then scheduling the communications. Figure 4 shows the best solutions found in both cases. The first column corresponds to the best execution time found by the partitioning tool without scheduling the communications. The second column shows the execution time of this solution once access conflicts are considered. The third column shows the best solution found when the partitioning tool schedules the communications.

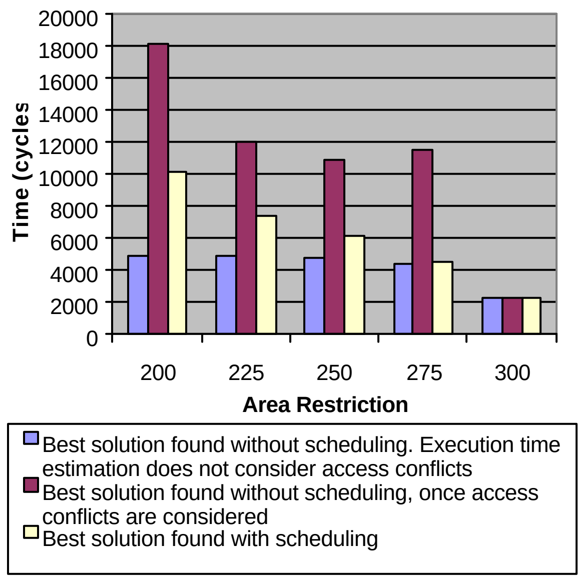


**Fig. 4** Result comparison

The results that emerge from this experiment confirm how communication access conflicts become a critical performance feature. It is remarkable how they increase execution time even up to three times, so ignoring them can make difficult to distinguish the goodness of a solution. It is not surprising that better solutions will be obtained once access conflicts are taken into account during the design space exploration. In this experiment up to a 2.5 speed-up has been achieved (for Area restriction =300 there is no improvement since the best solution found does not present any access conflict).

One of the main objectives of this work was trying to accomplish accurate time estimations during the design space exploration without increasing significantly the exploration time.

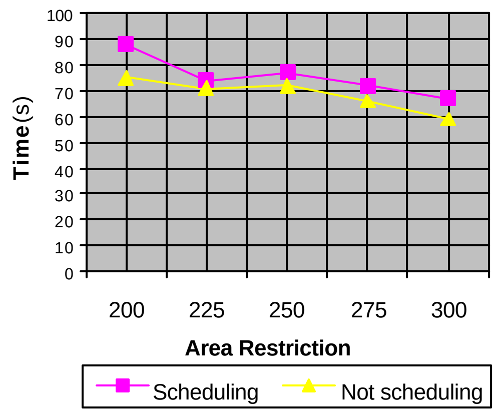


**Fig. 5** Desing space exploration time

Figure 5 compares the time needed for the partitioning tool in both approaches. The time measurement shows that the increment of computational time due to communication scheduler goes from 4% to 17%. These results appear to confirm that it is possible to accomplish more accurate time estimation without incrementing significantly the computational time.

## 8 Conclusion

The physical features of communication channels use to be one of the key factors of a codesign system performance. Taking into account these features are necessary for accurate communication time estimations. Moreover, even when accurate estimations for each communication have been done, access conflicts overhead must be consider, since the global execution time can increase significantly due to these conflicts.

Experiments have shown that, scheduling the communications with the heuristic presented in this paper allows the partitioning tool to perform more accurate time estimations, without increasing significantly the computational time needed for the design space search.

*References:*

[1] S. Narayan and D. Gajski. "*Syntesis of System Level Bus Interfaces*", Proc. of European Design&Test Conference 94, pp. 395-399, Feb. 1994.

[2] M. Gasteier, M. Munich, M. Glesner. "*Generation of Interconnect Topologies for Comunication Synthesis*", DATE'98, pp. 36-42, Feb. 1998.

[3] M. O'Nils, A. Jantsch. "*Device Driver and DMA Controller Synthesis from HW/SW Comunication Protocol Specifications*", Design Automation for Embedded Systems, 6, pp. 177-205. 2001.

[4] J.A. Rowson and A. Sangiovanni-Vicentelli. "*Interface-Based Design*", DAC'97, pp. 178-183, 1997.

[5] R. Ortega, G. Borriello "*Communication Synthesis for embedded Systems with Global considerations*" Proc. CODES/CACHE'97, pp. 69—73, March, 1997.

[6] P. V. Knudsen and J. Madsen. "*Integrating Communication Protocol Selection with Partitioning in HW/SW*" Trans. on CAD, pp. 1077-1095. 1999.

[7] . G. Gogniat, M Auguin, L. Bianco, A. Pegatoquet. "*Communication synthesis and HW/SW integration for Embedded System Design*", CODES/CASHE'98. pp. 49-53. March 1998.

[8] J.I. Hidalgo, J. Lanchares, R. Hermida. "*Partitioning and Placement for Multi-FPGA systems using Genetic algorithms*". Proc. of the 26th EUROMICRO conference, IEEE Press. pp. 204-5-7 2000

[9] F. Thomson Leighton. "*Introduction to Parallel Algorithms and Architectures*", pp. 210-213. Morgan Kaufmann Publishers 1992.