PROTOCOL SELECTION AND INTERFACE GENERATION FOR HW-SW CODESIGN

Jean-Marc Daveau, Gilberto Fernandes Marchioro
Tarek Ben-Ismail and Ahmed Amine Jerraya
TIMA/INPG Laboratory, System-Level Synthesis Group.
46 Av. Felix viallet. 38031 Grenoble, France.
E-mail : daveau@verdon.imag.fr

Abstract— **The aim of this paper is to present a communication synthesis approach stated as an allocation problem. In the proposed approach, communication synthesis allows to transform a system composed of processes that communicate via high level primitives through abstract channels into a set of processes executed by interconnected processors that communicate via signals and share communication control. The proposed communication synthesis approach deals with both protocol selection and interface generation and is based on binding/allocation of communication units. This approach allows a wide design space exploration through automatic selection of communication protocols. We present a new algorithm that performs binding/allocation of communication units. This algorithm makes use of a cost function to evaluate different allocation alternatives. We illustrate through an example the usefulness of the algorithm for allocating automatically different protocols within the same application system.**

Keywords— **Hardware/software codesign, Communication synthesis, Protocol selection/allocation, Interface generation**

I. INTRODUCTION

Recently the synthesis community has moved toward the highest level of abstraction commonly known as the system level [4] [9] [15] [16] [20] [32]. This move vas motivated by the increasing complexity of systems and by the need for a unified approach to allow the development of systems containing both hardware and software.

As the level of abstraction rise some problems heretofore non existing appear [12] [38]. At the system level, some of the main concepts are behaviour and communication [25]. These two concepts have brought new problems konwn as partitionning and communication synthesis. The goal of partitioning is to distribute a system functionality over a set of subsystems where each subsystem is to be executed either in software or in hardware processors [33]. The problem of communication synthesis [2], which appears after system level partitioning, is to fix the protocols and interfaces needed by the different susbsystems for the communication.

A. Objective

When designing distributed embedded systems, communication synthesis becomes essential as different subsystems inevitably need to communicate. Different communication schemes and protocols may be needed in embedded systems as well as different interconnection topologies. Communication topologies and protocol greatly influence the overall system performances and may lead to infeasible design if the designer underestimate communication load. Decision based on the average load only tend to forget peak load or communication delay due to bus sharing, which may degrade the system performances. Therefore a large design space exploration have to be explored to find a feasible solution. In this paper we describe a paradigm that allows a wide range of communication schemes to be modelled in a synthesis oriented approach. The main objective for our communication synthesis method are :

- to be able to choose between differents communication schemes.
- to be able to model the system behaviour independantly of the communication. System specification should be independant of the communication specification in order to allow changes in the communication scheme without any changes in the system specification.
- to be able to reuse existing communication models through a library.
- to have an automatic communication synthesis method based on a cost function and some constraints.

This paper introduces a new approach for communication synthesis. This task is formulated as an allocation problem aimed at selecting, from a library, a set of communication units that implement the data exchange between the subsystems.

B. Previous work

Most of the work in communication synthesis for codesign has focussed on interface synthesis assuming a fixed network structure [9] [15]. Only few works in codesign handle network synthesis [7] [13] [37]. In [13] Gong's network synthesis is guided by the mapping of variables (shared or private) to memory (local or global). In [37], Yen create a new processing element and a bus when it is not possible to assign a process to an already existing processing element or a communication on a bus without violating real time constraints. In [5] [30], Chou and Srivastava use a set of predefined interconnection models during communication synthesis. Several works on protocol selection are reported in the software synthesis for distributed systems [29]. Lots of previous work has focused on interface synthesis [6] [8] [21] [22] [26] [27] [28] and [35]. In [6], Ecker presents a method for transforming and optimising protocols. In [26], Narayan addresses the problem of bus interface generation between two different hardware mod-

ules of a partitioned specification. The focus is to optimise the bus utilisation by interleaving different point to point communications on it. As described in [21] [27], Lin and Nayaran consider the problem of interface synthesis with automatic protocol conversion with one or both sides having a fixed interface. Madsen interface synthesis approach [22] consider the problem of interface adaptation between a fixed interface and a communication medium chosen during partitioning. A state based model that describes both functional and timing properties of an interface is detailed by Ravn in [28]. Another model using extended signal transition graph allowing the specification of complex synchronous/asynchronous interface is proposed by Vanbekbergen in [35]. Approaches where communication is done through shared memory are detailed in [17] [14] and [5]. In [17] the problem of interface between a memory and a coprocessor or I/O processor is addressed. In [14] Gupta also address the problem of communication between a processor (software) and a coprocessor (hardware). In that approach the communication may be done through memory or through a direct bus between the processor and an ASIC. Different communication models (blocking, non blocking) are available. These approaches mainly address the hardware/software interface. In [30] Srivastava starts after partitioning with a process graph, an architecture template and map the communication on the physically available communication resources. Only one communication model is supported, the single reader single writer fifo. When the available communication resources does not directly support the fifo protocol it is emulated. This work mainly address the field of real time distributed heterogeneous systems.

To our knowledge none of the existing work tackle communication synthesis as an allocation problem. The main contribution of this paper is to present communication synthesis as an allocation problem.

Compared to classical communication synthesis approach the main advantages of our approach are :

- wide design space exploration through automatic selection of communication protocols.
- formulation as an allocation problem which allows numerous algorithm to solve it.
- complete communication synthesis approach :
  - network synthesis and protocol selection.
  - interface synthesis.
- component reuse through library.

The limitations of our approach are :

- the need for a library of communication that must be provided by the user. It is not possible to use a protocol that is not described in the library.
- The need for a realistic cost function for the algorithm and the need for communication estimator [37] [34] to lead network synthesis and protocol selection.

In the following sections we present our proposed communication synthesis method. The next section introduces the communication model. Section III introduces the concept of communication unit. The communication synthesis problem is detailed in section IV. Section V describes an algorithm for communication unit allocation. Finally, we will present the application of the communication synthesis method on an example before concluding the paper.

II. COMMUNICATION MODEL

In this paper we will use the communication modelling strategy described in [19]. At the system level, a system is

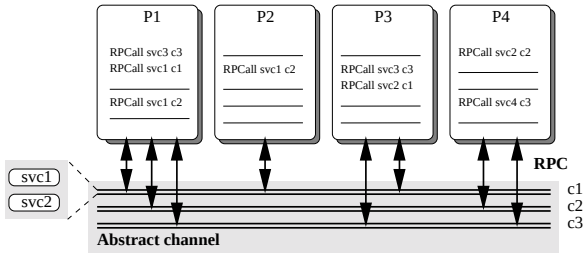

Fig. 1. Processes communicating through abstract channels

represented by a set of processes communicating through abstract channels (figure 1). An abstract channel is an entity able to execute a communication scheme invoked through a procedure call mechanism. These abstract channels offer high level communication primitives (services) that are used by the processes to communicate. Access to a channel is controlled by a fixed set of primitives and relies on remote procedures call [1], [3] of these communication primitives. A process that is willing to communicate through a channel makes a remote procedure call to a communication primitive (*send, receive*) of that channel. Once the remote procedure call is done the communication is executed independently of the calling process by the channel unit. The communication primitives are transparent to the calling processes and are the only visible part of a channel unit. This allows processes to communicate by means of high level communication schemes. There is no predefined set of communication primitives, they are defined as standard procedures and are attached to the abstract network. Each application may have different set of communication primitives (*send_int, send_short, send_atm, etc*). This model allows to hide the implementation details of the communication and separate communication from the rest of the design behaviour. In our approach the detailed I/O structure and protocols are hidden in a library of communication components. Figure 1 shows a conceptual communication over an abstract communication network. The processes communicate through three abstract channels *c1*, *c2* and *c3*. *C1* and *c2* offers services *svc1*, *svc2* and *c3* offers services *svc3*, *svc4*.

III. COMMUNICATION UNIT MODELLING

We define a communication unit as an abstraction of a physical component. Communication units are selected from the library and instantiated during the communication synthesis step.

From a conceptual point of view, the communication unit is an object that can execute one or several communication primitives with a specific protocol. These services can

share some common resources (bus arbiter, buffering memory, buses) provided by the communication unit. The communication unit can include a controller which determines the protocol of the communication. The complexity of the controller may range from a simple handshake to a complex layered protocol. The services interact with the controller which modifies the communication unit state and synchronizes the communication. All accesses to the interface of the communication unit are made through these services. Such services also fix the protocol of exchanging parameters between the processes and the communication unit. The use of services allows to hide the details of the protocol in a library where a service may have different implementations depending on the target architecture (hardware/software).

Communication units differs from abstract channels in the way that they implement a communication with a specific protocol and realisation (hardware/software). An abstract channel just specify the required services for a communication (figure 2). Therefore, several abstract channels

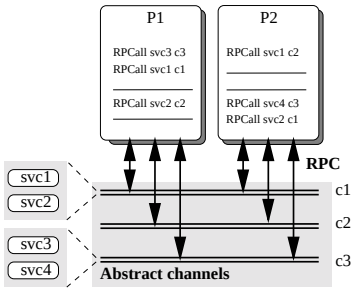

Fig. 2. Specification of communication with abstract channels

may be implemented by a single communication unit if it is able to provide all the required services. This operation is called a *merge* of abstract channels. Figure 3 represent a merge of two abstract channels *c1* and *c3* on a communication unit *cu1*. Communication unit *cu2* implement the communication offered by abstract channel *c2*.

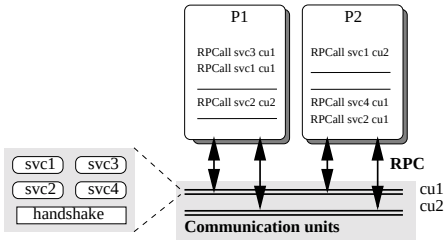

Fig. 3. Merge of abstract channels on a communication unit

This models enable the user to describe a wide range of communication schemes and most system level communication such as message passing or shared memory. Communication abstraction in this manner enables a modular specification, allowing communication to be treated independently from the rest of the design.

IV. COMMUNICATION SYNTHESIS

Communication synthesis aims to transform a system with processes that communicate via high level primitives into a set of interconnected processors that communicate via signals and share communication control. At this level the system is represented as a process graph [38]. The nodes represent the processes and the edges the communication. Communication through abstract channels is based on the remote procedure call of communication primitives (figure 1). Starting from such a specification two steps are needed. The first is aimed to fix the communication network structure and protocols used for data exchange. This step is called protocol selection or communication unit allocation. The second step adapt the interface of the different processes to the selected communication network.

A. Protocol Selection and Communication Unit Allocation

Allocation of communication units starts with a set of processes communicating through abstract channels (figure 1) and a library of communication units (figure 4). These communication units are an abstraction of some

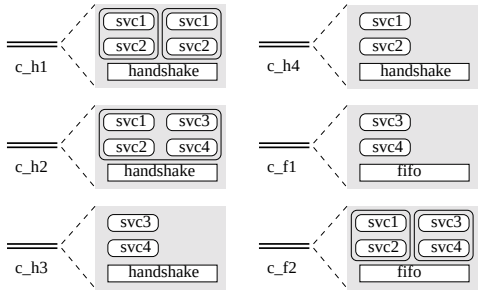

Fig. 4. Library of communication units

physical components. This step chooses the appropriate set of communication units from the library in order to provide the services required by the communicating processes. The communication between the processes may be executed by one of the schemes described in the library. This step fixes the protocol used by each communication primitive by choosing a communication unit with a specific protocol for each abstract channel.

Several abstract channels may be executed by a single communication unit if it is able to handle several independent communications. Merging several abstract channels on a single communication unit allows to share a single communication medium among several abstract communications. The different abstract channels will be time multiplexed over the communication unit. This step also determines the interconnection topology of the processes by fixing the number of communication units and the abstract channels executed on it.

Allocation of communication units to abstract channels is related to the classical speed/area trade off. The choice of a given communication unit will not only depend on the communication to be executed but also on the performances required and the implementation technology of the communicating processes. These features may be packed into a cost function to be reduced by the allocation algorithm. This is similar to the binding/allocation of functional units in classic high-level synthesis tools [11] [18].

Most of the allocation algorithms used in high level synthesis may be used to solve this problem [24].

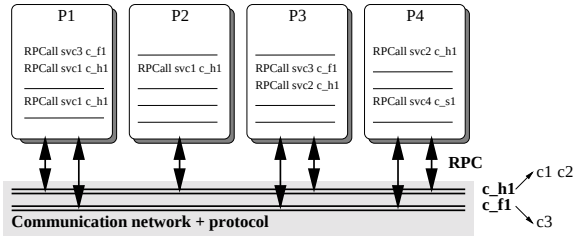

Fig. 5. System after allocation of communication units

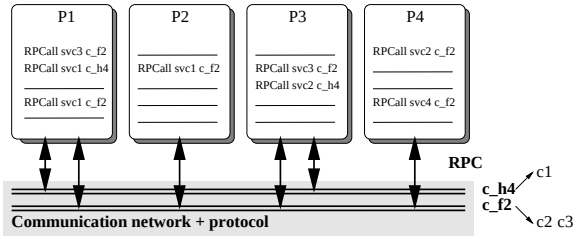

Fig. 6. Communication unit allocation alternative

An example of communication unit allocation for the system of figure 1 is given in figure 5. Starting from the library of communication units of figure 4, the communication unit *c_h1* has been allocated for handling the communication offered by the two abstract channels *c1* and *c2*. Communication unit *c_h1* is able to execute two independent communication requiring services *svc1* and *svc2*. Communication unit *c_f1* has been allocated for abstract channel *c3*. Another solution could have been to merge *c2* and *c3* and allocate *c_f2* for handling that communication. *C_h4* could have been allocated for *c1*. This solution is represented on figure 6.

B. Interface Synthesis

Interface synthesis selects an implementation for each of the communication units from the implementation library (figure 7) and generates the required interfaces for all the processes using the communication units (figure 8). The library may contain several implementations of the same communication unit. Each communication unit is realised by a specific implementation selected from the library with regard to data transfer rates, memory buffering capacity, and the number of control and data lines. The interface of the different processes are adapted according to the implementation selected and interconnected. An approach for determining the width of a bus that will implement a group of channels is presented in [10] and [26] or for interfacing incompatible protocol in [27]. The result of interface synthesis is a set of interconnected processor communicating through signals, buses and possible additional dedicated component selected from the implementations library such as bus arbiter, fifo. With this approach it is possible to map communication specification into any protocol, from a simple handshake to a complex protocol.

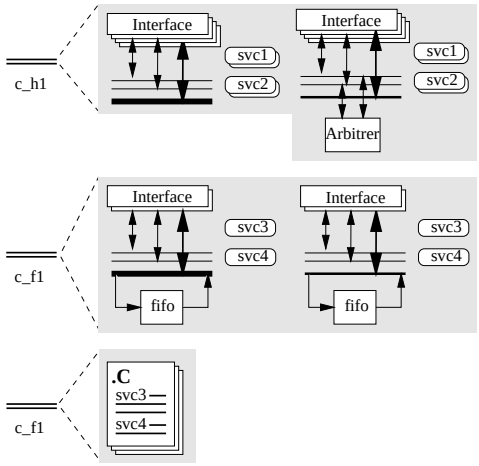

Fig. 7. Implementation library

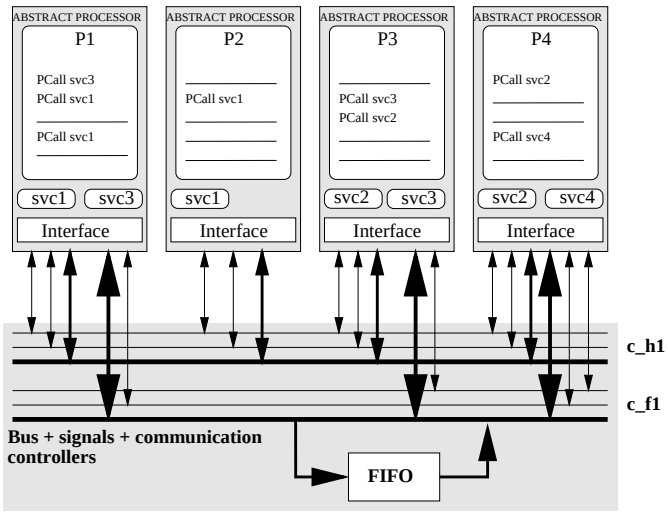

Fig. 8. System after interface synthesis

Starting from the system of figure 5, the result of interface synthesis task is detailed in figure 8. The communication unit *c_h1* has two possible implementations, one with an external bus arbiter for scheduling the two communication, one with the arbiter distributed in the interfaces. Any of the two implementation may be selected.

C. Statement of Communication Synthesis as an Allocation Problem

The communication synthesis is formulated as an allocation problem aimed at fixing the number and type of communication units needed to implement the abstract network. Given (1) a set of processes communicating via a set of primitives (figure 1) and (2) a library of functional communication units with their services and specific protocols (figure 4), the objective is to allocate a set of communication units that perform the task of the abstract network (figure 5, 6). Each communication unit hides a special kind of communication implementation. This scheme allows more than one form of communication protocol to exist within the same framework. The interface synthesis al-

lows to fix the implementation (actual signals and possible communication components) of the communication scheme (figure 8). Currently there is no system in our knowledge that performs the selection of the physical communication structure automatically.

V. COMMUNICATION UNIT ALLOCATION/BINDING ALGORITHM

A. Introduction

The proposed allocation/binding algorithm starts with a library of functional communication units and a process graph. The nodes of this graph are the processes and the edges are the abstract channels. The main task of the algorithm is to allocate from the library a set of instances of communication units to perform the task of the abstract network (figure 9). Allocation is based on a cost function that is to be reduced and some constraints that have to be met.

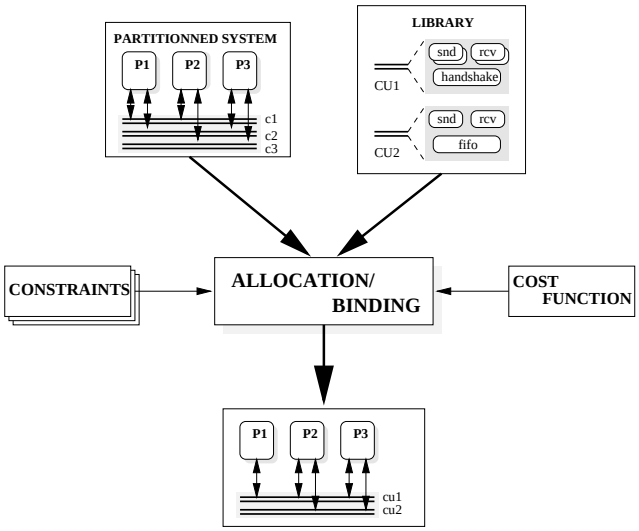

Fig. 9. Channel allocation/binding

For each abstract channel M_i we will use the same set of constraints defined in [26] :

- The protocol requested for the communication over that abstract channel noted $Protocol(M_i)$.
- The services provided to the processes noted $Services(M_i)$.
- The average transfer rate $AveRate(M_i)$. It is defined as the rate at which data is sent over the bus.
- The peak transfer rate $PeakRate(M_i)$. It is defined as the rate at which a single transfer occurs over the bus.

Both $AveRate(M_i)$ and $PeakRate(M_i)$ are specified in bits/clock. Those constraints can be set by the user or given by an estimation tool.

With each communication unit C_j from the library are given a set of properties :

- Its cost noted $Cost(C_j)$ which represents the intrinsic cost of the component due to its complexity, silicon area, buffering capacity, etc.
- The protocol implemented by that communication unit noted $Protocol(C_j)$.
- The maximum bus rate $MaxBusRate(C_j)$ at which the data can be transferred across the communication unit.
- The services offered noted $Services(C_j)$.
- The maximum number of independent communications it can support noted $MaxCom(C_j)$.

Given an abstract channel, a communication unit can be a candidate for allocation if it satisfies the three following conditions:

- It provides the required services : $Services(M_i) \subseteq Services(C_j)$.
- It provides the right protocol : $Protocol(M_i) = Protocol(C_j)$.
- It provides the minimum required bus bandwidth : $MaxBusRate(C_j) > AveRate(M_i)$.

During allocation of communication units we attempt to assign several abstract channels on the same instance of a communication unit to reduce cost. If some abstract channels need to transfer data at a certain average rate, after being merged onto the same communication unit they should be able to transfer data at the same rate [26]. If the $MaxBusRate$ is greater than the sum of the $AveRate$ of all the abstract channels merged on that communication unit we have a feasible implementation. All processes using that communication unit will be able to transfer data without being slowed by an insufficient bus bandwidth. Therefore we must have :

$$MaxBusRate(C_j) \geq \sum_{\text{All } M_i \text{ merged on } C_j} AveRate(M_i)$$

To ensure that a single data transfer does not take unnecessarily long time, the peak rate should be satisfied. This can be expressed as :

$$MaxBusRate(C_j) \geq PeakRate(M_i),\ \forall\, M_i \text{ merged on } C_j$$

If this constraint is not satisfied then the cost of that solution will increase. Since a finite number of abstract channels can be merged on a single instance of a communication unit we must have :

$$\text{number of abstract channels } M_i \text{ merged on a communication unit } C_j \leq MaxCom(C_j)$$

B. Algorithm

The proposed algorithm first builds the tree of all possible implementations. This decision tree enumerates for each abstract channel all the communication units from the library that are candidate for allocation (section V-A). The nodes of the tree are the abstract channels and the edges represent communication units. Each node will have as many candidates as communication units that may implement that abstract channel. The leaves of the tree correspond to empty nodes. Each path in the tree from the root to a leaf node is a possible solution.

The second step of the algorithm is to perform a depth first exploration of the tree in order to select the best solution. In order to handle the case where several abstract

channels are assigned on the same instance of a communication unit from the library, we use a procedure called *merge*. This procedure is used during the tree exploration in order to assign several abstract channels on the same instance of a communication unit. If a *merge* attempt fails, a new instance of a communication unit will be created. The algorithm is detailed in the rest of this section. The main program builds the tree. The procedure *traversal* explores the tree and procedure *merge* allows the assignment of several abstract channels on the same communication unit during the tree exploration.

The cost function to be reduced by the allocation algorithm takes into account a selected communication unit C_j from the library and several abstract channels M_i merged on it. This cost function is given below:

$$costfunction = K_1 * cost(C_j) + K_2 * \sum_{\text{All } M_i \text{ merged on } C_j} [PeakRate(M_i) - MaxBusRate(C_j)]^2$$

The second term of the cost function is taken into account only if the constraint on *PeakRate* is violated, i.e. only if $PeakRate(M_i) > MaxBusRate(C_j)$. K_1 and K_2 are user set parameters used to weight each term of the cost function. These allow trade-offs between component cost and performance. Let M_i be a abstract channel offering a set of services, called $Services(M_i)$, that have to be allocated on the same communication unit. Let C_j be an element of the library of communication units. C_j is a communication unit that offers a set of services, called $Services(C_j)$. Let $\mathcal{A}$ be a solution for the allocation/binding of the abstract channel network and *total_cost* its cost. Let $\mathcal{I}$ be a list of instances of communication units that have already been allocated along a path in the tree : $\mathcal{I} = \{I_1, I_2, ..I_f\}$. With each I_k comes a set of variables :

- The current bus load of that communication unit noted $BusRate(I_k)$. It is the sum of the $AveRate$ of all abstract channels allocated on that instance.
- The number of communications handled by that communication unit called $CurrentCom(I_k)$. It correspond to the number of abstract channels merged on that communication unit.

With each node of the tree is associated a abstract channel noted *AbstractChannel(node)* and to each outgoing decision edge a communication unit noted *CommunicationUnit(edge)*. Each edge is terminated by a node noted *Nextnode(edge)*

ALGORITHM

```
Algorithm Allocation/Binding {
    build the decision tree
    A = {∅}
    total_cost = +∞
    current_cost = 0
    TRAVERSAL (root, I, current_cost)
}
Procedure MERGE (InstanceList I, Instance CU : IN, Instance V,
Integer merge_cost : OUT) {
merge_cost = ∞
V = ∅
For each instance I_k == CU, I_k ∈ I Do {
    If protocol(CU) = protocol(I_k) and
        BusRate(I_k) + AveRate(M_i) ≤ MaxBusRate(I_k) and
        CurrentCom(I_k) + 1 ≤ MaxCom(I_k) Then {
        If PeakRate(M_i) > MaxBusRate(I_k) Then {
            current_merge_cost =
            K_2 * [PeakRate(M_i) - MaxBusRate(I_k)]^2
        }
        Else
            current_merge_cost = ∞
        If current_merge_cost < merge_cost Then {
            merge_most = current_merge_cost
            V = I_k
        }
    }
}
Return V and merge_cost
}

Procedure TRAVERSAL (Node n : IN, InstanceList I,
Integer current_cost : OUT) {
If n is a leaf Then {
    If current_cost < total_cost Then {
        /* new better solution found */
        A = I
        total_cost = current_cost
    }
}
Else {
    V = ∅
    M_i = AbstractChannel(n)
    For every edge e of n Do {
        CU = CommunicationUnit(e)
        MERGE (I, CU, V, merge_cost)
        If (V ≠ ∅) Then {       /* merge successful */
            bind M_i to V
            BusRate(V) += AveRate(M_i)
            CurrentCom(V) ++
            current_cost += merge_cost
            TRAVERSAL (Nextnode(e), I, current_cost)
        }
        Else {      /* allocate a new instance */
            If AveRate(M_i) ≤ MaxBusRate(CU) Then {
                /* feasible solution */
                If PeakRate(M_i) > MaxBusRate(CU) Then {
                    /* PeakRate constraint violated */
                    alloc_cost = K_1 * Cost(CU) +
                    K_2 * [PeakRate(M_i) - MaxBusRate(CU)]^2
                Else      /* no constraint violated */
                    alloc_cost = K_1 * Cost(CU)
                bind M_i to CU
                I += {CU}
                BusRate(CU) = AveRate(M_i)
                CurrentCom(CU) ++
                current_cost += alloc_cost
                TRAVERSAL (Nextnode(e), I, current_cost)
            }
            Else      /* not a feasible solution */
                current_cost = +∞
        }
    }
}
}
```

VI. RESULTS

In this section we show the results of applying our allocation algorithm onto a simple example. We consider a send and receive system. It is composed of two processes, a server, a host and two abstract channels ensuring a bi-directional communication. Let us assume that the communication synthesis starts with the system of figure 10. We give the following set of constraints on the abstracts channels:

- *AveRate(c1)* = 12 bits/clock, and *PeakRate(c1)* = 18 bits/clock.
- *AveRate(c2)* = 4 bits/clock, and *PeakRate(c2)* = 8 bits/clock.
- *Protocol(c1)* = *Protocol(c2)* = any.

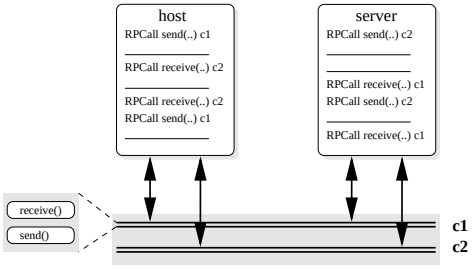

Fig. 10. Send and receive system

We set $K_1 = 1$ and $K_2 = 10$, therefore we favour the performance by setting its weight to 10 times the weight of the component cost. The communication library used for allocation is detailed on figure 11. It contains three communication units :

- a bi-directional handshake protocol.
- a single FIFO.
- a dual FIFO.

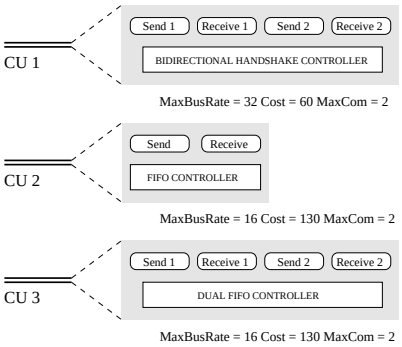

Fig. 11. Functional communication units library

Figure 12 shows the decision tree corresponding to the system described in figure 10 with the communication units library of figure 11. Each node of the tree correspond to an abstract channel. The edges corresponds to the different possible allocations for the abstract channel. The leaf nodes give the cost of the solution. When a communication unit can handle several abstract channels the leaf node contains two numbers corresponding to the path before/after *merge*. The first corresponds to the cost obtained by associating one communication unit for each abstract channel. The second is the cost obtained by sharing the communication unit. In figure 12, cu1 is a bi-directional handshake that can handle the communication of the two abstract channels c1 and c2. The leaf node includes two costs : 120 is the cost of two instances of cu1 assuming that the two abstract channels do not share the same communication unit, 60 is the cost of the corresponding solution with one shared instance of communication unit cu1.

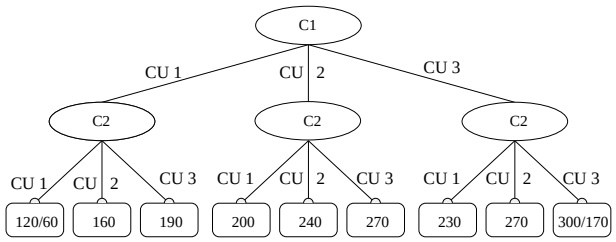

Fig. 12. Decision tree for the allocation/binding of system send-and-receive

Five of the possible allocation/binding alternatives are listed below and described in figure 13.

- (a) both c1 and c2 with a handshake, and total_cost = 60.
- (b) both c1 and c2 with a FIFO, and total_cost = 240.
- (c) c1 with a handshake and c2 with a FIFO, and total_cost = 160.
- (d) c1 with a FIFO and c2 with a handshake, and total_cost = 200.
- (e) both c1 and c2 with a dual FIFO, and total_cost = 170.

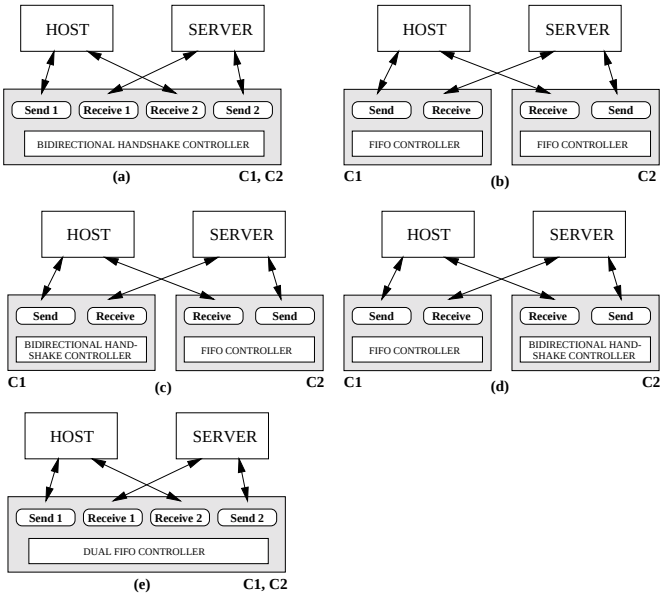

Fig. 13. Channel allocation/binding alternatives

These alternatives, presented on figure 13, make use of the functional communication units library presented in figure 11. From figure 13 we see that the allocation/binding of communication units determines the topology of the interconnection network. The total_cost is obtained by applying

the cost function detailed above. The algorithm is going to choose the solution with the lowest cost, therefore solution (a) will be retained. The two abstract channels will be physically implemented as a single bi-directional bus that multiplexes both accesses from abstract channel c1 and abstract channel c2. The interface synthesis is going to map a predefined generic interface onto the processes to obtain abstract processors using the implementation library (figure 14). Thus it will generate one bus with its control signals (figure 15). This step generates all the interfaces. This corresponds to an expansion (inlining) of procedure calls into the processes according to the communication units selected. The size of the bus will be fixed by the interface synthesis algorithm depending on the data transfer rate.

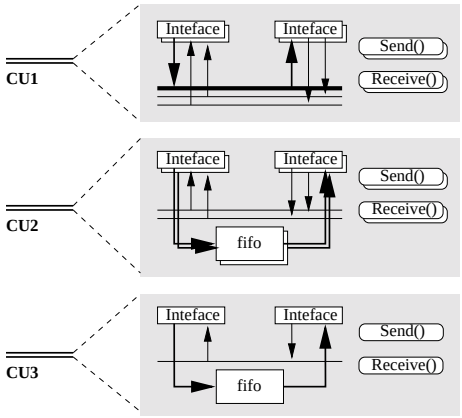

Fig. 14. Implementation library

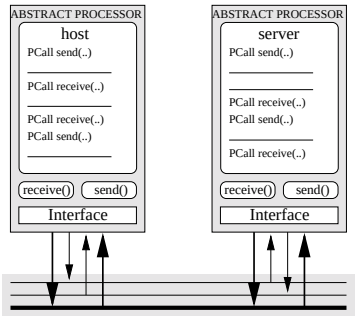

Fig. 15. System send and receive after interface synthesis

Table I gives a summary of the allocation/binding alternatives with their cost. The number of buses and external controllers neddeed are also reported. It includes the data and control lines. We assume the following interface for the library communication unit (figure 14).

- bi-directional handshake : 1 in and 1 out data port and 2 control ports.
- single FIFO : 1 in and 1 out ports and 1 control port.
- dual FIFO : 2 in and 2 out ports and 2 control ports.

VII. CONCLUSION

We have presented in this paper a means whereby communication synthesis is stated as an allocation problem. This approach allows a wide design space exploration through automatic selection of communication protocols. This problem can be solved by using most of the allocation algorithms used in high level synthesis. We have presented one possible algorithm for communication units allocation/binding. This algorithm is based on a decision tree. The interface synthesis task may be performed automatically. The key issue in this scheme is the use of an abstract and general communication model. The separation between communication and computation allows the reuse of existing communication models. A library of communication units offers a wide range of communication mechanisms allowing the designer to select the appropriate communication protocol for his application. Since no restrictions are imposed to the communication models this process can be applied to a large class of codesign applications.

protocol		cost	buses	controller
channel c1	channel c2			
handshake		60	1	none
handshake	single FIFO	160	3	1
handshake	dual FIFO	190	3	1
single FIFO	handshake	200	3	1
single FIFO	single FIFO	240	4	2
single FIFO	dual FIFO	270	4	2
dual FIFO	handshake	230	3	1
dual FIFO	single FIFO	270	4	2
dual FIFO		170	4	1

TABLE I
ALLOCATION/BINDING COST

ACKNOWLEDGMENTS

This work was supported by France-Telecom/CNET under grant 94 1B 113 and SGS-Thomson.

REFERENCES

[1] G.R. Andrews, *Concurrent Programming, Principles and Practice*, Benjamin/Cummings (eds), Redwood City, Calif., pp. 484-494, 1991.

[2] T. Ben Ismail, and A.A Jerraya, *Synthesis Steps and Design Models for CoDesign*, IEEE Computer, special issue on rapid-prototyping of microelectronic systems, Vol. 28, No. 2, pp. 44-52, February 1995.

[3] A.D. Birrell, and B.J. Nelson, *Implementing remote procedure call*, ACM Transactions on Computer Systems, Vol. 2, No. 1, pp. 39-59, February 1984.

[4] K. Buchenrieder, *A Prototyping Environnement for Control Oriented Hardware/Software Systems Using State-Charts, Activity-Charts and FPGA*, Proceedings of the European Design Automation Conference with Euro-VHDL, pp. 60-65, September 1994.

[5] P. H. Chou, R. B. Ortega, and G. Borriello, *The Chinook Hardware/Software Co-Synthesis System*, Proceedings of the 8th International Symposium on System Synthesis, pp. 22-27, September 1995.

[6] W. Ecker, M. Glesner, and A. Vombach, *Protocol merging : A VHDL Based Method for Clock Cycle Minimising and Protocol Preserving Scheduling of IO Operations*, Proceedings of the European Design Automation Conference with Euro-VHDL, pp. 624-629, September 1994.

[7] W. Ecker, and M. Huber, *VHDL Based Communication and Synchronization Synthesis*, Proceedings of the European Design Automation Conference with Euro-VHDL, pp. 458-462, September 1995.

[8] W. Ecker, *Semi Dynamic Scheduling of Synchronisation Mechanisms*, Proceedings of the European Design Automation Conference with Euro-VHDL, pp. 374-379, September 1995.

[9] R. Ernst, J. Henkel, and T. Benner, *Hardware/Software Co-Synthesis for Microcontrollers*, IEEE Design & Test of Computers, Vol. 10 No. 4, pp. 64-75, December 1993.

[10] D. Filo, D. Ku, C. N. Coelho, and G. de Micheli, *Interface Optimisation for Concurrent Systems Under Timing Constraints*, IEEE Transactions on VLSI, Vol. 1, pp 268-281, September 1993.

[11] C. H. Gebotys, *Optimal Scheduling and Allocation of Embedded VLSI Chips*, Proceedings of the IEEE Design Automation Conference, pp. 116-120, June 1992.

[12] D. Gajski, and F. Vahid, *Specification and Design of Embedded Hardware/Software Systems*, IEEE Design & Test of Computers, pp. 53-67, Spring 1995.

[13] J. Gong, and D. Gajski, *Model Refinement For Hardware Software Codesign*, Proceedings of the European Design & Test Conference, pp 270-274, March 1996.

[14] R.K. Gupta, C. N. Coelho and G. de Michelli, *Synthesis and Simulation of Digital Systems Containing Interacting Hardware and Software Components*, Proceedings of the IEEE Design Automation Conference, pp. 225-230, June 1992.

[15] R.K. Gupta, C.N. Coelho and G. de Michelli, *Program Implementation Schemes for Hardware Software Systems*, IEEE Design & Test of Computers, Vol. 27, No. 1, pp. 48-55, January 1994.

[16] R.K. Gupta, and G. de Michelli, *Hardware/Software Cosynthesis for Digital Systems*, IEEE Design & Test of Computers, Vol. 10 No. 4, pp. 29-41, December 1993.

[17] J. Henkel, R. Ernst, U. Holtman, and T. Benner, *Adaptation of Partitioning and High Level synthesis in Hardware/Software Co-Synthesis*, Proceedings of IEEE International Conference on Computer Aided Design, pp. 96-100, November 1994.

[18] C.Y. Huang, Y.S. Chen, Y.L. Lin, and Y.C. Hsu, *Data Path Allocation Based on Bipartite Weighted Matching*, Proceedings of the IEEE Design Automation Conference, pp. 499-504, June 1990.

[19] A.A. Jerraya, and K. O'Brien, *SOLAR: An Intermediate Format for System-Level Modelling and Synthesis*, in "Computer Aided Software/Hardware Engineering," J. Rozenblit, K. Buchenrieder (eds), IEEE Press, Piscataway, N.J., pp 147-175, 1994.

[20] A. Kalavade, and E.A. Lee, *Hardware/Software Codesign Methodology for DSP applications*, IEEE Design & Test of Computers, Vol. 10 No. 4, pp. 16-28, December 1993.

[21] B. Lin, and S. Vercauteren, *Synthesis of Concurrent System Interface Modules with Automatic Protocol Conversion Generation*, Proceedings of IEEE International Conference on Computer Aided Design, pp. 395-399, November 1994.

[22] J. Madsen, and B. Hald, *An Approach to Interface Synthesis*, Proceedings of the 8th International Symposium on System Synthesis, pp. 16-21, September 1995.

[23] A.J. Martin, *Synthesis of Asynchronous VLSI Circuits*, Formal Methods for VLSI Design, J. Staunstrup (eds), North Holland, 1990.

[24] P. Michel, U. Lauther, and P. Duzy, *The Synthesis Approach to Digital System Design*, Kluwer Academic Publishers, 1992.

[25] S. Narayan, and D. Gajski, *Features Supporting System-Level Specification in HDLs*, Proceedings of the European Design Automation Conference with Euro-VHDL, pp. 540-545, September 1993.

[26] S. Narayan, and D. Gajski, *Synthesis of System-Level Bus Interfaces*, Proceedings of the European Design Automation Conference with Euro-VHDL, pp. 395-399, February 1994.

[27] S. Narayan, and D. Gajski, *Interfacing Incompatible Protocols Using Interface Process Generation*, Proceedings of the IEEE Design Automation Conference, pp. 468-473, June 1995.

[28] A. P. Ravn, and J. Staunstrup, *Interface Models*, Proceedings of the IEEE Codes/Cashe workshop, pp. 157-164, September 1994.

[29] K. Salah, and R. Probert, *A Service-Based Method for the Synthesis of Communication Protocols*, International Journal of Mini and Microcomputers, Vol. 12, No. 3, pp. 97-103, 1990.

[30] M. B. Srivastava, and R. W. Brodersen, *SIERA : A Unified Framework for Rapid Prototyping of System Level Hardware and Software*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 14, No. 6, June 1995.

[31] A. Takach, and W. Wolf, *Scheduling Constraint Generation for communicating Processes*, IEEE Transactions on VLSI Systems, Vol. 3, No. 2, June 1995.

[32] D.E. Thomas, J.K. Adams, and H. schmit, *A Model and Methodology for Hardware/Software Codesign*, IEEE Design & Test of Computers, Vol. 10 No. 4, pp. 6-15, December 1993.

[33] F. Vahid, and D. Gajski, *Specification Partitioning For System Design*, Proceedings of the IEEE Design Automation Conference, pp. 219-224, June 1992.

[34] F. Vahid, and D. Gajski, *Closeness Metrics for System Level Functional Partitioning*, Proceedings of the European Design Automation Conference with Euro-VHDL, pp. 328-333, September 1995.

[35] P. Vanbekbergen, C. Ykman-Couvreur, B. Lin, and H. de Man, *A Generalised Signal Transition Graph Model for Specification of Complex Interfaces*, Proceedings of the European Design & Test Conference, pp. 378-384, February 1994.

[36] S. Vercauteren, B. Lin, and H. de Man, *Constructing Application Specific Heterogeneous Embedded Architecture from Custom HW/SW Application*, Proceedings of the IEEE Design Automation Conference, pp. 521-526, June 1996.

[37] T. Yen, and W. Wolf, *Communication Synthesis for Distributed Embedded Systems*, Proceedings of the International Conference on Computer Aided Design, pp. 288-294, November 1995

[38] W. Wolf, *Hardware/Software Co-Design of Embedded Systems*, Proceedings of the IEEE, Vol 82, No 7, pp 967-989, 1994.

[39] X. Xiong, P. Gutberlet, and W. Rosenstiel, *Automatic Generation of Interprocess Communication in the PARAGON System*, Proceedings of IEEE International Workshop on Rapid System Prototyping, pp 24-29, June 1996.