Interface Synthesis in Embedded HW/SW Systems

Michel AUGUIN, Cécile BELLEUDY, Guy GOGNIAT

I3S, Université de Nice Sophia/Antipolis - CNRS, 41 Bld. Napoléon III 06041 Nice Cedex, France.

- *Contact Person:* **Michel AUGUIN**
- *Phone:* **+33-93 21 79 58**
- *Fax:* **+33-93 21 20 54**
- *E-mail:* *Michel Auguin:* **auguin@alto.unice.fr**
- *Topic:* **System-Level Specification and Design - Design Systems and Tools**

Abstract

Since several years, embedded systems are used in an expanding number of domains. Furthermore, their complexity has significantly increased and most embedded systems must respect several design constraints. Hence, it becomes necessary to introduce major improvements in CAD methodologies in order to help designers. Hardware-Software codesign attempts to deduce automatically heterogeneous system solutions from a high level specification. The interface between heterogeneous resources can become critical in time and/or area for telecommunication applications, hence it is important to define techniques that minimize the communication overhead cost. In this paper we present an original method to synthesize efficient interfaces.

All appropriate organizational approvals for the publication of this paper have been obtained. If accepted the authors will prepare the final manuscript in time for inclusion in the Conference proceedings and will present the paper at the Conference.

1. Introduction

Since several years, embedded systems are used in an expanding number of domains, for example in medical devices, automobile cruise control, videoconference or wireless applications. Their complexity has also significantly increased due to integration density improvements and sophistication of digital applications. Furthermore most embedded systems must respect several design constraints (i.e., hardware cost, performance, power consumption). Hence, in order to handle these constraints designers use heterogeneous resources. For example a DSP and an ASIC. Hand-crafted techniques to design hardware-software systems are no longer sufficient due to the high complexity and the time to market pressure. Thus, it becomes necessary to introduce major improvements in CAD methodologies in order to help designers. For the considered applications the design cycle is generally composed of four main steps: Specification modelling, Hardware-software partitioning, Architecture refinement and Technology implementation. The hardware-software partitioning step precises, starting from the model of the specification, which parts will be handled by the software (DSP or RISC) and which functions will be supported by the hardware. Several partitioning methods and heuristics have already been published [4], [9], [14], [11], [17]. The architecture refinement follows the partitioning step and manages three interdependent tasks: the interface synthesis, the hardware synthesis and the code generation for the software part. These tasks lead to a RTL level description of the mixed hardware-software architecture which includes the code for the software entity. Finally the technology implementation step allows to generate the layout of the application. Each step of this design flow involves complex and sometimes open problems. In the case of the architecture refinement, the interface synthesis problem can become critical in time and/or area for applications that deal with an important number of data as in signal processing with spectral methods or in image processing applications. Hence, it is important to define techniques that allow to minimize the communication overhead cost. In this paper we present an original method to realize the interface synthesis.

The rest of this paper is organized as follows. First, we present most significant related works and detail our approach of the interface problem. Then, we describe the methodology that is used to minimize the hardware communication overhead cost. In Section 4, we introduce the example of an acoustic echo canceller to illustrate the considered approach and, finally, in Section 5, we conclude.

2. Related works

Communications between different entities can be considered at several levels of granularity. At the lowest level, communication primitives can be described with chronograms. These chronograms represent the behavior of each control signal that supports communications. At a higher level, protocols can be described with communicating processes using send and receive primitives. In this case the fine level behavior is not considered.

The method proposed by D.E. Thomas and J.A. Nestor [13] allows to generate hardware structures that support fine grain protocols. Relations between signals are described with a textual formalism where designers define behavior and timing constraints. The synthesized hardware architecture is based on finite state machines. In a same way, studies handled by G. Boriello and R.H. Katz [2] deal with interface synthesis at a fine grain level. The proposed method can manage hardware area minimization as well as bus bandwidth optimization. The methodology is based on formalized timing diagrams to specify the interface. The synthesis algorithms transform event graphs derived from the timing diagrams into a logic specification for the interface. The interface is based on elementary logic cells. In works of A.C. Parker, S.A. Hayati and J.J. Granacki [10] the description of the interface is also based on graphs. The

main feature of the method is the wide spectrum of protocols that can be described. For example synchronous and asynchronous protocols can be supported. The method proposed by S. Narayan and D. Gajski [12] supports the interface synthesis of incompatible protocols. The interface process generated between the communicating entities can manage different data widths. The method is based on duals of atomic protocol operations and can minimize the interface hardware area. All these methods deal with descriptions of protocols at a fine grain level. Designers need to have a precise knowledge of requested protocols. Such knowledge becomes difficult to get if we consider complex systems using heterogeneous resources. In this case it is important to consider a higher level of abstraction, so designers can concentrate on system design instead of the interface structure. In works handled by R.W. Brodersen and J.S. Sun [16] the considered applications are composed of a wide range of components (DSP, ASIC, FPGA or MCM subsystems). Generally, all these components have their own communication mechanisms. Thus, in order to allow data transfers between them it becomes necessary to introduce a hardware interface. The method is based on a specific formalism that describes the communication protocols with high level primitives. These primitives are then decomposed in event graphs before generating the interface. In works handled by J.M. Daveau, T.B. Ismail and A.A. Jerraya [3] the problem of interface synthesis is managed with an other approach. Designers specify protocols at a high level of description, for example a communication primitive is defined with a *rendez-vous* mechanism, maximum and average bandwidth can also be precised. Furthermore several protocols with their main features are defined in a library. The proposed method is based on an allocation algorithm: the problem is to find the best interface to support protocols specified by designers. The solution minimizes the hardware interface cost and respects all the communication constraints.

The method proposed by J. Gong, D. Gajski and S. Bakshi [8] allows to specify an interface between processors starting from the specification of the application. The application is decomposed into local and global variables and functions that realize computations. Four interface models are proposed that enable designers to explore different communication schemes. The choice of the interface is made according to design constraints and also depending of the number of local and global variables. When the specification is refined with interface features, behavioral synthesis and software synthesis permit to generate the hardware-software design. In this method one of the interesting point is that it relieves the designer from the protocol determination task. In a same way, works proposed by D. Filo, D. Ku, C. Coelho and G. De Micheli [6] enable to define automatically high level interfaces. In their method, application specifications are described with graphs where nodes represent computations and edges functional and temporal dependencies. The considered target architecture is fully synchronous i.e., the same clock signal is propagated through all the design. To minimize hardware communication costs the method schedules graph nodes in order to increase the use of synchronous protocols. The cost related to these protocols in a synchronous architecture is lower than asynchronous protocols. However, synchronous target architectures are not generally best solutions to implement complex application using heterogeneous resources due to synchronization problems. Furthermore, processor clock periods are mainly lower than hardware implementations using FPGA or gate arrays structures [15]. Hence, an asynchronous template architecture with concurrent resources timed by local clocks may yield more efficient mixed hardware-software implementations.

Optimizing the interconnection between units in such heterogeneous architectures suggests the following properties:

- (i) performing a global optimization of all communications involved by the application and not only between two entities,
- (ii) a high level of abstraction to model communication features.

Furthermore, relieving the designers from this time consuming task is one aim of the codesign paradigm. Above introduced methods are not well convenient for this purpose then we propose a new approach for the interface synthesis.

3. Interface synthesis

The interface synthesis process takes place after hardware-software partitioning (Fig. 1 •). We consider that the result of the partitioning step is a direct acyclic graph where nodes V_i represent computations and edges $e_{i,j}$ functional and temporal dependencies between nodes V_i and V_j. Functional dependency edges represent data transfers. Temporal dependency edges link all nodes that perform their computations on the same resource even if they are functionally independent. This relation exhibits the sequentiality of these nodes. Granularity levels of nodes depend on applications and several levels can be

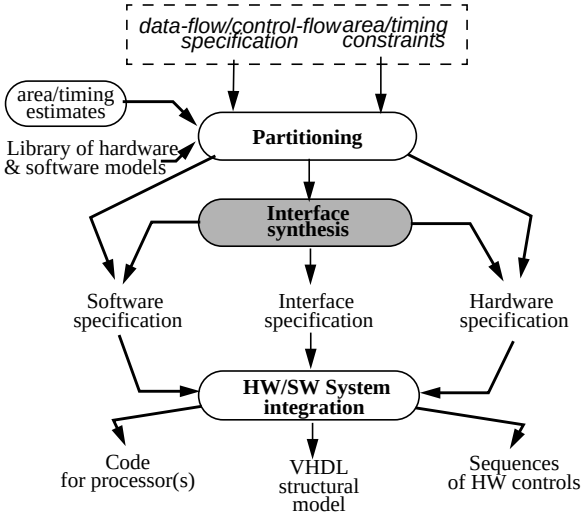


Fig. 1 • Overview of the codesign flow

mixed in the graph. For example, modelling of the acoustic echo canceller application requires nodes representing addition or FFT operations. Furthermore, all nodes are characterized with a parameter which indicates whether the considered node represents an operation that will be handled by a software unit or it will be performed on a hardware resource. Interface synthesis takes into account the underlying target architecture. In our case this architecture is composed of asynchronous functional units (Fig. 2 •). A functional unit is either a specific cell or a processor core or a synthesized cell. A controller is associated with each functional unit. This controller ensures the execution of corresponding instructions that are stored

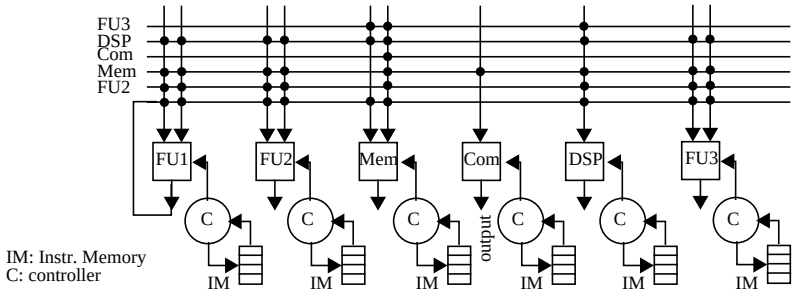


Fig. 2 • The considered target architecture

in a local memory. Asynchrony naturally encourages modularity by making functional units independent of the computation speed of other units, and simplifies hardware and software implementations by avoiding global timing references [15]. Furthermore, this computation model is well adapted to considered ap-

plications i.e., telecommunication and multimedia which are mainly data flow. For further information about this target architecture refer to [1]. In the proposed method, interface synthesis is performed before hardware synthesis and code generation. This approach allows to take into account results of interface synthesis in following steps. Interface synthesis generally consists in determining for each communication of the application: the transfer type, the communication support, the protocol and the transfer mode. This determination of the interface attempts to minimize the hardware communication area and the communication overhead. For the remainder of the interface synthesis presentation, we describe the considered communication model and how this model is taken into account in order to respect design constraints and to meet communication goals.

3.1 Communication model

Our communication model is based on (i) bufferized structures when communications are asynchronous, (ii) non bufferized structures when sender and receiver can be synchronized using a *rendez-vous* protocol (Fig. 3 •). A bufferized structure allows sender and receiver to emit and receive independently data since the communication is performed through a FIFO. This FIFO represents an additional hardware resource cost dedicated to communications. The shared memory communication model is not

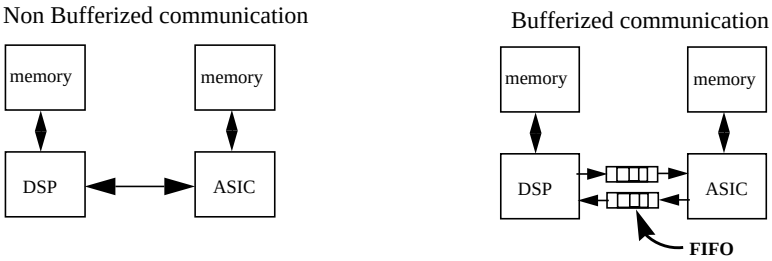


Fig. 3 • Communication resources

considered, this is due to the execution scheme of our data synchronized architecture which is based on a FIFO communication model. Using shared memories would impose to introduce local control logic which leads to the same complexity as a FIFO structure. The complexity of control mechanisms depends on protocols associated with asynchronous communications i.e., blocking or non blocking. The protocol is blocking if before reading or writing into a FIFO the corresponding unit respectively check the availabilities of data or verify that the FIFO is not full. With a non blocking protocol no verification needs to be done before writing or reading data. For communications using non bufferized structures, the hardware resource is a simple bus. Cost due to this transfer type is less important, however synchronization mechanisms must be introduced. As this solution represents a lower cost the interface synthesis attempts to minimize the utilization of bufferized structures. Note that performances can be reduced if the receiver is not ready as soon as the sender is and conversely. The last element in the communication model is the transfer mode. The selection of the transfer mode depends on capabilities integrated in target processors. Most recent processors can handle DMA transfers and memory mapped I/O move operations. When using DMA transfers, computations and communications can overlap. Nevertheless, initialization sequences are required and efficient overlapping is ensured only if data transfers induced by computation threads and I/O do not conflict on data buses. Hence, a trade-off must be found between DMA transfers and I/O memory mapped techniques in order to meet performance constraints.

3.2 Communication synthesis method

Starting from the partitioned specification and according to the communication model the aim of the interface synthesis is to define for each communication edge, (i) the type of transfer (synchronous or asynchronous), (ii) the needed hardware support and the associated protocol and (iii) the transfer mode

(DMA or memory mapped I/O). Communication edges are defined as links between hardware and software entities, between different software entities or between different hardware entities. Edges between nodes allocated to the same software or hardware entity are not considered as communication edges since in that case communications are supported through the local memory associated with the entity. Interface synthesis flow (Fig. 4 •) is composed of four main steps: (i) merging redundant links, (ii) transfer type determination, (iii) communication support and protocol determination and (iv) transfer mode determination. In the sequel each step of the interface synthesis flow in described.

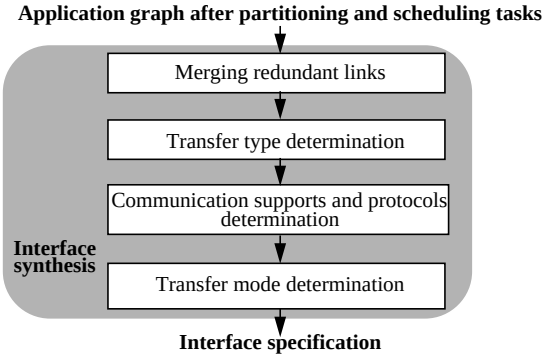


Fig. 4 • Interface synthesis flow

- **Merging redundant links**

Merging redundant links performs a preliminary hardware resource minimization. Communication edges representing data transfers between nodes can only be merged under several conditions: (i) There is a common source node, (ii) There is a single functional unit implementing destination nodes (Fig. 5 •). These edges represent the diffusion of the same data from a functional unit (associated to the

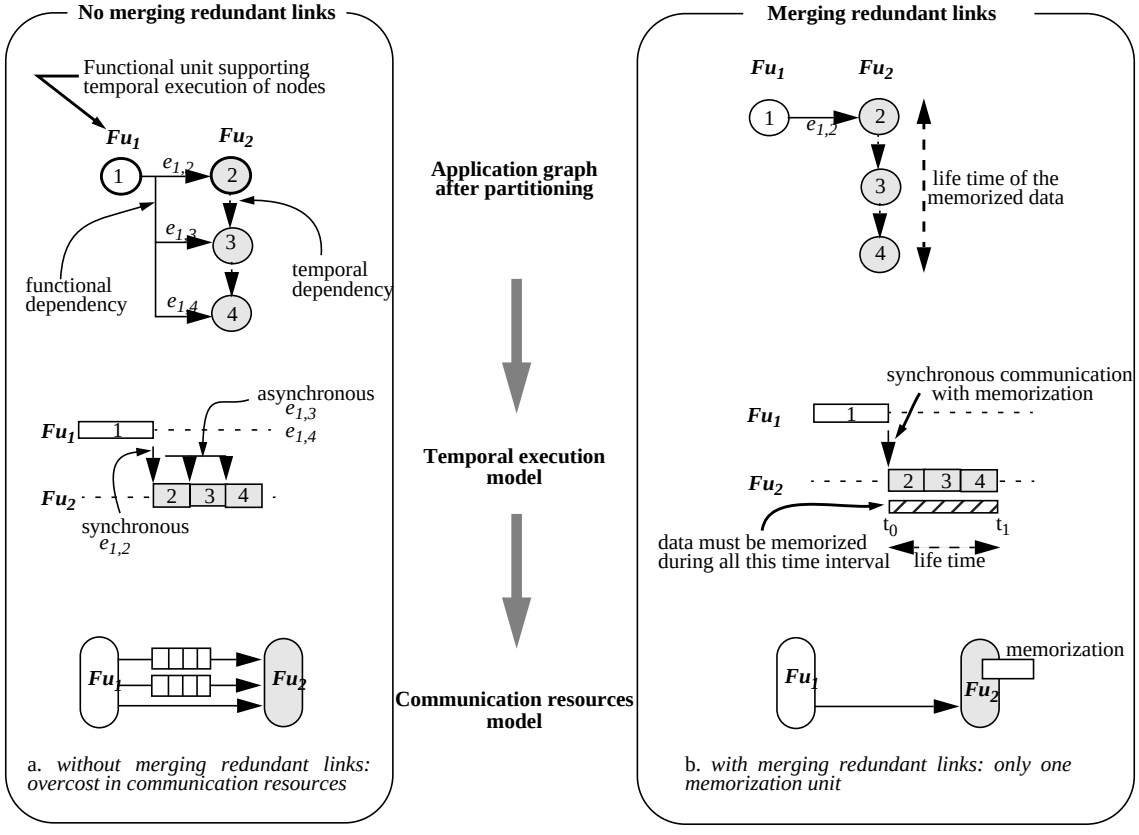


Fig. 5 • Merging redundant links: hardware resource minimization

source node) to another functional unit (associated to destination nodes). However destination nodes are scheduled at different times since their computations are performed on the same functional unit. If these edges are merged, the transferred data must be memorized in order to be read again at the right times. The Fig. 5 •a depicts hardware resources required if data transfers between nodes are duplicated. Since only one edge among the three communication edges can be synchronous, two FIFOs are required. In Fig. 5 •b merging redundant links is performed. Only one synchronous transfer is handled, but transferred data are memorized. Hence, the total amount of communication resources decreases. Consequently, this preliminary step allows local optimizations in order to minimize hardware resources.

- **Transfer type determination**

Transfer type determination constitutes the major step of the interface synthesis flow. The aim of this step is that each communication edge is labelled with a synchronous or an asynchronous transfer type. This determination is performed in order to maximize the use of synchronous communications, since in that case additional hardware cost is reduced. The communication resources in the final architecture hardly depends of that step. In order to define the set of synchronous communications, local rescheduling are performed. The local rescheduling notion allows to control if the delay due to a synchronization associated with a node respects timing constraints. If constraints are met then a synchronous communication is considered, otherwise an asynchronous communication is needed. The example described in Fig. 6 • illustrates the local rescheduling notion. The Fig. 6 •a presents a scheduling where the transfer of

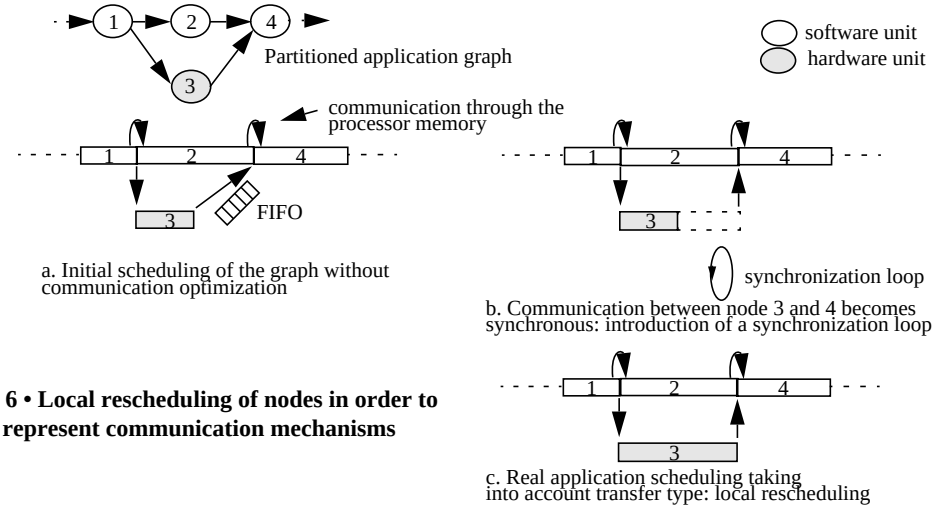


Fig. 6 • Local rescheduling of nodes in order to represent communication mechanisms

data between nodes 3 and 4 is performed through an asynchronous communication. In this solution, to ensure a correct transfer a memorization unit (FIFO) is required. This scheduling provides a correct communication sequence but hardware resources are not optimized. Another solution is shown Fig. 6 •b, in that case the transfer is performed through a synchronous communication. When computations associated with node 3 are completed, a request signal is activated and node 3 begins a synchronization loop until node 4 is ready to receive data. An acknowledge signal is emitted and the transfer takes place. Execution time of node 3 is increased due to the synchronization loop which represents the real execution behavior. This temporal dilatation of node 3 is called local rescheduling as illustrated in Fig. 6 •c. Note that the use of synchronous communications may involve overstepping of timing constraints (Fig. 7 •). In such case, the transfer of data is supported through an asynchronous communication.

In order to determine transfer type of each communication edge, two main functions are considered in the proposed algorithm: Node_characterization and Edge_characterization. For each node V_i of the application graph, Node_characterization computes the mobility interval Δ_{M_Vi} which is defined as

the interval between the ASAP starting time $ts_{(ASAP)i}$ and the ALAP ending time $te_{(ALAP)i}$ of the considered node: $\Delta_{M\_Vi} = [ts_{(ASAP)i}, te_{(ALAP)i}]$. Computations of interval mobilities take into account timing con-

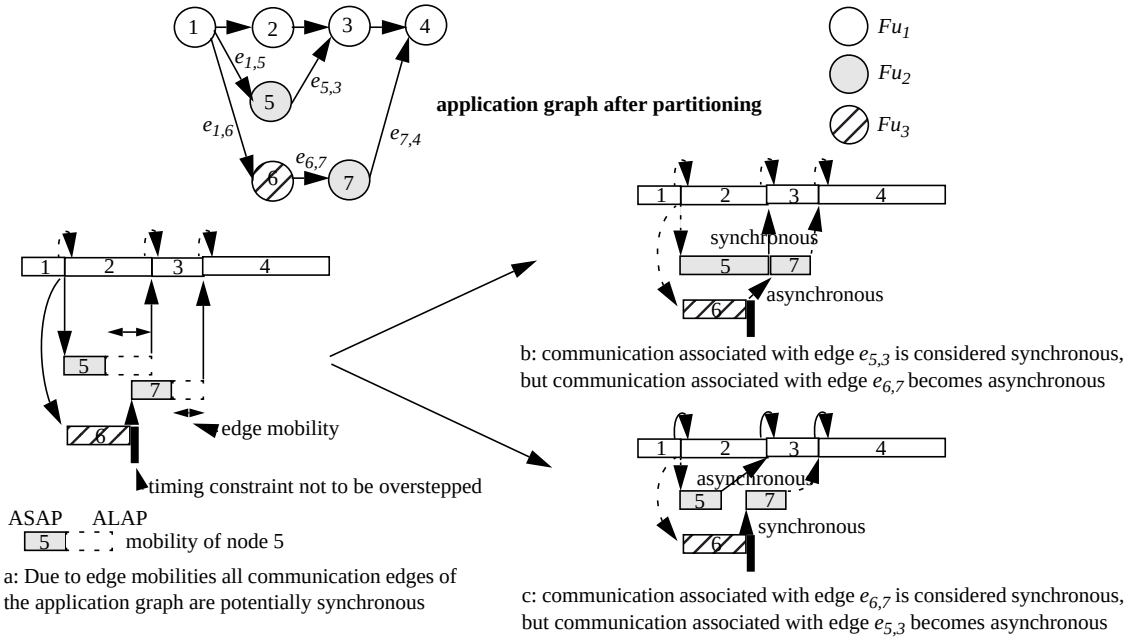


Fig. 7 • Potential synchronous communications can become asynchronous

straints. This parameter represents all the instants where one node can be scheduled without overstepping timing constraints.

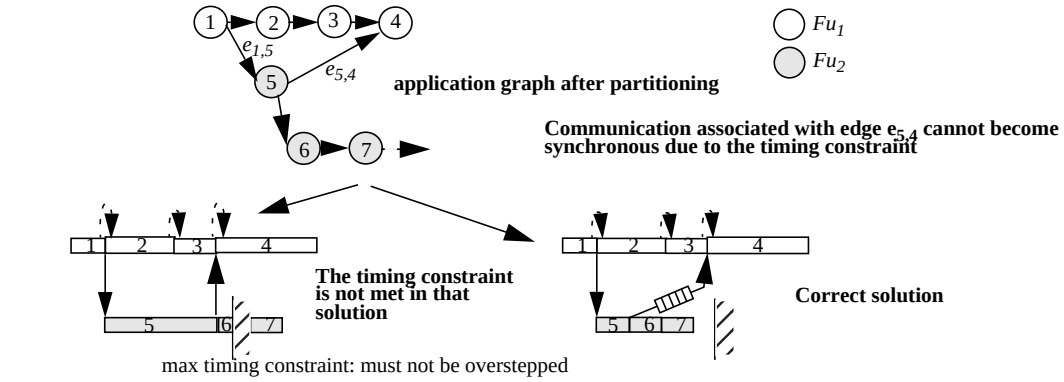


Fig. 8 • Impact of synchronous communications on the scheduling of the application

Edge_characterization computes for all the communication edges their mobility interval $\Delta Me_{i,j}$ and their mobility value $Me_{i,j}$. The mobility interval represents all the instants where a communication between two nodes V_i and V_j can take place:
$\Delta_{Mei,j} = [ts_{(ASAP)j}, te_{(ALAP)i}]$. The value $ts_{(ASAP)j}$ represents the ALAP starting time of the node that receives data and the value $te_{(ALAP)i}$ represents the ASAP ending time of the node that sends data. The mobility value $Me_{i,j}$ represents the length of the interval mobility: $Me_{i,j} = te_{(ALAP)i} - ts_{(ASAP)j}$. If $ts_{(ASAP)j} > te_{(ALAP)i}$ then $Me_{i,j}$ is negative and the communication is asynchronous since there is no timing overlap between the sender and the receiver. Otherwise the communication in considered as potential synchro-

nous. The example presented in Fig. 8 • illustrates the potential synchronous notion. In Fig. 8 •a all communications are potential synchronous, this is due to interval mobilities associated with communication edges. Fig. 8 •b and c describe two cases for which potential synchronous communications become asynchronous. For example, in Fig. 8 •b the communication associated with edge $e_{6,7}$ becomes asynchronous due to the temporal dependence between nodes 5 and 7 (these two nodes share the same functional unit). Consequently, the fact that initially all communications can be implemented with synchronous protocol does not involve that in the final solution all the communications will be effectively synchronous.

The Edge_characterization function also provides a cost function $\xi e_{i,j}$. This function is computed for each edge with a potential synchronous communication and represents the ratio between the amount of data that is transferred through this edge (volume of communication $Ve_{i,j}$) and its mobility value: $\xi e_{i,j} = Ve_{i,j} / Me_{i,j}$. Edges that have the highest cost function are considered first since if communications associated with these edges are synchronous a better hardware minimization is expected.

The algorithm is defined as follows: firstly, all nodes and edges of the graph are characterized. Edges supporting asynchronous communications ($ts_{(ASAP)j} > te_{(ALAP)i}$ i.e., $Me_{i,j} <0$) are labelled and are not considered for the remainder. An edge $e_{i,j}$ is labelled when a transfer type (synchronous or asynchronous) is assigned. For other edges (they are potential synchronous) an ordered list L according to the cost function $\xi e_{i,j}$ is created. Sender and receiver nodes corresponding to the first non labelled edge $e_{i,j}$ of L are preliminarily scheduled. Before validating this schedule, the impact of that solution on other communication edges is analyzed (the local rescheduling notion is considered). All nodes and edges are again characterized taking into account the preliminary scheduling performed on the edge $e_{i,j}$. If any transfer of data associated with communication edges becomes asynchronous the solution is convenient. The edge $e_{i,j}$ is definitively scheduled and is labelled in the list L with a synchronous transfer. If at least one communication becomes asynchronous another non labelled edge $e_{i,j}$ from L is considered in order to find a better solution. The process is iterated until all the communication edges that have no impact on other edges are labelled, i.e., their transfer type does not reveal asynchronous communications. After labelling an edge the list L is reordered since a local rescheduling may restrained mobilities of other edges.

If after this step it remains potential synchronous edges in L, each of them involves at least one asynchronous communication.

Let $\zeta_{i,j}$ be the cost function associated with the edge $e_{i,j}$ of L defined as the ratio of the total volume of data associated with non labelled edges of L that became asynchronous and the total volume of data associated with non labelled edges of L that remain synchronous: $\zeta_{i,j} = \sum$ *data of asynchronous edges* / $\sum$ *data of synchronous edges*. The edge $e_{i,j}$ of L with $\zeta_{i,j}$ minimum is labelled with a synchronous transfer since the objective is to minimize the area dedicated to FIFO structures. Edges with asynchronous communications, resulting from this choice are removed from the list L. The list is reordered according to the cost function $\xi e_{i,j}$ and the whole process is iterated until all communication edges are not labelled.

- **Communication support and protocol determination**

The FIFO size associated with a communication edge is defined in order to be sufficient to memorize the whole data transferred through this edge. Protocols associated with communications are determined in order to avoid resource access conflicts. Since synchronous communications are supported through a *rendez-vous* mechanism, protocols associated with senders and receivers are blocking. For asynchronous communications, since transfers have associated FIFOs, protocols associated with senders are non blocking. Furthermore, the above algorithm ensures that when the receiver reads from the FIFO,

all the data written by the sender are available. Hence, a non blocking protocol is associated with each receiver.

While all the communication edges are not labelled **do**

For each potential synchronous communication edge in the list *L* **do**

Preliminarily schedule the next edge $e_{i,j}$ that is not labelled;
Analyze the impact of that solution on other communication edges;

If no asynchronous communication edges is revealed **then**
Schedule definitively the edge $e_{i,j}$;
Label the edge $e_{i,j}$ with a synchronous transfer;
Reorder the list *L*;
End if;

End for;

// Each potential synchronous communication edge of L that is still not specified reveals at least one asynchronous communication edge //

Definitively schedule the edge $e_{i,j}$ that have the lowest cost function ζi,j;
Label the edge $e_{i,j}$ with a synchronous transfer;

// The schedule of the edge $e_{i,j}$ reveals at least an asynchronous communication edge $e_{k,l}$. Thus $e_{k,l}$ must be removed from the list L //

Remove asynchronous communication edges from the list *L*;
Reorder the list *L*;

End while;

Algorithm for the determination of transfer types

• Transfer mode determination

Generally, a transfer of an array of data is performed either by a direct memory access (DMA) or by memory mapped I/O move operations. The selection of the transfer mode depends on the capabilities integrated in the target processor. For example, the Motorola DSP56002 does not integrate a real DMA protocol, i.e., the DMA controller is located outside of the DSP and each value is transferred through an interrupt mechanism. Thus, on the DSP56002, CPU operations and DMA transfers are exclusive. On the TMS320C40 they may be overlapped but modelling the behavior of this DSP is more complex since numerous conflicts may occur due to movements of instructions and data that may conflict on buses. In Table 1• approximate models of the total execution times on these DSPs for a transfer of N values and C cycles of CPU operations (C ≥ 0) are depicted. Note that for DMA transfers these two tasks

Transfer mode	TMS320C40 (40ns)	DSP56002 (25ns)
DMA	ext->int: Max(N+14,C) int->ext: Max(2N+14,C) ext->ext: Max(3N+14,C)	C>8N => C+4N+4 C ≤ 8N => 12N + 4
Move	ext->int: N+4+C int->ext: 2N+4+C ext->ext: 3N+4+C	int/int: C+4N+8 int/ext: C+6N+8 ext/ext: C+8N+8

Table 1• Number of clock cycles for N I/O operations and C CPU operations

overlap and models are very different due to the disparate architectures of the DSPs. Notation *ext->int* represents a transfer from an external data memory to an internal data memory of the TMS320C40. Interactions between data and instruction flows are not modelled here. Execution time on the DSP56002 depends on the location of instructions and data. One extremity of the I/O transfer is necessary external to the DSP. Notation *int/ext* about the DSP56002 means that move instructions are located in the internal program memory and the other extremity of the transfer concerns an external data memory. The method is based on the evaluation of data transfer speeds for the DMA and I/O memory mapped technique. The solution that provides an efficient trade-off between the additional hardware cost (due to communication controls) and the communication speed (in order to meet constraints) is considered.

At this step of the interface synthesis flow, each communication of the application is characterized i.e., transfer type, transfer support, sender and receiver protocols as well as transfer mode are defined. Consequently, these interface features can be taken into account in the following steps in order to generate a compact code for the software unit and to synthesize an efficient target architecture.

4. Experiment: an acoustic echo canceller

Our experiment is the implementation of the GMDFα (Generalized Multi-delay frequency Domain Filter) algorithm which is used to perform echo cancellation in order to improve the quality of hand-free telephones. The acoustic coupling between the loudspeaker and the microphone of each terminal is a significant operating difficulty. The GMDFα algorithm is a frequency-domain block adaptive algorithm which allows to meet application constraints and to reduce arithmetic complexity. A detailed description is given in [5].

The data flow graph of the GMDFα [7] algorithm is given in Fig. 9 • (R=64, N=128, K=8, α=2). On each arc, the name and the volume of data are given. A dotted line means that this data will be used at the next iteration of the algorithm. Node 0 computes a FFT on the input signal (XT). Node 1 is the normalization block. The output of node 2 gives the estimated output in the frequency domain (Qx denotes arrays Qr and Qi of complex values) by a convolution product, and after an FFT^{-1} (node 3), we obtain the estimated output in the time domain (Ytest). The difference between the desired output (YT) and the estimated output is calculated in node 4. Node 5 provides the echo (YF). Node 6 transforms the error signal (Et) in the frequency domain (Ex) by one FFT. In nodes 7.i and 8.i, (i=1..K) filter coefficients are computed by FFT and FFT^{-1} operations of each elementary cells. Hardware resources that can support

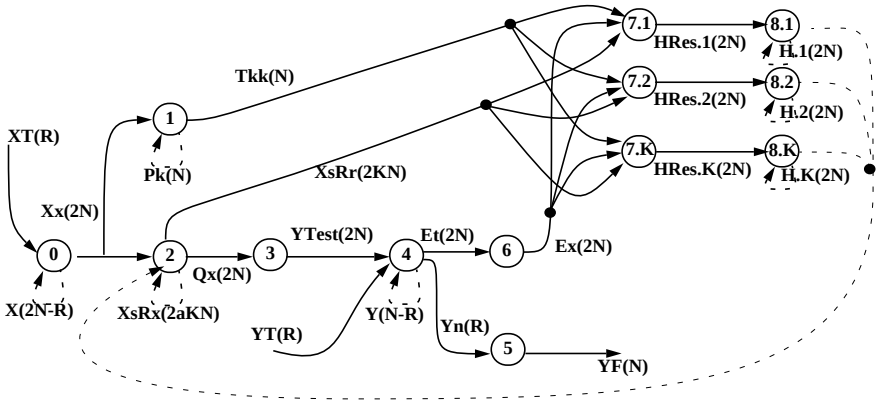


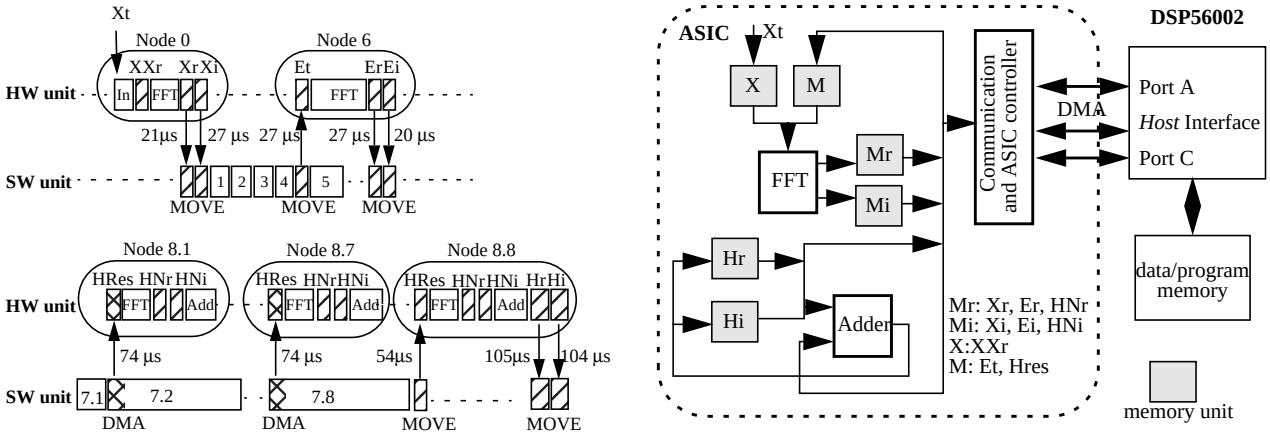
Fig. 9 • Flow graph of GMDFα

the application are: one DSP (DSP56002 from Motorola), several specific functional units (FFT, FFT^{-1},...) and some glue logic. Several hardware-software partitionings are possible, but two solutions are of

special interest. In the first one nodes 0, 6 and 8.i (i=1..K) are implemented in hardware. This solution minimize hardware area. The second partitioning (nodes 2, 3, 4, 5, and 7.i in hardware) provides an efficient trade-off between execution time and hardware area. The volume of HW/SW communication required in both partitioning is respectively V_1 = 4864 words and V_2 = 4992 words.

- **First partitioning**

In the first partitioning, mobilities of nodes enable to use only synchronous communications. Hence, no memorization resource is required. Hardware cost is due to control associated with synchronization mechanisms. Execution scheme (Fig. 10 •) is supported with DMA and I/O memory mapped technique in order to respect total execution time. Note that software to hardware data transfers after nodes 4 and 7.i (i<8) use different modes even if some parallelism exists between HW and SW units: the volume of computations of node 5 is too limited to exploit efficiently the DMA transfer mode of the DSP (see Table 1•). The DMA transfer in node 7.i has a duration of 25µs and overlaps during 74µs with computations of node 7.i+1. Each node 7.1 to 7.7 has an execution time of 574µs including the DMA transfer. At the end of node 7.8 the transfer is performed with explicit *move* instructions and requires 54µs. Consequently, the total elapse time due to hardware/software communications is 411µs and the total execution time of the GMDFα algorithm is 6.25ms that is less than the limit of 8ms imposed by the sampling. The ratio of HW/SW communications represents less than 7% of the total execution time. Utilization of the DMA mode allows to improve execution time of 2% compared to an execution with only a I/O memory mapped technique.



Scheduling and elapse times of communications with DSP56002

HW/SW architecture for GMDFα.

Fig. 10 • Hardware-software implementation of GMDFα.

- **Second partitioning**

Mobilities of nodes and dependencies between nodes do not allow to only use synchronous communications. The volume of asynchronous communication is 128 words what represent 2,56% of the total amount of communications. One memorization unit is needed in order to support all asynchronous data transfers and two bus for the synchronous ones. DMA and I/O memory mapped techniques are exploited. This second partitioning involves an extra hardware area compared to the first one. This overcost is mainly due to communication resources. The solution that will be considered for the following steps in the design flow (code generation and hardware synthesis) is the first one since a lower hardware area is obtained and timing constraints are met.

5. Conclusion

Interface synthesis represents one of the main step in the hardware-software system synthesis flow. Its aim is to define an interface supporting all the communications between heterogeneous resources. The interface determination takes place after hardware-software partitioning in order to be independent of partitioning heuristics even if communication estimations can be handled during the partitioning task. The communication interface must be adapted to the target architecture in order to obtain an efficient hardware-software implementation. In our case the target architecture is composed of asynchronous functional units since this computation model is well adapted to considered applications i.e., telecommunication and multimedia which are mainly data flow. The considered communication model is based on bufferized (through FIFOs) and non bufferized (bus) resources that allows to adjust the communication resources according to the transfer requirements. Furthermore, DMA and I/O memory mapped techniques are handled in order to take benefit of computation and transfer overlapping. Hence, the proposed method produces an interface that minimizes hardware area and respects timing constraints. Following steps in the codesign flow take into account the interface features to generate compact code and to synthesize efficient target architecture. But to achieve this result other tasks remain such as the construction of communication control sequences, which define the order of communications associated with a functional unit implementing several nodes and the optimization of communication links, which takes benefit of the times associated with data transfers in order to share communication resources (FIFOs or buses).

6. References

[1] AUGUIN M., BELLEUDY C., GOGNIAT G., JEGOU Y. A multi-granularity data synchronized architecture for HW/SW embedded DSP systems. *Proceedings Int. Conference on Signal Processing Applications & Technology.* Boston, october 7-10, 1996.

[2] BORIELLO G., KATZ R.H.,. Synthesis and Optimization of Interface Transducer Logic. *Proceeding of ICCAD*, pages 274-277. Nov., 1987.

[3] DAVEAU J.M., ISMAIL T.B., JERRAYA A.A. Synthesis of System-level communication by an allocation-based approach. *Int. Symposium on System Synthesis*, pages 150-155. Cannes - France, september 13-15, 1995.

[4] ERNST R., HENKEL J., BENNER T. Hardware-Software Cosynthesis for Microcontrollers. *IEEE Journal Design and Test of Computers.* 64-75, december, 1993.

[5] F. ROUSSEAU, J.M. BERGE, M. ISRAEL. Hardware/Software Partitioning for Telecommunication Systems. *COMPSAC'96*. Seoul, Aug., 1996.

[6] FILO D., KU D., COELHO C., De MICHELI G. Interface optimization for concurrent systems under timing constraints. *IEEE Trans. on VLSI*. 268-281, september, 1993.

[7] FREUND L., ISRAEL M., ROUSSEAU F., BERGE J.M., AUGUIN M., BELLEUDY C., GOGNIAT G. A codesign experiment in acoustic echo cancellation: GMDFα. *ISSS*. IEEE-ACM, La Jolla California, Nov. 6-8, 1996.

[8] GONG J., GAJSKI D., BAKSHI S. Model Refinement for Hardware-Software Codesign. *European Design & Test Conference*. Paris, France, March, 1996.

[9] GUPTA R.K., DE MICHELI G. Hardware-Software Cosynthesis for Digital Systems. *IEEE Journal Design and Test of Computers.* 29-41, september, 1993.

[10] HAYATI S.A, PARKER A.C, GRANACKI J.J,. Representation of Control and Timing Behavior with Applications to Interface Synthesis. *Proceeding of Int. Conf. Computer Design*, pages 382-387. Oct., 1988.

[11] KALAVADE A., LEE E. The extended partitioning problem: hardware/software mapping and implementation-bin selection. *Proceedings Int. Workshop on Rapid System Prototyping*, pages 12-18. Chapel Hill, NC, June 7-9, 1995.

[12] NARAYAN S., GAJSKI D. Interfacing Incompatible Protocols using Interface Process Generation. *Proc. Design Automation Conference*, pages 468- 473. San Francisco, CA, USA, June, 1995.

[13] NESTOR J.A., THOMAS D.E. Behavioral Synthesis with Interfaces. *Proceeding of Design Automat. Conf.*, pages 112-115. june, 1986.

[14] ROUSSEAU F., BENZAKKI J., BERGE J.M., ISRAEL M. Adaptation of force-directed scheduling for hardware/software partitioning. *Proceedings Int. Workshop on Rapid System Prototyping*. Chapel Hill, NC, June 7-9, 1995.

[15] SRIVASTA M.B., BRODERSEN R.W.,. SIERA: A Unified Framework for Rapid-Prototyping of System-Level Hardware and Software. *IEEE Transactions on Computer-Aided Design.* 14(6):676-693, June, 1995.

[16] SUN J.S., BRODERSEN R.W.,. Design of system interface modules. *in Proc. IEEE Int. Conf. Computer-Aided Design*, pages 478-481. Nov., 1992.

[17] VAHID F., GONG J., GAJSKI D. A binary-constraint search algorithm for minimizing hardware during hardware/software partitioning. IEEE CS Press (editor), *Proc. European Design Automation Conference*. 1994.