A Codesign Experiment in Acoustic Echo Cancellation: GMDFα

L. FREUND, M. ISRAEL, F. ROUSSEAU

LAMI, Université d'Evry
Bld des Coquibus
91025 Evry - FRANCE
{name}@lami.univ-evry.fr

J. M. BERGE

France Telecom - CNET
Chemin du Vieux Chêne
38243 Meylan - FRANCE
berge@cns.cnet.fr

M. AUGUIN, C. BELLEUDY, G. GOGNIAT

I3S, Université de Nice
CNRS, 41 bld Napoléon III
06041 Nice - FRANCE
{name}@alto.unice.fr

Abstract

Hardware/software codesign approaches consist generally in HW/SW partitioning and scheduling, constrained code generation, hardware and interface synthesis. This paper presents the codesign of an industrial experiment in acoustic echo cancellation (GMDFα algorithm) and emphasizes the partitioning and communication synthesis steps. This experiment points out interesting problems such as data and programs distribution between system memories and modeling communications in the partitioning process.

1. Introduction

The continuous advances in processor and ASIC technologies allow for the integration of increasingly complex specific systems. However, the improvements in design automation techniques for system production are increasing more slowly than those of integration capabilities of ASICs. Consequently, there is a strong interest in higher levels for system modeling and synthesis and particularly hardware/software codesign approaches ([11], [7], [18], [9] for example). Starting from a high level specification of a system, codesign techniques attempt to quickly find refined specifications of effective mixed implementations by a systematic exploration of various trade-offs. Codesign generally comprises several tasks including HW/SW partitioning and scheduling, constrained code generation, hardware and communication synthesis.

Acoustic echo cancellation is a good example of embedded system design since the problem of its real time implementation is still a challenging one, especially in broad band teleconference systems. Numerous algorithms have been proposed [10] but performances of the well known NLMS algorithm are too limited to deal with large teleconference rooms. The GMDFα algorithm [2] has good convergence and tracking performances and is a good candidate for echo cancellation. Nevertheless, for a long impulse response this algorithm involves numerous computations on a large set of data. Therefore, the real time implementation of the GMDFα algorithm constitutes a significant challenge for codesign methodologies.

The outline of this paper is as follows: Section 2 describes the GMDFα algorithm, Section 3 introduces our codesign approach and its application to the GMDFα algorithm, and the design results are presented in Section 4 followed by concluding remarks.

2. Description of GMDFα

2.1 Introduction

For some years, there has been considerable interest shown in improving high quality handfree telephone and acoustic conference applications. However, the acoustic coupling between the loudspeaker and microphone of each terminal is a significant operating difficulty.

One solution to this problem may be the application of the adaptive filtering method to an acoustic echo canceller. The adaptive filter corresponds to the impulse response of a FIR filter structure with variable coefficients. An algorithm computes the variation in filter coefficients and minimizes the matching error energy (Fig. 1). However, the adaptive filter length L may be several thousand coefficients at a sampling rate of 16 KHz. This fact leads to difficulties in real-time implementation of the algorithm.

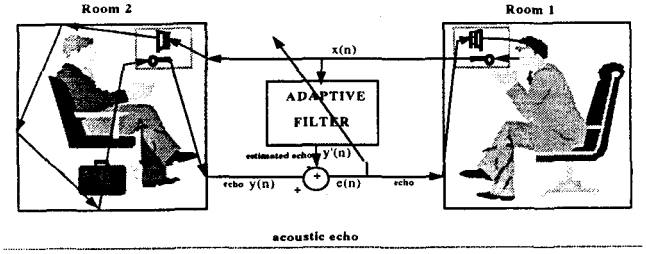

Fig. 1 : Acoustic Echo Cancellation method

The GMDFα (Generalized Multi-delay frequency Domain Filter) algorithm is one of the frequency-domain block adaptive algorithms. A block formulation involving a filter with fixed coefficients during N samples (N is the block size) allows various techniques (FFT, Fast FIR) to be applied in the frequency domain to reduce the arithmetic complexity.

0-8186-7563-2/96 $5.00

2.2 Specification of GMDFα

Different parameters define the complexity of the GMDFα algorithm: N is the block size, K the number of blocks, and L the filter length (with K = L / N). R new samples are processed at each iteration of the algorithm, and the filter is adapted α (overlapping factor) times per block (with R = N / α). The size of data (8, 16 bits for example), their type (integer, real), and their representation (fixed point, floating point) are other parameters, which affect implementation. This oversampling implies an increased arithmetic complexity, but allows a fast convergence rate. The functional description (Fig. 2) shows the arithmetic complexity of the algorithm.

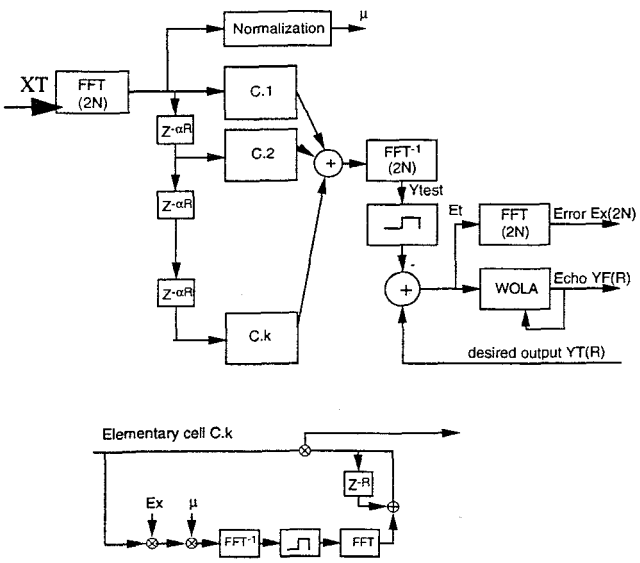

Fig. 2 : Functional description

The starting point of our experiment concerns the Directed Acyclic Graph (DAG) of the GMDFα, depicted in Fig. 3. This decomposition is manually obtained by study of the specification (functional description) or a VHDL model at the behavioral level. In this DAG G=(N, A), nodes N represent computation tasks and arcs A describe data transfert and control precedences between nodes. The precedence constraints between two nodes are expressed by an arc. 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 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 which 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) into an error signal 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 cell.

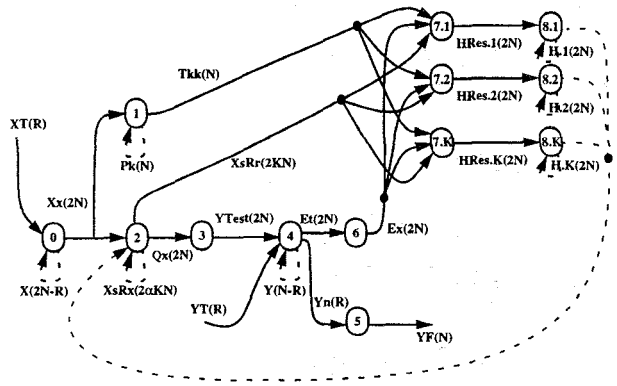

Fig. 3 : DAG of GMDFα

2.3 Software implementations

Several studies have tried to implement this algorithm on general purpose DSPs. [1] shows that a real time implementation is difficult on the TMS320C30 for a medium length filter (L = 1024). This real time implementation is very slow, compared with the maximum run time (the ratio is between 2 and 4). In [13], GMDFα is implemented on two TMS320C40s interconnected by a parallel bus. In this experiment, the second processor is active for only 50% of the total processing time, with a simplified version of GMDFα. This simplified version uses only 3 FFTs (or FFT^{-1}) instead of 2K + 3.

2.4 Hardware implementations

The hardware implementation has been studied in [5] and the feasibility has been demonstrated on a specific architecture. In this hardware implementation, some computations (division) are assumed to be executed by a processor. This architecture uses digital delay lines with an optimized area as memorization elements. It involves FFT and FFT^{-1} operators, and a specific Operating Unit (OU) composed of 2 multipliers with multiplexed inputs and one accumulator. It allows the execution of a complex multiplication in only two clock cycles. Memorization elements are composed of delay lines (a dynamic memory with sequential access). This solution of a read-write cycle in one clock cycle to be executed, thus enabling the operating unit to be used at maximum rate. In the 0.5 μm CMOS technology, the global area is about 15 mm^2, but this solution requires a very long design time.

3. Codesign of GMDFα

3.1 Decomposition of GMDFα

Regardless of these results, it seems interesting to mix hardware and software parts to implement GMDFα. Using

previous results, Table 1 summarize the execution times of DAG nodes.

Node	Hardware	TMS320C40	DSP56002
0	87	552	450
1	XXX	321	472
2	123	517	730
3	87	552	470
4	8	26	62
5	8	32	34
6	87	552	406
7.i	117	648	548
8.i	98	584	451
Total	2120 µs	12408 µs	10616 µs

Table 1 Execution times for various implementations

Hardware execution times come from [5] in the 0.5 µm CMOS technology. For the software part, the execution times of TMS320C40 and DSP56002 are given (in µs) with the following parameters: N=128, L=1024, α=2, data are words of 16 bits, real, and coded as fixed points. Following the sampling rate of the audio signal (16 KHz), the maximum execution time allowed for one iteration is 8 ms.

Since the algorithms used to implement each node and distribution of arrays into parallel memories of the DSPs are not identical for the two processors, there are differences in the node execution times. Note that the execution times of the TMS are estimated and those of the 56002 are real.

3.2 HW/SW partitioning

During the Hardware/Software Partitioning step, designers look for the best trade-off between hardware and software parts. We assume that in data flow systems (such as telecommunications systems), three problems have to be solved: *assignment* (choice of implementation), *scheduling* and *resource allocation*.

The main goal of the HW/SW partitioning is to find an *assignment* in hardware or software for all parts of the system specifications (all nodes of the DAG in our case). Their dates of execution are determined during *scheduling*. The study of scheduling allows different objectives to be targeted: minimization of operators, minimization of HW/SW communications, minimization of execution time for example. The problem of *resource allocation* is to find the type and number of resources used to implement the system. Trying to share resources between functionalities is the main difficulty. A resource can be an operator (more or less complex), such as UAL, adder. FFT or memorization elements such as register, FIFO, RAM.

In High Level Synthesis (HLS), scheduling and resource allocation problems exist. As they are NP-complete problems, numerous and various heuristic techniques have been published and provide good, but not optimal solutions. It seems that the two problems are similar in both HLS and HW/SW partitioning. One of the main differences concerns communications. In HLS, the communication times are generally ignored between functional units. In HW/SW partitioning for data flow systems, data can be vectors or matrices, and the partitioning step must take this parameter into account.

We have proposed a heuristic technique in [17] which solves scheduling and assignment problems for a DAG specification. This algorithm is an extension of the Force Directed scheduling algorithm [16], adapted for the HW/SW partitioning of tasks. It attemps to find the minimal cost schedule able to satisfy the set of constraints, given the global time constraint. Refer to [17] for further details and methodology.

3.3 HW/SW Partitioning of GMDFα

We propose two different HW/SW partitionings for GMDFα, obtained using our algorithm. The first implementation is a trade-off between execution time and hardware area. The scheduling and dates of execution are summarized in Table 2. The TMS320C40 (DSP56002 resp.) is active for 91% (90%) of the total processing time, but only 71% (57%) of the 8 ms corresponding to the sampling rate. The hardware operators used are one FFT^{-1} (1.8 mm^2) and one OU (0.5 mm^2). Note that the hardware execution of nodes 7.2 to 7.8 in the first partitioning overlaps with the software computation of nodes 8.1 to 8.7. Similar behavior is obtained for hardware nodes 8.1 to 8.8 in the second partitioning.

SW Nodes	HW Nodes	Scheduling 320C40 begin -> end (µs)	Scheduling 56002 begin -> end (µs)	Hardware operators
0		0 -> 551	0 -> 449	
1		552 -> 872	450 -> 921	
	2	552 -> 675	450 -> 572	OU
	3	676 -> 762	573 -> 659	FFT^{-1}
	4	763 -> 770	660 -> 667	OU
	5	771 -> 778	668 -> 675	OU
6		873 -> 1424	922 -> 1327	
	7.1	1425 -> 1541	1328 -> 1444	OU, FFT^{-1}
8.1...8.8		1542->...->6213	1445->...->5053	
		6213 µs	5053 µs	OU, FFT^{-1}

Table 2 First partitining of GMDFα

With the second partitioning (Table 3) which minimizes the hardware operator area, the TMS320C40 (DSP56002 resp.) is active for 97% (97%) of the total processing time, and 73% (70%) of the 8 ms. This solution requires only one FFT (1.8 mm^2) and one adder (0.07 mm^2). The number of data transfers between hardware and software units is $T_{com} = 2\ N\ (3 + 2\ K)$ words of 16 bits, i.e., $T_{com} = 4864$. These partitionings do not take into consideration any side effects on total execution time and area due to communications, controls and

allocation of DSP memories. Thus, in the following sections, synthesis of the complete architecture is described.

SW Nodes	HW Nodes	Scheduling 320C40 begin -> end (µs)	Scheduling 56002 begin -> end (µs)	Hardware operators
	0	0 -> 86	0 -> 86	FFT
1		87 -> 407	87 -> 558	
2		408 -> 924	559 -> 1288	
3		925 -> 1476	1289 -> 1758	
4		1477 -> 1502	1759 -> 1820	
5		1503 -> 1534	1821 -> 1854	
	6	1503 -> 1589	1821 -> 1907	FFT
7.1 7.2...7.8		1590 -> 2237 2238->...->6773	1908 -> 2455 2456->...->5743	
	8.8	6774 -> 6871	5744 -> 5840	FFT, Adder
		6871 µs	5840 µs	FFT, Adder

Table 3 Second partitining of GMDFα

3.4 Allocation of arrays and code sections to DSP memories.

After partitioning we obtain a set of nodes implemented by the software part. Since the DSP processors considered in our design contain internal data and program memories, the mapping of code and data sections corresponding to these nodes is of great importance for optimizing resource utilization. Furthermore, these DSP processors contain two internal parallel data memories to sustain the data throughput with the core. A process may be accelerated by a factor of up to 3 if its data and instructions are properly moved from outside to inside. In the same way, the throughput of communications can be accelerated if data are placed in internal memories. At present, the mapping of arrays and code is performed after partitioning but it would be desirable to consider this operation during the partitioning step[12]. Firstly, arrays accessed by nodes are distributed among the parallel memories in order to maximize the possibilities of parallel moves of data. This mapping does not depend on the internal/external allocation. Hence, the aim of internal/external allocation of code sections and arrays is to assign to internal memories objects that have high utilization rates. This allocation can be performed for example with a knapsack algorithm[3]. After array location in memories, more accurate data tranfer behaviors may be determined for communication synthesis.

Generally, data array transfer is performed either by direct access mode (DMA) or by memory mapped I/O move operations. The selection of 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 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 modeling the behavior of this DSP is more complex since numerous conflicts may occur due to movements of instructions and data sharing buses. In Table 4, approximate models of the total execution times on these DSPs for a transfer of N values and C cycles of CPU operations ($C \geq 0$) are depicted. Note that for DMA transfers these two tasks overlap, and models are very different due to the disparate architectures of the DSPs. Notation *ext->int* represents a transfer from an external to an internal data memory of the TMS320C40. Interactions between data and instruction flows are not modeled here. The execution time on the DSP56002 depends on the location of instructions and data. One extremity of the I/O transfer is necessarily external to the DSP. Notation *int/ext* concerning 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.

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 $C \leq 8N$ => 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 4 Number of clock cycles for N I/O operations and C CPU operations

3.5 Synthesis of communications

Data transfers in the GMDFα application deal with arrays of data. Thus synthesis of communications consists in determining for each HW/SW transfer:
- the type of transfer (synchronous or asynchronous),
- the needed hardware support and the associated protocol,
- the transfer mode (DMA or memory mapped I/O).

Previous works have focused on communication synthesis after HW/SW partitioning. In [4] and [15] the aim is to maximize utilization bandwidths of buses by analyzing peak and average data transfer rates over communication channels but they do not consider the scheduling of tasks after partitioning. However, this scheduling constitutes a set of timing constraints that communications must respect. In [14] the synthesis of channels uses as specification a sequence of timed events each one corresponding to a single data transfer. Transfers of arrays is not supported by this approach. The most relevant work to our problem is presented in [6]. Interface optimization attempts to maximize the use of non-blocking protocol in order to minimize control logic on channels. But the side effects consist in the introduction of control delays in both sender and receiver that impose a global reference clock.

In our approach [8], to avoid the problem due to a global reference we consider that both sender and receiver do not share the same clock. With such an assumption communica-

tions can be handled thanks to an asynchronous or a synchronous transfer. The asynchronous transfer allows both sender and receiver to be independent at the communication schedule, but hardware support may be important because all the transferred data need to be memorized, for example through FIFOs. The synchronous transfer needs less resources since handshake signals and data buses are only required, but with that type of transfer, sender and receiver need to be synchronized at the communication schedule in order to exchange data without memorization. The synchronization mechanism corresponds to a rendez-vous primitive. This synchronization may introduce delays with respect to the initial schedule. Consequently, before allocating a synchronous transfer to a communication between two nodes, it must be checked that the total time required to execute the application is not overstepped. In our communication synthesis method we minimize the hardware area by using as much as possible synchronous transfer even if a local reschedule is necessary. The result must respect the total execution time settled for the application.

The next step in the communication synthesis flow outlines the nature of communication protocols. Two types of communication protocols are considered: blocking or non-blocking. When using a blocking protocol for a communication, the sender (resp. receiver) must verify that the receiver (resp. sender) is ready before starting the transfer. On the contrary, when using a non-blocking protocol no verification by the sender and the receiver is performed since all the communication resources requested for the transfer are assumed to be available (for example a buffer). In our case, sender and receiver use blocking protocols if a synchronous transfer type is applied. For an asynchronous transfer the receiver uses a blocking protocol to ensure that the data emitted for that communication are all available. The sender uses a non-blocking protocol since requested memorization elements (FIFO) are allocated.

When protocols are defined, transfer modes can be associated with all communication links. The selection of the transfer mode depends on the capabilities integrated in the target processor. As mentioned above most recent processors can handle DMA transfers and memory mapped I/O move operations. The choice of the transfer mode is done to ensure the best communication timing performance. We use the cost function given in Table 4 to determine the transfer mode. This cost function takes into account the volume of data to be transferred and the total amount of potential instructions that can overlap the transfer.

These steps define all the communication supports. Next, instruction threads that manage the communications are generated for the processor and a controller that handle the hardware communication signals is synthesized. The communication interface produced allows the complete execution of the application.

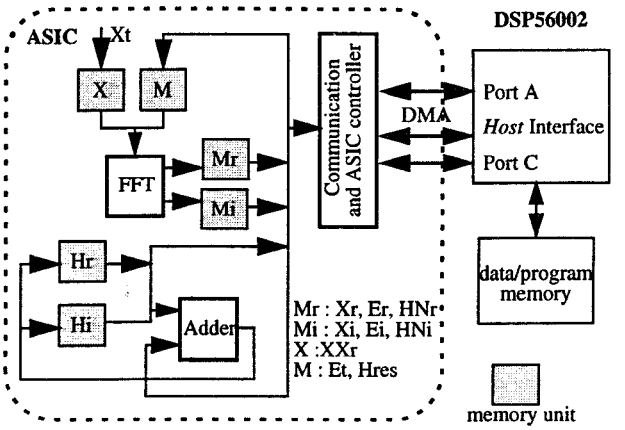

Fig. 4 : HW/SW architecture for GMDFα.

After synthesis of the whole hardware entities we get the architecture depicted in Fig. 4 corresponding to the second partitioning. Data arrays pointed out in the figure share the same memory units in the ASIC part. Since execution times of hardware nodes of GMDFα are small compared to those of the software nodes, there is room for rescheduling hardware nodes without increasing the overall execution time. Thus, all communications are implemented using a synchronized transfer. Data are not buffered and are exchanged directly from the internal memory of the sender to the internal memory of the receiver.

4. Design results

The following results are given for the DSP56002. The partitioning of the GMDFα application leads two solutions. If delays due to communications are omitted and data and code sections are both mapped into external memories of the DSP, the total execution times of these solutions are 12.5ms and 13.96ms. However, if location of array and code sections is optimized between external and internal memories, the execution time of the second partitioning is reduced by a factor of 2 (5.84 ms). This point illustrates the importance of considering different software implementations of nodes during the partitioning/scheduling of the application [12].

Elapse times for communications associated with the second partitioning are given in Fig. 5.

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 efficiently exploit the DMA transfer mode (see Table 4). The DMA transfer in node 7.i has a duration of 25µs and overlaps for 74µs with the computations of node 7.i+1. Each node from 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 ex-

ecution time of the GMDFα application is 6.25ms, i.e. less than the limit of 8 ms imposed by the sampling.The ratio of HW/SW communications represents less than 7% of the total execution time.The idle time of the DSP is only 2% of the total execution time whereas the idle time of the hardware unit is 77%. The second partitioning attempts to minimize the hardware area, and thus helps to maximize the utilization of the DSP.

The required sizes of program and data memories in the DSP56002 are 528 words and 9.4Kwords respectively. Internal/external distribution of data arrays is of prime importance with this DSP since only 512 words of internal data memories are available.

The area of the hardware unit including the DMA controller, FSM controller, 2.8Kwords of data memories, one FFT module and one adder is 4.5mm^2 in a 0.5 µm technology.

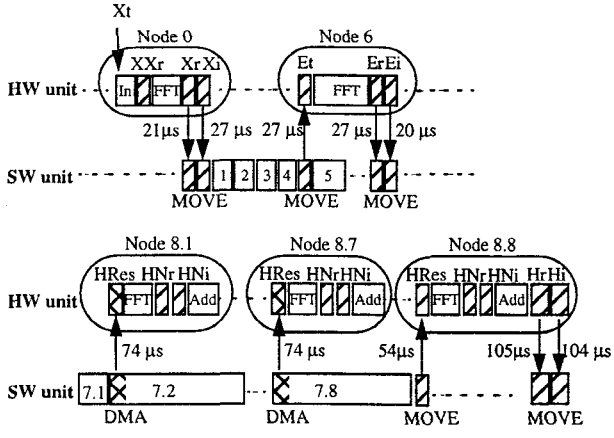

Fig. 5 : Scheduling and elapse times of communications with DSP56002

5. Conclusion and future works

The codesign flow for an acoustic echo cancellation algorithm described by a DAG was presented. An adapted force directed scheduling is applied on nodes of the DAG to distribute tasks to hardware and software units. With the knowledge of this partitioning/scheduling, communications are synthesized and the global architecture is constructed. This experience illustrates interesting results. The use of DSP or core processors implies to consider the allocation of data and program into internal or external data memories during the partitioning phase. Consequently, different software solutions have to be considered during partitioning/scheduling/allocation of tasks.

Moreover, since delays due to data transfers between HW and SW units are very dependent on this data distribution, effects of communications on produced solutions would be considered during partitioning of the specification in order to exhibit HW/SW systems that respect firmly cost and timing constraints. Futhermore, some intensive computing applications require transfers of data arrays between processing units. These transfers may be pipelined or overlapped with computations of nodes. Then, more efficient solutions would be achieved if these communication schemes are considered during partitioning step.

Generally, codesign methodologies consider target architectures composed of one ASIC and a single processor. But complex applications (for example full MPEG2 codec for videoconference) that are (soon) integrable on one chip require often more sophisticated systems able to exploit different levels of grain of parallelism. Therefore, there is a challenge in more general system architectures and co-synthesis methods able to deal with these intensive computing applications.

6. References

[1] AIT AMRANE O., *Identification des systèmes à réponse impulsionnelle longue par filtre adaptatif en fréquence: Application à l'annulation d'échos acoustiques.* Thèse de Doctorat, ENST - Paris, France, september, 1992 (in french).

[2] AMRANE A., MOULINES E., GRENIER Y., Structure and convergence analysis of the generalized multi-delay adaptative filter. *Int'l. Conf. EUSIPCO-92*, Brussels, August, 1992.

[3] CORMAN T.H., LEISERSON C.E., RIVEST R.L., *Introduction to algorithms.* MIT Press, 1992.

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

[5] EL HELWANI A., LESCAN P., VLSI Architecture of the Generalized Multi Delay Frequency-Domain Algorithm for Acoustic Echo Cancellation. *Proc. of ICASSP.* Detroit, may 9-12, 1995.

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

[7] GAJSKY D., VAHID F., Specification and design of embedded hardware-software systems. *IEEE Journal Design and Test of Computers.* pages 53-67, spring, 1995.

[8] GOGNIAT G., *Etude de la synthèse des communications dans les systèmes logiciels/materiels.* Technical Report RR96-12, I3S, Sophia-Antipolis, France, march, 1996 (in french).

[9] GUPTA R.K., COELHO C.N., De MICHELI G. Program implementation schemes for hardware software systems. *IEEE Computer Journal.* 48-55, january, 1994.

[10] HANSLER E., The hands-free telephone problem: an annoted bibliography. *Signal Processing.* 27259-271, 1992.

[11] JERRAYA A., ISMAIL T., Synthesis steps and design models for codesign. *IEEE Computer,* 28(2):44-52, february, 1995.

[12] 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.

[13] LE TOURNEUR G., THOMAS J.P., GILLOIRE A., Real time implementation of the GMDF Alpha algorithm on a multi DSP TMS320C40 board. *EUSIPCO-94*, pages 1007-1010. Edinburg - Scotland, september 13-16, 1994.

[14] MADSEN J., HALD B., An approach to interface synthesis. *Int. Symposium on System Synthesis*, pages 16-21. Cannes-France, september 13-15, 1995.

[15] NARAYAN S., GAJSKI D., Synthesis of System-level Bus Interface. *European Conference on Design Automation*, pages 395-399. Paris-France, february, 1994.

[16] PAULIN P.G., KNIGHT J.P., Force-Directed Scheduling for the Behavioral Synthesis of ASIC's. *IEEE Trans. on Computer-Aided-Design.* 8(6):661-679, june, 1989.

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

[18] WOO N.S., DUNLOP A.E., WOLF W., Codesign from cospecification. *IEEE Computer Journal.* 42-47, january, 1994.