A Codesign Experiment in Acoustic Echo Cancellation: GMDFα

L. FREUND, M. ISRAEL
Université d'Evry
F. ROUSSEAU
Ecole Supérieure d'Ingénieurs de Marseille
J. M. BERGÉ
France Telecom
and
M. AUGUIN, C. BELLEUDY, G. GOGNIAT
CNRS, Université de Nice Sophia-Antipolis

Continuous advances in processor and ASIC technologies enable the integration of more and more complex embedded systems. Embedded systems have become commonplace in recent years. Since their implementations generally require the use of heterogeneous resources (e.g., processor cores, ASICs) in one system with hard design constraints, the importance of hardware/software codesign methodologies increases steadily. HW/SW codesign approaches consist generally of HW/SW partitioning and scheduling, constrained code generation, and hardware and interface synthesis. This article presents the codesign of an industrial experiment in acoustic echo cancellation (GMDFα algorithm) and emphasizes the partitioning and communication synthesis steps. This experiment brings to light interesting problems such as data and program distribution between system memories and the modeling of communications in the partitioning process.

Categories and Subject Descriptors: C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems

General Terms: Design, Experimentation

Additional Key Words and Phrases: Codesign, communications, HW/SW partitioning

Authors' addresses: L. Freund, M. Israel, LAMI, Université d'Evry, Boulevard des Coquibus, 91025 Evry, France; F. Rousseau, ESIM, Technopôle de Chateau Gombert, 13451, Marseille cedex, France; J. M. Bergé, CNET, Chemin du Vieux Chêne, 38243 Meylan, France; M. Auguin, C. Belleudy, G. Gogniat, I3S, CNRS, Université de Nice Sophia-Antipolis, 41 Boulevard Napoleon III, 06041 Nice cedex, France; email: ⟨auguin@alto.unice.fr⟩.

1. INTRODUCTION

Embedded systems are being used in an expanding number of domains. Examples can be found in medical devices, automobile cruise control, and videoconference and wireless applications. Their complexity has significantly increased due to integration density improvements and sophistication of digital applications. To achieve a cost-effective implementation within design constraints (e.g., hardware cost, performance, power consumption) heterogeneous hardware and software resources must in general be considered. Examples of resources include RISC and DSP cores and ASIC parts. Hand-crafted design of significant parts of hardware/software systems is no longer sufficient due to the high complexity and the time-to-market pressure. Since design automation techniques for system production improve more slowly than the integration capabilities of ASICs, there is a major interest in system modeling and synthesis at high level and particularly in hardware/software codesign.[1] From a high-level specification of a system, codesign techniques attempt to quickly find refined specifications of mixed implementations by a systematic exploration of various tradeoffs. Commonly, codesign comprises several tasks, including system specification, HW/SW partitioning and scheduling, constrained code generation, communication synthesis, and HW/SW integration. Each task is itself composed of complicated problems and is still an area of research investigation. Providing efficient HW/SW embedded architectures requires not only that each step be itself efficient but also that taken together, these steps constitute an effective methodology. Consequently, in order to ensure the development of powerful codesign frameworks, it is important to consider case studies.

Acoustic echo cancellation is a good example of embedded system design since its real-time implementation is still a challenging problem, especially in wide-band teleconference systems. Numerous algorithms have been proposed [Hansler 1992] but performances of the well-known NLMS algorithm are too limited to deal with large teleconference rooms. The GMDFα algorithm [Amrane et al. 1992] has good convergence and tracking performance and is a good candidate for echo cancellation. Nevertheless, for long impulse responses, this algorithm implies 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 article is as follows. Section 2 describes the GMDFα algorithm and Section 3 introduces our codesign approach and its application to the GMDFα algorithm. Design results are presented in Section 4 followed by concluding remarks.

2. DESCRIPTION OF GMDFα

For some years, there has been considerable interest in improving high quality handfree telephone and acoustic conference applications. However,

[1]See Adams and Thomas [1996], Chou et al. [1995], Gajski and Vahid [1995], Gupta et al. [1994], Jerraya and Ismail [1995], and Woo et al. [1994].

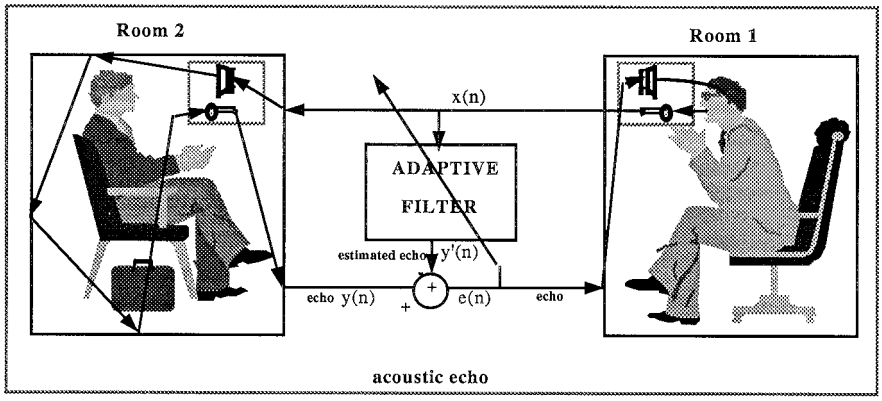

Fig. 1. Acoustic echo cancellation method.

the acoustic coupling between the loudspeaker and the microphone of each terminal is a significant operating difficulty.

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

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

2.1 Specification of GMDFα

Different parameters define the complexity of the GMDFα algorithm: N is the block size, K the number of blocks, 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 $R = N/\alpha$). The size of data samples (e.g., 8 or 16 bits), their type (integer, real), and their representation (fixed point, floating point) are other parameters that affect implementation. The oversampling implies an increased arithmetic complexity, but allows a fast convergence rate. The functional description (Figure 2) shows the arithmetic complexity of the algorithm.

The starting point of our experiment concerns the directed acyclic graph (DAG) of the GMDFα, depicted in Figure 3. This decomposition is obtained by hand at the procedure level from the VHDL model of GMDFα, using HW and SW units issued from a library developed at CNET. Further investigation could be performed to determine the best level of decomposition [Henkel and Ernst 1996]. The precedence constraints between two nodes

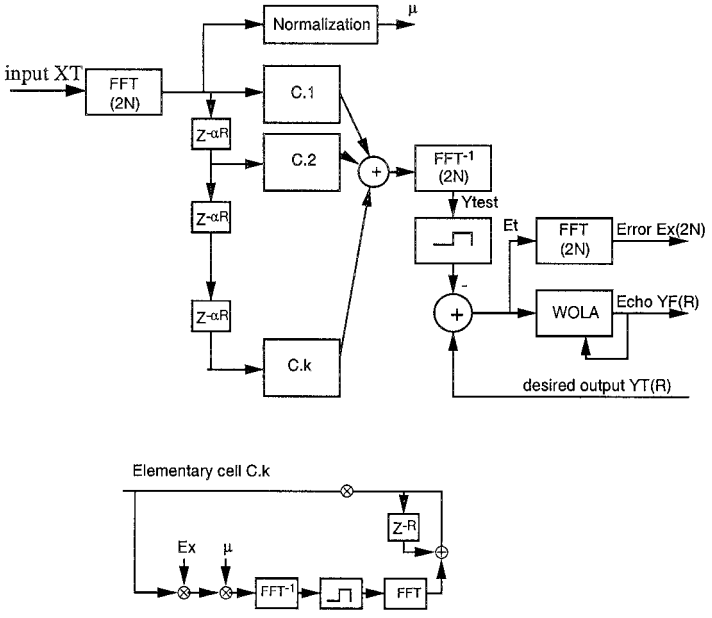

Fig. 2. Functional description of GMDFα.

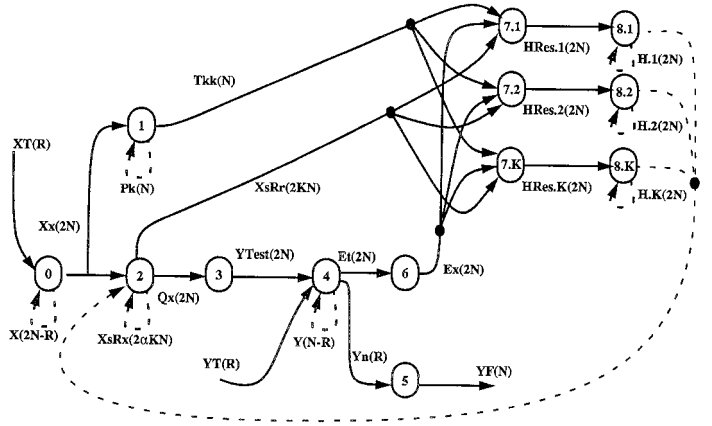

Fig. 3. Direct acyclic graph of GMDFα.

are expressed by an edge. The name and the volume of data are given on each edge. A dotted edge means that the data will be used at the next iteration of the algorithm. These edges are opened for partitioning to operate on a DAG.

Node 0 computes an 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 inverse FFT (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 updated by FFT and inverse FFT operations.

2.2 Software-Only Implementation

Several studies have tried to implement this algorithm on general purpose DSPs. Work presented in Amrane [1992] shows that a real-time implementation was difficult on the TMS320C30 for medium length filter (L = 1024, K = 8, α = 2). This real-time implementation is very slow, compared with the maximum run-time (the ratio is between 2 to 4). In Le Tourneur et al. [1994], GMDFα was implemented (L = 1024, K = 4, α = 4) on two TMS320C40s interconnected by a parallel bus. But in this experimentation only a simplified version of GMDFα has been implemented. This simplified version uses three FFTs (or inverse FFT) instead of 2K + 3.

2.3 Hardware-Only Implementation

The hardware implementation has been studied in El Helwani and Lescan [1995] 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 involves FFT and inverse FFT operators, and a specific operating unit (OU) composed of two 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 allows the execution of a read-write cycle in one clock cycle, which enables the operating unit to be used at maximum rate. In 0.5 μm CMOS technology, the global area is about 15 mm^2, but this solution requires a very long design time and a processor for the division operation.

3. CODESIGN OF GMDFα

Considering these results, it seems interesting to mix hardware and software parts to implement GMDFα. Using previous results, we collected in Table I the execution times of the nodes of the DAG. Hardware execution times proceed from El Helwani and Lescan [1995] in 0.5 μm CMOS technology. For the software part, execution times of the TMS320C40 and DSP56002 are given (in μs) with the following parameters: N = 128, L = 1024, α = 2, K = 8, data are words of 16 bits, real, coded as fixed points. Following the sampling rate of the audio signal (8 KHz), the maximum execution time allowed for one iteration is 8 ms. Since the algorithms used to implement each node and the distributions of arrays into parallel memories of the DSPs are not identical for the two processors, there are differences between node execution times. Notice that the execution times for the TMS are estimated whereas those for the 56002 are actual ones.

Table I. Execution Times for Various Implementations

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

3.1 HW/SW Partitioning

During the hardware/software partitioning step, designers look for the best tradeoff between hardware and software parts. We assume that in data-flow systems (such as telecommunication systems), three problems have to be solved: the *assignment* (choice of implementation), the *scheduling*, and the *resource allocation*. Several heuristics dealing with these classical problems have been published which differ by their underlying methodologies and objectives (e.g., the considered target architecture can be monoprocessor [Kalavade and Lee 1994] or multiprocessor [Lee and Potkonjak 1996; Bjorn-Jorgensen and Madsen 1997]).

The main goal of the HW/SW partitioning is to find an assignment to hardware or software units for all parts of the system specification (i.e., all nodes of the DAG in our case). The times at which their execution starts are determined during scheduling. The study of assignment and scheduling allows different objectives to be aimed at, for example, minimization of operators or HW/SW communications or execution time. The problem of resource allocation is to find the type and the number of resources used to implement the system. In our approach we consider a target architecture composed of a single DSP and HW units connected with a point-to-point topology. Trying to share resources among tasks is the main difficulty. Hardware resources can be operators, such as ALU, adder and FFT, or memorization elements such as register, FIFO, and RAM.

In high level synthesis (HLS), scheduling and resource allocation problems exist. Since these are NP-complete problems, various heuristics have been published that provide good, but nonoptimal solutions. In HW/SW partitioning these two problems are similar. But a main difference is due to the two-dimensional HW/SW space where each dimension has its own constraints. For example, the SW unit imposes sequential execution of tasks whereas the HW unit permits parallel execution of tasks. We propose an extension of the force directed scheduling algorithm [Paulin and Knight 1989] introduced for HLS. This heuristic solves scheduling and assignment

problems for a DAG specification [Rousseau et al. 1995]. It tries to find the minimal cost that satisfies the global time constraint.

However, other differences between HLS and HW/SW partitioning exist. For example, in HLS, communications between functional units deal generally with a single data word whereas in HW/SW partitioning, computations and transfers may involve vectors or matrices. This point is not currently considered in our heuristic, but will be taken into account in future works.

3.1.1 *Partitioning Algorithm.* The partitioning method considers the time step notion well known in high-level synthesis [Paulin and Knight 1989]. For each node, execution times of possible implementations (ts_i and th_i) are issued from a library and are given in time steps as the total execution time constraint of the application. At each iteration, the algorithm proceeds through all the nodes in the graph and schedules one of them according to the selected cost function. The designer's cost function determines the goal of the partitioning.

An ASAP (as soon as possible) scheduling and an ALAP (as late as possible) scheduling determine the time frame of each node. For each node i and each time step j of the time frame a couple $F_j(i)$ of "repel forces" $f^j_{soft}(i)$ and $f^j_{hard}(i)$ are calculated:

$$F_j(i) = \{f^j_{soft}(i), f^j_{hard}(i)\}.$$

Each repel force is the sum of a "self force" and a "total induced force." The "self force" is the quantity that reflects the effect of an assignment of a node to a time step regardless of the impact on other nodes. Assigning a node to a time step may reduce the time frames of other nodes. Thus, to take this constraint into account, an "induced force" is defined. For a graph of n nodes, the "total induced force" of a node i expresses the sum of the constraints induced by node i on all the other nodes:

$$f^j_{soft}(i) = Self\_force^j_{soft}(i) + \sum_{k=1\ (k\neq i)}^{n} Induced\_force(k)$$

$$f^j_{hard}(i) = Self\_force^j_{hard}(i) + \sum_{k=1\ (k\neq i)}^{n} Induced\_force(k).$$

The higher the repel force is, the higher is the cost of assigning the node to this time step (Figure 4). A repel force can take the value $+\infty$ if it is not possible to compute the repel force: for example, due to the sequential nature of the software unit, the repel force $f^j_{soft}(i) = +\infty$ if the processor is already active at time step j executing another node. The self force of a node i is the sum of the resource costs induced by this node. But if the architecture already contains parts of these resources their sharing allows

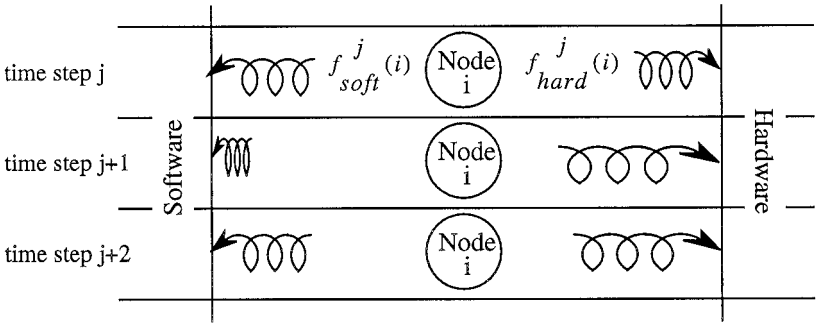

Fig. 4. Example of repel forces.

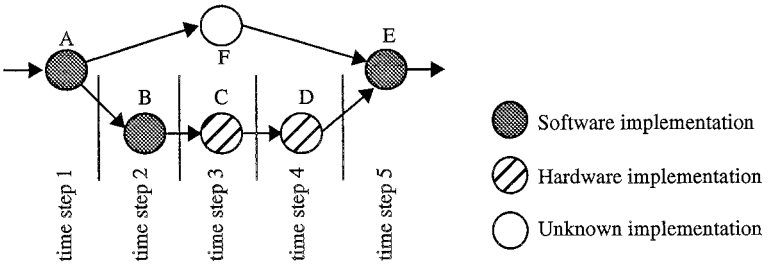

Fig. 5. Evaluation of time frame.

reducing the self force:

$$Self\_force^{j}_{soft}(i) = \sum_{x=1}^{p} A^{j}_{s_x} \times Cost(s_x)$$

$$Self\_force^{j}_{hard}(i) = \sum_{x=1}^{q} A^{j}_{h_x} \times Cost(h_x),$$

where $\{s_1, s_2, \ldots, s_p\}$ (respectively, $\{h_1, h_2, \ldots, h_q\}$) are software (respectively, hardware) resources needed to execute node i at time step j. The variable A^{j}_{r} denotes the availability of the resource in the architecture: A^{j}_{r} is equal to 0 if at time step j there is a free resource r in the architecture and 1 otherwise. To compute the time frame of a node, the smallest execution time of possible implementations is used. This determines the largest time frame but some adjustments can be necessary.

Consider the case depicted in Figure 5 where nodes A, B, C, D, and E are assigned to HW and SW units and the total execution time constraint is 5 time steps. Assuming that the HW and SW execution times of node F are $ts_{\mathrm{F}} = 2$ and $th_{\mathrm{F}} = 1$, the time frame of node F is three time steps (time step 2 through time step 4) but the evaluation of $f^{4}_{soft}(F)$ is not possible since the time constraint is violated. Therefore we get $F_4(F) = \{+\infty, f^{4}_{hard}(F)\}$. In the

```
Repeat until all nodes are scheduled
    Evaluate time frames using ASAP and ALAP schedules;
    Calculate the self forces for every feasible time step;
    Sum the total induced force;
    Schedule the node with the lowest repel force;
End repeat;
```

Fig. 6. Partitioning algorithm.

same way, since the processor executes the task B at time step 2, it cannot execute the task F: $F_2(F) = \{+\infty, f^2_{hard}(F)\}$. Only at time step 3 $f^3_{soft}(F)$ can be evaluated.

The total induced force due to a node i expresses the sum of constraints induced by node i on all other nodes of the graph. We define the induced force on a node k, $k \neq i$, as the mean value of its software self forces and its hardware self forces. If we assume that there are c (resp., d) time steps for which its software (resp., hardware) self force exists (not equal to $+\infty$), then the induced force is defined as:

$$Induced\_force(k) = \frac{\sum_{j=1}^{c} Self\_force^j_{soft}(k) + \sum_{j=1}^{d} Self\_force^j_{hard}(k)}{c + d}.$$

When there is no possible implementation for node k ($c = 0$ and $d = 0$), induced_force(k) is set to $+\infty$. The global partitioning algorithm may be summarized as illustrated in Figure 6.

The node i with the lowest repel force $F_j(i)$ induces the weakest constraints on other nodes. This node is scheduled at time step j. The choice between the hardware or software implementation is given by the smallest repel force that composes $F_j(i)$. Due to the assumption of a single processor in the target architecture, if node i is assigned to the software unit, all software repel forces of unscheduled nodes cannot be evaluated over time steps corresponding to the execution of node i. This model describes the sequential nature of the software unit.

3.1.2 *Application to GMDFα.* Two different HW/SW partitionings for GMDFα are automatically provided with our heuristic. The first implementation is a tradeoff between execution time and hardware area. The scheduling and the dates of execution are summarized in Table II. 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. Hardware operators used are one inverse FFT (1.8 mm^2) and one OU (operating unit of 0.5 mm^2). The amount of data transfers between hardware and software units is $T_{com} = N(7 + 4K) = 4992$ words of 16 bits. Notice that hardware execution of nodes 7.2 to 7.8 in the first partitioning overlaps with software computation of nodes 8.1 to 8.7. A similar behavior is obtained for hardware nodes 8.1 to 8.8 in the second partitioning depicted in Table III. In this partitioning which minimizes hardware operator area, the TMS320C40 (resp., DSP56002) is active for 97% (96%) of

Table II. First Partitioning of GMDFα

Software Nodes	Hardware 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 III. Second Partitioning of GMDFα

Software Nodes	Hardware 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->...->6291	
	8.8	6774 -> 6871	6292 -> 6389	FFT, Adder
		6871 μs	6389 μs	FFT, Adder

the total processing time, and 83% (77%) of the 8 ms. This solution requires only one FFT (1.8 mm^2) and one adder (0.07 mm^2). The amount of data transfers between hardware and software units is $T_{com} = 2\ N\ (3 + 2\ K)$ words of 16 bits (i.e., $T_{com} = 4864$). This partitioning does not consider side effects on total execution time and area due to communications, controls, and allocation of DSP memories. The synthesis of the complete architecture is described in the following sections.

3.2 Allocation of Arrays and Code Sections to DSP Memories

After partitioning we get a set of nodes associated with the software part. Since DSP processors considered in our design contain internal data and

Table IV. Number of Cycles for N I/O Operations and C CPU Operations

Transfer mode	TMS320C40 (40ns)	DSP56002 (25ns)
DMA	ext->int : Max(2N+14,C) int->ext : Max(3N+15,C) ext->ext: Max(4N+13,C)	C>8N => C+4N+4 C ≤ 8N => 12N + 4
Move	ext->int : 2N+4+C int->ext : 3N+4+C ext->ext: 4N+4+C	int/int : C+4N+8 int/ext : C+6N+8 ext/ext: C+8N+8

program memories, the mapping of code and data sections corresponding to these nodes is of great importance to optimize 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 up to 3 if its data and instructions are properly moved from outside to inside. In the same way, throughput of communications can be sped up if data are placed in internal memories. At this time we perform the mapping of arrays and code after partitioning but it would be better to consider this operation during the partitioning step [Kalavade and Lee 1995]. First, 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. Then, 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 [Corman et al. 1992]. After localization of arrays in memories more accurate behaviors of data transfers may be settled for communication synthesis.

Generally, a transfer of an array of data is performed either by direct memory access (DMA) or by CPU controlled I/O move operations [Vercauteren et al. 1996]. 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; that is, 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 modeling the behavior of this DSP is more complex since numerous conflicts may occur due to instruction and data movements sharing buses. In Table IV approximate models of the total execution times on these DSPs for a transfer task of N values and a CPU task of C cycles ($C \geq 0$) are depicted. Notice that for DMA transfers these two tasks overlap and models are very different due to the disparate architectures of the DSPs. Notation $ext \rightarrow int$ represents a transfer from an external data memory to an internal data memory of the TMS320C40. Interactions between data and instruction flows are not modeled here. Execution time on the DSP56002 depends on the location of instructions and data. Either the source or the destination of the I/O transfer is necessarily external to

the DSP. Notation *int*/*ext* about the DSP56002 means that instructions controlling the I/O transfer are located in the internal program memory of the DSP and data to be exchanged are placed in the external data memory of the DSP.

3.3 Synthesis of Communications

Synthesis of communications deals with communication edges connecting nodes assigned to different units. This step consists of determining for each communication edge:

—the type of transfer (synchronous or asynchronous),
—the needed hardware support and the associated protocol, and
—the transfer mode (DMA or CPU controlled I/O).

Previous works have focused on communication synthesis at different levels of abstraction. Methods proposed by Borriello and Katz [1987], Hayati et al. [1988], Nestor and Thomas [1986], and Narayan and Gajski [1995] consider low-level descriptions of communication protocols and take place after HW/SW partitioning. The synthesized hardware architectures are based on finite state machines with resource optimization. In these approaches, designers need to have a precise knowledge of requested protocols. For complex systems using heterogeneous resources these techniques may become inextricable. Thus, to alleviate this problem, other approaches consider a higher level of abstraction. In Daveau et al. [1995] and Narayan and Gajski [1994] the aim is to maximize bandwidth utilization of buses by analyzing peak and average data transfer rates over communication channels but they do not consider the scheduling of tasks defined by the partitioning. However, this scheduling constitutes a set of timing constraints that communications must respect. In Madsen and Hald [1995] the synthesis of channels uses as specification a sequence of timed events each corresponding to a single data transfer. Transfers of arrays are not supported by this approach. Since data transfers in the GMDFα application deal with arrays of data, the most relevant work to our problem is presented in Filo et al. [1993]. Interface optimization attempts to maximize the use of nonblocking protocols in order to minimize control logic on channels. But the side effects consist of adding control delays in both sender and receiver that impose a global reference clock.

In our approach [Gogniat 1996; Gogniat et al. 1997] 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 communications can be handled thanks to an asynchronous or a synchronous transfer. An asynchronous transfer is used when the sender and the receiver cannot be synchronized to communicate. Hence, data emitted by the sender are memorized in a FIFO. The receiver can then access the data located in the FIFO independently of the sender. The synchronous transfer is implemented using a rendezvous mechanism between the sender and the receiver. This type of communication needs fewer resources since handshake

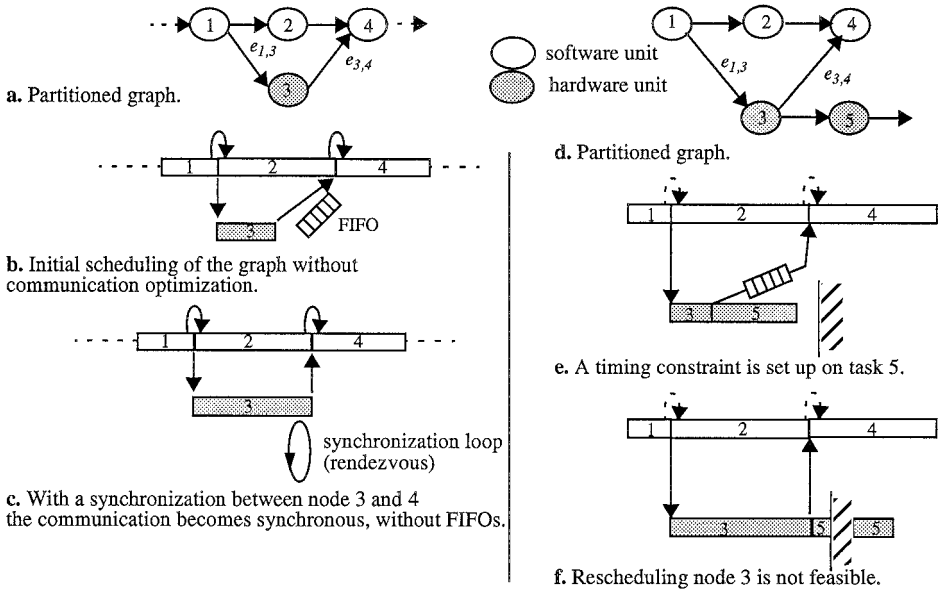

Fig. 7. Local rescheduling of nodes.

signals and data buses are only required. However, the synchronization (synchronization loop executed by the node) may introduce delays on other nodes (local rescheduling) with respect to the initial schedule if one unit is not ready to communicate. Consequently, before allocating a synchronous transfer to a communication between two nodes, it must be verified that the timing cost due to the rendezvous protocol does not lead to a violation of the timing constraint of the specification. Figures 7 (a), (b), and (c) depict an example of a feasible local rescheduling whereas Figures 7 (d), (e), and (f) illustrate the case of an imposed asynchronous communication due to the timing constraint imposed on node 5. In our communication synthesis method we minimize the hardware area by using synchronous transfers as often as possible. The result must respect the timing constraints settled for the application.

3.3.1 *Algorithm for Transfer Type Determination.* After partitioning, a schedule of tasks on functional units (FUs) is provided. The aim is to maximize the use of synchronous communications by local rescheduling of operations. However, even with local rescheduling it is not always possible to use only synchronous communications. In such cases asynchronous communications are requested and both transfer types are used to implement all the communications of the application in the final architecture [Gogniat 1996]. The algorithm is based on two functions: Node_characterization and Edge_characterization. For each node V_i Node_characterization computes the mobility interval Δ_{M_Vi} defined as the interval between the ASAP starting time $ts_{(ASAP)i}$ and the ALAP ending time $te_{(ALAP)i}$:

$$\Delta_{M\_Vi} = [ts_{(ASAP)i}, te_{(ALAP)i}].$$

```
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 edge 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;
    Definitively schedule the edge e_{i,j} that has the lowest cost function ζ_{i,j};
    Label the edge e_{i,j} with a synchronous transfer;
    Remove asynchronous communication edges from the list L;
    Reorder the list L;
End while;
```

Fig. 8. Algorithm for transfer type determination.

Computations of interval mobilities take into account timing constraints. Edge_characterization computes the mobility interval $\Delta Me_{i,j}$ of a communication edge $e_{i,j}$ (edge-connecting nodes assigned to different units) and its mobility value $Me_{i,j}$. The mobility interval represents all the instants at which 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 ASAP starting time of the node that receives data and the value $te_{(ALAP)i}$ represents the ALAP ending time of the node that sends data. The mobility value $Me_{i,j}$ is defined by:

$$Me_{i,j} = te_{(ALAP)i} - ts_{(ASAP)j}.$$

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 is considered as potentially synchronous.

The Edge_characterization function also provides a cost value $\xi e_{i,j}$ for edges with a potentially synchronous communication which represents the ratio of 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 value are considered first since if communications associated with these edges are synchronous, a better hardware minimization is expected.

The algorithm operates as follows (Figure 8). First, nodes and edges are characterized. An edge $e_{i,j}$ is labeled when a transfer type (synchronous or asynchronous) is assigned to this edge. Edges with asynchronous communications ($ts_{(ASAP)j} > te_{(ALAP)i}$ i.e., $Me_{i,j} < 0$) are labeled and are not

considered for the remainder. The ordered list L of potential synchronous edges is created according to $\xi e_{i,j}$. Nodes V_i and V_j corresponding to the first nonlabeled edge $e_{i,j}$ of L are preliminarily scheduled (local rescheduling). The impact of this schedule on other communication edges is analyzed by characterizing nodes and edges again. If any communication edge $e_{k,l}$ ($k \neq i$ and $l \neq j$) becomes asynchronous, $e_{i,j}$ is definitively scheduled and is labeled with a synchronous transfer. Otherwise another nonlabeled edge $e_{i,j}$ from L is considered. The process is iterated until all the communication edges that have no impact on other edges are labeled.

After this step remaining potential synchronous edges in L involve at least one asynchronous communication. Let

$$\zeta_{i,j} = (\sum \textit{data of asynchronous edges})/(\sum \textit{data of synchronous edges})$$

be the cost function associated with $e_{i,j}$ defined as the ratio of the total volume of data associated with edges of L that become asynchronous and the total volume of data associated with edges of L that remain synchronous. The edge $e_{i,j}$ with $\zeta_{i,j}$ minimum is labeled with a synchronous transfer since the aim is to minimize the area dedicated to FIFOs.

3.3.2 *Resources and Transfer Modes.* The next step in the communication synthesis flow outlines the nature of communication protocols. Sender and receiver use blocking protocols if a synchronous transfer is applied. In the case of an asynchronous transfer the receiver can use a blocking or a nonblocking protocol. The sender uses a nonblocking protocol since memorization elements (FIFO) are available. 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 previously, most recent processors can handle DMA transfers and CPU-controlled 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 IV 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 handles the hardware communication signals is synthesized. The communication interface produced allows the complete execution of the application.

4. DESIGN RESULTS

The following results are given for the DSP56002. The partitioning of the GMDFα application produces two solutions. Although both solutions respect specification constraints, partitioning synthesis at the RTL level and optimization of DSP code have been performed only for the second solution since it leads to a 17% smaller hardware area. If delays due to communica-

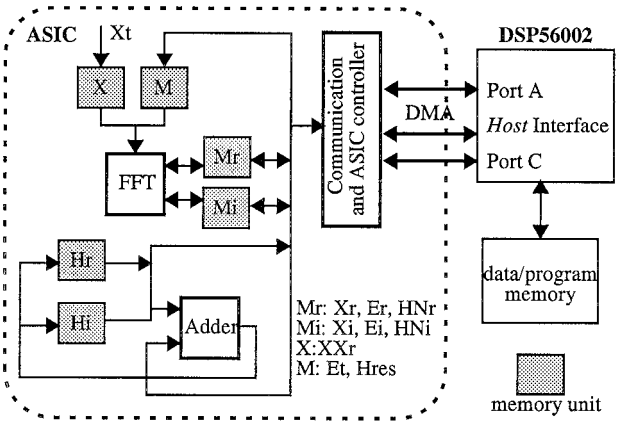

Fig. 9. HW/SW architecture for GMDFα.

tions are omitted and data and code sections are both mapped into the external memories of the DSP, the total execution time of the second partitioning is 13.96 ms. If instead localization of data and code sections is optimized in external and internal memories, the execution time is reduced by a factor 2 (6.39 ms). This point illustrates the importance of considering different software implementations of nodes during the partitioning/scheduling of the application [Kalavade and Lee 1995]. After synthesis of the whole hardware entities we assemble manually the architecture depicted in Figure 9 corresponding to the second partitioning. Data arrays pointed out in the figure share the same memory units in the ASIC part. Since the execution times of the hardware nodes of GMDFα are small compared to those of the software nodes, there is room for rescheduling the hardware nodes without increasing the overall execution time. Thus, all communications are implemented with a synchronous transfer. Data are not buffered and are exchanged directly from the internal memory of the sender to the internal memory of the receiver. Elapse times for communications are given in Figure 10.

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 (see Table IV). The DMA transfer in node 7.i lasts 25 μs and overlaps during 74 μs with computations of node 7.i + 1. Nodes 7.1 to 7.7 have each an execution time of 574 μs, including the DMA transfer. At the end of node 7.8 the transfer is performed with explicit *move* instructions and requires 54 μs. Consequently, the total elapse time due to hardware/software communications is 411 μs and the total execution time of the GMDFα application is 6.8ms, below the limit of 8ms imposed by the sampling. The ratio of HW/SW communications represents about 6% 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

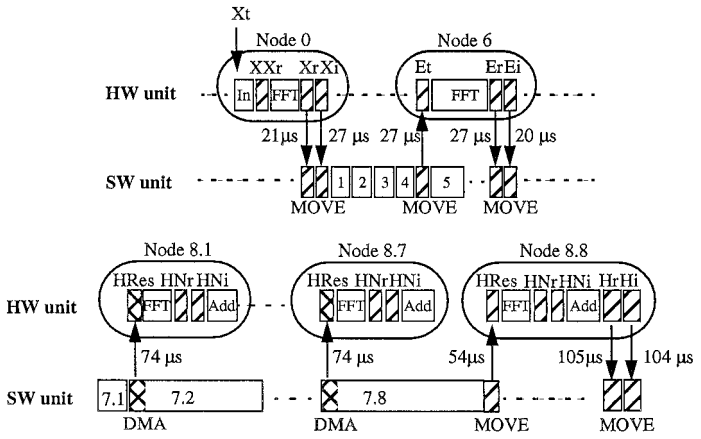

Fig. 10. Scheduling and elapse times of communications with DSP56002.

to minimize the hardware area that contributes to maximize the DSP's utilization. The requested sizes of program and data memories in the DSP56002 are 528 words and 9.4K words, 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, the FSM controller, 2.8K words of data memories, one FFT module, and one adder is 4.5 mm^2 in a 0.5 μ technology.

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 to the 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. Partitioning and communication synthesis algorithms have been implemented but the whole framework is still under construction. This experiment illustrates interesting results. The use of a DSP implies that the allocation of data and program into internal or external data memories be considered during the partitioning phase. Consequently, different software solutions have to be considered during partitioning/scheduling/allocation of tasks.

Since delays due to data transfers between HW and SW units are very dependent on this data distribution, effects of communications on produced solutions should also be considered during partitioning of the specification in order to exhibit HW/SW systems that respect firmly cost and timing constraints [Freund et al. 1997]. In addition, some computationally intensive applications require transfers of arrays between processing units that may be pipelined or overlapped with computations of nodes. In such cases, more efficient solutions would be achieved if these communication schemes are considered during the partitioning step. Our codesign methodology

considers target architectures composed of one ASIC and a single processor. However, complex applications (e.g., a full MPEG2 codec for videoconference) that may be or soon will be integrable on one chip often require more sophisticated systems, able to exploit different levels of granularity of parallelism. Therefore, there is a raw challenge for devising more general system architectures and cosynthesis methods able to deal with these computationally intensive applications.

REFERENCES

ADAMS, J. K. AND THOMAS, D. E. 1996. The design of mixed hardware/software systems. In *Proceedings of the Design Automation Conference* (Las Vegas, June), 515–520.

AMRANE, A. 1992. *Identification des systèmes à réponse impulsionnelle longue par filtre adaptatif en fréquence: Application à l'annulation d'écho acoustiques*. Thèse de doctorat (1992) ENST–Paris, September.

AMRANE, A., MOULINES, E., AND GRENIER, Y. 1992. Structure and convergence analysis of the generalized multi-delay adaptative filter. In *Proceedings of the International Conference EUSIPCO-92* (Brussels, August), 115–118.

BJORN-JORGENSEN, P. AND MADSEN, J. 1997. Critical path driven cosynthesis for heterogeneous architectures. In *Proceedings of the International Workshop on Hardware/Software Codesign* (Braunschweig, Germany, March 24–26), 15–19.

BORRIELLO, G. AND KATZ, R. H. 1987. Synthesis and optimization of interface transducer logic. In *Proceedings of ICCAD* (Nov.), 274–277.

CHOU, P. H., ORTEGA, R. B., AND BORRIELLO, G. 1995. The Chinook hardware/software co-synthesis system. In *Proceedings of the Eighth International Symposium on System Synthesis* (Cannes, France, Sept. 13–15), 22–27.

CORMAN, T. H., LEISERSON, C. E., AND RIVEST, R. L. 1992. *Introduction to Algorithms*. MIT Press, Cambridge, MA.

DAVEAU, J. M., ISMAIL, T. B., AND JERRAYA, A. A. 1995. Synthesis of system-level communication by an allocation-based approach. In *Proceedings of the International Symposium on System Synthesis* (Cannes, France, Sept. 13–15), 150–155.

EL HELWANI, A. AND LESCAN, P. 1995. VLSI architecture of the generalized multi delay frequency-domain algorithm for acoustic echo cancellation. In *Proceedings of ICASSP* (Detroit, MI, May 9–12), 3195–3198.

FILO, D., KU, D., COELHO, C., AND DE MICHELI, G. 1993. Interface optimization for concurrent systems under timing constraints. *IEEE Trans. VLSI 1* (Sept.), 268–281.

FREUND, L., DUPONT, D., AND ISRAEL, M. 1997. Interface optimization during hardware-software partitioning. In *Proceedings of the Fifth International Workshop on Hardware/Software Codesign* (Braunschweig, Germany, March 24–26), 75–78.

GAJSKI, D. AND VAHID, F. 1995. Specification and design of embedded hardware-software systems. *IEEE J. Des. Test Comput. 12*, 1 (Spring), 53–67.

GOGNIAT, G. 1996. *Etude de la synthèse des communications dans les systèmes logiciels/matériels*. Tech. Rep. RT96-01 (1996) I3S, Sophia-Antipolis, France.

GOGNIAT, G., AUGUIN, M., AND BELLEUDY, C. 1997. Interface synthesis in embedded HW/SW systems. In *Proceedings of the International Conference on Computer Hardware Description Languages and Their Applications* (Toledo, Spain, April 20–25), 80–82.

GUPTA, R. K., COELHO, C. N., AND DE MICHELI, G. 1994. Program implementation schemes for hardware software systems. *IEEE Comput. J. 27*, 1 (Jan.), 48–55.

HANSLER, E. 1992. The hands-free telephone problem: An annotated bibliography. *Signal Process. 27*, 259–271.

HAYATI, S. A., PARKER, A. C., AND GRANACKI, J. J. 1988. Representation of control and timing behavior with applications to interface synthesis. In *Proceedings of the International Conference on Computer Design* (Oct.), 382–387.

HENKEL, J. AND ERNST, R. 1996. The interplay of run-time estimation and granularity in HW/SW partitioning. In *International Workshop Codes/CASHE* (Pittsburgh, PA, March 18–20), 52–58.

JERRAYA, A. AND ISMAIL, T. 1995. Synthesis steps and design models for codesign. *IEEE Comput. 28*, 2 (Feb.), 44–52.

KALAVADE, A. AND LEE, E. 1994. A global criticality/local phase driven algorithm for the constrained hardware/software partitioning problem. In *Proceedings of the International Workshop on Hardware-Software Co-Design* (Grenoble, France, Sept. 22–24), 42–48.

KALAVADE, A. AND LEE, E. 1995. The extended partitioning problem: Hardware/software mapping and implementation-bin selection. In *Proceedings of the International Workshop on Rapid System Prototyping* (Chapel Hill, NC, June 7–9), 12–18.

LE TOURNEUR, G., THOMAS, J. P., AND GILLOIRE, A. 1994. Real time implementation of the GMDF Alpha algorithm on a multi DSP TMS320C40 board. In *Proceedings of EUSIPCO-94* (Edinburgh, Scotland, Sept. 13–16), 1007–1010.

LEE, C. AND POTKONJAK, W. W. 1996. System level synthesis using A* search and generalized force-directed heuristics. In *International Symposium on System Synthesis* (La Jolla, CA, Nov.), 2–7.

MADSEN, J. AND HALD, B. 1995. An approach to interface synthesis. In *Proceedings of the International Symposium on System Synthesis* (Cannes, France, Sept. 13–15), 16–21.

NARAYAN, S. AND GAJSKI, D. 1994. Synthesis of system-level bus interface. In *Proceedings of European Conference on Design Automation* (Paris, Feb.), 395–399.

NARAYAN, S. AND GAJSKI, D. 1995. Interfacing incompatible protocols using interface process generation. In *Proceedings of Design Automation Conference* (San Francisco, CA, June), 468–473.

NESTOR, J. A. AND THOMAS, D. E. 1986. Behavioral synthesis with interfaces. In *Proceedings of Design Automation Conference* (June), 112–115.

PAULIN, P. G. AND KNIGHT, J. P. 1989. Force-directed scheduling for the behavioral synthesis of ASIC's. *IEEE Trans. Comput. Aided Des. 8*, 6, 661–679.

ROUSSEAU, F., BENZAKKI, J., BERGE, J. M., AND ISRAEL, M. 1995. Adaptation of force-directed scheduling for hardware/software partitioning. In *Proceedings of the International Workshop on Rapid System Prototyping* (Chapel Hill, NC, June 7–9), 33–37.

VERCAUTEREN, S., LIN, B., AND DE MAN, H. 1996. Constructing application specific heterogeneous embedded architectures from custom HW/SW applications. In *Proceedings of the 33rd Design Automation Conference* (Las Vegas, NV), 521–526.

WOO, N. S., DUNLOP, A. E., AND WOLF, W. 1994. Codesign from cospecification. *IEEE Comput. J. 27*, 1 (Jan.), 42–55.

Received January 1997; accepted July 1997