A generic multi-unit architecture for codesign methodologies

Guy GOGNIAT, Michel AUGUIN, Cécile BELLEUDY

I3S, Université de Nice Sophia/Antipolis - CNRS, 41 Bld. Napoléon III 06041 Nice Cedex, France. - "name"@alto.unice.fr

Abstract: *This paper introduces a template architecture for codesign methodologies. This architecture is based on a data synchronized control scheme that is well adapted to the implementation of numerous telecommunication applications specified with a data flow model. The template architecture permits an easy integration of HW and SW coarse grain units. Communications between internal units are assumed to have an asynchronous protocol that is the more general transfer mechanism but also the more expensive in hardware resources. Hence, a communication synthesis method is presented that transforms asynchronous communications into synchronous ones. Results on an acoustic echo canceller illustrate the interest of the approach.*

1. Introduction

The complexity of embedded systems for multimedia and wireless applications imposes the use of heterogeneous resources (i.e. processor cores, dedicated functional units). Codesign methodologies of such systems must respect cost and performance constraints with a reduced time to market. The need for codesign techniques results from the increasing complexity of applications and the advances in HW and SW technologies. Due to the lack of a general formalism able to provide models for various application domains with efficient HW or SW implementations, codesign methodologies focus on a specific application domain with a dedicated generic architecture. For example, codesign methods consider generally a template architecture composed of a single processor and an ASIC [5],[7],[10]. But, with current advances in VLSI technologies, vendors propose systems on a chip composed of a DSP core, a RISC core and customized hardware cells. This limit of two processors on a chip will be widely overstepped shortly. Hence, more general architecture models targeted by codesign or system synthesis methodologies have to be investigated. Generally, the more template architectures are targeted to a restricted set of applications, the more efficient implementations can be achieved. But, codesign methodologies that focus on a too restricted application domain are somewhat uninteresting. Therefore, trade-offs between applicability and efficiency have to be explored. Since we focus on telecommunication and multimedia applications we consider a data flow oriented static model ([3] for example) that permits to describe a wide range of applications. In this paper a template architecture adapted to application and technology requirements is introduced. The following section depicts characteristics of this architecture. Section 3 presents a communication synthesis method that promotes synchronous transfers in that architecture to reduce the interconnect area. Before concluding, results about an acoustic echo canceller are depicted.

2. Template architecture for codesign

A static data flow model of an application can be described by a directed acyclic graph where nodes correspond to tasks of the application and edges represent data transfers between tasks. In this paper we focus on a coarse grain task model of applications. Generally, codesign techniques [2],[13],[7],[5], begin with a HW/SW partitioning of tasks which provides an assignment of implementation units to tasks. Some partitioning techniques are based on scheduling algorithms [9],[11]. The partitioning must take into account a target architecture in order to provide realistic solutions.

Our target architecture is based on the DSPA (data synchronized pipeline architecture) architecture [8] which was developed for high performance scientific computing. The initial DSPA model is composed of functional units (ALU, Mul, Mem,...) receiving their instruction flows through FIFOs from a VLIW type instruction memory. An instruction is executed by a functional unit (FU) as soon as data are available on its input ports. Outputs and inputs of FUs are connected through an incomplete crossbar network. Each crosspoint corresponds to a FIFO. This model presents attractive characteristics:

1. FUs may be asynchronous since they are synchronized on data arrivals. Then, communications between FUs are asynchronous. The Siera template architecture [12] considers also this approach to avoid the distribution of a global clock over the whole system.
2. Since FUs have a local data flow behavior, the mapping of a data flow model is facilitated.
3. Pipelining techniques can be applied to speed-up computations on data flows.

0-8186-7895-X/97 $10.00

4. FUs are controlled and interconnected with the same scheme, including communication ports. This facility allows to consider heterogeneous FUs in the same structure: for instance DSP and RISC processors and pre-designed FUs.

The DSPA model is a good candidate for codesign methodologies dealing with telecommunication and multimedia applications. Analog architectures where computation and memorization units are modeled at a fine grain level have already been developped for video and signal processings [4],[1]. The control scheme of these architectures allows an efficient mapping of the specification but the important use of FIFOs constitutes a major drawback for embedded system design. Therefore, refinements and improvements to the basic model are required to get a real template architecture for HW/SW codesign.

2.1 Control scheme of the model

Since FUs are either processor cores or pre-designed units, they cannot be controlled efficiently with a fine grain VLIW type instruction. Hence, a main controller (MC) is associated with each FU. This controller ensures the execution of corresponding instructions that are stored in a local instruction memory (Fig. 1 •). More precisely, the partitioning and the scheduling of tasks or nodes of the data flow graph

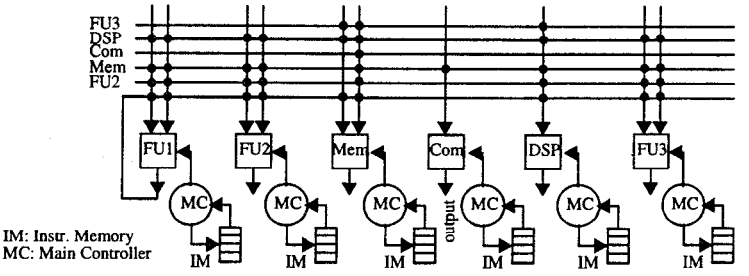


Fig. 1 • The considered target architecture

of the application [10],[11] provide coarse grain instruction segments for each FU. A coarse grain instruction contains information required to execute the operation: the location of the input or output data crosspoints in the communication network, indication of associated protocols to perform their transfers, and, if the unit supports several operations (for instance, FFT and FFT^{-1}), the op-code of the operation. The main controller sends to the FU this instruction and wait for an end of execution before sending the next one. Since we consider applications with a static behavior, only a global loop involving all instructions is required in the main controller. In order to facilitate the HW/SW system integration, we assume that all HW or SW FUs have this common interface with the network and the main controller. Therefore, input/output data transfers and FU's computations are considered to be internal to the FU, controlled by a specific controller (Fig. 2 •) or by a program memory of a core processor. This specific controller has in charge the management of network protocols and control signals of the FU.

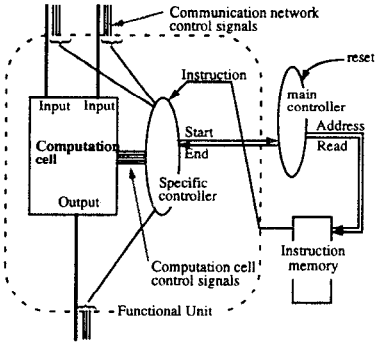


Fig. 2 • Control structure of functional units

2.2 Communication model

In the initial DSPA model, interconnection crosspoints are FIFO queues. This asynchronous communication model leads to an ASAP (As Soon As Possible) execution style of operations by FUs. Since considered applications have a static behavior, a fixed scheduling of operations may be defined during partitioning. Then, some communications may be changed into synchronous transfers (*rendez-vous* mechanism) without overstepping timing constraints. The protocol associated with a transfer can be blocking or non blocking. The protocol is blocking if before reading or writing into a crosspoint the FU

must check availabilities of data. With a non blocking transfer no verification needs to be done. To limit the cost due to FIFOs, the static schedule of tasks has to minimize the use of asynchronous communications. This point is addressed in the following section.

3. Communication synthesis

After partitioning communication edges represent links between HW and SW units, between different SW units or different HW units. Communication synthesis consists in determining for each communication edge the type of transfer (i.e. synchronous or asynchronous), the communication resources, the transfer protocol (blocking or non blocking) and the transfer mode (DMA or memory mapped I/O for processors). As mentioned above, the raw template architecture considers asynchronous communications with FIFOs. Hence, it is of prime importance to minimize their use in order to reduce communication resources. In the sequel, we address this point and details on other steps of the communication synthesis flow can be found in [6].

The communication synthesis method assumes that after partitioning a schedule of tasks on FUs is provided. The aim is to transform asynchronous communications into synchronous ones by local reschedulings. Fig. 3 • a,b and c depict an example of a feasible rescheduling whereas Fig. 3 • d,e and f illustrate the case of an imposed asynchronous communication.

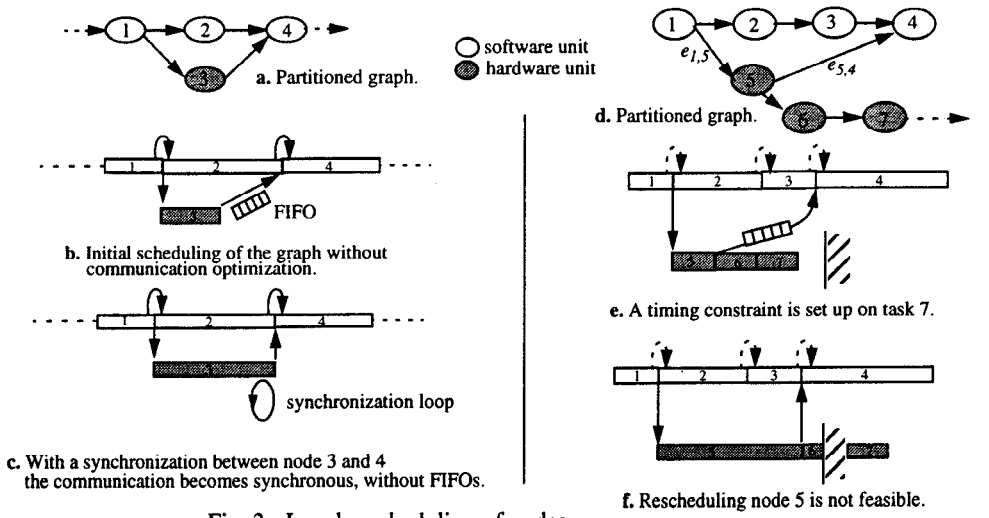


Fig. 3 • Local rescheduling of nodes

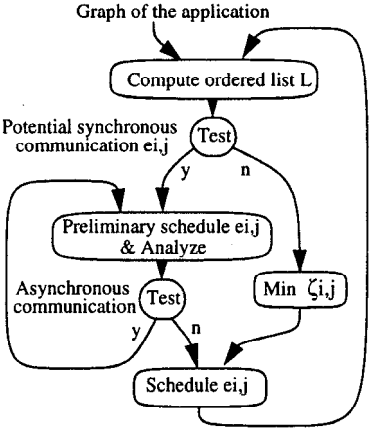
The algorithm is based on two functions: Node_characterization and Edge_characterization. For each node V_i Node_characterization computes the mobility interval Δ_{M_Vi} defined as the interval between the ASAP starting time $ts_{(ASAP)i}$ and the ALAP ending time $te_{(ALAP)i}$: $\Delta_{M\_Vi} = [ts_{(ASAP)i}, te_{(ALAP)i}]$. Computations of interval mobilities take into account timing constraints. Edge_characterization computes the mobility interval $\Delta Me_{i,j}$ of a communication edge $e_{i,j}$ and its 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 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 equal to $te_{(ALAP)i} - ts_{(ASAP)j}$. 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 synchronous.

The Edge_characterization function also provides a cost valuc $\xi e_{i,j}$ for edges with a potential synchronous communication and 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 (Fig. 3 •) operates as follows. Firstly, nodes and edges are characterized. An edge $e_{i,j}$ is labelled when a transfer type (synchronous or asynchronous) is assigned. Edges with asynchronous communications ($ts_{(ASAP)j} > te_{(ALAP)i}$ i.e., $Me_{i,j} < 0$) are labelled 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 non labelled edge $e_{i,j}$ of L are preliminarily scheduled (local rescheduling).

Impacts 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 labelled with a synchronous transfer. Otherwise another non labelled 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 labelled.

After this step remaining potential synchronous edges in L involve at least one asynchronous communication. Let $\zeta_{i,j}$ 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: $\zeta_{i,j}$ = $\sum$ *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 FIFOs.



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; -- local rescheduling may alter mobilities of other edges
    End if;
  End for;
  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;
  Remove asynchronous communication edges from the list L;
  Reorder the list L;
End while;

Fig. 4 • Communication synthesis algorithm

4. Example: GMDFα

To illustrate the principles of this generic architecture we consider a frequency domain block adaptative algorithm for acoustic echo cancellation (GMDFα). The flow graph is depicted in Fig. 5 •. The prece-

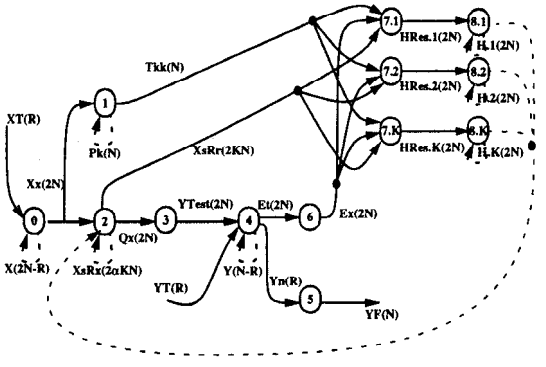


Fig. 5 • Flow graph of GMDFα and partitioning

SW Nodes	HW Nodes	Scheduling 56002 begin -> end (µs)	Hardware operators
	0	0 -> 86	FFT
1		87 -> 558	
2		559 -> 1288	
3		1289 -> 1758	
4		1759 -> 1820	
5		1821 -> 1854	
	6	1821 -> 1907	FFT
7.1 7.2...7.8		1908 -> 2455 2456->...->5743	
	8.8	5744 -> 5840	FFT, Adder
		5840 µs	FFT, Adder

dence constraints between two nodes are expressed by an edge. On each edge, 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 an FFT on the input signal (XT). Node 1 is a 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 cells. In Fig. 5 • is given a HW/SW partitioning of the flow graph [11]. This partitioning attempts to minimize the hardware area with a timing constraint of 8

ms. The software unit is a DSP56002 processor and the hardware part is composed of a FFT operator, an adder and two data memories Hr, Hi (Fig. 6 •.a).

Before synthesis of communications all units are connected through FIFOs placed in a network of six bus. The schedule of application nodes allows to label all communication edges with a synchronous transfer mode avoiding FIFOs. With the knowledge of timings of data transfers, a minimization of the number of bus can be performed (for example with a graph coloring method) and the network can be reduced to only two bus (Fig. 6 •.b). This example illustrates that the template architecture is designed

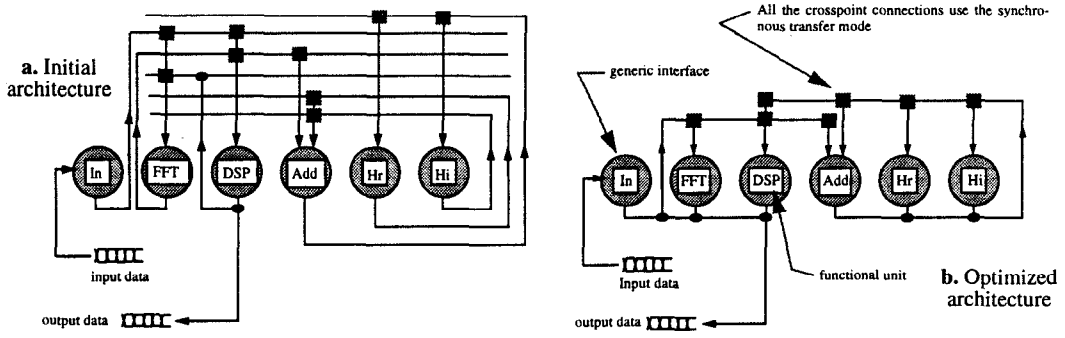


Fig. 6 • HW/SW architectures for GMDFα.

to support a general communication mechanism through FIFOs but the particularization induced by the knowledge of a specific application can lead to efficient implementations.

5. Concluding remarks

In this paper a generic architecture for HW/SW codesign methodologies is presented. Main characteristics of this model include asynchronous behaviors of functional units with synchronizations on data arrivals, common encapsulation of functional units that allows easy extension capabilities and integration of heterogeneous HW/SW units. But the communication model can lead to excessive communication resources, therefore a communication synthesis method that performs local reschedulings of transfers, permits to replace asynchronous communications involving FIFOs with synchronous transfers supported by bus. Our template architecture is therefore a good candidate model for codesign methodologies dealing with applications with a static behavior. For applications with a dynamic behavior, important extensions to this model are required and are area for future works.

6. References

[1] AARTS E.H.L., ESSINK G., De KOCK E.A. Recursive bipartitioning of signal flow graphs for programmable video signal processors. *European Design and Test Conference*, pages 460-466. Paris, March, 1996.

[2] ADAMS J. K. and THOMAS D. E. The Design of Mixed Hardware/Software Systems. *DAC*. Las Vegas, NV, USA, june, 1996.

[3] BUCK J., HA S., LEE E.A., MESSERSCHMITT D.G. Ptolemy: a framework for simulating and prototyping heterogeneous systems. *Int. Journal of Computer Simulation: special issue on Simulation Software Development*. 4april, 1994.

[4] CORPORAAL H., HOOGERBRUGGE J. Cosynthesis with the MOVE framework. *CESA'96 IMACS Multiconference*, pages 184-189. Lille, France, july, 9-12, 1996.

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

[6] GOGNIAT G. *Etude de la synthese des communications dans les systemes logiciels/materiels*. Technical Report RT96-01, I3S, Sophia-Antipolis, France, juillet, 1996.

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

[8] JEGOU Y., SEZNEC A. Data synchronized pipeline architecture: pipelining in multiprocessor environments. *Journal of Parallel and Distributed Computing*. 3508-526, 1986.

[9] KALAVADE A., LEE E. A global criticality/local phase driven algorithm for the constrained hardware/software partitioning problem. *Proceedings Int. Workshop on Hardware-Software Co-Design*, pages 42-48. Grenoble, France., September 22-24, 1994.

[10] 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.

[11] 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.

[12] 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.

[13] 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.