A Codesign Back-End Approach for Embedded System Design

G. GOGNIAT
Université de Bretagne Sud
and
M. AUGUIN, L. BIANCO, and A. PEGATOQUET
Université de Nice Sophia-Antipolis

Continuous advances in processor and ASIC technologies enable the integration of more and more complex embedded systems. Since their implementations generally require the use of heterogeneous resources (e.g., processor cores, ASICs) in one system with stringent design constraints, the importance of hardware/software codesign methodologies increases steadily. Interfacing heterogeneous hardware and software components together through a communication structure is particularly error prone and time consuming. Hence, on the basis of a generic architecture dedicated to telecommunication and multimedia applications, we propose an extended communication synthesis method that provides characterization of communications and their implementation schemes in the target architecture. This method takes place after the partitioning and scheduling phase and may constitute the basis of a back-end of a codesign framework leading to HW/SW integration.

Categories and Subject Descriptors: B.0 [**Hardware**]: General; J.6 [**Computer Applications**]: Computer-Aided Engineering

General Terms: Design, Experimentation, Measurement

Additional Key Words and Phrases: Codesign, communications synthesis, HW/SW integration, template architecture

1. INTRODUCTION

The use of programmable processors and the advances in HW and SW technologies have led to a generalization of digital systems in numerous application domains (e.g., automotive, telecommunication). These systems are generally dedicated to one application, since 98% of sold microproces-

Authors' addresses: G. Gogniat, Université de Bretagne Sud, 2 rue le Coat Saint Haouen, Lorient cedex, 56100, France; M. Auguin, L. Bianco, and A. Pegatoquet, Université de Nice Sophia-Antipolis, I3S, 42 Bd Napoléon III, Nice cedex, 06041, France.

sors deal with embedded systems. Wireless and multimedia are typical application examples and belong to the fastest growing industrial activity today. The growth rate of GSM in the next four years will be over 300%, from 280 million parts in 2000 to an expected 1 billion parts by 2003. Moreover, the importance of the time-to-market constraint is becoming commonplace because customer requirements are always increasing. Consequently, the average time window for a new product may be as short as 18 months. The complexity of embedded systems is also growing and often imposes the use of heterogeneous resources in a single chip (i.e., several processor cores, coprocessors, dedicated functional units). Hence, codesign methodologies of such systems must respect cost and performance constraints with a reduced time-to-market. However, due to the lack of a general formalism able to provide models for various application domains with efficient implementations, codesign methodologies concentrate on a specific application domain with a dedicated generic architecture. Since we focus on telecommunications and multimedia applications, we consider a dataflow-oriented static model that permits us to describe a wide range of applications in this area [Buck et al. 1994]. At each step in the design flow (i.e., HW and SW estimation, partitioning, code generation, communication synthesis, HW/SW integration), codesign methodologies target a generic architecture composed of interconnected heterogeneous resources. For embedded system design, it is of prime importance to develop methods that minimize the area and delays induced by interconnections. We propose a model of a generic architecture dedicated to embedded telecommunication applications and an associated extended communication synthesis method. This pairing, whose aim is to define efficient HW/SW integrated system models, takes place during the back-end steps in codesign flow. Hence, to provide high flexibility to this process, it is essential to define an efficient architecture model that includes functionalities adapted to the requirements of the application domain and authorizes an automatic generation of control patterns. The following section depicts state of the art architectures for codesign and communication synthesis. Section 3 presents characteristics of our generic architecture, including the communication scheme. Section 4 details an extended communication synthesis method that promotes synchronous transfers in the architecture to reduce the interconnect area and introduces an HW/SW integration method. Before concluding, we depict results of an audio decoder and an acoustic echo canceller.

2. STATE OF THE ART

Integration rates and chip sizes are increasing steadily every year. Hence, design of very highly complex embedded systems in a single chip are expected in the near future. Further, such core-based designs offer numerous advantages [Kalavade 1995]: performance improvement, field and mask programmability, and area and power reduction. The most relevant work at the chip level is introduced in Kalavade [1995] and Van Rompaey et al. [1996]. In Kalavade [1995] the target architecture consists of a single programmable processor core and multiple hardware modules connected to a single system bus. It is assumed that hardware and software components communicate via a memory-mapped, asynchronous, blocking communication mechanism. The various components in the architecture are configured as a globally asynchronous, locally synchronous model that enables each component to be used at its maximum speed. However, this architecture model targets single processor systems only, which may constitute a limitation for complex application design. In COWARE [Van Rompaey et al. 1996], the integration of heterogeneous (e.g., DSP or microcontroller cores from different vendors) and dedicated hardware modules into a custom target embedded architecture is achieved by encapsulating each component into a module template. Communications between components are through a point-to-point communication structure. For example, a software unit template is composed of three main components: the processor core itself, an internal memory structure for storing the program instructions and runtime data, and a hardware I/O unit that implements the hardware communication interface to the external environment. Communication with the external environment is accomplished through at least one input or output port attached to the I/O unit. These ports implement channel controls using a circuit-level handshaking protocol. However, this generic protocol implies the duplication of transferred data and the point-to-point connection limits the reusability of I/O ports.

One of the most important factors in selecting an architecture template is the choice of the interconnect scheme to link the various modules together [Srivasta and Brodersen 1995]. However, the task of interfacing the hardware and the software components together to support communication between them is particularly error prone and time consuming [Vercauteren et al. 1996]. Hence, several efforts have focused on communication synthesis at different levels of abstraction. Methods proposed by Nestor and Thomas [1986]; Boriello and Katz [1987]; Hayati et al. [1988]; Narayan and Gajski [1995] consider low-level descriptions of communication protocols and take place after HW/SW partitioning. The synthesized hardware interfaces are based on finite state machines with resource optimization. In these approaches, designers need to have a precise knowledge of the protocols being requested. For complex systems using heterogeneous resources, these techniques may become inextricable. Thus, the following work deals with communication synthesis at 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 task scheduling that results from partitioning. However, this schedule constitutes a set of timing constraints that communications must respect. In Madsen and Hald [1995], channel synthesis uses a sequence of timed events as specification, each event corresponding to a single data transfer. Transfers of arrays are not supported by this approach. The most relevant work for our problem is presented in Filo et al. [1993]. Interface optimization attempts to maximize the use of the nonblocking protocols in order to minimize control logic on channels. But the side effects consist of control delays introduced by both sender and receiver that impose a global reference clock.

3. AN ARCHITECTURE FOR CODESIGN

Our target architecture takes place at the chip level and promotes modularity, reusability, flexibility, and uniformity. The principles stated by Srivasta and Brodersen [1995] lead to a definition of an efficient generic architecture for complex digital applications. This architecture encourages (i) modularity, which means viewing the system as interacting hardware and software modules with well-defined interfaces; (ii) reusability allows us to use already defined modules in several applications with low overhead cost; (iii) flexibility permits us to easily improve the architecture with new modules in order to take late modifications into account in the design cycle; (iv) uniformity enables us to make modules, whether hardware or software, interact in a uniform way using a small number of well-defined communication mechanisms. Since defining an efficient architecture dealing with a broad spectrum of applications is a very hard problem, better results can be obtained if we consider a restricted application area. Hence, we must point out the main characteristics of the targeted applications handled by a codesign method. The important characteristics in embedded signal-processing telecommunication applications under consideration are as follows:

- At the upper level, applications are generally modeled by functions connected in a static dataflow style (as this model was successfully used to model DSP-oriented systems [Lin 1996], e.g., the SDF/DDF domains in PTOLEMY and the GRAPE II system). Commercial offerings include SPW from Alta/Cadence, COSSAP from Synopsis, and the DSP-Station from Mentor.
- Image and frequency domain processing leads to array or vector computation and communication, whereas time domain processing of monodimensional signals implies scalar computation and communication.

According to these characteristics and principles, we propose a new generic model for an architecture that matches efficiently embedded signal-processing telecommunication applications. The control of each resource integrated in the architecture is triggered by data arrivals. Several architectures dedicated to signal processing [Corporall and Hooger 1996; Aarts et al. 1996; Yeung and Rabaey 1992] and to intensive numerical computing [Jegou and Seznec 1986] using this mechanism have proven their efficiency. However, they correspond to ASIP with operations at a low level of granularity, which does not allow us to take into account the high complexity of codesign applications. The proposed architecture can be composed of several heterogeneous resources (e.g., DSP core, RISC core, coprocessor) configured in a globally asynchronous locally synchronous computing model. This model contributes to a modular and extensible architecture because its overall control is distributed.

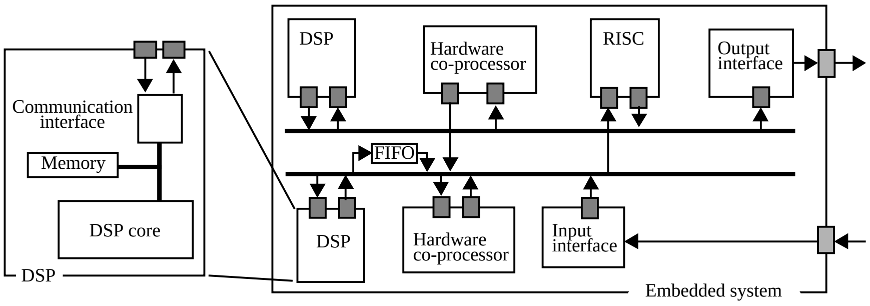


Fig. 1. Target architecture.

3.1 A Generic Model of the Target Architecture

The template architecture can be composed of software, hardware, memory, and interface components. Software components represent processor cores, since telecommunication applications generally require several DSP and RISC cores in a chip. For example, the VLSI VVS3670 system integrates two cores, a RISC (ARM7TDMI) and a DSP (OAK). Hardware components correspond to specific computation units issued by a library of cells or by a synthesizer. Memory components consist of shared memory that may be helpful in cases of high communication volume. Interface components implement the hardware communication interface connecting the system to the external environment (e.g., D/A converter). All these components have their own internal memory and communicate through a bus structure. Communication mechanisms are either synchronous using a handshaking protocol or asynchronous. In the case of a synchronous communication, both sender and receiver need to be synchronized during data transfer. With asynchronous communications, data written by the sender is memorized in a FIFO to make it available when the receiver is ready to read. Since it is rather difficult to determine the communication interfaces between heterogeneous resources, the components are encapsulated using hardware wrappers and communicate through well-defined protocols. The wrappers may result from a low-level description interface synthesis [Boriello and Katz 1987]. Encapsulation is managed by the communication interface as shown in Figure 1.

3.2 A Generic Model of Communication

Communication of the application is supported by the interconnection of components through buses and managed by well-defined protocols. Using a structure composed of buses instead of point-to-point connections is motivated by the encapsulation of each component, allowing greater flexibility: the communication interface associated with a component is not affected by the number of connections to other components, since input and output ports are only the result of the number of data channels imposed by the component itself. Late modifications of the architecture do not imply a redesign of the communication interfaces. Moreover, compared to a point-to-point scheme, a bus-based interconnection allows communication savings for broadcasting [Yen and Wolf 1995]. For static applications, a schedule of operations can be computed at compilation time. Consequently, an optimized communication network using the minimum number of buses can be defined.

Communicating through buses can be either synchronous or asynchronous. Since a handshake protocol is used between the sender and the receiver in a synchronous case, two control signals are managed by the communication interface (i.e., request and acknowledge). The receiver initiates the communication, and both sender and receiver use a blocking protocol. In the case of asynchronous communication, the sender writes the data to be transferred in a FIFO. Communication synthesis must verify that at least one FIFO is always available with adequate memory to contain all the transferred data. Thus, the protocol used by the sender is nonblocking. The protocol associated with the receiver can be either blocking or nonblocking. The protocol is blocking if the receiver must check the availability of data before reading from the FIFO . This situation may occur when the receiver is faster than the sender. In return, if the data is expected to be in FIFO, no verification is performed and a nonblocking protocol is considered. The same I/O port can be used for synchronous and asynchronous transfers, which allows for simple communication interfaces. Generally, both types of communication are used to implement an application as shown in Figure 1, where FIFOs and buses interact in the final architecture.

3.3 Architecture Software Components

The architecture's software components are based on processor cores and are composed of four main units: the processor core itself (e.g., OAK, ARM7, DSP56009), an on-chip internal memory (either on core or off core) to store data and programs, a communication interface to communicate with other components and mixed memory (Figure 2). Mixed memory has a particular function in the component since it can be controlled either by the processor core or by the communication interface. When communication is initiated, the communication interface takes control of the address and of the data buses connected with the mixed memory in order to manage communication through the network. As long as communication proceeds, the processor core cannot reach this memory. When the software component performs an output, the data is read by the communication interface from mixed memory and sent through the communication network. In the case of an input, data issued by other components is written into mixed memory. When communication is completed, the processor core can have access to data in the mixed memory. This mechanism permits the use of generic protocols through the communication network and avoids duplication of data in a dedicated buffer [Vercauteren et al. 1996]. Furthermore, according to the processor core used, since communication generally corresponds to memory-mapped access to mixed memory, an easy implementa-

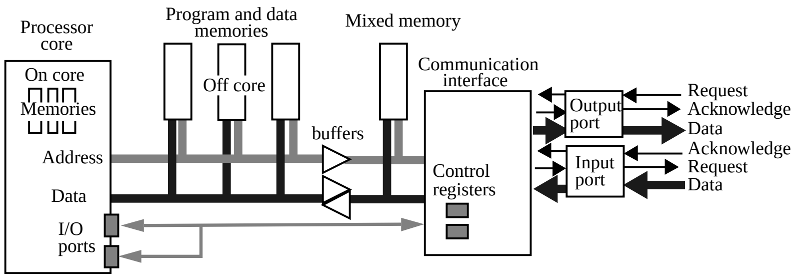


Fig. 2. A generic model of software components.

tion is made possible.When the processor core has dedicated I/O ports, the communication interface can also handle transfers through the network by buffering the corresponding data.

The software communication interface can be managed either in a sequential mode or in an overlapping mode. The easiest strategy is the sequential mode that consists of exclusive accesses to mixed memory. When communication begins, the processor core configures the communication interface using memory-mapped control registers and waits until the transfer is completed to continue its computation. With the overlapping strategy, communication and the computation are executed in parallel. In this case, while the mixed memory is controlled by the communication interface, the processor core can have access to its data memory to execute its operations. This technique can only be applied if the transferred data is not needed by the processor core to complete its computation.

3.4 Architecture Hardware Components

The architecture's hardware components mainly correspond to three categories: computing, interface, and memory. These components are encapsulated in order to use generic communication protocols to provide a standardized interface. The internal structure of hardware components is similar to that of software, except for the use of a main controller that manages the scheduling of communication and computation (Figure 3). This controller, adapted to each category of hardware component, configures the communication interface and activates the functional unit to execute one of its associated functions. All the communication and function activities performed by a hardware component during the execution of an application are stored in the instruction memory associated with the main controller. Thus, the functional unit corresponds to a data path and an internal controller triggered by the main controller.

4. COMMUNICATION SYNTHESIS AND HW/SW INTEGRATION

The codesign back-end has two phases: communication synthesis followed by HW/ SW integration. In this approach, the communication synthesis problem is addressed as a characterization of the communication involved in the application. At this step, only a general model of architecture is

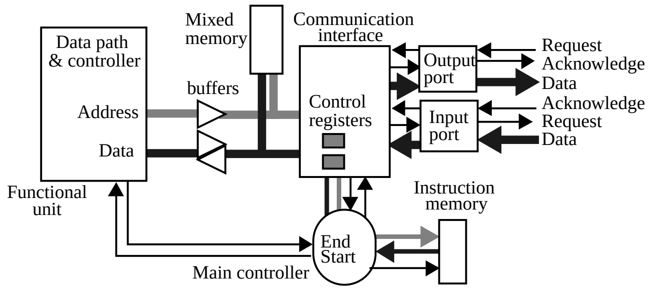


Fig. 3. A generic model of hardware components.

considered. The extended synthesis method detailed in this paper is based on a refined communication model that achieves accurate synthesis (compared to Freund et al. [1997]). The aim of HW/SW integration is to optimize communication resources and to generate the corresponding controls, depending on the target architecture. Communication synthesis and HW/SW integration occur after the partitioning and scheduling phase, allowing the use of various partitioning heuristics.

4.1 Graph Model of Partitioned Applications

Static signal-processing applications can be modeled by a direct acyclic graph where nodes represent computations and edges correspond to data dependencies. By $G(V, E)$ we denote the graph of the application after partitioning and scheduling. An edge $e_{i,j} \in E$ between two nodes $V_{i,}$, $V_j \in V$ denotes a dependency. Three types of dependency are considered: temporal, functional, and internal. A temporal dependency (TD) edge connects two nodes that do not communicate but are allocated to the same component. This link describes the order of node execution imposed by scheduling. Functional dependency edges (FD) represent data transfers between nodes. These edges are annotated with volume $\mathrm{d}_{\mathrm{i,j}}$ and word size ($\mathrm{L}_{\mathrm{data}}$) of transferred data. An internal dependency (ID) edge connects two communicating nodes implemented on the same component. In this case, no transfers are needed since the data is stored in the internal memory of the component.

4.2 Communication Synthesis Flow

The aim of communication synthesis is to minimize the amount of resources and overhead due to communication delays, while respecting the designer's constraints. The communication characteristics used for the synthesis are based on a general model of architecture in order to evaluate communication implementations on several architectures, and are defined by the following features:

- the transfer mode (i.e., overlapping or sequential);
- the transfer type (i.e., synchronous or asynchronous);

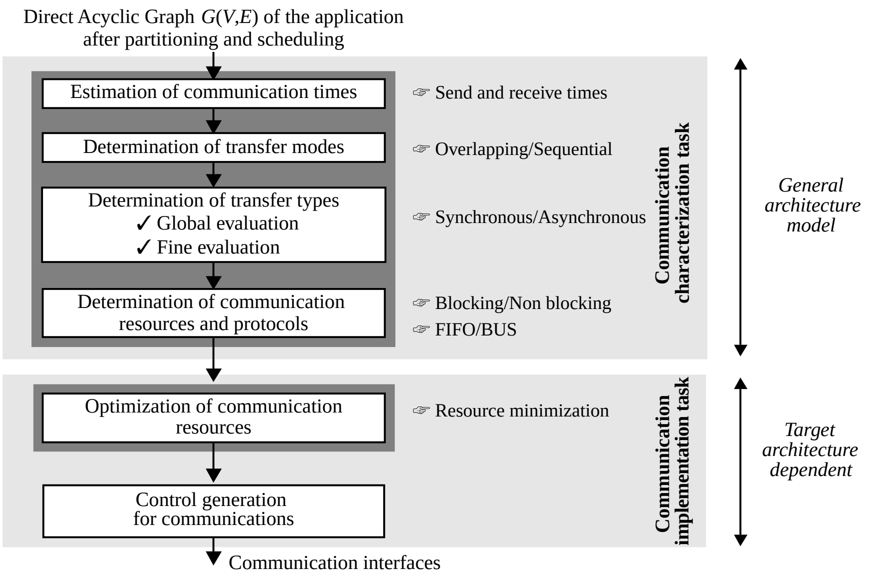

Direct Acyclic Graph G(V,E) of the application after partitioning and scheduling
Estimation of communication times
Determination of transfer modes
Determination of transfer types
✓ Global evaluation
✓ Fine evaluation
Determination of communication resources and protocols
☞ Send and receive times
☞ Overlapping/Sequential
☞ Synchronous/Asynchronous
☞ Blocking/Non blocking
☞ FIFO/BUS
Communication characterization task
General architecture model
Optimization of communication resources
☞ Resource minimization
Control generation for communications
Communication implementation task
Target architecture dependent
Communication interfaces

Fig. 4. Communication synthesis flow.

- communication supports and protocols (i.e., memory, bus; blocking, nonblocking).

With the overlapping transfer mode, a communication and a computation thread can be executed in parallel using, for example, a DMA mechanism or the mixed memory of our model (Section 3.3). When processors do not support DMA or equivalent transfer modes, only sequential executions are considered. The transfer type corresponds to the execution scheme (i.e., synchronous or asynchronous). Communication supports are the hardware resources required to execute the set of data transfers resulting from allocation and scheduling of the graph on componenets of the architecture (i.e., memory, bus). The protocol associated with a communication can be blocking or nonblocking. A blocking protocol must be selected if the consistency of data is not guaranteed by a nonblocking protocol. Figure 4 represents the synthesis flow that is divided into two main tasks. The first one characterizes each communication in the partitioned and scheduled graph, whereas the second task optimizes the number of resources and determines the set of controls required to manage communication. This last task is architecture-dependent and leads to the complete overall definition of the final target architecture.

4.2.1 *Characterizing Communication Edges.* The main task in characterizing communication is determining the transfer type, since this maximizes the number of synchronous communications, and consequently minimizes the number of FIFOs. Two evaluation methods are addressed in order to reach this goal, based on a global and a fine behavioral model of the system. The global model is introduced in Freund et al. [1997] and is

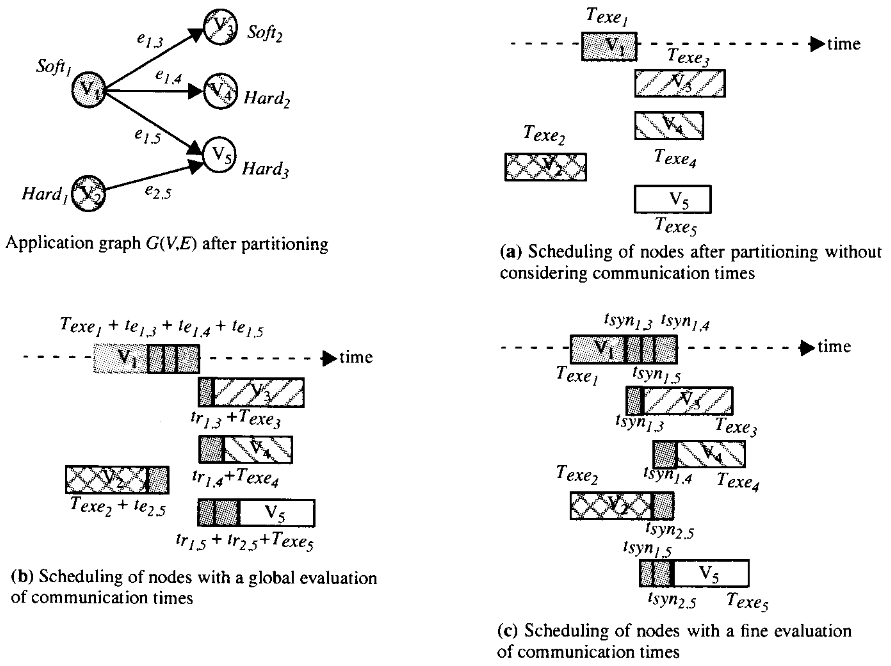

Soft1
Soft2
Hard2
Hard3
Hard1
e1,3
e1,4
e1,5
e2,5
Application graph G(V,E) after partitioning
Texe1
time
Texe3
Texe2
Texe4
Texe5
(a) Scheduling of nodes after partitioning without considering communication times
Texe1 + te1,3 + te1,4 + te1,5
tr1,3 +Texe3
tr1,4+Texe4
Texe2 + te2,5
tr1,5 + tr2,5+Texe5
(b) Scheduling of nodes with a global evaluation of communication times
tsyn1,3 tsyn1,4
tsyn1,5
tsyn2,5
(c) Scheduling of nodes with a fine evaluation of communication times

Fig. 5. Execution schemes for communication.

based on the usual communication and execution model [Lee and Messerschmitt 1987]. As shown in Figure 5(b), this model, in a single time frame, merges the computation (i.e., $Texe_i$) and communication times (i.e., send times $te_{i,j}$ and/or receive times tr_{ij}) associated with a node in the graph. For example, node V_1 sends data to V_3, V_4, V_5, hence its new execution time corresponds to its computation time increased by the three data emission delays. With this model, the execution of a node can only start when all the data produced by its preceding nodes is transferred. Hence, communication synthesis is based on a global model of communication. In the fine evaluation method, the communication model does not merge computation and communication, leading to a more realistic behavioral description of the system (Figure 5(c)). Furthermore, a single communication time $tsyn_{i,j}$, corresponding to the elapse time induced by the data transfer, is considered. This fine model permits a global and a local optimization of communication, since each data transfer associated with a node is locally ordered before scheduling the whole sequence of transfers. This point is detailed in the sequel. Since the global evaluation approach has been published in Freund et al. [1997], we focus mainly on the fine evaluation method.

Communication synthesis starts with an estimate of all the transfer times associated with communication edges (i.e., functional dependencies) of the partitioned graph. These communication edges correspond to data transfers between the different instances of the architecture (e.g., DSP and RISC, DSP and a coprocessor). With a general model of architecture, these

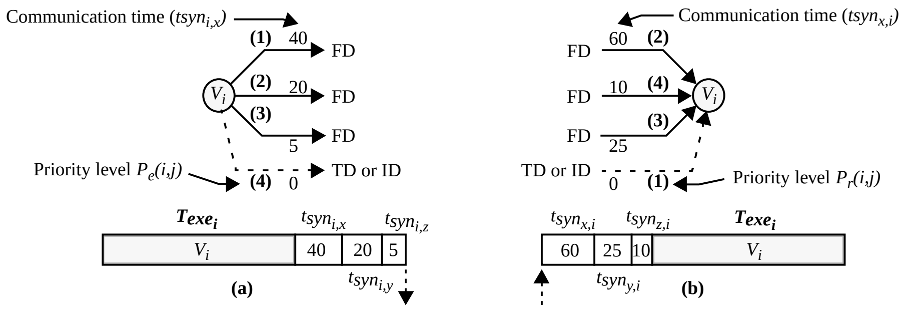

Communication time ($tsyn_{i,x}$)
(1) 40 FD
(2) 20 FD
(3) 5 FD
V_i
Priority level $P_e(i,j)$
(4) 0 TD or ID
$Texe_i$ $tsyn_{i,x}$ $tsyn_{i,z}$
V_i 40 20 5
$tsyn_{i,y}$
(a)
Communication time ($tsyn_{x,i}$)
FD 60 (2)
FD 10 (4)
FD 25 (3)
TD or ID 0 (1)
Priority level $P_r(i,j)$
$tsyn_{x,i}$ $tsyn_{z,i}$ $Texe_i$
60 25 10 V_i
$tsyn_{y,i}$
(b)

Fig. 6. Priority levels for senders and receivers.

estimates represent the data transfer times of the sender ($te_{i,j}$) and the receiver ($tr_{i,j}$), and are given by

$$T_{com} = d_{i,j} \times \left\lceil \frac{L_{data}}{L_{bus}} \right\rceil \times N_c \times T_c$$

where T_{com} is either $te_{i,j}$ or $tr_{i,j}$, $d_{i,j}$ represents the amount of data to be transferred between the sender and the receiver; L_{data} corresponds to the size of the data; N_c is the number of clock cycles to access a data; T_c is the clock period of the processor or coprocessor; and L_{bus} corresponds to the communication bus size. Since the aim of communication synthesis is to promote synchronous communication, the communication time $tsyn_{i,j}$ between two nodes V_i and V_j is estimated as the maximum of the sending and the receiving times, e.g., the slowest instance imposes its communication rate with a handshake protocol.

The fine evaluation model illustrated in Figure 5(c) leads us to reschedule nodes in the graph in order to take into account data transfers between components. This is achieved by computing a preliminary schedule of data transfers in a sequence of communications associated with a node. The order of analysis of these data transfers is given by scheduling rules. On the sender, these rules first schedule the outputs of data corresponding to functional dependency edges (Figure 6(a)). We assume that the execution time of a node depends on the amount of data consumed. Therefore, the highest priority level $P_e(i, j) = 1$ is associated with the edge with the longest communication time in order to promote execution of time-consuming nodes. Priorities for other edges are set up according to their decreasing communication times. The receiver can only start if temporal and internal dependencies are verified, hence the highest priority level $P_r(i, j) = 1$ is associated with these edges (Figure 6(b)). For the remaining edges a decreasing rule is used again.

Local scheduling attempts to maximize the use of synchronous transfers. However, even with local rescheduling, it is not always possible to use only synchronous communication [Gogniat 1997]. In such cases, asynchronous

```
For all the nodes $V_x \in \mathrm{pred}(V_i)$    $t_{val} = te_{ASAP_x} + \sum t_{syn_{x,j}}$    ($V_j \in \mathrm{suc}(V_x)$, $j \neq i$ and $P_e(x,j) > P_e(x,i)$)
    If $t_{val} > ts_{ASAP_i}$ Then $ts_{ASAP_i} = t_{val} + t_{syn_{x,i}}$ ;
    End if;
    If $t_{val} < ts_{ASAP_i}$ Then $ts_{ASAP_i} = ts_{ASAP_i} + t_{syn_{x,i}}$ ;
    End if;
End for;
```

```
For all the nodes $V_x \in \mathrm{suc}(V_i)$    $t_{val} = ts_{ALAP_x} - \sum t_{syn_{x,j}}$    ($V_j \in \mathrm{pred}(V_x)$, $j \neq i$ and $P_e(x,j) < P_e(x,i)$)
    If $t_{val} < te_{ALAP_i}$ Then $te_{ALAP_i} = t_{val} - t_{syn_{x,i}}$ ;
    End if;
    If $t_{val} > te_{ALAP_i}$ Then $te_{ALAP_i} = te_{ALAP_i} - t_{syn_{x,i}}$ ;
    End if;
End for;
```

Fig. 7. Determining the ASAP node starting and ALAP node ending times.

communication is required, and both transfer types are implemented in the final architecture (see, for example, Figure 1). The algorithm is based on two functions: node_characterization and edge_characterization. For each node V_i node_characterization computes its mobility interval $\Delta_{M\text{-}Vi}$ defined as the interval between the ASAP starting time ts_{ASAP_i} and the ALAP ending time te_{ALAP_i} of V_i. Computing interval mobilities takes into account timing constraints and local scheduling associated with nodes. Determination of ASAP node starting time and ALAP node ending time is presented in Figure 7. For each node V_i all the predecessors and all the successors are scanned in order to define ts_{ASAP_i} and te_{ALAP_i}.

Edge_characterization computes the mobility value $Me_{i,j}$ of a communication edge $e_{i,j}$ defined as the difference between te_{ALAP_i} and ts_{ASAP_i}. If $ts_{ASAP_i} > te_{ALAP_i}$ 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 the edges with a potential synchronous communication that represents the ratio of the amount of data $d_{i,j}$ transferred through this edge and its mobility value. Edges with the highest cost values are considered first, since a better hardware minimization is expected if a synchronous transfer mode is associated with these edges.

The algorithm for determining the transfer type with the fine evaluation method is similar to the one with the global evaluation model [Gogniat 1997] (except the evaluation of node_characterization and edge_characterization functions) and is briefly described here (Figure 8). First, nodes and edges are characterized. An edge $e_{i,j}$ is labelled when a transfer type (synchronous or asynchronous) is assigned to this edge. Edges with asynchronous communication (i.e., $Me_{i,j} < 0$) are labelled and are not considered further. 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 nonlabelled edge $e_{i,j}$ of L are scheduled preliminarily (local rescheduling). The impacts

```
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 denitively 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;
    Denitively schedule the edge $e_{i,j}$ that have the lowest cost function $\zeta_{i,j}$;
    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 determining transfer type.

of this schedule on other communication edges are 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 nonlabelled 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, the remaining potential synchronous edges in L involve at least one asynchronous communication. Let $z_{i,j}$ 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 remain asynchronous and the total volume of data associated with edges of L that become synchronous. The edge $e_{i,j}$ of L with $z_{i,j}$ minimum is labelled with a synchronous transfer, since the objective is to minimize the area dedicated to FIFOs.

The next step in communication synthesis flow outlines the nature of communication protocols and resources. For a synchronous communication, only one bus is required to support the data transfer between the sender and the receiver, but with an asynchronous communication three resources are necessary: the bus from the sender to the FIFO, the FIFO itself, and the bus from the FIFO to the receiver. All these resources are characterized by their width, throughput, and the depth of the FIFOs. A sender and a receiver involved in a synchronous transfer use a blocking protocol. With an asynchronous transfer, the receiver can use either a blocking or a nonblocking protocol (see Section 3.2). The sender uses a nonblocking protocol, since memory elements (FIFOs) are available. This ends the communication characterization task: all communications are characterized with their mode, type, resource(s) and protocol(s).

4.2.2 *Communication Implementation Task.* The communication implementation task corresponds to HW/SW integration, which depends on the target architecture. The aim of this task is to minimize the number and the size of memory units and buses required in the communication network to support all the data transfers [Gogniat 1997]. In order to merge communication resources with exclusive lifetimes, we use an extended weighted bipartite matching algorithm [Gajski et al. 1992]. With the target architecture (see Section 3.1), this step leads to the minimization of FIFO and bus hardware costs. Next, data is distributed between internal and mixed memories of components, according to possible overlapping execution of transfers and computations. Control resources (e.g., buffer, multiplexer) necessary to control the communication network are also determined. Then, for each component allocated to the executions of several nodes and edges, the sequences of computation activations and communications are generated. All these operations end the HW/SW integration of the application. Hence, starting from a partitioned and scheduled graph, the proposed approach first performs the communication synthesis that characterizes all the communications of the application according to a general architectural model, and then integrates the HW/SW, which defines the optimized communication network and the set of controls associated with each component in the final target architecture. This last step is architecture-dependent, and must be adapted to the model of the target architecture.

5. DESIGN RESULTS

To illustrate the principles of this codesign back-end, we consider two applications. With a simplified AC3 audio decoder we show the ability of the communication characterization step (Figure 4) to assist the designer in selecting an architecture model. The second application, a frequency domain block adaptative algorithm for acoustic echo cancellation (GMDFα), illustrates the full communication synthesis flow depicted in Figure 4.

5.1 Simplified AC3 Audio Decoder

The dataflow graph of a 3–channel AC3 audio decoder [ATSC 1995] corresponding to worst-case behavior is depicted in Figure 9 with execution times (μs) and area values for HW and SW implementation. Software execution times are estimated values using the Motorola DSP56009. Values for SW and HW areas are given in gate equivalents. The software area of a task corresponds to the size of the RAM and ROM memory required to store the task's variables and instructions. The size of the DSP core is not included in area values. After partitioning and scheduling the specification, we retain four potential solutions satisfying a time constraint of 2ms. The communication characterization step produces the results given in Table I. Communication is supported through a single bus in the four architectures. Only the simplest architectures (i.e., 1 and 2) require a memory unit to store data due to asynchonous transfers. Despite this additionnal memory cost, architecture 1 has the lowest gate count. Compared to other solutions this architecture realizes the best execution time/area tradeoff. These results also show that solutions with a high degree of parallelism (i.e., 3 and 4) induce an interconnection structure without memorization. This is due to the fact that a higher tasks mobility is the result of a lower unit utilization rate. This example highlights the capacity of the communication

Table I. Architecture Characteristics

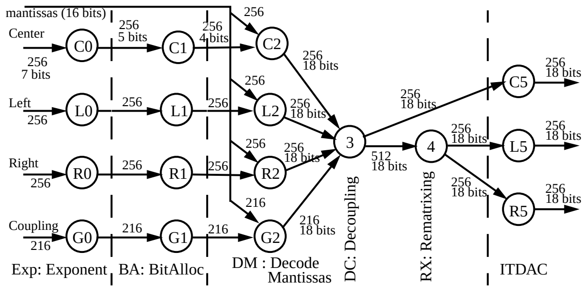
Solutions		1	2	3	4
Units		DSP, BA	DSP, BA, DM	Exp, BA, DM, DC, RX, Itdac	DSP, 2xBA, Exp, DM, RX
Execution time		1778	858	711	452
Nb Bus		1	1	1	1
Nb memorization unit/size		1/768x4 bits	1/512x16 bits	0/0	0/0
Sync./Async. Transfers		3976/768	5216/512	7008/0	6752/0
Area	HW/SW	2119/1557	2816/1362	8091/0	5903/637
	Memorization unit	490	1300	0	0
	Total	4266	5478	8091	6540

characterization method to quickly evaluate potential implementations, including communication.

5.2 Acoustic Echo Cancellation

The GMDFα algorithm is shown in detail in Freund et al. [1997], which also shows a partitioned task schedule. From this schedule an optimized hand-crafted approach and the method presented here were applied to synthesize communication and to perform HW/SW integration in the final architecture. In the hand-crafted design, the components of the architecture are directly connected to each other without using encapsulation through the generic interface. The timing characteristics of the hand-crafted approach are the result of logic simulation and synthesis. The global and fine evaluation methods were applied in order to compare both models. The solutions are close, since all the transfers are implemented using synchronous communication on a single bus. Moreover, several communications took advantage of the overlapping scheme in order to reduce global timing overhead costs. The estimated schedules and the real one provide similar execution times, as shown in Table II. However, communication synthesis using the global evaluation method provides overestimated communication delays, since there is a difference of 70% between the estimated solution and the real one. The behavioral model in the fine evaluation method provides a more realistic estimate of communication times, since a difference of 34% is obtained compared to the hand-crafted result.

Finally, the automatically constructed architecture and the manually optimized one lead to a similar hardware area (Table II). Even if the distributed control and the generic communication interface integrated in each component contribute to an increase in the area (i.e., 0.45 mm^2 instead of 0.15 mm^2), this augmentation is still negligible (i.e., less than 0.5%), compared to memory and data path areas. Figure 10 represents the target architecture with the communication interfaces, the mixed memories, and the main controller associated with each hardware component.



Implementation characteristics

Task	HW area	SW area	T_{HW}	T_{DSP}
Exp	734	645	10	69
BA	2119	767	73	545
DM	697	195	33	237
DC	527	187	14	49
RX	234	80	10	26
ITDAC	3780	450	32	130

Fig. 9. Flow graph of a simplified AC3 audio decoder.

Table II. Comparing the Final Results

	After communication synthesis (fine estimate)	After communication synthesis (global estimate)	After communication synthesis (real)
Hardware area	6.97 mm^2+0.45mm^2	6.97 mm^2+0.45mm^2	6.97 mm^2+0.15mm^2
Execution time	6683 μs	6722 μs	6800 μs
Communication time	550 μs	700 μs	411 μs
Processor utilization rate	92.7%	92.2%	91%
Hardware unit utilization rate	14.5%	14.4%	14%

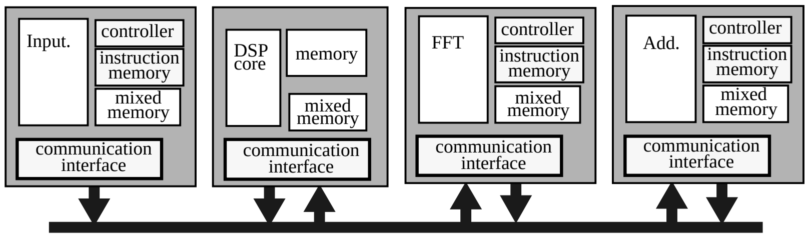


Fig. 10. Target architecture for the GMDFa application.

6. CONCLUSION AND FUTURE WORK

The proposed codesign back-end performs communication synthesis and resource optimization from a static partitioned/scheduled graph and a generic modular architecture in order to provide a dedicated realistic signal-processing embedded architecture. This architecture presents interesting features for embedded systems, since it is integrated in a single chip and encourages component reusability by encapsulating computation units in hardware wrappers with a standard external interface. A generic communication network built on a bus-based scheme constitutes the underlying communication model. Furthermore, in order to avoid error-prone and time-consuming manual determination of interfaces between heterogeneous components of the architecture, an extended communication synthesis method is defined. This new method has proven its efficiency on static applications and permits a better estimate of the communication schemes implemented in the final architecture. However, this static model may be inadequate for dealing with multimedia and telecommunication services that require complex controls due to the key role of user interaction and environmental data communication, coupled with sophisticated signal processing. Hence, our work will be extended in order to take dynamic application execution schemes into account.

REFERENCES

AARTS, E. H. L., ESSINK, G., AND DE KOCK, E. A. 1996. Recursive bipartitioning of signal flow graphs for programmable video signal processors. In *Proceedings of the European Conference on Design and Test* (Paris, France, Mar. 1996), 460–466.

ATSC. 1995. Advanced Television System Committee. Digital Audio Compression Standard (AC3).

BORIELLO, G. AND KATZ, R. H. 1987. Synthesis and optimization of interface transducer logic. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design* (ICCAD, San Jose, Calif., Nov. 9 - 12), IEEE Computer Society Press, Los Alamitos, CA, 274–277.

BUCK, J., HA, S., LEE, E. A., AND MESSERSCHMITT, D. G. 1994. Ptolemy: A framework for simulating and prototyping heterogeneous systems. *Int. J. Comput. Simul. 4*, 4 (1994).

CORPORAAL, H. AND HOOGERBRUGGE, J. 1996. Cosynthesis with the MOVE framework. In *Proceedings of the IMACS Multiconference on CESA'96* (Lille-France, July 9-12, 1996), 184–189.

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 Eighth International Symposium on System Synthesis* (Cannes, France, Sept. 13–15, 1995), P. G. Paulin and F. Mavaddat, Eds. ACM Press, New York, NY, 150–155.

FILO, D., KU, D., COELHO, C., AND DE MICHELI, G. 1993. Interface optimization for concurrent systems under timing constraints. *IEEE Trans. Very Large Scale Integr. Syst. 1*, 3 (Sept.), 268–281.

FREUND, L., ISRAEL, M., ROUSSEAU, F., BERGÉ, J. M., AUGUIN, M., BELLEUDY, C., AND GOGNIAT, G. 1997. A codesign experiment in acoustic echo cancellation: GMDFα. *ACM Trans. Des. Autom. Electron. Syst. 2*, 4, 365–383.

GAJSKI, D. D., DUTT, N. D., WU, A. C.-H., AND LIN, S. Y.-L. 1992. *High-Level Synthesis: Introduction to Chip and System Design*. Kluwer Academic Publishers, Hingham, MA.

GOGNIAT, G. 1997. Architecture gènèrique et synthëse des communications pour la conception conjointe des systëmes embarquès logicel/matèriel. Ph.D. Dissertation. Universitè de Nice Sophia - Antipolis.

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* (ICCD '88), 382–387.

JÉGOU, Y. AND SEZNEC, A. 1987. Data synchronized pipeline architecture: pipelining in multiprocessor environments. *J. Parallel Distrib. Comput. 3*, 4 (Dec. 1, 1987), 508–526.

KALAVADE, A. 1995. System level codesign of mixed hardware-software systems. Ph.D. Dissertation.

LEE, E. A. AND MESSERSCHMITT, D. G. 1987. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. Comput. C-36*, 1 (Jan. 1987), 24–35.

LIN, B. 1996. A system design methodology for software/hardware co-development of telecommunication network applications. In *Proceedings of the 33rd Annual Conference on Design Automation* (DAC '96, Las Vegas, NV, June 3–7), T. P. Pennino and E. J. Yoffa, Eds. ACM Press, New York, NY, 672–677.

MADSEN, J. AND HALD, B. 1995. An approach to interface synthesis. In *Proceedings of the Eighth International Symposium on System Synthesis* (Cannes, France, Sept. 13–15, 1995), P. G. Paulin and F. Mavaddat, Eds. ACM Press, New York, NY, 16–21.

NARAYAN, S. AND GAJSKI, D. 1994. Synthesis of system level bus interfaces. In *Proceedings of the 94 Conference on European Design and Test* (Paris, France, Feb. 1994), 395–399.

NARAYAN, S. AND GAJSKI, D. D. 1995. Interfacing incompatible protocols using interface process generation. In *Proceedings of the 32nd ACM/IEEE Conference on Design Automation* (DAC '95, San Francisco, CA, June 12–16, 1995), B. T. Preas, Ed. ACM Press, New York, NY, 468–473.

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

SRIVASTA, M. B. AND BRODERSEN, R. W. 1995. SIERA: A unified framework for rapid-prototyping of system level hardware and software. *IEEE Trans. Comput.-Aided Des. 14*, 6, 676–693.

VAN ROMPAEY, K., BOLSENS, I., DE MAN, H., AND VERKEST, D. 1996. CoWare—a design environment for heterogenous hardware/software systems. In *Proceedings of the Conference on European Design Automation* (EURO-DAC '96, Geneva, Switzerland, Sept. 16\), G. Symonds and W. Nebel, Eds. IEEE Computer Society Press, Los Alamitos, CA, 252–257.

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 Annual Conference on Design Automation* (DAC '96, Las Vegas, NV, June 3–7), T. P. Pennino and E. J. Yoffa, Eds. ACM Press, New York, NY, 521–526.

YEN, T.-Y. AND WOLF, W. 1995. Communication synthesis for distributed embedded systems. In *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design* (ICCAD-95, San Jose, CA, Nov. 5–9), R. Rudell, Ed. IEEE Computer Society Press, Los Alamitos, CA, 288–294.

YEUNG, A. AND RABAEY, J. M. 1992. A data-driven architecture for rapid prototyping of high throughput DSP algorithms. In *Proceedings of the Conference on VLSI Signal Processing* (Napa Valley, CA, Oct. 1992), 225–234.

Received: December 1997; revised: September 1998; accepted: November 1998