CODEF: A SYSTEM LEVEL DESIGN SPACE EXPLORATION TOOL

M. Auguin†, Laurent Capella†‡, Fernand Cuesta†‡, E. Gresset‡

‡ Philips Semiconductors Sophia, Sophia Antipolis, Valbonne.
† Laboratoire I3S, Université de Nice Sophia Antipolis

ABSTRACT

The increasing complexity of embedded applications combined with the advances in chip integration make the design process a very challenging task. Due to this rising complexity, the design under performance, area and consumption constraints of a system-on-a-chip (SOC) composed of mixed software-hardware units, becomes increasingly intricate. This paper presents a method and an associated tool (CODEF) which allow the designer to do an automatic and/or interactive system design space exploration in order to construct cost effective embedded real-time architectures dedicated to complex signal processing applications. The method is based on a recursive partitioning algorithm followed by a communication synthesis procedure.

1. INTRODUCTION

System design is one of the key steps in the design flow of real time signal processing heterogeneous embedded systems [1]. Starting with a functional specification of a target application, it is of prime importance to determine as soon as possible a precise system architecture specification. This system specification must be able to support successive and concurrent design refinements leading to an actual real time system with reduced area and energy consumption. Approaches based on cosimulation (e.g., systems like Polis [2], VCC from Cadence) impose a preliminary manual hardware/software partitioning and a development of precise simulation models. Cosimulation techniques permit the validation of system specification after partitioning.

We have developed a SOC design framework (CODEF) that improves the system-level design flow and allows a significant reduction of time to market. From a time constrained functional specification, CODEF performs an automatic or interactive design space exploration and constructs a set of system architectures. From this set of solutions, a particular system can be selected for further cost/performance improvements. This is achieved through architecture adjustments proposed by the designer whose impacts on performance and costs can be quickly evaluated by CODEF.

2. APPLICATION MODEL

We focus on embedded systems including significant parts of signal processing. A data flow model is well adapted to describe signal processing functions which operate on regular streams of samples [3]. However, the rising number of functionalities embedded in multimedia and telecommunication systems and the improved accuracy of signal processing computations lead to increase continuously the complexity of applications. For example, orders of computations are controlled by external events or are dependent on values of variables internally updated at runtime. The execution of such applications with a data flow semantic is not effective since the model assumes that conditional computations are always executed whatever the values of controls.

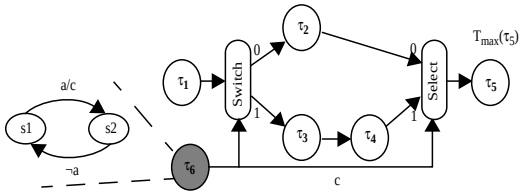

Fig. 1 • The BDF controlled data flow model

We consider the controlled data flow model [3], which includes switch and select nodes (figure 1). According to the value of the boolean signal c the switch node produces a token to the output 0 or 1 and the select node consumes a token from the input 0 or 1. This boolean-controlled data flow (BDF) model is used in tools such as Ptolemy, SPW and COSSAP. A static schedule of a BDF can be found [3] but in the firing sequence of the tasks it may be necessary to check the value of the control signals. For example, <τ1, τ6, c.τ2 (1-c).(τ3, τ4), τ5> is a conditional firing sequence of the graph given in figure 1. If c is true, the task τ2 is executed else τ3, τ4 are fired. The task τ6 produces the value of the control signal. This control is either the result of a processing task or a value driven by an external input or the output of a finite state machine (FSM) as depicted in figure 1. In this case, the firing of the FSM task must respect the BDF semantics: upon each firing, a token is consumed on each incoming edge and a token is produced on each outgoing edge. Furthermore, reacting to a firing, the FSM makes a single state transition. A time constraint is set up on each terminal task of the graph, (e.g. the task τ_5 in figure 1).

3. TARGET ARCHITECTURE MODEL

Due to the continuous pressure for embedding sophisticated services with higher quality in telecommunication systems, embedded system architectures are changing with a shorter and shorter period. These system on a chip architectures include IPs or in-house components such as RISC and DSP core processors connected to coprocessors and sharing hardware accelerators (e.g. ASIP). In an heterogeneous structure, inter-unit communications

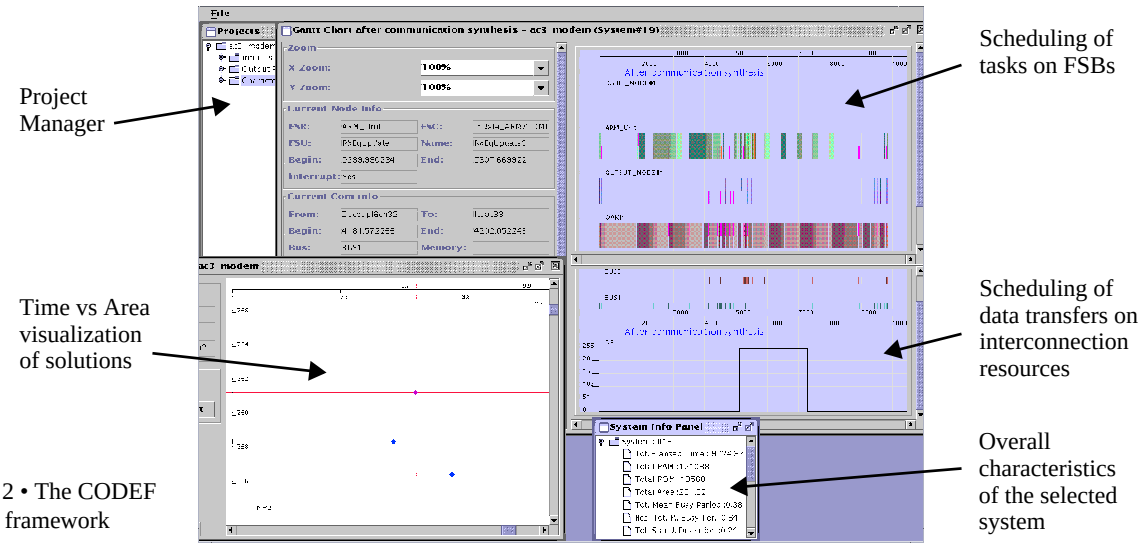

Fig. 2 • The CODEF framework

are supported through tailored data buses or through a one to one specific connection model where the protocols are adapted to the type of each transfer. We consider a data bus structure rather than a one to one connection model [4] in order to provide better extension capabilities and to improve the architecture reusability.

As system architecture specification must be pertinent with respect to further design steps i.e., it must support architecture design refinements with no need of backward modifications. If the top-level system specification is too optimistic or imprecise, derived refined systems will impose some architectural modifications to match the application requirements. In order to limit these feedbacks, the initial system architecture and its main characteristics must be set up precisely.

The system level characteristics and parameters taken into account by CODEF are: number and type of processors, size and access time of memories (i.e., RAM, ROM, cache memories associated with programmable processors and communication memories), internal or external distribution of data and code sections in processor memories, contribution of coprocessors or hardware accelerators on performance and cost, DMA or CPU controlled data transfers, communication protocols between units (synchronous or asynchronous), characteristics of I/O ports of units and overhead due to interrupts.

4. CODEF SYSTEM DESIGN FRAMEWORK

Inputs and outputs of CODEF are depicted in figure 3. From an application graph model, a set of constraints, a library of potential hardware or software units (FSBs), a library of task or process implementation models on the FSBs and an optional predesigned system, CODEF performs a system level design space exploration and provides a set of solutions that respect the constraints. Currently, CODEF takes into account time and area constraints. Each solution is described as a set of interconnected FSBs, a mapping of the tasks of the application graph model on the FSBs, a set of characteristics (e.g. the mean busy period of each FSB) and a schedule of the tasks (Gantt chart). In figure 2 is illustrated the graphical user interface of CODEF.

In order to take into account the designer's skill, CODEF is able to deal with an optional pre-designed system. This capability allows the designer to perform iterative system design optimizations by tuning the architecture and its parameters. The impacts on performance and cost are immediately evaluated with CODEF. This pre-designed system may be a previously designed system (off-the-shelf) on which an application has to be mapped with or without potential architecture modifications. This facility allows the reuse of designs or IPs, which is of prime importance for a rapid development of cost effective architectures.

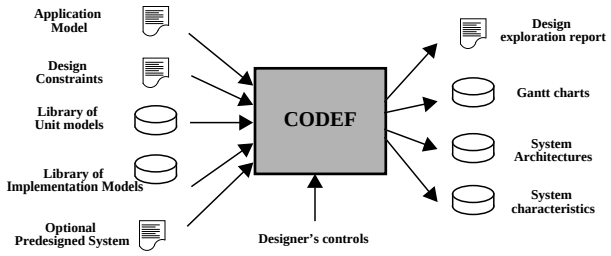

Fig. 3 • I/Os of CODEF

5. SYSTEM SYNTHESIS PRINCIPLES

System architectures are constructed in two steps: a partitioning/scheduling step which operates on the tasks of the functional specification to map these functionalities on the processing units, followed by a communication synthesis step that determines an optimized architecture interconnection.

5.1. System partitioning

The aims of system partitioning are i) to define an architecture built from the FSBs of the library, ii) to assign the tasks of the application on the FSBs and iii) to define a scheduling of the tasks that respects the application time constraints. The FSBs in the architecture are selected such that the total area is minimized. The partitioning of a boolean-controlled data flow graph (BDF) is performed in two steps. The first step consists in identifying the worst cases involved in the BDF graph. Each worst case corresponds to

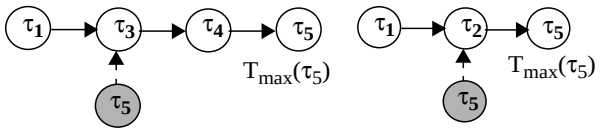

Fig. 4 • The two states of the DFG given in figure 1.

a static data flow graph on which a partitioning/scheduling is performed.

Identification of the worst cases

Setting a value to each control input of the routing nodes determines a specific routing of the data in the graph [5]. The nodes and edges activated by this routing define a state of the application and correspond to a sub-graph which is a pure data flow graph (Fig. 4). The dotted edges correspond to control precedences in the original graph. However, it is not necessary to partition all the states of the application to derive a system architecture that supports the execution of the whole application [5]. Indeed, if a system architecture supports the real time execution of a state G_i, it is able to execute all the states $G_j \subseteq G_i$. Consequently, only the prime states $G_i / G_i \cap G_j \neq \emptyset, \forall j \neq i$ have to be considered to construct a system architecture that includes all the resources required to execute the whole application.

Incremental partitioning of the graph

To determine a system architecture that supports the execution of the prime states we consider an incremental partitioning method [5]. At step i, $0 \leq i \leq p\text{-}1$ we use as predesigned system, the system architecture A_i provided at step i-1 to partition the graph G_i. The architecture A_{i+1} provided at step i, supports the real time execution of G_i (figure 5). Since the successive steps of the incremental partitioning can only extend this architecture (if needed), the final architecture A_p contains the resources required to support the real time execution of all the application states.

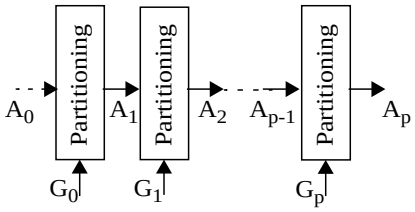

Fig. 5 • Incremental partitioning of the prime states

Partitioning of a single data flow graph (a state)

The partitioning/scheduling algorithm of a data flow graph is based on the static analysis of the time urgency of task executions according to the application time constraints [6]. The algorithm uses this time metric for selecting the hardware or software units required to implement each task of the functional specification. Depending on the time urgency of each task, the partitioning goal of CODEF is either a performance criteria (in order to avoid time violations) or the total system area, or a mix. The partitioning process is controlled by a reusability and an instantiation threshold parameters that can be set up by the designer. These parameters allow the selection of FSBs according to their time efficiency to map the functionalities of the application. Automatic or designer-controlled variations on the reusability and instantiation threshold parameters allow various system architectures to be explored. The partitioning algorithm uses a simplified communication model. In order to get an accurate interconnection between units of the architecture, a communication synthesis is performed once partitioning and scheduling are performed.

5.2. Communication synthesis

From the whole set of communications involved in the execution of the partitioned application, communication synthesis builds a structure composed of optimized resources required to supports these communications [7]. The minimization of the interconnection is realized by maximizing the use of synchronous transfers through a *rendez-vous* or an interrupt mechanism. These transfers are supported with a simple bus whereas asynchronous transfers require a memorization in the interconnection. While the partitioning proceeds communications cannot be synchronized easily. Therefore, the objective of communication synthesis after partitioning is to synchronize asynchronous transfers such that there is no time constraint violations and the total interconnection area is minimized.

A data transfer can be synchronized with a *rendez-vous* mechanism if the rescheduling of the tasks induced by the synchronization between the sender and the receiver do not generate violations of the application time constraints. A data transfer from a task τ_i to a task τ_j can be synchronized with an interrupt triggered by the sender to the receiver (resp. the receiver to the sender) if the receiver (resp. sender) has a sufficient free memory space to store the transferred data until they are consumed (resp. requested) by τ_j. Interrupt-based synchronous transfers or semi-synchronous transfers, introduce delays on executions of tasks. Note that the available memory in the communicating FSBs may vary in the scheduling according to the size of the local variables of the tasks executed by the FSBs. Due to these delays, semi- synchronous transfers are considered only if they do not induce time constraint violations. The communication synthesis method attempts first to synchronize communications with a *rendez-vous* protocol. If asynchronous communications remains, the methods tries to transform them according to the semi-synchronous protocol.

6. EXPERIMENTAL RESULTS

The capabilities of CODEF are illustrated through the design of a subset of a top box composed of a AC3 audio decode and a V22bis modem. In the 10ms period of the V22bis modem we consider that two audio decoding functions are executed (instead of 11ms). The whole application graph model is then composed of three data flow graphs: a V22bis graph constrained at 10 ms, a AC3 graph constrained at 5 ms and a second AC3 graph starting at 5ms and constrained at 10ms. The whole graph composed of these three functions is composed of 253 nodes and 335 edges.

The library of FSBs is composed of a ARM7TDMI processor, a OAK DSP core, a hardware unit (hw_bitAlloc) dedicated to the execution of the *bit allocation* function of the audio decoder, a coprocessor (psd_copro) of the DSP to speed up the bit-allocation function and a coprocessor of the DSP to speed up the inverse time division aliasing cancellation function of the AC3 decoder. The result of a design space exploration realized by CODEF is il-

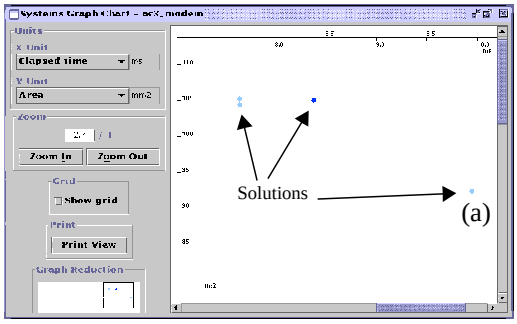

Fig. 6 • Result of the design space exploration.

lustrated in figure 6. This exploration is realized with an empty initial predesigned architecture. Each point corresponds to a different solution. The architecture corresponding to point (a) includes a ARM7TDMI, a OAK DSP and a coprocessor psd_copro connected to it. This solution has no hw_bitAlloc hardware unit. All the others solutions contain this FSB: they are faster but are more costly.

In figure 7, is given the schedule of the two AC3 decoders and the V22bis modem after partitioning. The mean busy period of the OAK is 98% and that of the ARM7TDMI is only 43%. Note that the tasks scheduled close to the time limit (10000 µs) on the ARM7TDMI are issued from the V22bis receiver. Their predecessors are scheduled after the second AC3 decoder which ends at 9200 µs on the OAK DSP. In order to get a lower execution time on the V22bis modem, we do a new partitioning using the modified architecture (a) as a predesigned system. The modifications

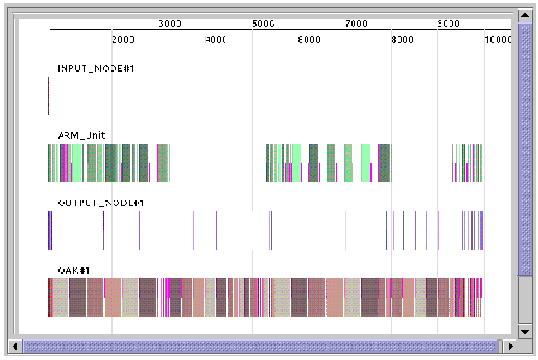

Fig. 7 • Full schedule provided by CODEF

consist in moving to the ARM7TDMI the V22bis receiver tasks scheduled after 9200 µs on the OAK. After partitioning and communication synthesis we get the schedule given in figure 8.The tasks of the AC3 decoder and V22bis modem are mapped on the two processors according to their execution time/memory tradeoffs. The end times of the two AC3 decoders and the V22bis modem are respectively 4.2 ms, 9.3 ms and 7.8 ms. At bottom of figure 8 are illustrated the results of communication synthesis: all the transfers are synchronized and are supported by two buses. Note that the mean busy periods of the buses are low. Since we dispose of an interval of 10.0 - 9.3 = 0.7ms there is room for optimization in order to remove one bus. Improvements of the communication synthesis method to achieve this result are under progress. The Ram and Rom sizes estimated for the processors are:

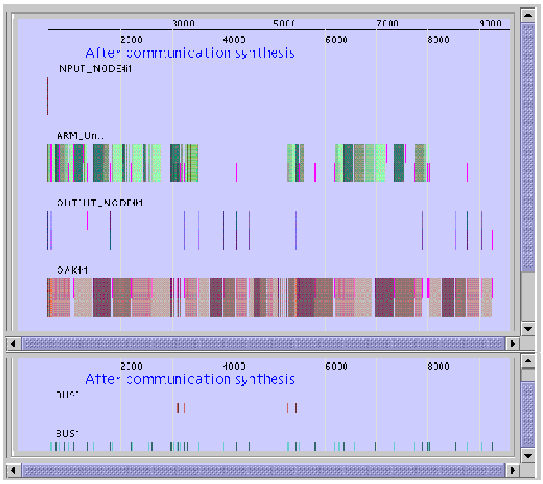

Fig. 8 • Final schedule after communication synthesis

- ARM7TDMI: 4.6kbytes Rom and 89kbytes Ram,
- OAK DSP: 3.6kbytes Rom and 61kbytes Ram.

This example shows the ability of CODEF to explore various system solutions and to provide an efficient interactive assistance for system architecture optimizations.

7. CONCLUDING REMARKS

Our approach is able to determine efficient system specifications from a functional application description. This is achieved through a constructive partitioning/scheduling algorithm followed by an interconnection synthesis step. Partitioning exploits different ratio of reusability and choice of implementation supports of tasks. Interconnection synthesis attempts to minimize the communication resources using synchronous transfers. As illustrated in the experimental result section, after this automatic system design space exploration, a particular solution can be selected for further optimizations using the tool. The algorithm is able to start from scratch or from an initial system architecture. In this case, the tool tries to make the best re-use of the FSBs and the interconnection resources present in the predesigned system.

8. REFERENCES

[1]G. De Micheli, R. Gupta, Hardware/Software Co-Design, Proc. IEEE, vol. 85, No 3, pp 349-364, 1997.

[2]F. Balarin et al. Hardware-Software Co-Design of Embedded Systems: The Polis Approach, Kluwer Academic Press, 1997.

[3]. Buck, E.A. Lee, The token flow model, Data Flow Workshop, Hamilton Island, Australia, mai 1992.

[4]. Bolsens, H.J. De Man, B. Lin, K. Van Rompaey, S. Vercauteren, D. Verkest, Hardware/Software Co-Design of Digital Telecommunication Systems, Proc. IEEE, vol. 85, No 3.

[5]M. Auguin, L. Bianco, L. Capella, E. Gresset, Partitioning conditional data flow graphs for embedded system design, ASAP 2000, Boston, Massachusetts, july 10-12, 2000.

[6]L. Bianco, M. Auguin, A. Pegatoquet, A prototyping method of embedded real time systems for signal processing applications, EUROMICRO'99, 7-10 septembre, Milano, 1999.

[7]F. Cuesta, M. Auguin, E. Gresset, System level communication synthesis for embedded signal processing applications, ICS-PAT, Dallas, october 16-19, 2000.