HLS design flow for the synthesis of multimode systems under multiple constraints

Bertrand Le Gal, Lilian Bossuet
IMS Laboratory, CNRS UMR 5218
ENSEIRB, Bordeaux 1 University, France
Email: surname.name@ims-bordeaux.fr

Shafqat Khan, Emmanuel Casseau
IRISA Laboratory, R2D2
ENSSAT, Lannion, France
Email: surname.name@irisa.fr

Abstract—In a mobile society, more and more devices need to continuously adapt to changing environments. Such mode switches can be smoothly done in software using a general purpose or digital signal processor though hardware components can cope with throughput and power constraints. In this paper we propose a methodology to implement multiple configuration (or mode) and multi-constraint systems into a single circuit using conventional hardware technologies. Results show the interest of the methodology.

I. INTRODUCTION

In many applications, such as personal computers or high-performance computing, the main focus of processor design is to increase the performance of the processor when running a wide set of applications. ASIC allows designer to optimize the hardware for one or more parameters. However due to lack of flexibility, ASICs can not cope with the evolving standards and applications of today’s world. A general purpose processor’s hardware contains all the basic blocks needed to build any logic of mathematical function imaginable but the limitations are memory latencies, limits in the parallelism available in the program, i.e. performance, and power consumption. FPGA provides flexibility but at the cost of large performance, area, power and reconfiguration time penalties.

Keeping in view the requirement of digital design, multimode systems are proposed to realize a set of selected configurations into a single system. The main intention of multimode systems is to implement multiple configurations or modes using conventional hardware technologies. Multimode systems provide both reconfigurability and efficiency in terms of area, performance, power consumption and reconfiguration time. One of the goals of multimode system design is to minimize area by reusing resources effectively among different configurations. High-level synthesis (HLS) generates RTL architectures based on resource sharing. The main idea of this paper is to make use of HLS for the synthesis of multimode systems.

The paper is organized as follow. Section II presents HLS basics and related work on the implementation of multimode architectures. Our technique for the implementation of multi constraint multimode systems is presented in section III. Section IV presents first experimental results obtained applying the proposed methodology.

II. RELATED WORK

A. High Level Synthesis

The increasing DSP application complexity coupled to the time to market constraint urge designers to increase the abstraction level of their work using design tools to implement their applications [1]. High-level Synthesis (HLS) copes with this issue. HLS is analogous to software compilation transposed to the hardware domain with application specification written in a high-level language (Matlab, C, SystemC, etc.). The HLS design flow is constraint-based (timing, area, power consumption, etc.) (Figure 1): hardware resources are selected from libraries of components designed and characterized for a targeted technology. The first synthesis task is the transformation of the algorithmic description to an internal representation model, which captures the algorithmic semantics. From this internal model, four main tasks [2], [3] are performed: (1) internal representation analyzing, computation identifying, (2) hardware resource selection and allocation, (3) operation scheduling optimizing hardware reuse, power consumption and interconnect costs, (4) optimized RTL architecture generation (datapath, control finite-state machine and other units if required).
B. Multi-mode design flows

In order to get better performance of a system, the best option is a hardware implementation rather than a software implementation. In order to automate the design of multimode architectures, designers have tried to formalize methodologies for incorporating additional steps at different levels in conventional HLS processes.

An approach proposed by Chiou and al. [4] for obtaining multimode architectures consists of representing different configurations with data flow graphs (DFGs) and then schedule each DFG separately under given resource/timing constraints. These scheduled DFGs are then concatenated after the scheduling stage using dummy nodes to obtain a unified DFG. In the next step resource binding of the concatenated DFG is performed using maximal weighted bipartite matching algorithm. This technique ensures that resource utilization will be high in the final design. However this technology does not take care of the complexity of the controller.

Another approach for the development of multimode systems was proposed by Kumar and al. [5]. This technique makes use of small scale reconfigurability along with high level synthesis of multimode system. In small scale reconfigurability the chip area is divided into fixed logic and reconfigurable logic areas. Although the design will not be fully reconfigurable like FPGA, this will incorporate the reconfigurability up to the desired extend. In the proposed process for the synthesis of multimode systems, data flow graphs of each configuration are first traversed for critical path and slake analysis (slake is the difference between latency constraint and critical path of a particular DFG). Each DFG is then scheduled in the order of increasing slake. Control step matching is done to increase the resource sharing within control steps. Then resource binding is done in the order of decreasing resource usage. Finally the controller is designed to drive the system.

III. MULTI-CONSTRAINTS MULTIMODE DESIGN FLOW

A. Overview

In this section we propose an approach for the synthesis of multimode architectures that can support different constraint natures. For example, if we have two configurations, one can be optimized for area and the other for performance. Figure 2 shows the proposed design flow. It consists of 3 major steps:

1) Analysis of the applications: In this step the analysis of all the applications which need to be integrated in a single architecture is done independently. The constraints for each application are also analyzed independently. The algorithmic description of each application is transformed into a formal model of representation such as a data flow graph DFG or a Control data flow graph CDFG [6]. The set of constraints (timing, resources,...) imposed by the designer for each application is added to the flow graph model of representation. From this model we find out the mobility of each node corresponding to the different applications in an independent way by calculating the as soon as possible (ASAP) and, if a performance constraint is targeted, as late as possible (ALAP) execution times.

2) Merging of the models of representation: In this step, all the annotated flow graphs are merged in a single flow graph

3) Unified behavioural synthesis: In this step, the high-level synthesis process is applied to the unified model of representation. HLS not only meets all the constraints corresponding to the different applications but also try to minimize the total cost of the multimode architecture (area, power consumption).

B. Design flow

The objective of the approach is to implement multimode systems using high-level synthesis algorithms. In this section we explain the design flow assuming the implementation of two configurations is considered. We assume the behaviour of configuration 1 can be depicted by equation (1) and the behaviour of configuration 2 by equation (2).

\[
E = (A \times B) + (C \times D) - 2
\]

\[
J = 3 \times (F \times G) + (H - I)
\]

1) Graph modelling: To carry out the process of synthesis of multimode architectures, the applications are modelled by their data flow graph DFG (Figure 3).

The configurations may have different constraints (i.e. timing, resources). The management of these constraints will be done in an unified manner during the scheduling and binding steps of the HLS process. In this example, we assume the main emphasis during the synthesis of these two configurations is to meet a timing constraint for configuration 1 and to implement configuration 2 under a resource constraint.
For the set of modes the synthesis begins with the selection of the operator for each operation kind. This selection is based on the type of components available in the hardware library provided by the designer.

After the selection step, the mobility of all the operations are calculated using as soon as possible (ASAP) and as late as possible (ALAP) execution time of each node. Calculation of ASAP and ALAP is done independently for each configuration. In this example the execution time (latency) of each operation is assumed to be one clock cycle. As said before, configuration 1 is constrained for speed and configuration 2 is constrained for limited resources. Figure 4 presents both DFGs annotated with the mobility [ASAP, ALAP] of each operation. Figure 4a shows that there is a minimum timing constraint of four clock cycles for configuration 1. Figure 4b shows that there is a hardware resource constraint for configuration 2 which is depicted by the absence of ALAP execution time.

After the generation of a formal model of each configuration, all the models are merged to obtain a single model. This merging of models will allow considering only one graph during the HLS process. In order to incorporate the notion of timewise mutually exclusive relationship between the different configurations, a "conditional" node is used to emphasize mutually exclusive branches. The merging of $DFG_1$ and $DFG_2$ is shown in Figure 5. As shown in Figure 5, all the configurations are sharing a conditional node whose condition will be evaluated before the execution of any application. All the nodes in this unified model preserve their ASAP and ALAP execution times which were calculated previously. After this step, a model of representation which contains all the operations to be performed for all the possible scenarios as well as their mobility is thus available.

2) Resource allocation: The main objective of the resource allocation is to calculate the number of arithmetic resources required to implement the final architecture while meeting the constraints imposed on each mode. In order to respect the constraints corresponding to each configuration, allocation is first done individually for each configuration (each timewise mutually exclusive branch of the unified model). In a second step, the allocated resources are combined to obtain the number of resources needed for the implementation of all the applications. Since the configuration are timewise mutually exclusive, all the resources are then shared without distinction of configuration.

In the case of a configuration to be synthesised under a resource constraint, it is the task of designer to specify the number of each type of arithmetic resources. On the contrary, in the case of a configuration to be synthesised under a timing constraint, the allocation is automatically performed by the synthesis process.

3) Scheduling and binding: One of the objectives of the scheduling of the proposed multimode architecture synthesis flow is to manage the parallelism of the flow graph while respecting all the synthesis constraints for the different modes. Another objective is to optimize the use of operators whereas the binding step aims at minimizing interconnect overhead. The scheduling algorithm used is based on list scheduling [2]. Mobility is used for main priority function. In [7], to reduce the latency of HLS generated architectures which are constrained by the number of resources, successor and predecessor information are used for ordering operations that have equal priority value. In the proposed synthesis process, successor and predecessor information are used to reduce as much as possible the extra cost due to inter-mode resource sharing. Binding is performed in the same time to maintain this feature.

The basics of the scheduling algorithm are the following. For the scheduling of each control step, first of all we search for the operation which has highest priority among the list of all eligible operations of the unified graph whatever the configuration. This operation is scheduled. In the next step we search in a different configuration the eligible operations.
which can be scheduled in this control step. To reduce the complexity of the controller, the list of eligible operations is processed to find out whether there is any operation which is compatible with the operation which has just been scheduled in this control step. Compatible means that the operation can be executed by the same arithmetic operator. The intention is to schedule that operation whose cost of merging is very small to share the associated resource compare to the use of this resource in one other control step where the operator is not used.

### IV. Experiments

To evaluate the effectiveness of the proposed methodology, several experiments with signal and image processing applications were made. These experiments aim to compare our approach with a conventional approach producing one architecture per couple (mode, constraints). A multimode architecture has been developed which implements in a single component 3 different applications: a 128 tap LMS filter, a 256 tap FIR filter and a subtraction tree (64 data). We target a Xilinx Spartan-3E FPGA and library with 32 bit resources. Results were obtained using the design flow presented in this article and implemented in the GraphLab\(^1\) tool.

An estimation step of the functional complexity of the three algorithms has been done. The FIR filter requires 255 additions and 255 multiplications, the LMS filter requires 255 additions and 256 multiplications and 1 subtraction (both FIR and LMS complexities are quite equal in term of computations but their timing repartition during their execution is different) and subtraction tree requires 64 subtractions.

Three different kinds of constraints have been used in the multi-mode design flow: the FIR filter synthesis is area constrained with 3 adders and 3 multipliers, the LMS filter is time constrained under 70 clock cycles and the subtractor tree is "unconstrained" for an "as fast as possible" implementation (lower area over cost).

To estimate the multi-mode architecture interests, we have performed the synthesis of each application in a mono-mode HLS design flow to obtain their performance parameters. These syntheses have been performed respecting strictly the designer constraints presented before. Area results\(^2\) provided by the Xilinx ISE tools and latency are presented in figure 7. Area results correspond to the complete architecture (data-path, its controller, the memory unit and the memory banks).

We have then applied our multimode design flow. The design area of each application is shown in table 7. As we can see, the multi-mode architecture is 30% smaller than the sum of the other architectures. At the same time, it is important to underline that the FIR filter application has also been speed-up compared to its mono-mode implementation (the latency falls down from 89 clock cycles to 67 cycles). This speed-up is issued from the previously allocated resources for the LMS filter which are shared between all the applications integrated in the multi-mode component. If we now consider the same performance for the mono-mode FIR filter, area increases to 1738 slices. So the total size for the 3 IP components having the same timing performance as the multimode component is about 4064 slices. In this case, the multi-mode component is 34% smaller.

### V. Conclusion

In this paper a methodology for implementing multiple applications with heterogeneous constraints into a single architecture is described. The methodology is based on high-level synthesis. Due to the sharing of resources among different configurations this methodology provides a efficient performance / area trade-off. Experiments show that the multimode design flow allows area saving compare to a monomode design flow but also permit to speed-up applications with efficient hardware sharing. Work is in progress to improve the binding step in order to better reduce the chip area and power consumption.

### References