HW/SW Codesign Techniques for Dynamically Reconfigurable Architectures

Juanjo Noguera and Rosa M. Badia

Abstract—**Hardward/software (HW/SW) codesign and reconfigurable computing are commonly used methodologies for digital-systems design. However, no previous work has been carried out in order to define a HW/SW codesign methodology with dynamic scheduling for run-time reconfigurable architectures. In addition, all previous approaches to reconfigurable computing multicontext scheduling are based on static-scheduling techniques. In this paper, we present three main contributions: 1) a novel HW/SW codesign methodology with dynamic scheduling for discrete event systems using dynamically reconfigurable architectures; 2) a new dynamic approach to reconfigurable computing multicontext scheduling; and 3) a HW/SW partitioning algorithm for dynamically reconfigurable architectures. We have developed a whole codesign framework, where we have applied our methodology and algorithms to the case study of software acceleration. An exhaustive study has been carried out, and the obtained results demonstrate the benefits of our approach.**

Index Terms—**Dynamic scheduling, dynamically reconfigurable architectures, HW/SW codesign, HW/SW partitioning.**

I. Introduction

THE CONTINUED progress of semiconductor technology has enabled the "system-on-chip" (SoC) to become a reality. In this sense, programmable logic manufacturers have also proposed new products. An example of this is the Altera's new device Excalibur, which integrates a processor core (ARM, MIPS, or NIOS), embedded memory and programmable logic [1]. New types of devices, which are run-time reconfigurable, have also been proposed thanks to the advents of the dynamically reconfigurable logic (DRL). An example of this, is the Virtex family from Xilinx, which is partially reconfigurable at run time [2]. A hybrid device, which combines both above explained features, is the CS2112 chip from Chameleon Systems, Inc. This device integrates a RISC core, embedded memory, and a run-time reconfigurable fabric on a single chip [3]. Clearly, all these devices could be used as the final-target architecture of a hardware/software (HW/SW) codesign methodology.

Reconfigurable computing (RC) [16] is an interesting alternative to application specific integrated circuits (ASICs) and the general-purpose processor systems, since it provides the flexibility of software processors and the efficiency and throughput of hardware coprocessors. DRL devices and architectures present new and exciting challenges to the design automation community. The major challenge introduced by DRL devices is the *reconfiguration latency*, which must be minimized in order to maximize application performance. In order to achieve run-time reconfiguration, the system specification (typically, a task graph) must be partitioned into temporal exclusive segments (called *reconfiguration contexts*). This process is usually known as *temporal partitioning* and it is a way to address the problem of reconfiguration latency. A different approach is to find an execution order for a set of tasks that meets system-design objectives (i.e., minimize the application execution time). This is known as *DRL multicontext scheduling*.

There are a great number of approaches to HW/SW codesign of embedded systems, which use different techniques for partitioning and scheduling. However, DRL devices and architectures change many of the basic assumptions in the HW/SW codesign process. The flexibility of dynamic reconfiguration (multiple configurations, partial and run-time reconfiguration, etc.) requires new methodologies and the development of new algorithms, as conventional codesign approaches do not consider the features of these new DRL devices and architectures.

A. Previous Related Work

Traditionally, HW/SW codesign challenges and DRL challenges have been addressed independently.

Earlier approaches to the HW/SW codesign, model the system based on a template of a CPU and an ASIC [11], [14]. HW/SW partitioning and scheduling techniques can be differentiated in several ways. For instance, partitioning can be classified as *fine-grained* (if it partitions the system specification at the basic-block level) or as *coarse-grained* (if system specification is partitioned at the process or task level). Also, HW/SW scheduling can be classified as *static* or *dynamic*. A scheduling policy is said to be static when tasks are executed in a fixed order determined offline, and dynamic when the order of execution is decided online. HW/SW tasks' sequence can change dynamically in complex embedded systems (i.e., control-dominated applications), since such systems often have to operate under many different conditions. Although, there has been a lot of previous work in static HW/SW scheduling, the dynamic scheduling problem in HW/SW codesign has only been addressed in a few research efforts. A strategy for the mixed implementation of dynamic real-time schedulers in HW/SW is presented in [27]. In [4] a review of several approaches to control-dominated and dataflow-dominated software scheduling is presented.

Manuscript received October 30, 2000; revised January 18, 2002. This work was supported by CICYT-TIC project TIC2001-2476-C03-02.

J. Noguera is with Hewlett-Packard Inkjet Commercial Division, Department of Research and Development, San Cugat del Valles, 08190 Spain (e-mail: jnoguera@bpo.hp.com).

R. M. Badia is with the Technical University of Catalonia (DAC-UPC), Computer Architecture Department, Barcelona, 08034 Spain (e-mail: rosab@ac.upc.es).

Digital Object Identifier 10.1109/TVLSI.2002.801575

On the other hand, several approaches can be found in the literature addressing reconfiguration latency minimization. *Configuration prefetching* techniques have been used for reconfiguration latency minimization. They are based on the idea of loading the next reconfiguration context before it is required, hence, overlapping device reconfiguration and application execution. Hauck first introduced configuration prefetching in [15], where a single-context prefetching technique is presented. Also, several references can be found in the literature addressing temporal partitioning and multicontext scheduling for reconfiguration latency minimization. See [24] as an example. All these previous approaches address the problem of reconfiguration latency minimization, but they do not address HW/SW partitioning and scheduling.

Recent research efforts have addressed this open problem. In [7], an integrated algorithm for HW/SW partitioning and scheduling, temporal partitioning, and context scheduling is presented. A more recent work [22] presents a fine-grained HW/SW partitioning algorithm (at loop level). Both previous approaches are similar to [12] which take the reconfiguration time into account when performing the partitioning, but they do not consider the effects of configuration prefetching for latency minimization. In [17], this topic is introduced, and a HW/SW cosynthesis approach for partially reconfigurable devices is presented. They do not address multicontext devices. Moreover, this approach, which is based on an ILP formulation, is limited by the high execution times of the algorithm, which hardly gives solutions for task graphs having more than ten tasks.

B. Motivation and Contributions of the Paper

New approaches are possible because 1) all existing approaches to DRL multicontext scheduling are based on static (compile time) scheduling techniques and 2) no previous work has been carried out in order to define a HW/SW codesign methodology with dynamic scheduling based on DRL architectures.

In this paper, we address these two open problems and present the following: 1) a novel HW/SW codesign methodology with dynamic scheduling for dynamically reconfigurable architectures, 2) a dynamic approach to DRL multicontext scheduling, and 3) an automatic HW/SW partitioning algorithm for DRL architectures. The proposed algorithm takes into account the reconfiguration latency when performing the HW/SW partitioning. The experiments carried out demonstrate that the benefits of using a prefetching technique, for reconfiguration latency minimization, can be improved if it is considered at the HW/SW partitioning level.

The rest of the paper is organized as follows: Section II introduces the HW/SW codesign methodology with dynamic scheduling, and the target architectures (named, local, and shared memory architectures). In Sections III and IV, we present two-different algorithms (HW/SW partitioning and DRL multicontext scheduling) for the shared and local memory architectures. In Section V, we apply our methodology and algorithms to the software acceleration of telecom networks simulation, and present the obtained results. An improved HW/SW partitioning algorithm for DRL architectures is presented in Section VI. Finally, Section VII presents the conclusions of this work.

II. HW/SW Codesign for Discrete Event Systems

Discrete event (DE) systems design has been recently addressed using HW/SW codesign techniques [13], [21]. However, none of these approaches is based on DRL devices as the hardware platform.

The methodology presented here addresses the problem of HW/SW codesign with dynamic scheduling for DE systems using a heterogeneous architecture that contains a standard off-the-shelf processor and a DRL based architecture. It is important to note that the proposed methodology follows an object orientation paradigm, and uses the object oriented concepts in all its steps (specification, HW/SW partitioning and scheduling, etc.). The majority of previous related work only uses this object oriented approach at the system specification (modeling) level. See [37], as an example.

A. Definitions

1) *Discrete event class* is a concurrent process type with a certain behavior, which is specified as a function of the state variables and input events. See Fig. 1(a).
2) *Discrete event object* is a concrete instance of a DE class. Several DE objects from a single DE class are possible. Given two DE objects (DEO_1 and DEO_2) they may differ in the value of their state variables. See Fig. 1(b).
3) *Event* E is a member of $T \times C \times O \times V$ where T is a set of tags, $T \in \Re^+$ (the real numbers), C a given set of DE classes, O a set of DE objects, and V a set of values. Tags are used to model time and values represent operands or results of event computation.
4) *Event stream* (ES) is a list of events sequentially ordered by tag. Tags can represent, for example, event occurrence or event deadline. See Fig. 1(c).
5) *Discrete event functional unit* is a physical component (i.e., DRL device or SW processor) where an event $e = (t, c_1, o_1, v_1)$ can be executed. A functional unit has an active pair (*class, object*), $p = (c_a, o_a)$. See Fig. 1(d).

   Our methodology assumes that: (1) several DE classes could be mapped into a single DE functional unit, and (2) all DE objects from a DE class are mapped into the same DE functional unit where the DE class has been mapped.
6) *Object switch* is the mechanism that allows a DE functional unit to change from one DE object to another, both DE objects belonging to a same DE class. For example, if an input event $e = (t, c_-, o_1, v_1)$ has to be processed in a DE functional unit with an active pair $p = (c_1, o_2)$ then an object switch should be performed. Object switch means a change of values in the state variables from the ones of a concrete DE object (o_1) to the others of another DE object (o_2).
7) *Class switch* is the mechanism that allows a DE functional unit to change from one DE class to another. For example, if an input event $e = (t, c_1, o_-, v_1)$ should be processed in a DE functional unit with an active pair $p = (c_2, o_2)$,

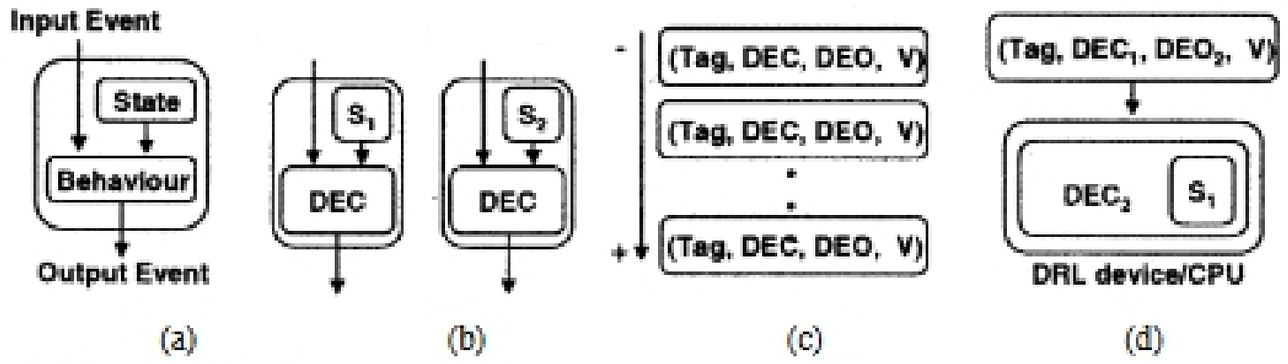

Fig. 1. HW/SW codesign methodology definitions.

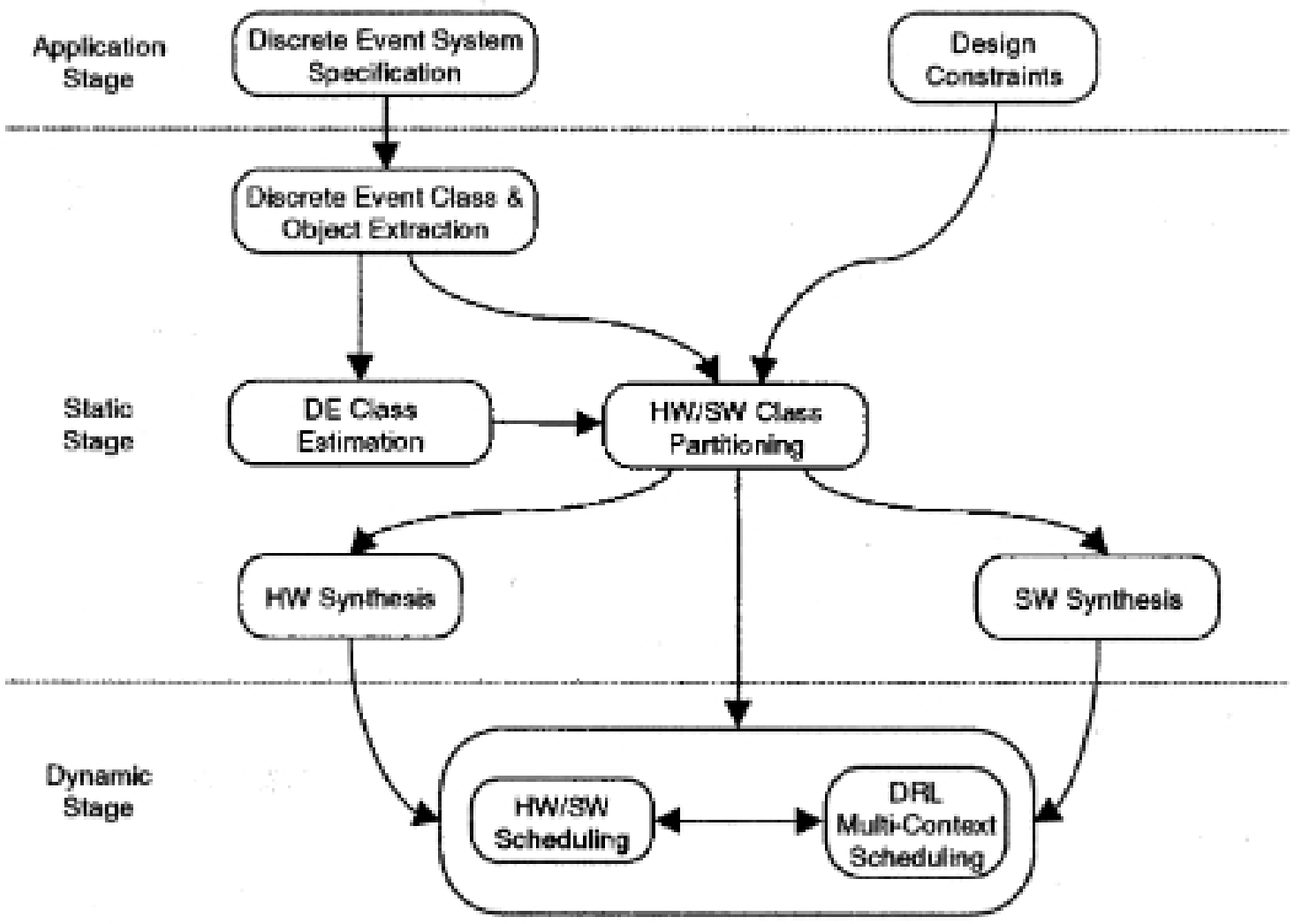

Fig. 2. HW/SW codesign methodology.

then a class switch should be performed. Class switch, in case of a DRL device, means a context reconfiguration.

B. HW/SW Codesign Methodology With Dynamic Scheduling

The methodology we propose is depicted in Fig. 2. It is divided in three stages: *application stage*, *static stage*, and *dynamic stage*. The key points in our methodology are: (1) *application* and *dynamic stages* handle DE classes and objects, and (2) *static stage* only handles DE classes.

The *application stage* includes *discrete event system specification* and *design constraints*. We assume the use of an homogenous modeling language for system specification, where a set of independent DE classes must be first modeled. Afterwards, these DE classes are used to specify the entire system as a set of interrelated DE objects, which communicate among them using events. These DE objects are interrelated creating a concrete topology. A DE object computation is activated upon the arrival of an event. By design constraints we understand any design requirement necessary when synthesizing the design (i.e., timing or area requirements).

The *static stage* includes typical phases of a codesign methodology: (1) *estimation*, (2) *HW/SW partitioning*, (3) *HW and SW synthesis*, and (4) *extraction*.

As previously stated, the *static stage* handles DE classes. However, the system has been specified as a set of interrelated DE objects, which are instances of previously specified DE classes. The final goals of the methodology's extraction phase are, for a given DE class, to obtain 1) a list of all its instances (DE objects) and 2) a list of all different DE classes and objects connected to it. Both lists are afterwards attached to each DE class found in the system specification. Once this step has finished, DE classes can be viewed as a set of independent tasks.

Note that although our methodology, addresses DRL architectures, a temporal partitioning phase can not be found. The DE object/class extraction phase should be viewed as the temporal partitioning algorithm. Indeed, the temporal partitioning algorithm is included within our concept of DE class, as DE classes are functionally independent tasks.

We classify our HW/SW partitioning approach as coarse-grained, since it works at the DE class level. Different HW/SW partitioning algorithms can be applied depending on the application. The solution obtained by the HW/SW partitioning algorithm should meet design constraints.

The estimation phase also deals with DE classes and the used estimators depend on the final application. Typically used estimators (HW/SW execution time, DRL area, etc.) can be obtained using high-level synthesis and profiling tools.

The *dynamic stage* includes *HW/SW Scheduling* and *DRL multicontext Scheduling*. Both schedulers base their functionality on the events present in the event stream. Our methodology assumes that both of them are implemented in hardware using a centralized control scheme. As it is shown in Fig. 2, these scheduling policies (HW/SW and DRL) cooperate and run in parallel during application run-time execution, in order to