Operating Systems for Reconfigurable Embedded Platforms: Online Scheduling of Real-Time Tasks

Christoph Steiger, Herbert Walder, *Member*, *IEEE*, and Marco Platzner, *Member*, *IEEE*

Abstract—Today's reconfigurable hardware devices have huge densities and are partially reconfigurable, allowing for the configuration and execution of hardware tasks in a true multitasking manner. This makes reconfigurable platforms an ideal target for many modern embedded systems that combine high computation demands with dynamic task sets. A rather new line of research is engaged in the construction of operating systems for reconfigurable embedded platforms. Such an operating system provides a minimal programming model and a runtime system. The runtime system performs online task and resource management. In this paper, we first discuss design issues for reconfigurable hardware operating systems. Then, we focus on a runtime system for guarantee-based scheduling of hard real-time tasks. We formulate the scheduling problem for the 1D and 2D resource models and present two heuristics, the *horizon* and the *stuffing* technique, to tackle it. Simulation experiments conducted with synthetic workloads evaluate the performance and the runtime efficiency of the proposed schedulers. The scheduling performance for the 1D resource model is strongly dependent on the aspect ratios of the tasks. Compared to the 1D model, the 2D resource model is clearly superior. Finally, the runtime overhead of the scheduling algorithms is shown to be acceptably low.

Index Terms—FPGA, partial reconfiguration, operating system, online scheduling, real-time.

1 Introduction

Embedded computing platforms are composed of a variety of different processing elements, memories, I/O devices, sensors, and actors. The choice of processing elements includes instruction-set processors, application-specific fixed-function hardware (ASICs), and (re)configurable hardware devices. SRAM-based reconfigurable devices, typically field-programmable gate arrays (FPGAs), store the hardware configuration in static memory cells. These devices can be reconfigured arbitrarily often with reconfiguration times on the order of milliseconds.

SRAM-based FPGAs were introduced as high-end devices for implementing random logic in the mid-1980s. In the following years, FPGAs have found a number of additional and novel uses in the design of embedded systems. One example is rapid prototyping and emulation. More recently, reconfigurable hardware is being used as an ASIC replacement with much shorter time-to-market and the novel ability to update hardware after product deployment. Today, the increasing densities of reconfigurable devices and the trend to integrate them with processors, memories, and special function blocks on configurable systems on a chip (CSoC) [1], [2] advocate more *dynamic uses* of reconfigurable hardware in embedded systems. A higher degree of dynamics is further facilitated by partial reconfiguration, a technique that allows us to reconfigure only a fraction of the reconfigurable hardware resources while the other part continues to execute [3].

Many promising application domains for *reconfigurable embedded systems* combine high performance demands with frequent changes of their workloads. Examples for such application domains are found in wearable computing [4], mobile systems [5], and network processors [6]. The dynamics in these systems are caused by user requests and packet flows in the communication networks. Mobile and wearable systems additionally operate in changing physical environments and contexts, which reflects in different types of workloads. Consequently, neither the set of functions nor the time at which these functions will be executed is exactly known in advance. A classic system design process with complete design-time synthesis and optimization is no longer possible. The required degree of flexibility paired with high computation demands asks for partially reconfigurable hardware that is operated in a true multitasking manner.

Multitasking reconfigurable hardware raises a number of novel issues, ranging from programming models to runtime systems. A programming model defines the executable objects and their interaction and provides the developer with a set of well-defined system services. A runtime system efficiently operates the system and resolves conflicts between executable objects. A programming model, together with a runtime system, forms a *reconfigurable hardware operating system* [7], [8]. A reconfigurable hardware operating system can be compared to embedded real-time operating system (RTOS) kernels for microprocessors. Such kernels have been industry standard for years and offer a minimal programming model by specifying a set of objects, e.g., tasks, buffers, semaphores, timers, and their possible

- *C. Steiger is with ESA/ESOC, Robert-Bosch-Strasse 5, 64293 Darmstadt, Germany. E-mail: Christoph.Steiger@esa.int.*
- *H. Walder and M. Platzner are with the Computer Engineering and Networks Lab, Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, 8092 Zurich, Switzerland. E-mail: walder@tik.ee.ethz.ch, marco.platzner@computer.org.*

Manuscript received 1 Dec. 2003; revised 11 Apr. 2004; accepted 8 June 2004. For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TCSI-0251-1203.

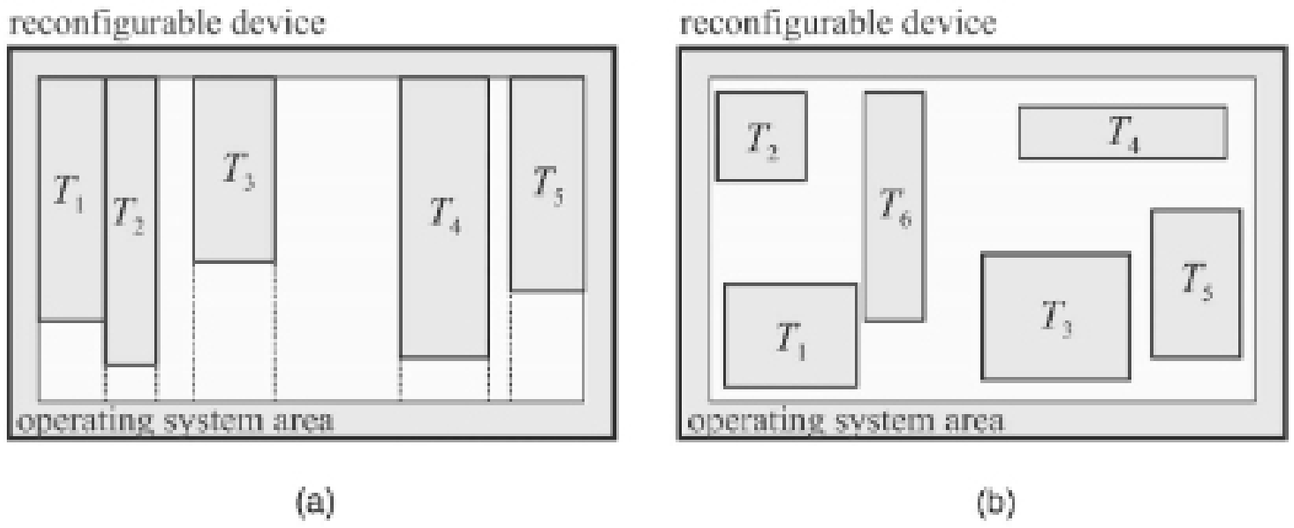

Fig. 1. Reconfigurable resource models: (a) 1D area model; (b) 2D area model.

interactions. The main abstraction is the introduction of a smallest unit of execution, which is mostly denoted as task (alternatively, the smallest unit of execution is called process or thread).

Our long-term goal is to develop an operating system for reconfigurable embedded platforms by building the same abstractions for partially reconfigurable hardware as an RTOS builds for microprocessors. Besides enabling *hardware multitasking*, the benefits of such an operating system are increased productivity and portability. The abstractions of system objects and tasks enable the reuse of tested and reliable code and circuitry, which can considerably speed up development cycles and shorten time-to-market. Application ports are greatly simplified, provided the operating system is available on different target platforms. Eventually, all processing elements of an embedded reconfigurable platform should be managed by one operating system that deals with software tasks running on a microprocessor and hardware tasks running on reconfigurable hardware. Tasks that are available for both software and hardware facilitate dynamic system repartitioning [9].

Reconfigurable hardware operating systems are a rather new line of research, where only a few issues have been addressed yet. In this paper, we first discuss conceptual and practical aspects of reconfigurable hardware operating system design. Then, we concentrate on the runtime functions of such an operating system. The main contribution of this paper is the development and evaluation of heuristics for online scheduling of hard real-time tasks to partially reconfigurable devices. The proposed scheduling heuristics are applicable to both commonly used reconfigurable resource models, the 1D and 2D area model.

The remainder of this paper is structured as follows: In Section 2, the basic operating system abstractions—task and resource models—are presented, followed by a survey of related work and a discussion of the limitations of current FPGA technology. The design of our reconfigurable hardware operating system is outlined in Section 3. Section 4 defines the online scheduling problem and presents two heuristics to solve it. An experimental evaluation of the proposed heuristics based on synthetic workloads is done in Section 5. Finally, Section 6 concludes the paper.

2 Models and Limitations

This section first presents the basic models for hardware tasks and reconfigurable devices that are being used for conceptualizing and constructing operating systems for reconfigurable platforms. Then, we review related work in the area. Finally, we discuss the limitations of task and resource models when it comes to practical implementation in currently available technology.

2.1 Basic Task and Device Models

A hardware task is a synthesized digital circuit that has been preplaced and prerouted. The task is stored in a position-independent way and can be relocated to different locations on the reconfigurable device by the operating system. Hardware tasks have several characteristics. The functional characteristic captures the task behavior and is not visible to the operating system. The structural and timing characteristics, however, are exposed to the runtime system for scheduling and placement.

The main structural characteristics are size (area) and shape. Hardware tasks have a certain area requirement, given in numbers of reconfigurable units (RCUs). The shape of a hardware task is mostly modeled by a rectangle including all RCUs as well as the routing resources used by the task. Compared to more complex shapes, such as polyominoes [10], [11], rectangular shapes simplify task placement. However, rectangular shapes also lead to *internal fragmentation*, i.e., the unused area inside the rectangle.

The main timing characteristic of a hardware task is the clock range at which the task can run. Design tools usually report an upper bound for the clock rate. A task may, however, require a specific clock rate, for example, to derive a timer object that relates events to physical time. A task might further require a clock rate in a certain interval to preserve timing requirements of I/O devices or memory. Further timing characteristics, that might or might not be known in advance, are the number of clock cycles to execute and the deadline of a real-time task.

The complexity of mapping tasks to devices depends heavily on the *area model* used. The two main area models are the 1D and the 2D model shown in Fig. 1. In both models, the reconfigurable device is represented by a rectangular area of RCUs. A part of the reconfigurable resource is reserved for operating system functions (denoted as operating system area in Fig. 1). The remaining part is the hardware task area. In the simpler 1D area model, tasks can be allocated anywhere along the horizontal device dimension; the vertical dimension is fixed and spans the total height of the hardware task area. The 1D area model leads to simplified scheduling and placement problems. However, the model suffers from two types of *external fragmentation*. The first type of external fragmentation is the area wasted when a task does not utilize the full height of the task area. The second type of fragmentation

arises when the remaining free area is split into several small but unconnected vertical stripes. External fragmentation can prevent the placement of further tasks although sufficient free area exists.

The more complex 2D area model allows us to allocate tasks anywhere on the hardware task area and suffers less from external fragmentation. Consequently, a higher device utilization can be expected. On the other hand, the high flexibility of this model makes scheduling and placement rather involved.

Formally, a task T_i is modeled as a rectangular area of reconfigurable units given by its width and height, $w_i \times h_i$. Tasks arrive at arbitrary times a_i. Real-time tasks require execution times e_i and carry deadlines d_i, $d_i \geq a_i + e_i$. The hardware task area of the reconfigurable device is modeled as a rectangular area $W \times H$ of reconfigurable units.

2.2 Related Work

A substantial body of work has been done in offline optimization of reconfigurable embedded systems. Examples are temporal partitioning for nonpartially reconfigurable systems, e.g., by Purna and Bhatia [12], or 3D placement of tasks in time and space dimensions for partially reconfigurable devices, e.g., by Fekete et al. [13]. In an offline scenario, one can afford to spend the time to derive optimal or near-optimal solutions. In contrast, operating systems work on online scenarios and require efficient algorithms.

Brebner [7], [14] was among the first to propose an operating system approach for partially reconfigurable hardware. He defines swappable logic units (SLUs), which are position-independent tasks that are swapped in and out by the operating system. Jean et al. [15] discuss an online scenario where a resource manager schedules arriving tasks to a farm of FPGAs. Since each task occupies exactly one FPGA, there is no partial reconfiguration and placement is not an issue. Merino et al. [16], [17] split the reconfigurable surface into an array of predefined subareas, so-called slots. The operating system schedules tasks to these slots based on a task allocation table that keeps track of currently loaded tasks. As each task fits into one slot, there is again no placement problem involved. Simmler et al. [18] discuss task switching in a preemptive environment. The authors further propose critical sections, i.e., periods of time during which a task must not be preempted. However, their device is not partially reconfigured. Burns et al. [19] describe several operating system functions, including a 2D transform manager that performs translation and rotation operations on tasks to better fit them to the device. Shirazi et al. [20] propose three runtime modules, a monitor, a loader, and a configuration store, to manage reconfigurable designs. The authors discuss trade offs between reconfiguration time and circuit quality depending on the reconfiguration method used and information about the configuration sequence that is available at compile time.

Several authors have addressed the problem of external fragmentation and tackled it by compaction, i.e., rearranging executing tasks in order to maximize the contiguous free area. While these techniques increase the chance of successful future placements, compaction requires preemptive systems and can delay task execution for a considerable period of time. Diessel et al. [21], [22] investigate compaction in the 2D area model. They perform task rearrangement by techniques denoted as local repacking and ordered compaction. Compton et al. [10], [23] discuss task relocations and, additionally, task transforms to reduce fragmentation. Task transforms consist of a series of rotation and flip operations. The authors also propose a novel FPGA architecture that supports efficient row-wise relocation. A different compaction approach for the 1D area model is presented by Brebner and Diessel in [24], where an FPGA itself contains circuitry that determines the relocation positions.

The problem of placement in the 2D area model has been addressed by Barzagan et al. [25]. The authors investigate efficient data structures and algorithms for fast online placement. Simulation experiments for variants of first-fit, best-fit, and bottom-left bin-packing algorithms are conducted. The 2D placer we use in this paper relies on our previous improvements [26] of the work of Bazargan et al. Further, we have also experimented with the placement of tasks that consist of a number of rectangular subtasks [11]. Such shapes result naturally from a core-oriented design style and can be modified by footprint transforms, i.e., by arranging the subtasks differently.

Recent work in reconfigurable hardware operating systems has covered a broader range of runtime functionalities as well as prototype construction. Wigley and Kearney [27] identify a set of services that should be offered by reconfigurable operating systems, including partitioning, allocation, placement, and routing. Mignolet et. al [9] present a networked reconfigurable platform for multimedia appliances that enables multitasking in hardware and software. In [28], the authors discuss interconnection networks on reconfigurable devices. Design issues for a reconfigurable hardware operating system have been presented in [29] and a prototype implementation in [5].

Like some of the related work, the main part of this paper addresses online task and resource management for partially reconfigurable devices. This work is an extension of a previously published conference contribution [30]. The difference from related work is that we target a real-time scenario where each incoming task is either accepted with a guarantee to meet the deadline or rejected. For such a scenario, we compare different scheduling heuristics under both the 1D and 2D area models. The feasibility of the 1D area model is backed by our prototyping work on Xilinx Virtex technology. The 2D model is not enabled by currently available FPGA technology. Despite this fact, the 2D area model is very popular and has often been used in related work.

2.3 Limitations of Current Technology

The main abstraction is that tasks are modeled as relocatable rectangles that can be placed anywhere on the device in the 2D area model and anywhere along the horizontal device dimension in the 1D area model. While the latest FPGA design tools [31] allow us to constrain tasks to rectangular areas at the price of some internal fragmentation, the relocatability raises a number of questions. These questions concern the device homogeneity, task communication and timing, and the partial reconfigurability.

The placement and scheduling algorithms of the runtime system assume homogeneous reconfigurable devices, which is in contrast to modern FPGAs that contain special resources such as dedicated memories and embedded multipliers [1]. Although these special resources are