A Unified HW/SW Operating System for Partially Runtime Reconfigurable FPGA based Computer Systems *

Qingxu Deng[1], Yi Zhang[1], Nan Guan[1] and Zonghua Gu[2]

[1] Northeastern University, Shenyang, China
[2] Hong Kong University of Science and Technology, Hong Kong, China

Abstract

Partially Runtime-Reconfigurable (PRTR) FPGAs allow hardware tasks to be placed and removed dynamically at runtime. We present an OS for hybrid computing systems consisting of both CPUs and PRTR FPGAs. The OS is based on Linux, and provides unified interfaces for both HW and SW processes to ease the design of such hybrid systems. The scheduler of HW processes is implemented on the hardware, to alleviate the performance penalty of the time-consuming HW task scheduling algorithms.

1 Introduction

A Partially Runtime-Reconfigurable (PRTR) FPGA (referred to as FPGA for short), such as the Virtex family FPGAs from Xilinx [1], is composed of a rectangular grid of Configurable Logic Blocks (CLBs) and the interconnects between them. A FPGA allows part of the area to be reconfigured while the remainder continues to operate without interruption and is regarded as a 2D continuous processing area that can hold a lot of HW tasks. In other words, HW tasks can be allocated and deallocated dynamically at runtime.

Traditionally, designing HW/SW hybrid systems is a very tough work. The HW and SW part were developed separately, and later pieced together. Since standard interfaces and services have not yet been established [9] [18], designers are forced to literally build systems from scratch. A unified interface for both HW and SW can provide clean separation of the system design and implementation, which is the base of system design standardization.

Furthermore, a HW/SW interface that is familiar and easy to understand will greatly facilitate the transition from past supercomputers or computer clusters based systems into HW/SW hybrid platforms [11].

*This work is partially supported by the National High Technology Research and Development Program of China (863 Program) under Grant No. 2007AA01Z181 and the National Natural Science Foundation of China under Grant No. 60773220

In this paper we present our on-going work on a Linux-based OS for hybrid systems consisting of both CPUs and FPGAs. The OS provides unified interfaces for both HW processes and SW processes. Since HW scheduling algorithms are usually very time-consuming, we migrate this part of work of OS into hardware in order to reduce the runtime overhead and improve real-time performance of the system.

The remained part of the article is organized as follows. We introduce the related work in Section 2, and then present our OS prototype design in Section 3. Finally future work is discussed in Section 4.

2 Related Work

The concept of OS for reconfigurable computing systems was firstly proposed by Brebner et al. [3]. Wigley et al. [17] discussed several issues in OS for reconfigurable computing systems, including HW tasks downloading, FPGA area management, HW tasks scheduling, storage management and protection, I/O communications, HW tasks communication and fragmentation metrics.

Walder and Platzner described a OS prototype for reconfigurable computing systems in [15] [12]. They discussed the online scheduling of HW tasks and implemented HW task scheduler and placer in their OS prototype. Their work shows a good paradigm of the runtime system for HW multitasking. However, they didn't consider the unified management of HW tasks and SW tasks.

Rissa and Niittylahti [10] introduced a HW/SW hybrid system, where FPGAs on a PCI-broad are connected to general computers via the PCI bus. Wiangtong presented the UltraSONIC system based on a similar architecture in [16], where they mainly focused on the system HW/SW co-design, and introduced a HW/SW co-design environment DAG.

Kwok-Hay et al. [11] introduced a Linux-based system BORPH, which provides unified interfaces for both SW and HW at the OS kernel level. BORPH provides HW processes with Unix-standard access interfaces and BOF (a ELF based file format) in order to unify the operations of both SW and HW tasks. BORPH is implemented on the BEE2 module hardware platform. Every BEE2 module contains five FPGAs (one

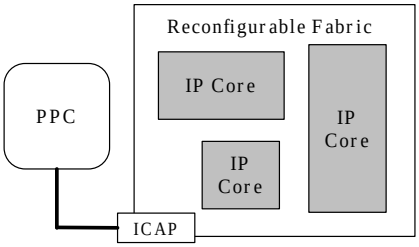

Figure 1. Hardware Platform

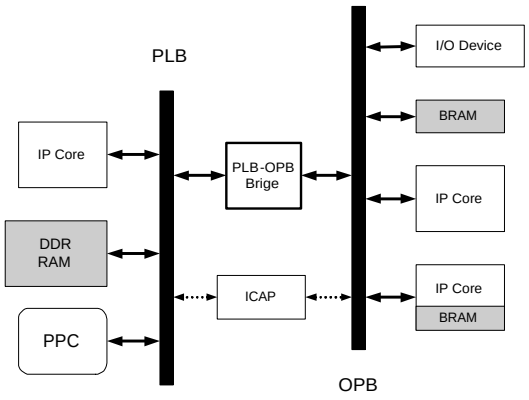

Figure 2. Hardware Platform

control FPGA and four User FPGAs). They defined a bus architecture for inter-FPGA communication. The BORPH system didn't consider the partially reconfiguration and HW tasks scheduling problem.

Agron et al. [2] proposed a CPU/FPGA hybrid system Hthread on CSoC (Configurable SoC), where some run-time system components like Thread Manager, Scheduler, Mutex Manager, and a new CPU Bypass Interrupt Scheduler (CBIS) are migrated into the reconfigurable fabric on an FPGA. Migrating these services into hardware brings significant performance benefits to software threads through more efficient invocation and processing mechanisms as well as helps in eliminating the hidden overhead of context switch times associated with entering and exiting the RTOS. The Hthread system is not based on Linux, but built around their own-developed APIs that are compatible with the POSIX thread standard.

3 System Design

3.1 Overview

The system is implemented on a Virtex-II Pro XC2VP30 FPGA, which contains a PowerPC405 hardcore and partially runtime reconfigurable fabrics, as shown in Fig. 2. Multiple HW tasks can simultaneously execute on the reconfigurable fabrics, be allocated and deallocated dynamically at runtime without interrupting other HW tasks. The off-chip DDR memory is used as the system main memory, and on-chip BRAMs (Block RAM) are used by IPCores as their own storage resource. IPCores are connected to the PLB (Processor Local Bus) or OPB (On-chip Peripheral Bus), depending on their communication bandwidth demand. The PowerPC hardcore accesses ICAP (Internal Configuration Access Port) via the high-speed PLB to configure the reconfigurable fabrics.

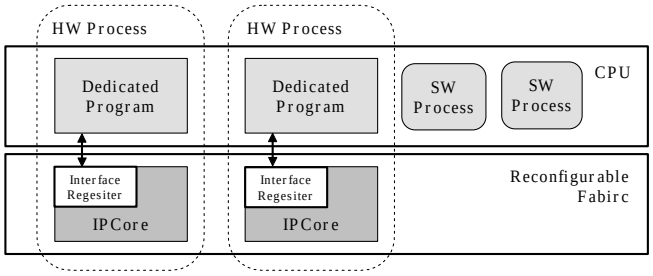

Figure 3. Hardware Platform

Our goal is to develop an OS in order to:

- Provide a unified process model for both SW tasks and HW tasks;
- Enable on-line placement and scheduling of HW processes at OS level.

We choose Linux 2.6 as the foundation of our OS prototype. In contrast to Linux 2.4, Linux 2.6 supports preemptions in the kernel mode, which benefits the on-line management of HW processes (will be discussed in Section 3.3).

3.2 The HW Process Model

Both the HW tasks and SW tasks are implemented as processes in our system. A HW process consists of two parts: the IPCore (hardware part) and the Dedicated Program (software part), as shown in Fig. 3. IPCores take charge of the computation work, while the Dedicated Program encapsulates the communication operations between the IPCore and the system. The Dedicated Programs are instantiated from the same template, and access IPCores by the same device driver, so HW process designers do not need to write any software program, but only need to implement their IPCore conforming to the pre-defined interface standard.

3.2.1 Communication

The Dedicated Program accesses the IPCore via some specific registers in the IPCore , which is named as Interface Register. The Interface Register consists of two parts: (1) State Registers, which show the current state of the IPCore and (2) Data Registers, which store the communication data.

The passive communication of the HW process is quite simple. When some process P wants to send data to a HW process H, the procedure is:

1. P sends data to H's Dedicated Program;

2. H's Dedicated Program write data to the Data Registers of the IPCore.

When some process P wants to get data from the HW process H, the procedure is:

1. P sends a message to H's Dedicated Program to denote which data are required;

2. H's Dedicated Program reads data from the assigned Data Registers of the IPCore.

3. H's Dedicated Program sends data to P.

The active communication of the HW process is a little more complicated. To enable IPCores to initiate communications, we bind a unique interrupt source to each HW process. When the HW process H wants to send data to some other process P, the procedure is:

1. H's IPCore updates the State Registers.

2. H's IPCore generates an interrupt requirement.

3. H's Dedicated Program answers this interrupt: looks up the State Registers and get data from the IPCore.

4. H's Dedicated Program send these data to P.

When the HW process H wants to get data from some other process P, the procedure is:

1. H's IPCore updates the State Registers.

2. H's IPCore generates an interrupt requirement.

3. H's Dedicated Program answers this interrupt: look up the State Registers and get data from P.

4. H's Dedicated Program sends Data to the IPCore.

The Interfaces Registers are mapped to the system address, and Dedicated Programs access the Interfaces Registers by direct reading/writing operation to the corresponding address. The address range of each IPCore is 4K, which equals to the size of a page. This is for future extension of mapping IPCore's internal BRAM to the system address, in order to facilitate the data-stream style communication.

Since all HW processes share the same Dedicated Program template, the communication operation supported by the Dedicated Program should be simple and application-independent. Currently only three simple communication mechanisms are supported for HW processes: (1) pipeline, (2) signal and (3) message.

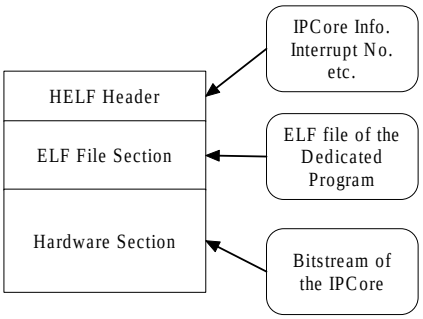

Figure 4. Hardware Platform

3.2.2 File Format

We design a new file format HELF for HW processes by extending the ELF file format. A HELF file consists of three parts, as shown in Fig. 4:

- HELF Header: HW process's basic information, like the width/height of the IPCore, the interrupt source no. etc.
- ELF Section: The ELF file of the Dedicated Program.
- Hardware Section: The bitstream of the IPCore.

Since all HW processes' Dedicated Programs are exactly the same, the ELF section could be compiled into a executable code in prior, and directly linked into each HW process's HELF file.

3.3 On-line Scheduling of HW Process

The on-line scheduling[1] of HW tasks on PRTR FPGA is much more complicated than SW scheduling. SW tasks only share computing resources in the time dimension, while HW tasks share computing resources in not only the time but also the space dimension. The on-line HW task scheduling algorithms are usually very time-consuming.

If the HW task scheduling algorithm is implemented in the software, its execution time could be quite long and it will heavily degrade the real-time performance of the system. So we implemented the HW task scheduling algorithm on hardware.

We add functions to the original "exec()", to recognize HELF files and extract the information of the HW process from the HELF Header, e.g., its width, height, WCET, deadline etc. These information are sent to the hardware-implemented scheduler as the input of the scheduling algorithms.

We have two choices to in "exec()" after sending information to the hardware scheduler:

1. "exec()" yields immediately and the OS scheduler starts to execute and selects other processes to execute. When the scheduler produces the result, it sends interrupt signals to "exec()" to finish the scheduling operations.

[1] Including task placement.

Table 1. The complexity of on-line HW task scheduling algorithms in literatures.

Author	Literatures	Free Area Management Method	Complexity
Handa et al.	[8] [7]	Maximal Empty Rectangles	$O(N^2 * W * H)$
Cui et al.	[4] [5]	Maximal Empty Rectangles	$O(N^2 * W * H)$
Tabreo et al.	[13] [14]	Virtex List	$O(N^2)$
Deng et al.	[6]	Reject Region	$O(N * (W + H))$

2. "exec()" does not yield execution, but waits for the result of the hardware-scheduler, and then continues to execute. Since Linux 2.6 is preemptable in kernel mode, "exec()" can be preempted if there is other more ungent processes.

Due to the strong computation power of hardware, the execution of the scheduling algorithm would be very fast, so in most case the scheduling decision will be obtained immediately. So we choose the second method in our system, which is much easier to implement.

4 Conclusion and Future Work

In this paper, we have reported the current progress of the project on a Unified HW/SW Operating System for Partially Runtime Reconfigurable FPGA based Computer Systems. In the next step, we will provide multiple templates for Dedicated Programs in order to support more complicated communication mechanisms for HW processes, like Semaphore and Mutex. We also plan to design the HW process interface for datasteam style applications by mapping IPCore's internal BRAM to the system address. Experiments with real applications will be conducted to evaluate the performance of our system.

References

[1] Xilinx website. In *Available: http://www.xilinx.com*.

[2] Jason Agron, Wesley Peck, Erik Anderson, David Andrews, Ed Komp, Ron Sass, Fabrice Baijot, and Jim Stevens. Run-time services for hybrid cpu/fpga systems on chip. In *RTSS*, 2006.

[3] G. Brebner. A virtual hardware operating system for the xilinx xc6200. In *The 6th International Workshop on Field-Programmable Logic and Applications (FPL)*, 1996.

[4] J. Cui, Q. Deng, X. He, and Z. Gu. An efficient algorithm for online management of 2d area of partially reconfigurable fpgas. In *DATE*, 2007.

[5] J. Cui, Q. Deng, X. He, and Z. Gu. An efficient algorithm for online soft real-time task placement on reconfigurable hardware devices. In *ISORC*, pages pp. 321 – 328, 2007.

[6] Qingxu Deng, Fanxin Kong, Nan Guan, and Yi Wang. Online placement of real-time tasks on 2d partially run-time reconfigurable fpgas. In *Technical Report, Northeastern University, China*, 2008.

[7] M. Handa and R. Vemuri. Area fragmentation in reconfigurable operating systems. In *ERSA*, pages pp. 77–83, 2004.

[8] M. Handa and R. Vemuri. An efficient algorithm for finding empty space for online fpga placement. In *DAC*, pages pp. 960–965, 2004.

[9] A. A. Jerraya and W. Wolf. Hardware/software interface co-design for embedded systems. 2005.

[10] T Rissa and J Niittylahti. A hybrid prototyping platform for dynamically reconfigurable designs. In *The international conference on Field-Programmable Logic and its Applications(FPL)*, 2000.

[11] Hayden Kwok-Hay So, Artem Tkachenko, and Robert Brodersen. A unified hardware/software runtime environment for fpga-based reconfigurable computers using borph. In *CODES*, 2006.

[12] C Steiger, H Walder, and Platzner M. Operating systems for reconfigurable embedded platforms online scheduling of real-time tasks. In *IEEE Transaction on Computers*, pages Vol. 53, NO. 11, 1393–1407, 2004.

[13] J. Tabero, J. Septien, H. Mecha, and D. Mozos. A low fragmentation heuristic for task placement in 2d rtr hw management. In *FPL*, pages pp. 241–250, 2004.

[14] J. Tabero, J. Septien, H. Mecha, and D. Mozos. Task placement heuristic based on 3d-adjacency and look-ahead in reconfigurable systems. In *ASPDAC*, pages pp. 396–401, 2006.

[15] H. Walder and M. Platzner. Reconfigurable hardware operating systems: From design concepts to realizations. In *ERSA*, 2003.

[16] T Wiangtong, Y.K. P. Cheung, and W. Luk. A unified codesign run-time environment for the ultrasonic reconfigurable compute. In *FPL*, 2003.

[17] Grant Wigley and David Kearney. Research issues in operating systems for reconfigurable computing. In *ERSAw*, 2002.

[18] T.-Y. Yen and W. Wolf. Communication synthesis for distributed embedded systems. 1995.