Microkernel on reconfigurable ARM-FPGA platform

Tian Xia, Jean-Christophe Prévotet, Fabienne Nouvel

Microkernel on reconfigurable ARM-FPGA platform

Tian XIA, Jean-Christophe Prevotet, Fabienne Nouvel
Université Européene de Bretagne
INSA, IETR, UMR 6164, F35708 Rennes, France
Email: {tian.xia, Jean-Christophe.Prevotet, Fabienne.nouvel}@insa-rennes.fr

***Abstract*—This paper proposes a custom microkernel on a ARM-FPGA platform which is capable of managing reconfigurable hardware parts dynamically. After describing the hardware platform on which the microkernel has been ported, We will focus on the proposed microkernel and the custom specific system task dealing with the reconfiguration management. Scheduling mechanism will also be discussed in this paper.**

***Keywords*—FPGA, embedded system, reconfigurable architectures, microkernel**

I. Introduction

The technique of Dynamic Partial Reconfiguration (DPR) has gained increasing attention in the embedded domain because of its runtime adaptivity for hardware algorithms and higher hardware utilization. However, the reconfiguration overhead remains a crucial issue for this technique. In modern FPGA devices, the reconfiguration of a computing-intensive module may be quite time-consuming [1]. Therefore, a dedicated efficient management is essential for DPR systems.

There have been wide researches in efficient hardware architecture management with OS support [2][3], which, however, were restricted to the limited computation power provided by embedded processors[4]. Such a limitation has been eliminated on the Zynq-7000 platform, where a fully capable processing system is provided with powerful ARM Cortex-A9 processor[5]. On Zynq-7000, the FPGA is considered as an auxiliary computing resource and thereby a specific kernel is required to efficiently dispatch both hardware and software resources. Microkernel technique seems to be a promising solution in this case, due to its features of task isolation and system security. In embedded circuits such as FPGAs, a microkernel may also be of interest since a new DPR management service may be easily developed in the existing kernel.

In this paper, we describe a custom embedded microkernel on a hybrid ARM-FPGA Xilinx Zynq-7000 platform. This microkernel is a revised version of the NOVA microhypervisor [6], and is capable of the management and scheduling of reconfigurable hardware resources. This architecture allows for dynamic management of SW/HW tasks, secure task isolation and efficient SW/HW communication.

II. Proposed Platform

The proposed platform is built on the Zynq-7000 platform framework. The intention of this framework is to develop a user-practical environment with a highly abstract microkernel, on the top of which the management of hardware resources is integrated as an user application. Both software and hardware tasks are monitored and scheduled by a custom microkernel based on the Mini-NOVA kernel. A block diagram of the proposed hybrid platform is shown in Fig. 1.

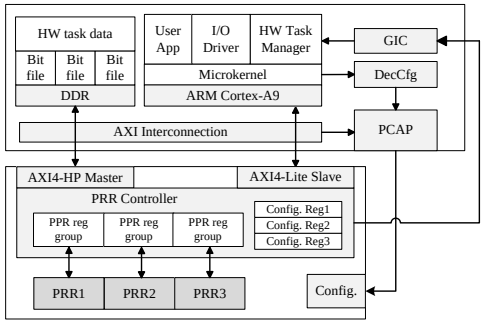

Fig. 1. Diagram of the proposed hybrid platform

The Zynq platform framework are composed of Processing System (PS) and Programmable Logic (PL). As shown in Fig. 1, on the PL side, the FPGA is consisting of multiple partial reconfigurable regions (PRR), which execute independently as hardware tasks containers. The hardware task running in each container is run-time switchable under the control of the hardware task manager. The fabric information of hardware task is held by a bitstream file. All bitstream files are loaded to the DDR memory space at the kernel bootload stage with unique IDs. The container corresponding to each HW task is pre-determined. A HW task is dispatched by downloading the target bitstream file to the pre-assigned PRR.

Several onboard resources are used to support the communication between PL and PS, as well as the downloading of bitstream file. Partial reconfigurations are conducted by launching a DMA transfer through the Processor Configuration Access Port (PCAP). Two memory-mapped AXI_GP interfaces are applied to configure the states of HW tasks. Four AXI_HP interfaces are used by HW tasks to access both on chip memory(OCM) and DDR at high throughput. Since AXI_HP is working in the slave mode, data are fetched and processed simultaneously with the processor.

To control and monitor the behavior of the HW tasks, we developed a PRR controller which consists of a state machine under the processor supervision. Through the AXI_GP interface, we have implemented a group of configuration registers which are mapped into the memory space and accessible to the processor. By accessing these registers, a SW service is able to configure the working state and parameters of HW tasks. One major feature of the PRR controller is to guarantee a smooth HW task switch at reconfiguration following these rules:

- The involved pipeline should be emptied before any

HW task switch, so that invalid output is avoided.

- To maintain the integrity of the data structure being processed, PRR reconfigurations should be launched in interval of data frames to protect data.
- A reset should be inserted to the reconfigured PRR to put it into a desired state before activation.

III. REAL TIME MICROKERNEL

To enhance the management of multiple guest SW applications and HW tasks, we developed a simplified microkernel based on Mini-NOVA, one revision of the NOVA hypervisor, by porting it from x86 to the ARM platform. In the process, additional management and scheduling mechanisms are added to the system to support dynamic PR management.

One major kernel object in our system is the execution context (EC), which is the abstraction of user threads or applications in the kernel space. Each user application is attached to one unique EC, which is able to maintain and configure user applications states such as the CPU/FPU registers and scheduling sequence. When sensitive operations (cache operation, task creation, page allocation, etc.) are required, the user space may access kernel services by generating system calls, which are also handled through an EC.

The HW task manager is defined as a special application service in user space. Any operation and switch of HW tasks should be accomplished by the HW task manager, so that the security of the FPGA fabric is ensured. A specific system call from user space will require the kernel to launch the HW task manager. The prototype of this specific system call is:

Syscall_HW_Manager(HW_id,irq_en,arg01,arg02,arg03)

The *HW_id* is used to identify the target HW task and PRR. The *irq_en* argument will indicate whether the PL interrupt will be enabled for the target HW task. Other parameters are transferred as *arg01-arg03*. At the completion of the required operation, the HW task manager gives control back to the previously interrupted application by generating another system call *Syscall_yield()* . To avoid the significant reconfiguration time overhead, the HW task manager aborts the polling-for-done mechanism. Instead, once the HW task manager launches the PCAP transfer, it gives up the CPU control and wait for the next call. The HW task is set to automatically start an operation as soon as the reconfiguration is done. Thus the reconfiguration time overhead is overlapped by CPU operations.

A priority-based round-robin mechanism is employed to schedule SW tasks. The scheduler manages the execution sequence of ECs. Each EC obtains its priority level at inception. Within the same priority level, SW tasks share the CPU through round-robin scheduling. Among different priority levels, high-priority tasks will always preempt low-priority tasks since the scheduler always selects the highest priority EC and dispatches the SW task attached to it.

Basically, all general SW tasks execute at the default priority level. To guarantee a quick response to PR requests, a higher priority is given to the HW task manager. The *run_queue* is composed of all executable ECs. In each priority level, ECs are organized as a round-robin queue. *list_prio[]* is a group of EC pointers which point respectively to the entrance EC of each priority level queue. There can be several priority levels within the *run_queue*, while the highest priority level is identified by the *prio_top* signal. When *reschedule()* is invoked, the highest-priority-level EC queue is dispatched by indexing *list_prio[prio_top]*. Once dispatched the queue will keep executing until another *reschedule()* is invoked. As shown in Fig. 2, The function *HW_Manager_Enqueue()* and *HW_Manager_Dequeue()* are used to add the EC of the HW task manager from to the *run_queue* when invoked and remove it when it finishes its task. Through this strategy, the PR of an HW accelerator is able to preempt other SW tasks, thus a quick response for the HW task management is guaranteed.

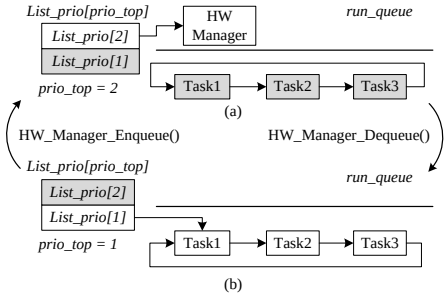

Fig. 2. Scheduling mechanism. (a) *prio_top*=2; (b) *prio_top*=1

TABLE I. SW/HW TASK EXECUTION AND SWITCH OVERHEAD

Task name	Execution time (ms)
HW task manager	0,0096
EC switch	0,00232
HW switch overhead	0.0142

The time overheads of SW/HW task switches are listed in Table I. The latency for HW task switch is composed of the execution time of the HW task manager and the time required to perform EC switches between HW task manager and user applications.

IV. CONCLUSION

In this paper, a custom microkernel on Zynq-7000 platform is proposed. This approach allows to dynamically manage reconfigurable HW accelerators and SW tasks by developing a HW task manager routine and specific scheduling mechanism.

REFERENCES

[1] S. Hauck, and A. DeHon, *"Reconfigurable computing: the theory and practice of FPGA-based computation,"* Morgan Kaufmann, 2010.

[2] F. Berthelot, F. Nouvel, and D. Houzet, "Partial and Dynamic Reconfiguration of FPGAs: a top down design methodology for an automatic implementation," in *20th International Parallel and Distributed Processing Symposium, 2006 (IPDPS 2006)* , IEEE, 2006. pp. 4-pp.

[3] J. C. Prvotet, A. Benkhelifa, B. Granado, et al. "A framework for the exploration of RTOS dedicated to the management of hardware reconfigurable resources," in *International Conference on Reconfigurable Computing and FPGAs, 2008 (ReConFig'08),* IEEE, 2008, pp. 61-66.

[4] K. Vipin, and S. A. Fahmy, "ZyCAP: Efficient Partial Reconfiguration Management on the Xilinx Zynq," in *IEEE Embedded Systems Letters,* 2014, vol. 6.

[5] *UG585: Zynq-7000 All Programmable SoC Technical Reference Manual,* Xilinx Inc., Mar. 2013.

[6] U. Steinberg and B. Kauer, "NOVA: a microhypervisor-based secure virtualization architecture," in *Proceedings of the 5th European conference on Computer systems,* ACM, 2010, pp. 209-222.