An Architecture and Programming Framework for Dynamic Reconfigurable Computing Systems

Qiang Wu Wei Xie Wei Wang

College of Computer and Communication, Hunan University, Changsha, China 410082

Abstract

Dynamic reconfigurable computing (DRC) system is becoming increasingly attractive with its potential to combine high performance and rich functionality. But problems exist in practical application of DRC, such as that designers need to know the architectural and physical details of reconfigurable device. To address this issue, a framework with hybrid architecture and transparent programming model has been proposed in this paper, which allows designers develop applications independently of the underlying physical devices. The hybrid architecture consists of microprocessors and reconfigurable hardware accelerators with corresponding control and management units. Hardware and software functions are described with function libraries that can be called in same manner by application designers. Compilation and synthesis processes are discussed to map the system description to the hybrid architecture. It is believed that this framework will be helpful to increase the development efficiency on reconfigurable computing platform.

Keywords: dynamic reconfigurable computing, hybrid reconfigurable architecture, transparent programming model

1. Introduction

Reconfigurable computing systems provide a third choice other than microprocessors and dedicated logic circuits, which has the potential to integrate the high performance of hardwired circuits and the rich functionality of programmable microprocessors, hence gain much research interests in these years[1]. However, problems still exist in the practical application of reconfigurable systems. Among them, programming model is a key issue in our point of view. In traditional designs, reconfigurable resource generally plays the role of hardware accelerators that is controlled and manipulated with specially designed circuits in system. This requires the knowledge of the architectural and physical details of the reconfigurable devices, which is a non-neglectable obstacle for application developers who have a software development background.

In previous works, many efforts have made to attack this problem. Representatively, Herbert Walder used operating system to abstract programmable device and proposed an operating system framework to support reconfigurable hardware in [2]. David Andrews proposed the opinion to adopt hardware function modules as hardware tasks to the systematic management level, and describe the embedded system with a unified multitasking model[3]. But there are several shortages in their models, such as that they only support hardware blocks created statically.

Miljan Vuletic studied the seamless hardware-software integration in reconfigurable computing systems by adding a system-level virtualization layer on reconfigurable device to support a transparent programming model which hides platform details from designers[4][5]. Whereas it's regrettable that dynamic reconfiguration is not included in the proposed model, while this ability is becoming poplar in currently released programmable devices.

We attempt to follow the efforts of previous approaches to provide a transparent programming model for designers, and take advantage of the dynamic reconfiguration ability of underlying programmable devices. Our idea is borrowed from the parallel computing for multiprocessors. We treat the hybrid architecture of dynamic reconfigurable system consisting of microprocessors and hardware accelerators implemented on reconfigurable resources as a cluster of heterogeneous processing units with a virtually shared memory. Designers write application description in a manner very similar to that on single processor architecture, while compiler and synthesis tools map the description to underlying reconfigurable architecture, hiding the physical details from the designers. To achieve this, reconfigurable resource management unit, hardware function library, as well as other necessary hardware and software components are considered and integrated in the architecture and programming framework in this paper.

The rest of the paper is organized as follows: Section 2 gives the details of the framework. Section 3 briefs the validating case study. Section 4 discusses the future work and draws the conclusion.

2. Framework

2.1. Outline

As mentioned in the above section, we intend to provide a transparent programming model similar to the single processor platform for the designers. For the reason, it should be noticed that the software development seems to be more expensive than the hardware design in nowadays. Chips are getting cheaper with the technology improvement while software development costs are rising rapidly. Furthermore, software developers have a large library of previously accumulated codes which is not only a helpful resource but also a heavy burden needing to maintain. This brings the designers much difficulty to shift to a novel development environment. Dataflow programming languages are such an example, which honestly to say are much suitable for development on reconfigurable computing platform consisting of both hardware and software components. However, due to the lack of the easy porting method for traditional software codes to newly proposed dataflow languages, the dataflow computing systems get very limited applications in real world.

Another important element of our framework is the architecture model of the dynamic reconfigurable computing systems. We focus on the hybrid architecture of dynamic reconfigurable system consisting of microprocessors and hardware accelerators implemented on reconfigurable resources. Actually this is popular in commercial reconfigurable computing development platforms[6][7]. Typically, these platforms have one or more soft-core microprocessors which can be integrated and implemented with other hardware logic blocks into the programmable device. The hardware logic blocks connect with microprocessors through a shared bus or some specially designed interconnecting network as the accelerator of specific functions. The microprocessors and hardware accelerators may use some of the on-device memory as local caches. Memory interface circuits are always integrated into the programmable device to provide access to the off-device large memory modules such as SDRAM units. Such architecture resembles the shared memory MIMD machine in supercomputer industry. So we take a view of the underlying hardware as a cluster of heterogeneous processing units with a virtually shared memory. In addition, since the programmable devices are capable of dynamic reconfiguration, this cluster of heterogeneous processing units can be changed dynamically, which obviously requires corresponding control and management components to handle dynamic processing unit adding and deleting, as well as interconnection reconstruction.

Obviously, there is a gap between the programming model and the architecture model mentioned in above. The programming model we want to provide intends to approach the single processor model, while the architecture model follows the multiprocessing manner. To fill the gap, compiler and synthesis tools are needed to map the application description to underlying hardware. The ideal compiler and synthesis tools are expected to do the mapping fully automatically. However, considering the experience in automatic parallelization compiler research, this task is hardly possible. For an easier translation process, we leave some work to designers to help the compiler and synthesis tools complete their jobs. Designers are required to specify the hardware-software partition of the system functions explicitly in the application description with the support of a hardware function library. Function calls to the hardware function library will be mapped to the reconfigurable fabric, while others are mapped to microprocessors. Such a partitioned description takes a look as same as the normal function calls, which is believed to be convenient for designers to specify the system function.

Putting the above altogether, the framework we proposed has the hierarchy as follows:

- *Application layer*: Provides an API for designers to describe the application. This API contains the functions from software and hardware function libraries.
- *Operating system layer*: Provides mechanism and system calls for process communication, synchronization and scheduling with the aid of the nether firmware.
- *Firmware layer*: Provides basic routines for control and access to the reconfigurable resource management unit, on-line synthesis and placement unit etc.
- *Architecture layer*: Contains the microprocessors and reconfigurable fabric as processing units, as well as reconfigurable resource controller, on-line synthesis and placement unit as the supporting units.
- *Hardware layer*: Specifies the real physical components in the system, such as the microprocessor instances, programmable devices, memory modules, interconnecting networks and so on.

The upper three layers belong to the programming model, and the bottom two layers belong to the architecture model. The details of them are described in next two subsections.

2.2. Architecture Model

The architecture model of the proposed framework is shown in Fig. 1.

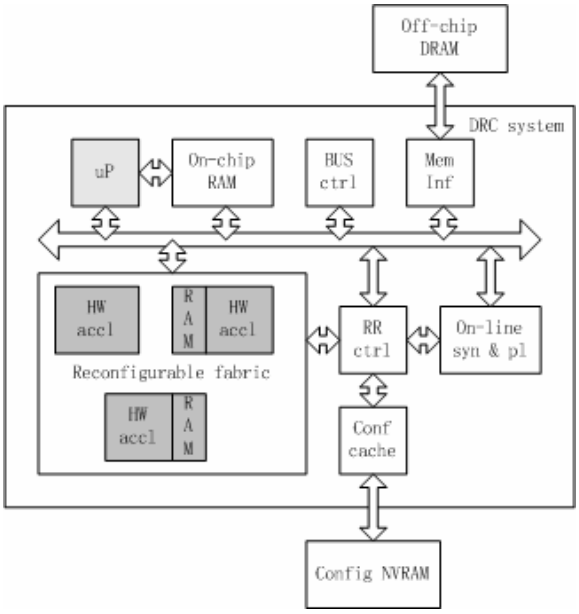

Fig. 1: Architecture model

In the following, we want to make a concise introduction on the key components of the model.

- *Processing units*: These include microprocessors and hardware accelerators implemented on the reconfigurable fabric, as the "uP" and "HW accl" depicted in the figure.
- *Memory system*: Microprocessors and hardware accelerators may have on-chip memory resources as local caches. They can also access off-chip memories through memory interface unit. Both the on-chip and off-chip memories share a uniform linear address space. Memory interface unit has circuits to manage accesses between on-chip and off-chip memories.
- *Interconnecting bus*: Processing units are connected with a shared bus. Each component connected on the bus has a unique address to identify itself. Bus controller manages the traffics on the bus. Snoopy logic is included to support concurrent executing of software and hardware functions.
- *Reconfigurable resource manager*: This component consists of reconfigurable resource controller (RRC) and on-line synthesis and placement unit (SPU). They are in charge of reading bit streams, performing dynamic synthesis and placement, and pre/configuring the programmable device.

2.3. Programming Model

We hope to present a familiar interface for application developers. A set of function libraries is defined to build an API for designers. Some of the libraries are of software. Some of them are hardware function libraries. They have same syntax forms but different implementation. Software functions are compiled to instructions as in normal way. Hardware functions have only encapsulating codes in the library binaries. These instructions redirect the execution to system calls of the operating system and/or basic routines of firmware layers that commands the reconfigurable resource manager to read, configure and run the bit streams of the corresponding hardware accelerator on the programmable device.

To reduce reconfiguration delay, pre-caching and pre-configuration should be used. This can be realized in this framework through static scheduling. In compilation procedure, calls to hardware functions are recorded. Then an assistant program reads these records and makes decisions on when to fetch the bit streams and configure the programmable device. The scheduling result will be packed with initializing codes and bit streams into the NVRAM to notify the reconfigurable resource controller about the time of pre-caching and pre-configuration.

Another potential performance improvement in dynamic reconfigurable computing systems is the parallelization of software and hardware tasks. This is supported this framework through result snooping. In traditional programming semantics, caller and callee functions execute mutually exclusively. In our programming model, calls to hardware functions return immediately after feeding the parameters to hardware accelerators. The microprocessor can continue to run succeeding instructions until it needs the result of the hardware accelerator. As mentioned before, bus controller has snoopy logic to monitor the accesses to these critical objects, and make sure the right accessing order is maintained.

Fig. 2 summarizes the above brief description of the programming model.

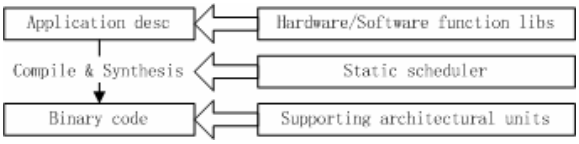

Fig. 2: Programming model in brief

3. Case Study

To validate the proposed framework, we plan to use the AVS[8] (Recently approved standard of video

compression in China) decoder as the experimental design example. The development board for test is the Xilinx XUPV2P development board, which has MicroBlaze soft-core processor built in the Virtex-II Pro programmable device. The software part of the resultant design will be run on this processor, while the hardware part will be fitted into the rest of the reconfigurable resources.

AVS forum has released a reference design of the decoder fully in C++ source code. The first step of the design can begin with the construction of the hardware function library. We search for or design by our own the hardware implementation of the software functions that may be partitioned to programmable devices as the accelerators. It should be noted that such a library construction may not be needed every time in designs, since these hardware implementations may be reused in future designs.

After the construction of hardware function library, the application description can be specified with the functions of both hardware and software libraries. Then the compiler will translate the codes to instructions. Calls to hardware functions will generate instructions communicates with the reconfigurable resource controller to configure, execute the corresponding hardware accelerators. Calls to software functions will generate instructions as in normal cases. Hardware scheduler can engage in this process to scan the generated instructions and build the pre-configuration list on the programmable device for better performance. In the next, these complied codes and pre-configuration information, as well as the configuration bit streams for hardware accelerators synthesized beforehand in hardware library construction will be stored in the non-volatile memory of the system. During the execution of the system, the operating system and the on-line reconfigurable resource management unit will cooperate to control and synchronize the running of hardware and software functions for a high performance and transparent implementation of system functionality.

Currently we are constructing the hardware function library and building the corresponding configuration bit streams. The whole decoder is expected to finish soon after this laborious task.

4. Conclusion

In this paper, we presented a framework for dynamic reconfigurable computing systems, which includes architecture and programming models. The architecture model takes a view on the dynamic reconfigurable computing system as a cluster of microprocessors and hardware accelerators as heterogeneous processing units with a shared memory. Reconfigurable resource controller, on-line synthesis and placement unit and snoopy logics are included to support efficient implementations of applications. The programming model intends to present an interface similar to normal software development environment for designers. Hardware accelerators are encapsulated in hardware function libraries and called as normal software functions. Static scheduling and result snooping are employed to improve the performance.

Current work is focused on validating the proposed framework with a case study on AVS decoder. Automatic hardware-software partitioning for dynamic reconfigurable computing systems is also noticed as one of our research topics in the future.

5. Acknowledgement

The authors would like to thank Xilinx University Program for the kind donations of the development boards and software packages.

6. References

[1] Tim J. Todman, George A. Constantinides, etc. "Reconfigurable Computing: Architectures and Design Methods". *IEE Proceedings on Computers and Digital Techniques, 152(2),* pp.193-207, 2005.

[2] H. Walder, and M. Platzner, Reconfigurable "Hardware Operating Systems: From Concepts to Realizations", *Proc. Int'l Conf. Eng. of Reconfigurable Systems and Algorithms*, pp. 284-287, 2003.

[3] David Andrews, Douglas Niehaus. "Programming Models for Hybrid FPGA-CPU Computational Components: A Missing Link". *IEEE Micro*, 24(4), pp. 42-53, 2004.

[4] M. Vuleti´c, L. Pozzi, "Seamless Hardware Software Integration in Reconfigurable Computing Systems", *IEEE Design & Test of Computers*, 22(2), pp. 102-113, 2005.

[5] M. Vuleti´c, L. Pozzi, "Virtual Memory Window for Application-Specific Reconfigurable Coprocessors", *Proc. of DAC*, pp. 948-953, 2004.

[6] Xilinx Inc. "Development boards", http://www.xilinx.com. 2006

[7] Altera Inc. "Development kits", http://www.altera.com. 2006

[8] AVS Organization. http://www.avs.org.cn. 2006