A Co-Design Flow for Reconfigurable Embedded Computing System with RTOS Support

Xiao-Wei Wang, Wei-Nan Chen, Ying Wang, Cheng-Lian Peng
School of Computer Science and Technology
Fudan University, Shanghai, China
Email:{051021035,clpeng}@fudan.edu.cn

Xiao-Wei Wang
Shanghai Branch Navy Equipment Technology Institute, Shanghai, China

***Abstract*--Reconfigurable system provides both flexibility of software and performance of hardware. It is a significant trend in embedded application domain. Some new reconfigurable technologies and technology-dependent tools have been developed, but the whole design flow for run-time reconfigurable systems with real-time operating system support is not proposed. RTOS plays an important role in the system and the co-design flow. The special requirements for reconfigurable embedded systems with RTOS support are analyzed, and a novel co-design flow is proposed in this paper. A design case is presented here, which shows the co-design flows of the implementation of an adaptive signal filtering system on a commercially available reconfigurable platform. The results show that using run-time reconfiguration can save over 66% area when compared to a functionally equivalent fixed system and achieve 24 times speedup in processing time when compared with a functionally equivalent pure software design.**

.Keywords: Reconfigurable embedded computing system, Co-Design Flow, RTOS Support, unified hardware task interface

I. INTRODUCTION

Modern embedded systems such as consumer appliances and military devices present ever-increasing computational power, flexibility, easy-to-upgrade and easy-to-maintenance requirements amidst tight budgets and Space, Weight, and Power (SWaP) challenges[1]. Modern SRAM-based FPGAs can be dynamically and partially reconfigured at runtime, without interrupting the operation of other logic within the FPGA, which presents intriguing possibilities for novel system architectures and applications. So, over the past decade, the reconfigurable computing platform has been an emerging approach in scientific research and in practical application to meet these requirements [2]. The design and implementation of dynamic reconfigurable systems (DRSs) is exceptionally challenging.

The hardware/software unified RTOS process ensures portability of the co-designed reconfigurable embedded computing applications and minimizes the performance loss by providing standardized means of interfacing and unified effective programming model [3]. But， the co-design flow for the reconfigurable embedded systems with the RTOS support was not proposed. We analyzed the special requirements for such systems and proposed co-design flow in this paper.

This paper is organized as follows: related work will be presented in section2. Section 3 will discuss the special requirements of design flow for reconfigurable embedded computing with RTOS support. Our co-design flow will be presented in section 4.Section 5 will describe the adaptive filtering system using commercial off-the-shelf reconfigurable devices and the results will be presented. Finally, the conclusions will be given in Section 6.

II. RELATED WORK

The traditional design flows for mixed hardware-software embedded systems are to select architecture, decide where in the architecture different pieces of the application will be implemented, design the hardware and software pieces, and finally integrate the pieces together [4,5].If the design criteria are not met, the functionality, architecture, and mapping can be modified6, 7]. System-level design covers various issues, such as partitioning, task

978-0-7695-3678-1/09 $25.00 © 2009 Crown Copyright
DOI 10.1109/ICESS.2009.84

scheduling, and synthesis. An SW/HW partitioning and online task scheduling approach is presented [8][9].

Nikolaos S.Voros and Kondtantinos Masselos described the system level design of reconfigurable system-on-chip [5], but they did not include the co-verification of the dynamic part of the reconfigurable system and they neglected the difference of the systems with real time operating system support. Most of the co-design methods were missing OS layer abstraction for reconfigurable application [10].

David Andrews proposed unified multithreading programming model for operating system support and implemented some critical operating components in hardware such as mutex and signal semaphore. But they did not discuss the design flow of the reconfigurable system [12]. Zhoubo proposed unified multiple task programming model and implemented a operating system prototype using ucosII for reconfigurable platform,but did not proposed implementation flow for partial reconfigurable system[15].Xilinx proposed early access partial reconfiguration implementation flow, but it did not include system co-design flow[18].

We proposed the co-design flow for embedded reconfigurable computing system with RTOS support based on the analyses of the special requirements including system level design flow, detailed design and implementation flow.

III. Special Requirements of Co-Design Flow for Reconfigurable Computing Systems

A. special requirements for system-level design

A generic system-level design flow is showed in figure1 [5].In system specification phase; designers need extra effort spent on identifying parts of the application that server as candidates of implementation with reconfigurable hardware. In architecture definition phase, designers need to model the reconfigurable technologies of different

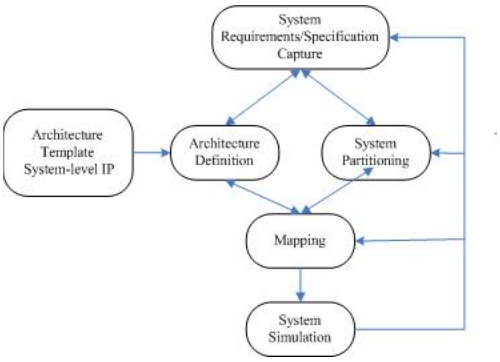

Figure1. A generic system-level design flow

types and vendors at abstract level, and designers need to include them in the architecture models. System partitioning needs to analyze and estimate the functions of the application for software, fixed hardware, and reconfigurable hardware. The parts of the targeted system that will be realized on reconfigurable hardware must be identified.

In hardware/software partitioning and mapping phase, for reconfigurable computing, a new dimension is added to the problem. Which part of the system that will be implemented on reconfigurable hardware must be identified according to the specification If the application has several roughly same hardware accelerators that are not used in the same time, a dynamic reconfigurable block may be a more optimized solution .If the application has some parts in which specification changes are foreseeable, the implementation choice may be reconfigurable hardware. If there are foreseeable plans for new generations of application, the parts that will change should be implemented with reconfigurable hardware.

B. special requirements for detailed design

In this phase, the individual portions of hardware, software and reconfigurable hardware are designed and verified. Reconfiguration requirements include communication mechanism for software and static hardware and reconfigurable mechanism to handle context multiplexing.

The integration and co-verification combines the reconfigurable hardware components with other hardware and software components into a single platform, which also accommodates external IP and provides co verification of overall design. The reconfigurable hardware is simulated in a HDL simulator or emulated in an FPGA emulator. Specific HDL modeling rules need to be followed for multiple dynamically reconfigurable contexts. The reconfigurable hardware modules must be implemented using the specified technology, including the required control and support function for reconfiguration.

C. *The special requirements for implementation design*

Dynamic reconfiguration requires configuration bitstreams of multiple contexts to be managed. Specific design rules and constraints must be followed for multiple dynamically reconfigurable modules. Reconfiguration overhead is determined by configuration bandwidth, frequency, and configuration granularity in the design. The generation for partial bitstream files depends on the specific tools. If Xilinx's FPGAs are used, we need Xilinx's partial reconfigurable tools and have to obey the partial design flow to generate partial reconfigurable bitstream files.

D. *The special requirements for OS support*

The use of a runtime resource allocation unit will get the full benefits of dynamic reconfiguration on high density FPGAs. In addition to runtime resource allocation, other services provided by an OS such as abstraction of I/O and inter-applications communication will provide additional benefits to the users of a reconfigurable computer. This will reduce the difficulty of application development and deployment. Reconfiguration result in the complexity of the system control requires RTOS for synchronization, communication, and concurrency management. OS for RC provides the following capabilities: transparent hardware and software communication services and configuration management. Another important RTOS advantage is the abstraction to increase portability and to facilitate reuse of code and repartitioning.

RTOS for reconfigurable computing system needs effective programming model, appropriate abstraction of hardware tasks, unified hardware task interface, efficient communication methods between tasks and optimized allocation and partition algorithms for reconfigurable hardware resource .The co-design flow must consider hardware resource that will be used for operating system components and software interface for reconfigurable hardware modules.

IV. CO-DESIGN FLOWS FOR RECONFIGURABLE COMPUTING WITH RTOS SUPPORT

The co-design flow for reconfigurable embedded computing system is different from conventional hardware/software co-design flow and it is also different from the conventional co-design flow for reconfigurable system-on-chip. The programming model and unified reconfigurable hardware module interface have to be considered before discuss the operating system for reconfigurable system.

A. *Hardware and software unified multiple task programming models*

In conventional co-designed system, FPGAs are used as hardware accelerator. User application program directly manage and use them. Without reconfigurable hardware resource abstract and resource management, reconfigurablity and task potential parallelism can not be used sufficiently. The conventional RTOS have to be improved for the reconfigurable embedded computing system. A unified multi-task model (as POSIX thread specification) was widely accepted. Under the model, the hardware function modules are viewed as hardware tasks, and the RTOS manages hardware tasks as well as the software ones. The RTOS for RC provides unified application program interface (API) and necessary hardware components, which makes it easy to communicate between function modules, including communication and synchronization between tasks. Because it is naturally transferred from conventional method, it is easy to be accepted by designers.

B. *Unified hardware task interface library*

To enable the RTOS to effectively manage hardware tasks, special functions have to be added to the implementation of the hardware. Hardware units respond to the basic communication primitives. We proposed a set of virtual hardware task interfaces, which compose unified hardware task interface (UHIF) library to satisfy the requirements of typical data stream-based applications. The interfaces can adjust parameters according to system's structure and application requirements. UHIF combine with user hardware task as macro module. A set API functions are proposed to make it easy to use hardware tasks as usual software task.

C. *RTOS for reconfigurable computing system*

RTOS introduces an abstract layer between the application designer and the systems software. The software part of RTOS for reconfigurable system includes the software task

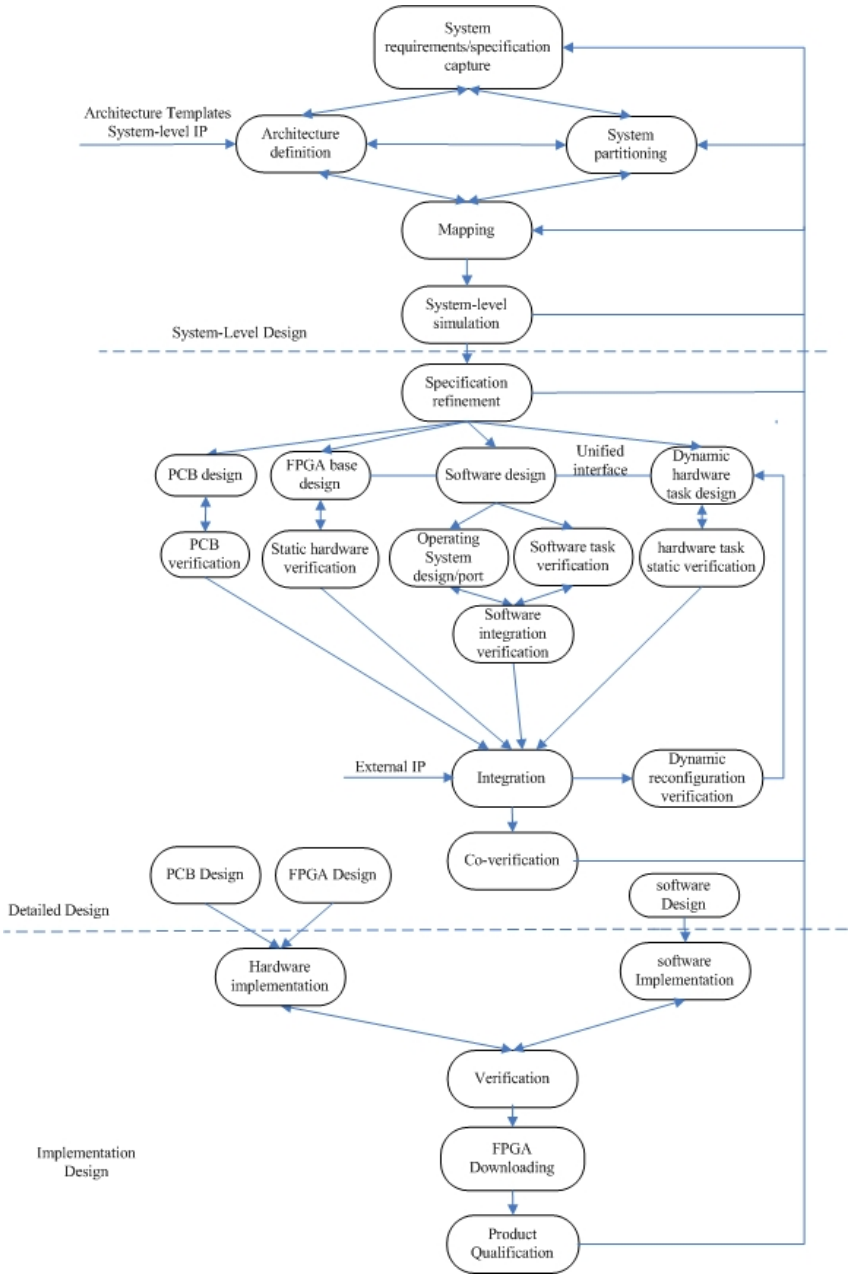

Figure 2. co-design flow for reconfigurable system with RTOS support

interface, task scheduler and resource manager. The hardware part of RTOS is called the hardware task manager, usually implemented on the FPGAs, including the communication controller, standard hardware-task interface, configuration interface and hardware-task configuration controller.

Linux is widely used in real time embedded application. We improved it for reconfigurable system based on FPGAs by extending the standard Linux kernel to support dynamic reconfigurable embedded system. The services of RTOS are provided to deal with input and output, load and execute pre-designed circuits. User application processes can therefore be either software programs running on processors, or hardware implementations running on FPGAs. We term hardware implementations on FPGA as hardware tasks. RTOS maintains a consistent unified interface for both software and hardware processes. Therefore, to the rest of the system, communicating with a hardware process is no completely different from communicating with a normal software process. This homogeneous handling of hardware and software in the kernel forms the foundation of coarse grain hardware/software co-design boundary.

D. Co-Design Flow for reconfigurable computing system with RTOS support

1) System level design

The requirements including RTOS are captured and analyzed in the specification phase. The results are fed to the next phase of design flow. In the partitioning phase, the function of application is partitioned in software, hardware and reconfigurable modules. The operating system support is one of the most important parts of software under unified multi-task programming mode. In higher abstraction level, a reconfigurable module is viewed as a task of the RTOS to schedule.

System-C and MATLAB can be used to create the system model. By running the model, enough information of the communication between tasks can be captured, which make it easy to schedule .The reconfiguration way is determined by the reconfiguration specification.

2) Detailed design

The specifications are refined and verifications are planned according to target implementation technologies and processors type etc .The hardware designs include system circuit board and FPGA hardware design. For the unified management of the hardware tasks and software tasks, HW/SW unified interfaces are needed. The interfaces stride on the boundary between the reconfigurable hardware tasks and the software tasks. Software includes operating system components for reconfigurable hardware design and application. We improved linux2.6.24 version kernel, a part of the standard kernel is improved and drivers are added. Because the reconfigurable embedded system with operating system is more complex than usual embedded system, every part should be verified respectively, such as FPGA design and verification, software design and verification. The base design of FPGA is the static part of the reconfigurable system on which the operating system and application program run. The application software must verify on operating system.

The implementation flows of reconfigurable system are tedious, and it is hard to debug reconfigurable hardware. So, the correctness of reconfigurable hardware must be ensured. Each version of the hardware task must be verified in static condition. That is to say, in order to ensure the correctness of the function and the timing order, every reconfigurable module must be integrated into the system as a static part to be verified and tested before it is used as reconfigurable hardware task.

The co-verifications integrate the reconfigurable hardware tasks, static hardware, and software components into whole system. The co-verification environment is provided for the overall design. The reconfiguration parameters are inserted into the co-verification phase. Some commercial tools such as

MATLAB,CoDeveloper and synthesis tools are used to co-verify the design.

3) Implementation design

In implementation design phase, the embedded system components containing circuit board, FPGA configuration bitstream files, OS images and application software executable files are prepared. If commercial devices are used, the bitsreams files of hardware tasks will be generated by the device provider's tools. If Xilinx virtex family FPGAs are used, designers may use the modular based design flow to generate all the bitstreams of the FPGA and use Xilinx embedded development kit(EDK) to compile software, generate board support package(BSP) for RTOS and combine the hardware and software into one image file.

The unified interfaces are inserted into the system as IP cores in embedded development kit. The hardware's bitsreams and software's images are downloaded to FPGA to verify, and the dynamic reconfigurable tasks are verified by downloading into FPGA through configuration port such as Xilinx's SelectMap,ICAP,JTAG etc.

V. . A CASE: CO-DESIGN FLOWS FOR ADAPTIVE FILTERING SYSTEM-ON-CHIP

A. application and target platform

We use a real case to demonstrate our design flow for reconfigurable embedded system with RTOS support. The design case is an adaptive filtering design that targets to meet all the objectives (low pass, band pass, and high pass) on the Xilinx virtex-5 (XC5VLX50T-FF1136) FPGA on ML505 board. The application architecture shows in figure 3. The FIR filters are special kinds of digital filters and have a wide applicability because they have a good characteristic such as linear phase and stability. They are employed in the majority digital signal processing (DSP) based electronic systems.

B. system design and implementation

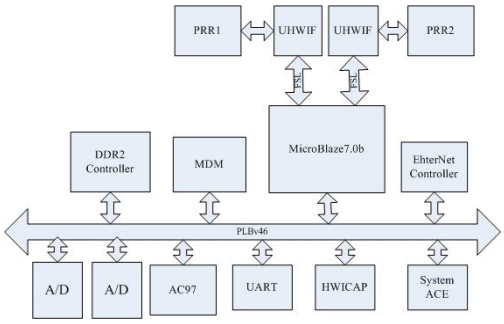

Figure 3 Adaptive filtering system

The specification of the application is clear. It is adaptive filtering system for sound signal. The sample frequency, high passes frequency, and low pass frequency are pointed out clearly. The flexibility requirement for the application is to adaptive different signals. In system level design, we capture the reconfiguration requirements that were to satisfy the flexibility and performance. To make it easy to port the application and share the FIR and other hardware resource, RTOS is required.

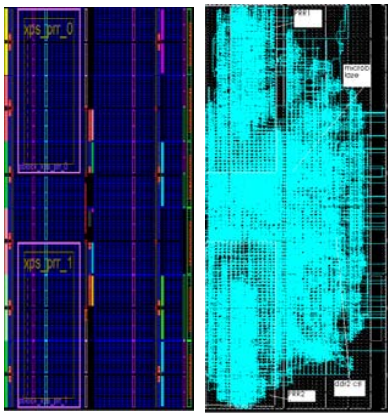

Figure4 Two PRRs in the system FPGA Editor View and PlanAhead Device View

Adaptive filtering system implemented by a reconfigurable hardware. The FIR filters may need a large number of resources to satisfy the desired specification. The partial reconfiguration technique can be used in this case since the various types FIR filters have so many similarities in their structure. Therefore, partial reconfiguration addresses the reduced reconfiguration overhead, coefficient flexibility

and area efficiency for higher order FIR filters. Reconfigurable FIR filter offers both the flexibility of computer software, and the ability to construct custom high performance computing circuits.

We selected Mircoblaze7.0b with MMU support because the system needs RTOS. We improved uclinux to make it support RC. All versions of FIR (low pass, high pass and band pass FIR) were generated in MATLAB with System Generator, which can create cycle-accurate and bit-accurate model for DSP application and co-simulate the hardware and software model. After successfully simulated, we integrated the FIR module with the base system of the FPGA and connected the FIR with Microblaze by unified hardware task interface After base design completed, the improved uclinux was ported into the system to verify the RTOS and application software.

We used Xilinx's EAPR flow to generate the partial bitstreams of the different version of FIR with different parameters. We verify the partial bitstreams by download to FPGA through JTAG port. Application software was downloaded to FPGA to test and verify the configuration function. Then,RTOS image combine the bitstream file of the static full to generate ace file to verify the RTOS in reconfigurable system. The next step, we co-verified the partial reconfigurable modules and the software with RTOS. Two partial reconfigurable regions (PRR) placed in the system as figure 4 shows. There are 3 reconfigurable modules (PRMs) for each PRR. The most resources required for FIR are listed in table1, and the executing time cost for FIR is listed in tableII The configuration overhead are listed in table III.

We used one third of resources to implement the three versions of FIR by dynamic partial reconfiguration. The processor Microblaze ran at 125MHz.We tested the time cost including data transfer time for FIR computing using timer at 125MHz, reconfigurable hardware FIR ran 24 times faster than pure software implemented FIR.

TABLE I. RESOURCE REQUIRES FOR FIR

Resource Type	Reconfigurable Requires	Static Requires
LUT	849	2547
FF	1346	4038
SLICEL	291	873
SLICEM	120	360
DSP48E	16	48
RAMBFIFO	0	0

TABLE II. EXECUTION TIME

Software FIR	79849600 cycles	631.5ms
Hardware FIR	3240105 Cycles	25.9ms
Speed up	24.38	

TABLE III. CONFIGURATION TIME

Reconfigurable Hardware FIR	6374683cycles	50.997ms

C. Comparisons with other approaches

In addition to partial reconfigurable implementation, a fixed hardware implementation and pure software implementation were made as a reference design. In the fixed hardware implementation, the processing blocks were mapped onto static accelerator and 3 versions of FIR requirs triple resources as table1 shows. That is to say, we saved about 66% of the resources for adaptive FIR by reconfiguration.

For the full software implementation, the design was implemented as an application on linux, and the data are transferred through FSL bus using Xilinx Microblaze customized instructions. All ran on Microblaze at 125 MHz, data store in BRAM and the RTOS and application software ran in DDR2.The processing time of one data was 631.5ms, which was over 24 times of the processing time in partial reconfigurable case.

As we know, reconfigurable hardware tasks need to be reconfigured when the scene changed, while the software implementation only need few cycles to jump to another process. We use Xilinx internal configuration access port (ICAP) to implement dynamically reconfiguration.

VI. . CONCLUSION AND FUTURE WORK

The main advantages of reconfigurable embedded system with RTOS support are the combined flexibility, performance, portability and resource sharing. The implementation this kind of system requires extra efforts in every design stage. In this article, we present a design flow for reconfigurable system with RTOS support, which combines design support at system level for partitioning and mapping, programming model, unified hardware task interface library. A set of FIR filter are used in case study. Compared to completely fixed hardware implementation, reconfiguration reduced over 66% of resource; compared with full software implementation, reconfiguration is over 24 times faster. The commercial off-the-shelf FPGA platform caused limitation on the implementation of dynamic reconfiguration. If we added the configuration overhead, the speed ratio down to 12.We will improve our system to reduce the configuration overhead in the future.

. REFERENCE

[1] John Wemekamp, Emerging Trends in Mil/Aerospace Embedded Systems,Controls Embedded Computing Ecnmag.com May 01, 2007

[2] Tarek ElGhazawi, Esam ElAraby, and MiaoqingHuang, The promise of high-performance reconfigurable computing, Computer, February, 2008

[3] David Andrews, Ron Sass, Erik Anderson, Achieving Programming Model Abstractions for Reconfigurable Computing, IEEE transactions on very large scale integration(VLSI)system, VOL. 16, No. 1, January, 2008

[4] IAN ROBERTSON and JAMES IRVINE,A design flow for reconfigurable hardware, ACM Transactions on Embedded Computing Systems, Vol. 3, No. 2, May 2004, Pages 257–283.

[5] NikolaosVoros,Knostantinos,Masselos, system level Design of reconfigurable Systems-on-Chip,2005,Springer

[6] Yang Qu,Kari Tiensyrj Juha-Pekka Soininen,and Jar Nur , Design Flow Instantiation for Run-Time Reconfigurable Systems: A Case Study, Volume 2008

[7] Florent Berthelot,Fabienne Nouvel,and Dominique Houzet , A Flexible System Level Design Methodology Targeting Run-Time Reconfigurable FPGAs, Volume 2008.

[8] B. Knerr, M. Holzer, and M. Rupp, RRES: A Novel Approach to the Partitioning Problem for a Typical Subset of System Graphs,Volume 2008

[9] Chun-Hsian Huang,Pao-Ann Hsiung, Software-Controlled Dynamically Swappable Hardware Design in Partially Reconfigurable Systems, Volume 2008,

[10] S´ebastien Pillement, Olivier Sentieys , DART: A Functional-Level Reconfigurable Architecture for High Energy Efficiency, Volume 2008,

[11] Aric D. Blumer and Cameron D. Patterson , Exploiting Process Locality of Reference in RTL Simulation Acceleration, Volume 2008

[12] WeiWang,Qiang,WuWei,Hardware Software Co-design for Dynamic Reconfigurable Computing with Collaborative Supports of Architecture and Operating System, Proceedings of the 2007 11th International Conference on Computer Supported Cooperative Work in Design

[13] Kamana Sigdel, Mark Thompson, Andy D. Pimente,Todor Stefanov,System-Level Design Space Exploration of Reconfigurable Architecture, Springer-Verlag Berlin Heidelberg 2008

[14] Carlos Paiz, Boris Kettelhoit, and Mario Porrmann, A design framework for FPGA-based dynamically reconfigurable digital controllers, 1-4244-0921-7/07 2007 IEEE.

[15] Zhou Bo, Wang Shiji, Qiu Weidong, SHUM-UCOS: A real-Time operating system for reconfigurable systems using uniform multi-task model [J]. Journal of Computers, 2006, 29(2): 208-218 (in Chinese)

[16] Zhou Xuegong,LiangLiang,Huang Xunzhang,On-line scheduling and placement of real-time tasks for reconfigurable computing system [J]. Journal of Computers, 2007, 30(11): 1901-1909 (in Chinese)

[17] J.A.Williams and N. W. Bergmann, "Embedded Linux as a platform for dynamically self-reconfiguring systems-on-chip," in Proc. Int. Conf. on Engineering of Reconfigurable Systems and Algorithms (ERSA '04), Las Vegas, Nevada, 2004.

[18] Xilinx,"VirtexII FPGA platform datasheet",May,2007,http://www.xilinx.com