PROCEEDINGS OF SCIENCE

A Linux-based Dynamic Partial Reconfiguration System Applied on Xilinx Zynq

Dongming Luo[1]
Beihang University
Beijing ,100191, China
E-mail: ldm520@buaa.edu.cn

Guoqing Pan
Beijing Aerospace Measurement & Control Technology CO.LTD
Beijing, 100037,China
E-mail: pan_guo_qing@163.com

Guohua Wang
Beihang University
Beijing ,100191, China
E-mail: wgh@buaa.edu.cn

Dynamic Partial Reconfiguration(DPR) optimizes the resource utilization in the Field Programmable Gate Array(FPGA) over time and space nowadays. But the lack of standard tools and interface makes the extensive application of DPR rather difficult . With the open source Linux widely used in the Xilinx Zynq, users' demand for the reconfiguration becomes urgent. Linux-based dynamic partial reconfiguration system(LDPRS), combining the unified multi-thread programming model of software tasks and hardware tasks with FPGA dynamic partial reconfiguration, provides reliable and perfect reconfiguration system services. LDPRS supplies Linux standard reconfiguration programming interface and service of dynamic loading bitstreams. In the experiment, reconfiguration system is applied in the reconfigurable computing and the results show that FPGA reconfiguration design based on LDPRS has a short development period and high flexibility, supporting the dynamic loading bitstreams. This system efficiently solves the problem of the lack of standard reconfiguration development tools under LINUX, increasing the development flexibility of reconfiguration design and decreasing the difficulty and time in development.

CENet2017
22-23 July 2017
Shanghai, China

[1]Speaker

PoS(CENet2017)047

1.Introduction

FPGA are increasingly adopted in embedded systems due to their flexibility and relative low cost compared to the fixed application of specific integrated circuits (ASICs) [1]. In modern FPGAs, the flexibility of the device is further enhanced by the dynamic partial reconfiguration feature(DPR). DPR allows the functionality of certain blocks to change in the FPGA at run-time without interrupting the operation of the system[2].

Xilinx Zynq retains the partial reconfiguration feature of Virtex Series FPGA and supports the reconfiguration through the PS to control the configuration interface. However, the lack of the unified interface and system support leads to the partial reconfiguration unable to be efficiently applied in Zynq. In the paper, the author designs the DPR based on the open source Linux , combining the partial reconfiguration feature of Zynq. The LDPRS provides the reconfiguration system based on the Linux and API which control the reconfiguration process. By the standardized API proposed by LDPRS, users can load dynamicalbitstreams to the designated logical resource regions without interrupting the operation of other regions. Compared with Xilinx official reconfiguration, LDPRS provides the unified reconfigurable API, which can support the relocation of bitstreams and combine the reconfiguration features with the operating system. Thus the development cycle of reconfiguration can be shortened and the difficulty of using reconfiguration can be reduced as users need't to concern the complex substructure when they use the reconfiguration features.

The main contribution of this paper is the presentation of the novel reconfiguration system LDPRS, including the programming model for hardware tasks and a runtime system for Xilinx Zynq FPGAs.

The remainder of this paper is organized as follows: Section presents an overview of related work; Section 3 is dedicated to describing the architecture of our system; The practical application of reconfiguration system and the results analysis are presented in Section 4;The final section summarises the conclusion and the future research prospects.

2.Related Work

Reconfiguration operation system(ROS), firstly proposed by Brebner, refers to the system which provides some functions to manage and control the reconfiguration hardware(HW).The ROS can directly provide some efficient services to users, avoiding the substrate complex architecture. These services include the management and controll of the whole reconfiguration through API.

As the early ROS, OS4RS were developed by IMEC[3]. It is mainly applied in multimedia, while little relevant design information about OS4RS is available. The idea that hardware tasks (Hwt) are controlled by software tasks (Swt) proposed in OSR4S, is applied extensively in the later PRS designs.

ReconOS was developed by the University of Paderborn. It integrates the hardware thread, software thread and delegate thread. However, all the reconfiguration modules must be generated in the design phrase in the reconfiguration and all the bitstreams must be stored in the external memory for the system's relocation, thus this system lacks flexibility.

R3TOS was developed by the University of Edinburgh, the main CPU of which adopted the Xilinx's Microblaze. This architecture is not suitable for Zynq's dual ARM processor [4].

There are no systems directly designed for the Zynq and no one suitable for Linux among the above systems. In this case, the combination of reconfiguration features with the popular Linux can not be realized.

3.LDPRS Architecture

The architecture of LDPRS is shown in Figure 1. LDPRS provides the standardized system services and interfaces for the software and hardware which lends support for the hardware tasks based on the Linux's multi-thread, thus the hardware tasks and software tasks use the same system mechanism.The whole system is made up of bottom hardware and Linux software. Designed by VHDL, the bottom hardware includes reconfiguration CPU, memory

sub-system, ICAP controller and the reconfiguration hardware tasks(Hwt). The software controls the bottom hardware based on the Linux's multi-thread.

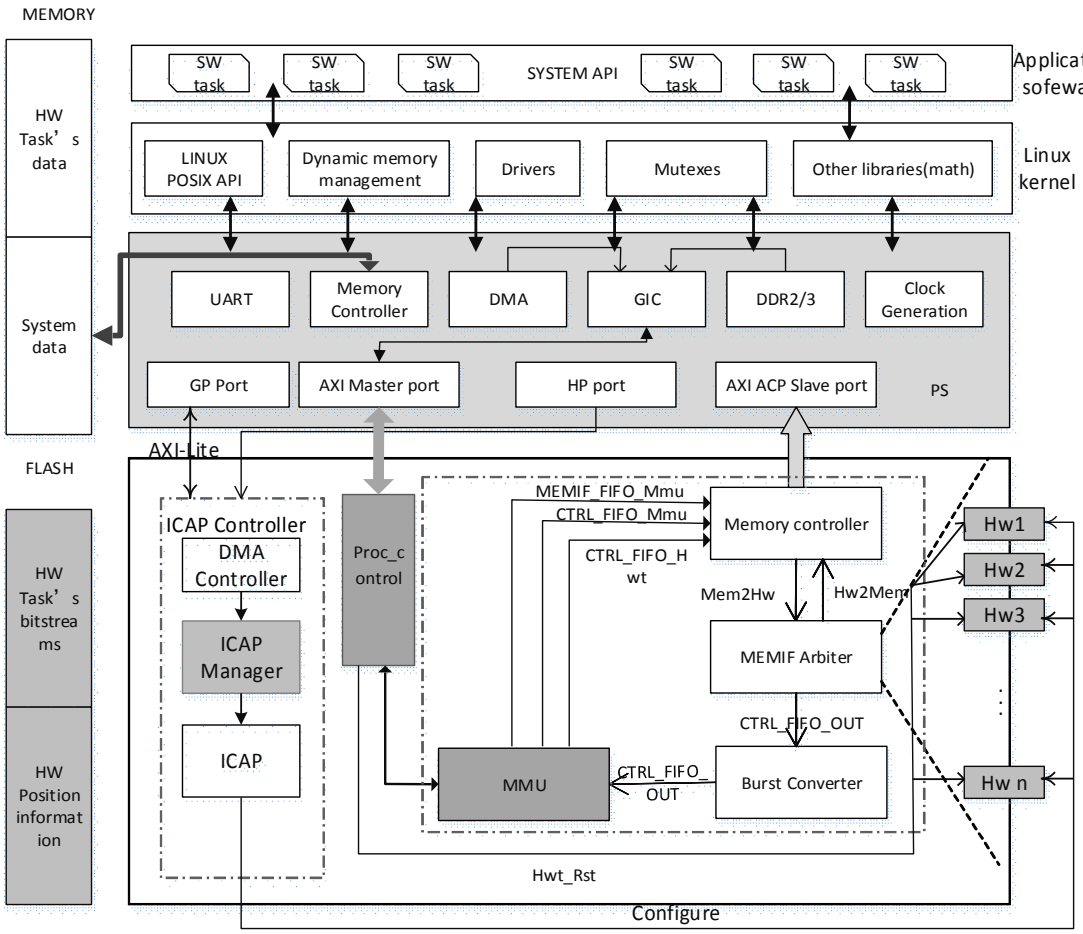

Figure 1: Architecture of LDPRS

The whole reconfiguration is completed by hardware and software tasks. The hardware tasks are the function circuit running on the FPGA's logical resources, while the software is running on the Linux, matched with the hardware. Each hardware task corresponds to a software task. Users send the reconfiguration command to the system, acknowledging the information of hardware tasks. The reconfiguration system loads the hardware tasks to FPGA from the external memory according to the users' command. The tasks' bitstream is generated by Xilinx's PlanAhead and then stored in the external memory. After the establishment of hardware tasks, the system will create the corresponding software task, which will bridge the gap between hardware tasks and the reconfiguration system. The tasks make the inter-task communication through the shared memory.

The system's memory stores the reconfiguration tasks and system data, and the external memory stores partial bitstreams and the location profile of the hardware tasks. The location profile describes the location of each hardware task. In the reconfiguration, the system loads the bitstreams, inquires the position in the location profiles and modifies the bitstreams' configuration information, which will then be loaded to the designated location.

3.1The Architecture of the Memory Subsystem

In the ROS, there's data interaction among all the software tasks. LDPRS conducts the data transmission by means of shared memory. Before reconfiguration, the system applies to a piece of memory space and then divides the space according to the hardware tasks. In the running, hardware tasks store the generated data into the designated memory space and send the

access command to memory controller when access to other hardware tasks is required. The memory controller will search the data in the memory space according to the hardware tasks ID and transmit the data to the designated the hardware task.

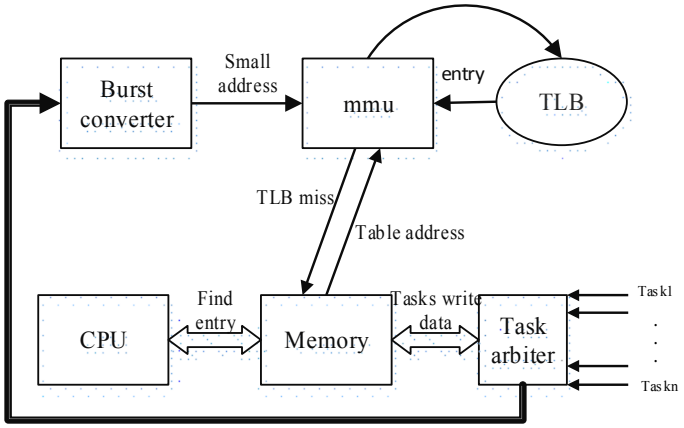

Figure 2: The Memory Subsystem

The memory controller links the memory subsystem to the memory bus of the system as an AXI master. The memory controller is responsible for searching the address entry when the memory manage unit(MMU) can't find the relevant entry. The entry searched will be sent back to the MMU. The MMU will generate a physical address based on the received entry and then send the physical address to the memory controller. The memory controller will inquire the system memory based on the physical address and then transmit the searched data to the designated hardware task. The detailed process is shown as follows (shown in Figure 2):

- The hardware task writes the data to the memory system:

The hardware tasks apply to the memif arbiter for writing the data to the memory. The memif arbiter will transmit the data to the memory controller if one of the hardware tasks buffers is not empty. Then the memory controller will write the data to the system memory through the AXI bus.

- CPU writes the data to the designated hardware task:

The task arbiter chooses the hardware tasks in a traversal way and writes the transmitted data into the corresponding hardware task buffers.

- Hardware tasks access to the memory:

The task arbiter selects the hardware tasks and sends the data address to the burst converter, which will truncate the data address according to the address length and send the short address to the MMU. When the the MMU doesn't find the entry in translation lookaside buffer (TLB), the memory controller will transform the physical address in the page table and then send the transformed address back to the MMU to generate the physical address. The physical address will be transmitted to memory controller for access to memory data, which will be sent back to task arbiter. The specific process is shown in Figure 2.

3.2 The Architecture of Reconfiguration Hardware Task

The reconfiguration task, as the hardware task is located in the designated slot, is controlled by CPU and has access to the memory. All the hardware tasks have the same interface for the unified management and only one hardware task can access to the memory at the same time. Therefore, the synchronization mechanism needs to be established.

There is a synchronization finite state machine in each hardware task to manage the access to the memory. See Figure 3 for the specific procedure. The hardware thread waits on a semaphore, sets the hardware thread as read-lock, reads a block of data from shared memory into a local RAM, unlocks and processes it and then writes the result back to shared memory. The hardware thread performs the void-write lock and unlock. The synchronization finite state machine and user logic communicate via the two handshakes signals data_in and data_out.

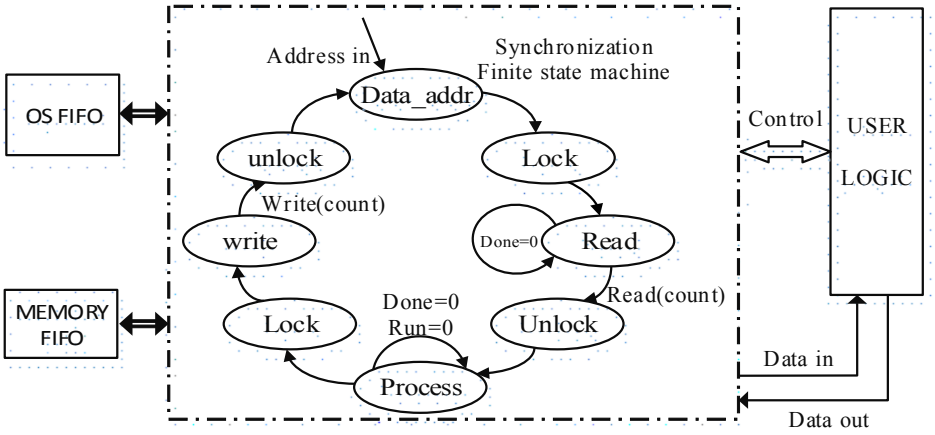

Figure 3: Architecture of Reconfiguration Hardware Task

Each hardware task has four FIFO buffers. Two of them(OS FIFO) are used for data interaction between hardware task and CPU and the other two (MEMORY FIFO) are employed to access the memory sub-system. Users can modify the function of hardware tasks by changing the user logic. When the users design the reconfiguration tasks, they shall ensure that each reconfiguration task shall have the same port consistent with the port of CPU and memory subsystems.

3.3 Architecture of ICAP Controller

Zynq can be reconfigured through internal configuration access port (ICAP) or processor configuration access port (PCAP), but PCAP always blocks CPU at the moment of loading the bitstreams, which is not suitable for ROS. Therefore, we adopt ICAP and design the ICAP controller as follows. Direct Memory Access (DMA) controller loads the bitstreams from the external memory to the ICAP under the control of CPU. This process doesn't consume CPU resources, thus efficiently reducing the CPU load and increasing the reconfiguration speed (shown in Figure 4).

Our ICAP controller supports the modification of the bitstreams' location. The architecture of bitstreams is shown in Figure 5. Each bitstream has the unique 32-bit address called FAR (frame address register), which determines the initial position of the bitsreams. The ICAP controller relocates the bitstreams by modifying FAR[5]. In the bitsreams, there is also a cyclic redundancy check (CRC). The CRC parameter is placed in the footer to check the validity of the bitstreams. To download a Partial Reconfiguration Bitstreams (PRB), we have to invalidate the CRC; to invalidate the CRC, we have to modify the access command to the CRC register. This is represented as (3000 0001) in footer command, and the subsequent word is CRC value. The (3000 0001)must be modified to (30008001), and the subsequent word must be modified to (0000 0007). (3000 8001) is an access command to the command register(CMD), and (0000 0007) is the CRC reset command.

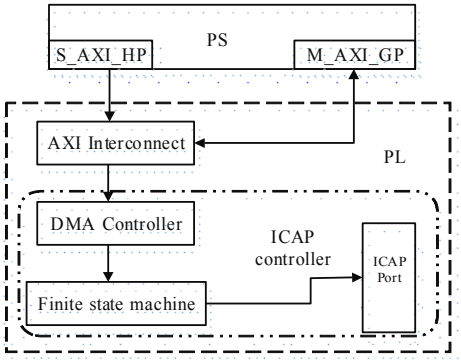

Figure 4: The ICAP Controller

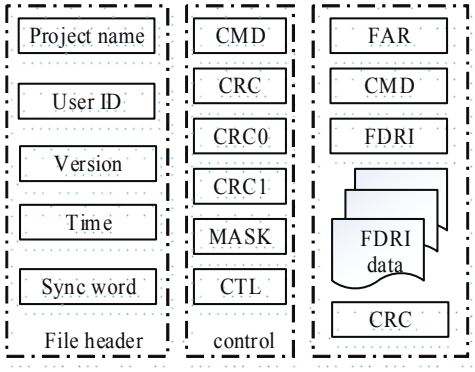

Figure 5: Content of the Bitstreams

The software driver allows user applications to utilize ICAP controller at a high speed without being aware of the low-level operations associated with PR. The driver handles high-level tasks such as management of bitstreams and DRAM memory as well as low-level operations like configuration of the hardware, reconfiguration manager as well as the bitstream movement. We also integrate the ICAP controller into Linux as a driver. Users need to modify the device tree before employing API to control the ICAP controller in Linux.

4. Case Study

We use the zedboard to confirm the efficiency of the reconfiguration system.The hardware task is a bubble sort algorithm. In the reconfiguration, the system generates some random numbers and divides the numbers evenly to each hardware task based on the number of hardware tasks. The hardware tasks sort the number separately and then one of them will make the overall sorting after all the hardware tasks finish the separate sorting. Therefore, the total time in the sorting process consists of time of generating data, time of loading bitstream, the sorting time of hardware thread and the whole sorting time. We will record the time used in the whole process and make comparison. In this experiment, we can test whether API of the reconfigurable system operates smoothly and whether ICAP controller can realize the relocation of bitstream and verify the application of our reconfigurable system in other fields.

In the experiment, the reconfigurable system loads the generated sorting bitstream from the external flash according to the designated sorting task amount by the users and online modify the configuration location based on the bitstream configuration location files. Without interrupting the system operation, the modified bitstream will be loaded to the designated location. Reconfigurable sorting tasks concurrently sort the data, check the sorting result and output the sorting time.

Number of Data	Number of Hwt	time of generating data (ms)	time of loading bitstream(ms)	Partial sorting time(ms)	Total sorting time(ms)
131072	2	5	20	350	530
131072	4	5	45	270	442
131072	6	5	50	230	310
131072	8	5	90	146	256
131072	10	5	120	100	190
131072	12	5	200	79	340
131072	14	5	278	46	500
131072	-	5	0	0	2577

Table 1: Result of the Experiment

As is shown in the Table 1, when the total number of sorting data is certain, the sorting time will decrease with the increase of hardware tasks number, because ICAP controller can load only one bitstream at the same time. However, the partial sorting time decreases with the number of hardware threads, because more hardware tasks make sorting time for each task

decrease, besides, the hardware tasks are performed concurrently. However, the total sorting time decreases firstly and increases then with the increase of hardware thread number. This is because the increase of time of loading bitstreams is larger than the decrease of partial sorting time by the increase of hardware tasks. The results show that the reconfiguration system provides normally functional API and ICAP operates the bitstreams correctly.

The bottom line shows the sorting result of Linux thread mechanism by establishing 14 software threads without using the hardware tasks. From the table, we can see that the time consumed after using the hardware thread is much less than that consumed after using the software thread because the operation of hardware thread is concurrent while the operation of software thread is sequential; therefore, our reconfiguration system can be applied to other fields, such as reconfigurable computing.

5. Conclusion and Future Work

In this paper, we have proposed the linux-based partial dynamic reconfigurable system (PDRS) to provide the unified reconfigurable API for the linux users and reduce the time and complexity of the development. At meanwhile, we also proposed a novel ICAP controller, which can modify the online configuration location of bitstreams and realize the relocation of bitstreams.Our reconfiguration system is based on the slots. Later, more efforts shall be made on the slotless management of FPGA resources.

References

[1] A. Ebrahim, K. Benkrid, X. Iturbe, C. Hong. *A novel high performance fault-tolerant ICAP controller*[C]. 2012 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), Nuremberg, Germany, Jun, 2012:259-263

[2] J.A. Williams, N.W. Bergmann, X. Xie. *FIFO communication models in operating systems for reconfigurable computing*[C]. Proc IEEE 13th Symposium on Field-Programmable Custom Computing Machines(FCCM' 05), Los Alamitos, American, Apr, 2005:277-278

[3] X. lturbe, K. Benkrid, C. Hong, A. Ebrahim, R. Torrego, I. Martinez. R3TOS:A Novel Reliable Reconfigurable Real-Time Operating System for Highly Adaptive[J], Efficient and Dependable Computing on FPGAs. IEEE Trans. Computers.2013,62(8): 1542-1556

[4] X. Iturbe, A. Ebrahim, K. Benkrid, C. Hong, T. Arslan, J. Perez. R3TOS Based Autonomous Fault-Tolerant Systems[J], IEEE Micro.2014, 34(6):20-30

[5] H. Styles, W. Luk. Compilation and Management of Phase-Optimized Reconfigurable Systems, The International Conference on Field Programmable Logic and Applications (FPL), Tampere, Finland, Aug, 2005:311-316