Remote Embedded Simulation System for SW/HW Co-design Based On Dynamic Partial Reconfiguration

Jiaqi Gu[1]*, Ruoyao Wang*, Jian Wang*, Jinmei Lai, Qinghua Duan[2]

1. State Key Laboratory of ASIC and System, Fudan University, Shanghai 200433, China
2. Chengdu Sino Microelectronics Technology Co., Ltd.

*Email:14307130160@fudan.edu.cn
*Email:wsxzwps43@126.com
*Email:wjian@fudan.edu.cn

Abstract

Projects involving both software design and hardware design are usually retarded by expensive equipments, complex simulation and challenging modification. In order to retrench the designers' time and economic costs in SW/HW (Software/Hardware) co-design simulation, this paper demonstrates a remote embedded simulation system to help multiple users manage and simulate their SW/HW co-design projects remotely while scheduling the access to on-chip FPGA resources. We built a small-scale, high-concurrent multiuser management service system on board. The system offers TCP/IP connection and transmission, flexible Wi-Fi network, secure multiuser information and files management, real-time task progress notification, compilation and execution of multiple programming languages, run-time FPGA configuration and simulation, which considerably augments the exploitability for embedded simulation service. Meanwhile, we offer users a supporting PC (Personal Computer) application to attain pertinent features, which has a multithreading GUI (Graphical User Interface). In order to verify this design, we deploy a prototype on a Xilinx Zynq™-7000 AP (All Programmable) SoC (System on Chips) Z-7010 on a ZYBO Board. We apply an image processing SW/HW co-design project to our prototype. The experiment result demonstrates the system's portability and efficacy when dealing with remote access, and flexibility when simulating SW/HW co-design projects through PR (Partial Reconfiguration) technique within reasonable latency. The latency for the end-to-end reconfiguration of a 306.60KB partial bitfile is 8.819ms.

1. Introduction

FPGA dynamic partial reconfiguration has the advantages of low configuration latency and low power consumption. FPGA manufacturers like Xilinx gave support to partial reconfiguration both on hardware and software in recent years. Basing on such tools, many researchers have advocated models for dynamic partial reconfiguration application. K. Vipin and S.A.Fahmy present a prototype that can automatically complete partial reconfiguration based on the status of the system in [1]. This model requires a pre-stored partial bitstream files library and thus can only be implemented to a system with several fixed statuses.

With the rapid development of cloud services, a system that can be configured remotely is needed to share FPGA resources on a cloud server. In [2], Chen et al. designs a model that is suitable for FPGA to be implemented on a cloud server, Ethernet being the connection between them. Users' FPGA designs are allowed to be configured within certain PR regions so that users only need to know the offered interfaces and resources instead of the whole details of the FPGA. In [3], an application of such a model is introduced. Several sensors are connected to the FPGA as peripheral devices. With the help of partial reconfiguration, the system can change the sensors' drivers remotely.

In [2] and [3], FPGA partial reconfiguration is controlled by a micro-processer, which is called the PS (Processing System). However, Mishra in [4] pointed to the resource redundance in such mechanism since the PS is not necessarily involved in the control of partial reconfiguration. Under circumstances where only FPGA is needed, Mishra's design achieves a good performance.

However, the PS part is indispensable in respects of easy Wi-Fi connection and multiple programming languages execution. In this paper, we adopt the basic idea in [2] and [3] to achieve regulative sharing of FPGA resources by utilizing PR (partial reconfiguration) technique. Instead of obviating a CPU on board as introduced in [4], we foreground the cardinal role of an embedded CPU. We implement a Linux system on the PS which can run codes that require FPGA acceleration without occupying enormous CPU resources. Our prototype synthesizes the novel usage of partial reconfiguration technique, remote service through Wi-Fi, and overall management system on the PS, which can offer more features on remote multiuser simulation system of SW/HW co-design projects in terms of flexible combination of multiple programming languages and run-time remote FPGA reconfiguration without network latency. This whole design is deployed on a Xilinx Zynq™-7000 AP (All Programmable) SoC (System on Chips) Z-7010 on a ZYBO Board.

2. System Architecture

The system adopts the classical C/S (Client/Server) structure, which is coded mainly by Python. Figure 1 illustrates the overall structure of the management system.

The client can get access to functions available on a simple GUI. Each user can register an account on the

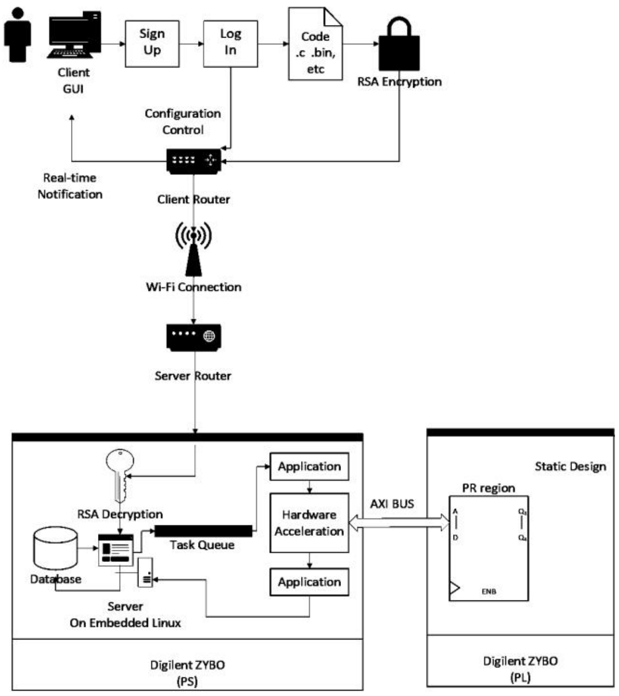


Figure 1. Structure of the Management System

server and set his/her own password. After logging in the server with their accounts, the client can not only be used to upload or download files, but also send commands to the remote server to execute codes programmed with C, C++ or Python. The result of the execution will be registered in the database and displayed in form of real-time notification in the GUI. Users can download the processed data from the server at any time. Aside from software codes, users can also offer a partial bitstream of an FPGA design to apply the dynamic partial reconfiguration to the server's FPGA remotely.

The Wi-Fi-based remote communication between the server and the client is through TCP/IP protocol, and AES-RSA-combined encryption algorithm is adopted to improve the confidentiality of the communication process.

The server is running on an embedded Linux system, which is implemented on the PS of a Zybo board. The database serves to manage users' accounts, check the limitation of registration, etc. AES encryption is used to protect users' privacy.

A high-concurrent "epoll" model is adopted to deal with multiuser visits, while "distributed task queue" is applied in the producer/consumer model to handle certain remote commands. When the server receives an execution command, it will check the validity of the task and then submit it to a task management queue, from which the scheduling model will fetch and deal with the execution, including running codes, configuring the PL, sending data to or receiving data from the PL. Any necessary record of the whole process will be pushed to the client. Since PL configuration and hardware acceleration can be inserted in a software project, the simulation and application of SW/HW co-design projects will benefit from it accordingly.

3. Software Features

The server adopts the I/O multiplexing model "epoll" to address multiuser access. Given the practical application situation and Zybo's processing capacity and capability, non-blocking LT (level triggered) mode instead of ET (edge triggered) mode is used, which circumvents the likely error induced by the ET mode.

Subsequent operations like local database read/write, directory refreshing, notification operations, etc, are immediately processed in the main thread, while time-consuming operations like code compilation or execution, PL reconfiguration, etc, are dealt with by Distributed Task Queue "Celery" especially designed for Linux and Python. This distributed task management framework is principled in producer/consumer model.

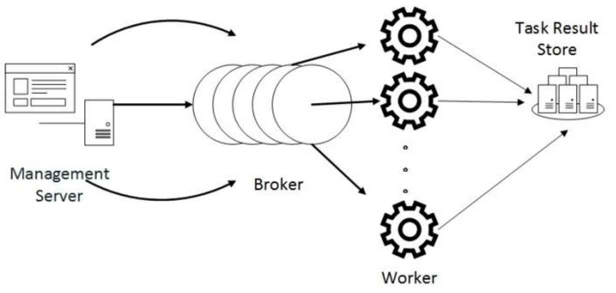


Figure 2. Framework of Celery Distributed Task Queue

Celery's framework is composed of message broker, serving as the communication mediator, worker and task result store, which can be achieved with third parties like Redis, RabbitMQ, etc. Here we use Redis as the result store. Therefore, in a role of producer, the remote simulation management server distributes all project simulation requests to broker as tasks, while celery fetches tasks from the task queue and appoints workers as consumers to execute them, whose processed results will be registered in notification database available to be pushed to users. The task scheduling framework utilized in this design is shown as Figure 2.

4. Hardware Framework

The hardware design includes running embedded Linux on board and dynamic partial reconfiguration of bitstream.

4.1 Build embedded Linux on board

Running the whole system on embedded Linux offers an optimum environment for Wi-Fi connection, compilation of multiple languages and equipment of Python and other future exploitable applications. The Linux in this design is deployed in a micro-SD card and harbors a 3.18.0 linux kernel and a device tree from Digilent and Ubuntu 14.04 file system from Linaro.

With Vivado Design Suite and Digilent Github source codes, the embedded Linux can successfully be booted on Zybo Board formulated by the following flow shown in Figure 3.

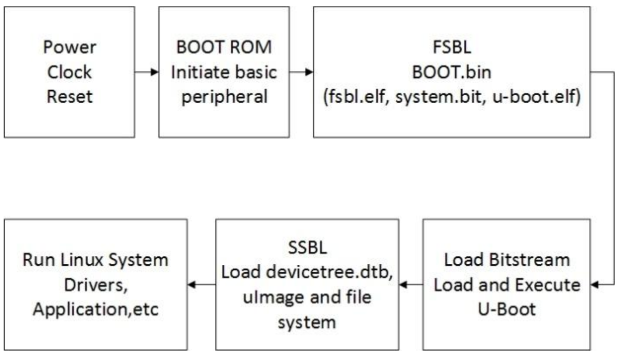


Figure 3. Embedded Linux Boot Flow

4.2 Partial reconfiguration design

Given the method of allowing users to utilize circumscribed region in the PL on Zybo, it is necessary to set the attribution of a fixed instantiation in the top design as “partial reconfigurable”. After the partition definition, a gray box which matches the PR region’s interface ports but without any logic design should be configured as a special RM (reconfigurable module). The PR design requires a specific region for PL to hold the RM, thus a pBlock must be drawn on the PL according to exact regulations, which contains certain amount of logic resources. Every RM has to undergo an OOC (out-of-context) synthesis and generate a partial bitstream after being inserted in the static logic design whose routing must be locked after the initial implementation.

The configuration of partial bitstream considerably reduces the latency and power dissipation to configure a new function in FPGA as illustrated in Xilinx tutorial ug585[5]. Since the partial bitstream has already contained information necessary to PL configuration regarding the accurate location, size, etc, the configuration can be achieved in a mechanism involving PCAP Bridge.

The partial bitstream is first loaded into memory and AXI DMA directly transfers the file data with a relatively great throughput to PCAP interface through the receiver FIFO. Since we are not using AES mode to improve the confidentiality of the configuration, the partial bitstream is finally written into the PL configuration module to implement the fabric.

4.3 Experiment and Results

With the intention to demonstrate the simulation system and given the disparate characteristics of different program languages, we applied a small-scale image processing project involving Python code as the pre-process module, C code as the control module and Verilog design as the hardware acceleration module.

First, we used C code to control the dynamic partial reconfiguration through modifying the register in the DevC device and loading the partial bitstream to partially reconfigure the PL through PCAP Bridge and then call Python code to pre-process the RGB image. After the image data was abstracted and transformed into an apposite format, C code would control, read and write the slave AXI peripheral through mapping its fixed physical address into Linux virtual address. At this time, the peripheral was configured to exert a Gaussian filter (kernel size 7×7) to the gray image data transformed from the original RGB image, and successively the C code would dynamically reconfigure the partial reconfigurable module into a Sobel filter. The gray image, through the previous smoothing processing, would be later transformed into a binary image when edge detection finished. Finally, a piece of Python code saved the processed data as a file in picture format available for the user to download remotely. During the whole process all the task information would be sent to the client as real-time notification. The photo of the remote embedded simulation system is shown in Figure 4.

Figure 5 shows the client shortcut while the user is controlling the remote simulation of the example project on the remote server.

The simulation result was registered in the database and pushed to client as notification while user can download the desired data files from the server. In this case the original image, the output image of the Gaussian filter and the final output are displayed as Figure 6. All these results conform to the expectation of the project’s design.

The two partial bitfiles used in this project are both 306.60KB. Partial bitstream downloading and reconfiguration latency are measured by counting CPU clock which contains C code overhead. The latency of partial reconfiguration is 8.819ms. It is mainly limited by the transmission from the data buffer of the partial bitfile loaded in memory to the DevC device.

In comparison to the features and performance of the design in [4], our design has similar performance but more features. The end-to-end partial reconfiguration latency of a 325.97KB partial bitfile in [4] is 5.629ms. Our design manages to use PS to reconfigure PL in application level, while the design in [4] directly utilizes Gigabits Ethernet and static FPGA design to reconfigure the PR region on the same FPGA in hardware level.

Moreover, the Zybo board we use has an ICAP max clock rate of 100MHz, while the design in [4] is deployed on NetFPGA-SUME Virtex-7 FPGA development board with an ICAP max clock rate of 400MHz. Due to the two reasons above, the latency of our design is relatively reasonable compared with the partial reconfiguration latency in [4]. More detailed performance and features comparison are shown in Table 1.

	Design in [4]	Our Design
Development Board	NetFPGA-SUME Virtex-7 FPGA	Zynq™-7000 AP SoC Z-7010
Partial Bitfile Size	325.97KB	306.60KB
Partial Reconfiguration Latency	5.629ms	8.819ms
Downloading Network Latency	3.197ms	0ms
Wi-Fi Connection	NA	Yes
Multiple Programming Languages	NA	Yes

Table 1. Performance and features comparison between our design and the design in [4]

5. Summary

This paper describes a remote embedded simulation system for SW/HW co-design projects over Wi-Fi based on dynamic partial reconfiguration managed by Linux-based server on board.

On the client, users can upload their simulation projects and related data and utilize both the PS and the PL resources on the portable hardware server to accomplish their comprehensive project development. Compared with the traditional remote FPGA reconfiguration, this system allows users to insert the controlling and partial reconfiguration of PL into software execution. With wireless network as the communication approach, the whole simulation management system harbors relatively great mobility and spatial independency. This system has the ability to implement scheduling of different tasks involving software execution and hardware simulation without the potential FPGA access conflicts by setting the concurrency of Celery's worker to 1 and make real-time registration and feedback of the tasks progress.

References

[1] K. Vipin and S. A. Fahmy, "Automated Partial Reconfiguration Design for Adaptive Systems with CoPR for Zynq," *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, Boston, MA, 2014, pp. 202-205.

[2] Fei Chen, Yi Shan, Yu Zhang, Yu Wang, Hubertus Franke, Xiaotao Chang, Kun Wang, "Enabling FPGAs in the Cloud", *the 11th ACM Conference on Computing Frontiers*, 2014, pp. 1-10

[3] A. A. Prince and V. Kartha, "A framework for remote and adaptive partial reconfiguration of SoC based data acquisition systems under Linux," *2015 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, Bremen, 2015, pp. 1-5.

[4] V. Mishra, Q. Chen and G. Zervas, "REoN: A protocol for reliable software-defined FPGA partial reconfiguration over network," *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Cancun, 2016, pp. 1-7.

[5] Xilinx, "UG585: Zynq-7000 All Programmable SoC", 2015.

[6] Nishmitha Naveenchandra Kajekar, "Tutorial on Partial Reconfiguration of Image Processing Blocks using Vivado and SDK".

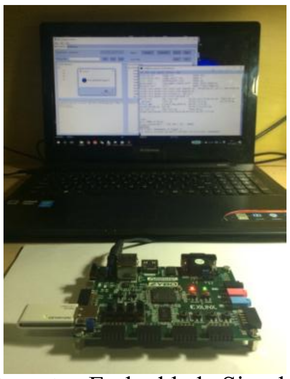
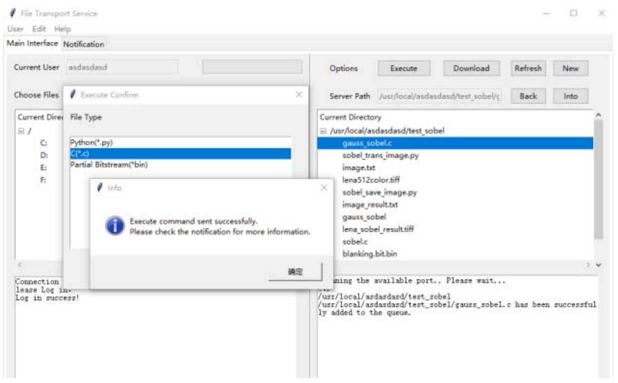


Figure 4. Remote Embedded Simulation System in Experiment



Current User asdasdasd

Server Path /usr/local/asdasdasd/test_sobel/

Current Directory
/usr/local/asdasdasd/test_sobel
gauss_sobel.c
sobel_trans_image.py
image.txt
lena512color.tiff
sobel_save_image.py
image_result.txt
gauss_sobel
lena_sobel_result.tiff
sobel.c
blanking.bit.bin

Execute command sent successfully.
Please check the notification for more information.

```
Connection
lease Log in.
Log in success!
```

```
ning the available port.. Please wait...
/usr/local/asdasdasd/test_sobel
/usr/local/asdasdasd/test_sobel/gauss_sobel.c has been successful
ly added to the queue.
```

Figure 5. The client successfully sends execution command to the remote server.

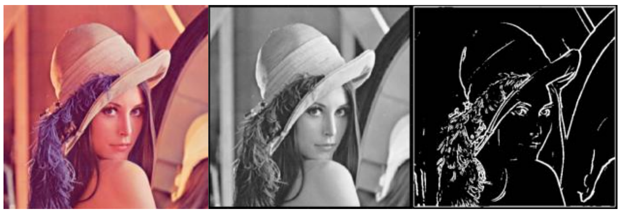


Figure 6. Input image (left), the output image of the Gaussian filter (middle) and final output image (right) of the SW/HW co-design project