Remote FPGA Lab Platform for Computer System Curriculum

Yuxiang Zhang
Tsinghua Univ.
zhangyuxiang13@mails.tsinghua.edu.cn

Yu Chen
Tsinghua Univ.
chenyu15@mails.tsinghua.edu.cn

Xiaojian Ma
Tsinghua Univ.
maxj14@mails.tsinghua.edu.cn

Yuhan Tang
Tsinghua Univ.
yh-ta14@mails.tsinghua.edu.cn

Yilin Niu
Tsinghua Univ.
niuyl14@mails.tsinghua.edu.cn

Shanshan Li
Tsinghua Univ.
liss02@mails.tsinghua.edu.cn

Weidong Liu
Tsinghua Univ.
liuwd@mail.tsinghua.edu.cn

ABSTRACT

This paper presents a MOOC-ready online FPGA laboratory platform which targets computer system experiments. Goal of design is to provide user with highly approximate experience and results as offline experiments. Rich functions are implemented by utilizing SoC FPGA as the controller of lab board. The design details and effects are discussed in this paper.

CCS CONCEPTS

• **Social and professional topics** → **Computing education programs**; • **Hardware** → **Reconfigurable logic and FPGAs**; *Logic circuits*; • **Computer systems organization** → *Architectures*;

KEYWORDS

FPGA; MOOC; Remote Lab; Digital Circuit

1 INTRODUCTION

The computer system is one of the key curriculum for computer science major students. Courses such as Digital Circuit, Computer Organization, Compiler and Operating System introduce computer system in different aspects. To deepen the understanding of the computer architecture, experiments are usually necessary in these courses. For comprehensive and systematic understanding of computer system, the knowledge units introduced in all of these courses should be covered in the experiments.[7]

In Computer Organization course, students are required to build some function units with digital circuits,including ALU (Arithmetic Logical Unit), serial port, memory controller, and simple processors. At this stage, the processor is capable of running a monitor program.

Light-weight operating system can be ported to this processor as the training of Operating System course. Finally, students may build a self-contained computer system which implements a specific ISA (e.g. MIPS32) based on prior works. The light-weight operating system running on the computer verifies the correctness of implementation. Students may also modify the compiler to support their computer system.

To support the experiments mentioned above, platform must provide programmable logic and essential I/O functions. The FPGA based development boards are the common choice. However, the emerging ways of remote education like MOOC(Massive Open Online Course) makes it hard for students to do experiments with boards in hand. The experiment have to be done online. Furthermore, MOOC requires more resources since the students enrolled can be much more than offline classes.

In this paper we represent a remote FPGA experiment platform optimized for experiments of computer system curriculum. We have designed an FPGA lab board and software, which supports both online and offline experiments. Figure 1 gives the remote user interface and photo of lab board. The goal is that experience of doing experiment online should approximate to offline experiment, and the result of online experiment should be reproducible offline. In online scenario, a number of lab boards are installed in the server room, connected to server via network. Students operate a visualized board in web browser, all of the operations are executed on assigned board.

2 RELATED WORK

Some online FPGA lab designs have been proposed in previous works. In [1], by connecting I/O ports of FPGA development board to parallel port of PC, the concept of remote FPGA lab was tested. To display the state of real hardware, Webcam was used in [6] and [2]. [6] utilized the Altera's In System Memory Content Editor with logic inside FPGA to implement virtual switches. [2] used low-cost MCU as the an adapter between FPGA and PC. Only basic user I/O control provided in implementations above, they are not suitable for complicated experiments. In [3] a laboratory system with visualized view was presented. It was based on NI lab equipment and LabView software. In [4] and [5] the system-specific logic in FPGA cooperated with external USB controller to transfer data between user logic and servers. To run on this platform, user's logic have to be integrated to given wrapper code

ACM TUR-C '17, May 12-14, 2017, Shanghai, China
 ISBN 978-1-4503-4873-7/17/05…$15.00
DOI: http://dx.doi.org/10.1145/3063955.3063958

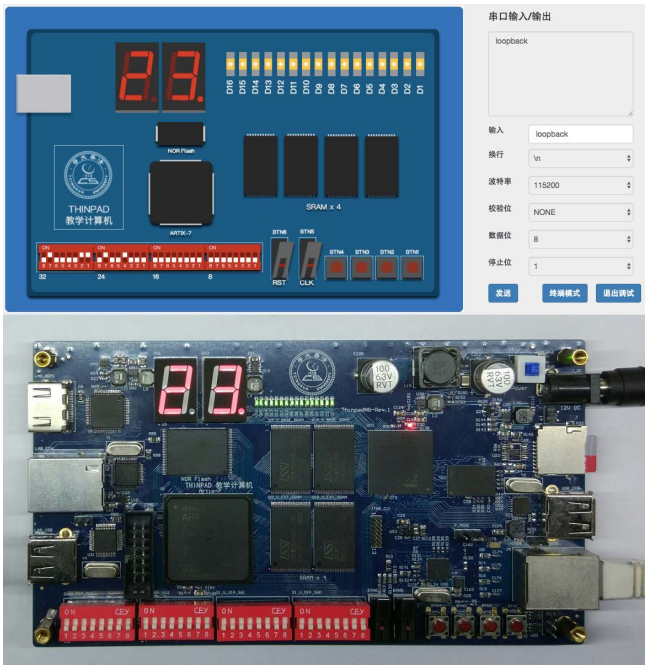


Figure 1: Web UI and lab board

before compilation. Communication through USB also limited the scalability of system, because one server may work with few boards considering the bandwidth of USB host controller.

Another approach of online digital circuit laboratory is building a platform based on simulation. [8] presented a web-based electronic circuit simulation system, in which the SPICE simulation engine is chosen. A major issue of using such simulation system for experiments of computer system curriculum is that, the performance of cycle accurate simulator is not enough to support a complicated design(e.g. MIPS32 CPU). Simulation of a processor running operating system is almost not possible. Besides, the experiment results of simulation may not be as accurate as the results produced by real hardware.

3 TYPICAL EXPERIMENTS

This section lists some typical experiments related to computer system curriculum. Platform design should meet the requirements of the following experiments.

3.1 Digital Circuit Basics

As a tutorial of digital system design, basic input, output and state machine design methodologies are introduced in this experiment. For example, the task can be building a counter with segment display. Additionally, the counter can be controlled by switches, then switch debouncing logic is required.

Another example is ALU(Arithmetic Logic Unit) implementation. ALU designing requires understanding of combinational arithmetic circuits. In this experiment, the ALU designed inside the FPGA takes input from switches, and displays results on LED.

3.2 RAM Access

Memory is one of the main parts of computer system. Memory reading and writing are the basic operations of the processor. In the RAM access experiment, students have to design a memory controller, which generates the signals meeting the timing of SRAM. If data read from RAM matches they written, the implementation is considered correct.

3.3 Serial Port Communication

Serial port communication is a basis of follow-up experiments. Students need to make communication with other device(e.g. PC) by reading/writing registers of serial port controller chips. Besides the communication itself, the techniques of bus arbitration are also important. Because serial controller chip shares bus with memory, so bus contention have to be properly handled by hardware logic.

3.4 MIPS CPU Implementation

Implementing a MIPS processor can be a harder task, the theory of computer organization is applied to the practice. Students should implement a processor, which compatible to a given simplified MIPS ISA. A monitor program is also provided to verify the processor implementation.

As a challenge task, additional functions such as TLB and exception are required to be implemented. This task requires deeper understanding of the processor architecture and implementation techniques. With these essential functions, an operating system can be ported and runs on the processor.

4 LABORATORY PLATFORM DESIGN

4.1 Overview

The proposed platform consists of a number of lab boards and at least one server, shown in Figure 2. All of the user interactivity and laboratory management jobs are processed on the server, while the user-implemented FPGA logic runs lab boards. Each user will be assigned to one lab board after logged in. When users exit the system, the board assigned will be recycled and ready for next user.

Each lab board consists of two subsystems: The laboratory subsystem and controller subsystem. All resources of the laboratory subsystem are open to the user assigned to this lab board, which means the FPGA is fully programmable for user, nothing is reserved for remote control. The controller subsystem monitors the laboratory subsystem, and exchanges data between laboratory subsystem and servers.

For example, if users turn on a switch on remote UI, the server will send this request to controller subsystem, which sets the logic level of corresponding switch of laboratory subsystem to high. Then, the user-programmed logic in laboratory subsystem may drive an LED high, this level change will be captured by controller subsystem, which sends the event to server, resulting in the corresponding LED highlighted on remote UI.

4.2 Laboratory Subsystem

4.2.1 On-board Resources. Laboratory subsystem features a Xilinx Artix-7 series FPGA, with 101,440 logic cells, which is capable of running most experiments related to computer architecture

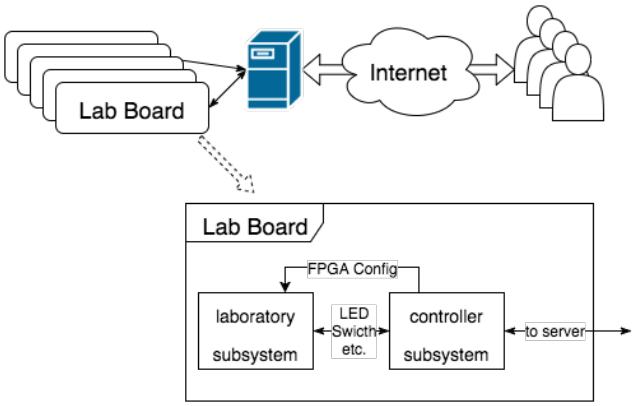


Figure 2: Remote laboratory platform architecture

curriculum. Large logic capacity makes it possible for students to implement more advanced functions such as Caches.

Four 2 MB 16-bit SRAM are organized as two independent units, with 1 Mega × 32-bit in each unit. The address, data and control signals of two units are separately routed to FPGA. This memory schema is useful for some simple architecture, where instructions and data are stored in two different memory spaces. Because instructions and data can be simultaneously accessed, structural hazards can be avoided. Users may also merge two memory units into a single 8 MB memory unit with glue logic. By using all 8 MB memory on board, running most embedded operating systems is possible.

The parallel bus signals between FPAG and SRAM are routed to controller subsystem, hence the controller is able to monitor the memory access transactions on bus.

An 8 MB 16-bit NOR Flash memory is presented as non-volatile storage. User may use this storage to implement file system. This is also useful for storing OS image. Loading image form Flash is faster than serial port, which saves time while debugging hardware logic.

Simple I/O components are also provided for users. 32-bit DIP switch, 2 push-button with hardware debouncing and 4 push-button without debouncing are directly connected to FPGA, acting as the user input. Two 7-seg display and 16 LEDs are connected to FPGA as observable signal output.

Several wires connecting lab FPGA and controller subsystem directly are reserved for extension functions. More peripherals (e.g. PS/2 keyboard) can be emulated by controller with these wires.

Additionally, a parallel-to-DVI encoder chip on board makes it possible for user logic to generate graphical output in offline scenario.

4.2.2 FPGA In-system Configuration. One of the essential functions of remote FPGA lab is to configure the FPGA remotely. The common way to configure FPGA through JTAG requires Xilinx proprietary tools, making it hard to be integrated into third party system.

Our platform utilizes the "Slave Serial Configuration" mode of Xilinx FPGA. As shown in Figure 3, only one INIT_B signal and two serial data serials are required to configure the FPGA. An additional DONE signal can be connected to monitor the status of configuration.

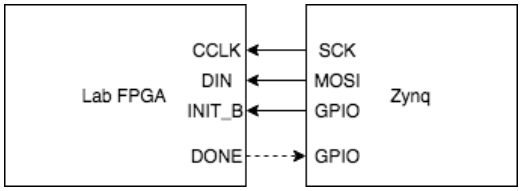


Figure 3: Slave configuration mode connections

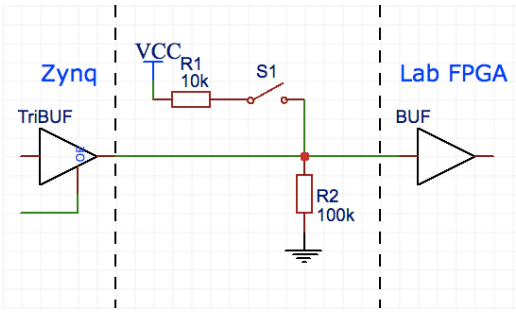


Figure 4: Switch and FPGA connection circuit

The bitstream file (*.bit) uploaded by user is firstly parsed by libxbf [1] to extract the configuration data. Then Zynq sends a low-level pulse on INIT_B pin to put the lab FPGA into configuration state, the configuration data can be transmitted to lab FPGA through 2-wire SPI connections. With SPI clock running up to 50 MHz, the whole configuration process finishes in less than 1 second.

4.3 Controller Subsystem

4.3.1 Controller Core. The controller subsystem is constructed of a Xilinx Zynq-7 SoC FPGA, which integrated an ARM Cortex-A9 processor into the FPGA fabric. FPGA and ARM CPU share the DDR memory.

Considering that some functions (e.g. LED sampling and switch control) require high I/O capacity, and some (e.g. bus analyzer) requires real-time processing, the FPGA is a suitable choice. However, network communication with server requires software involvement. By implementing hardware interface in FPGA, communication software in CPU, this design takes advantages of both FPGA and CPU.

The connection between CPU and server is established on 1000M Ethernet. The Ethernet port on board is design to be PoE-compatible. With the support of powering the board over Ethernet cable, massive deployment is simplified.

4.3.2 I/O Sampling and Control. All user I/O signals for lab FPGA, such as LED and switch connections, are also routed to Zynq for remote display and control. The GPIO core implemented in programmable logic of Zynq handles the I/O sampling and control. The sample rate of LED is over 100Hz, enough for human observation. To avoid the collision of different logic level set by physical switch and Zynq, the circuit design described by Figure 4 is used.

[1] https://github.com/wkoszek/libxbf

In the offline use scenario, the pin of Zynq is set to high-impedance mode, then physical switch works. While the pin of Zynq is set to drive mode in online scenario, then remote control overrides the state of physical switch.

For several physical switch without hardware debouncing, the switch bouncing is emulated by logic in Zynq, to ensure that experiment results online matches offline ones.

4.3.3 Memory Bus Analyzer. The memory bus analyzer function is designed to help students debug their memory access timing. Like most logic analyzer, this function is able to record the transactions on memory bus and display them as waveform. With the help of analyzer, common problems like insufficient address setup time can be easily figured out. Wrong control flow implementation can also be located by checking the memory access sequence.

Since all memory singles between SRAM and lab FPGA are also connected to programmable logic part of Zynq, all memory accesses can by monitored by controller. When memory bus acquisition started, the logic implemented in Zynq waits for $\overline{WR}$ and $\overline{RD}$ assertion, then stores the address, data and timing information to internal FIFO for buffering. Finally the data in FIFO is written to DDR memory, and transferred to server for display. The acquisition can be started and stopped manually, or stops automatically when running out of internal storage.

4.3.4 Serial Port Controller Emulation. To support the serial port experiment, this platform emulated a 8250 UART compatible serial port controller. It is a hardware logic implemented in programmable logic instead of a real chip. Data received from serial port is wrapped and transferred to server via network, while user input from server is unwrapped and written to data FIFO of serial port controller. Because of the requirement that serial controller shares bus with memory, the address and data port of the controller can reuse the pins of memory bus analyzer, which saves I/O pins of Zynq.

4.3.5 Static Memory Controller. Static memory controller is a part of the programmable logic in controller subsystem. This memory controller is able to read and write the SRAM of lab FPGA through interconnects on board. It only works when lab FPGA is unconfigured, otherwise user logic may also access SRAM and cause contention. The basic idea to integrate such a memory controller is to help assess the work of students. After design of students writes data into SRAM, teacher may put lab FPGA into unconfigured state then check the content in SRAM using this function. SRAM read and write function is exposed on web UI like serial communication function.

4.3.6 Control Software. In the ARM side of Zynq SoC, an embedded Linux controls all of the hardware modules in programmable logic such as I/O sampling and serial port. The main procedure of controlling software is an event loop, which handles events from server and programmable logic. Corresponding actions are taken when receiving an event.

Serial port communication function was implemented with the Boost[2] Asio framework. Data sent by lab FPGA triggers pre-registered callback functions, which then sends the data to server.

[2]http://www.boost.org/

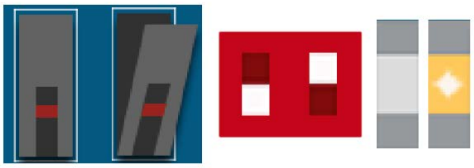


Figure 5: Different images used to emulate the physical state of buttons, DIP switches and LEDs

The Linux system boots from a SD card connected to Zynq for now. However, considering the convenience of software upgrade in massive deployment, the design can be changed that, only bootloader stored on board, while root file system mounts from network(i.e. the server).

4.4 Remote User Interface

4.4.1 Server Architecture. In our platform, server is the center of the whole system. It manages all of the lab boards by assigning free boards to authenticated users and recycling resources from disconnected users.

The server program is built over Node.js, an event-driven asynchronous I/O framework, which is capable of handling large number of concurrent requests. The major role of server is to exchange data between boards and browsers. Because board I/O state (e.g. LED) changes and serial port communication produce very small amounts of data, the server may theoretically handle hundreds of online users at the same time. However, this number is usually limited by the number of physical boards deployed.

Both server-to-browser and server-to-board date transfers are based on Socket.IO [3] , which is a cross-platform bidirectional communication library written in Javascripts. A native C++ implementation of Socket.IO is also available, and is used in Zynq on board. For the server-to-browser data transfers, modern Websocket protocol is used to achieve lower latency than traditional long polling method. Data transferred over Socket.IO is serialized to JSON(JavaScript Object Notation) instead of raw binary data, preventing compatibility issues among various system environment.

We choose MongoDB [4] as the database, storing user information and metadata. Currently we have logged user operations in an experiment. These data may be analyzed to optimize the platform design.

4.4.2 Visualized Operation. The visualized board is shown in the main panel, with three kinds of components: LED (including segment display), DIP switch and button. LED state changes between normal and highlighted to represent the actual hardware state changes. One clicking on a DIP switch toggles its state between 0 and 1. Pressing and holding on button sets its state to 1, while releasing sets its state back to 0. Different images are used to emulate the physical state of switches.

[3]http://socket.io/

[4]https://www.mongodb.com/

The web UI is written in pure HTML, CSS and Javascripts without dependency of obsolete Flash or Applet technologies, achieving quick page response and high browser compatibility.

4.4.3 Serial Port. Two methods of serial communications are provided on UI, simple serial input box and terminal emulator. For simple communication with serial port, user enters text on input text box and click send button, while reply from board shows on receiving text area. This function is usually used by students who want to test the hardware design. They may send some commands to hardware and get replies of hardware.

In operating system experiments, an unix-like shell is commonly required. To support the interactive shell in OS experiments, our platform provides a terminal emulator based on xterm.js [5] library. It behaves just like a normal terminal emulator in unix-like system.

The parameters of serial port such as baudrate, stop bits and parity check are adjustable with dropdown menu before opening the port. In terminal emulator, the automatic newline character conversion between "\r", "\n" and "\r \n" can be manually selected by users, making it convenient for users to work with all kinds of target operating systems inside the board.

An serial port API is also provided for user-developed application to communicate with program inside board. The API is useful for user-customized functions. As an example, a user implemented file transfer tool could work with this API to transfer file to board from computer.

4.4.4 Bus Signal Waveform Display. As the front-end of memory bus analyzer, a timing diagram display with WavaDrom [6] is implemented referring BeagleLogic [7]. After the signal acquisition, the singles of memory bus are displayed on the page with time bar and measurement tools. Users may visually find out the timing problem by comparing the actual timing value with timing characteristics specified by memory chip datasheet.

5 DESIGN FEATURES

Design of our platform is optimized for massive deployment to support the MOOC. With PoE function presented, only an Ethernet cable is needed for one board, no more accessories are required.

All signal sampling and controls are non-intrusive, no modification of user design is needed for remote experiment. The results of online experiments would be the same as the offline ones.

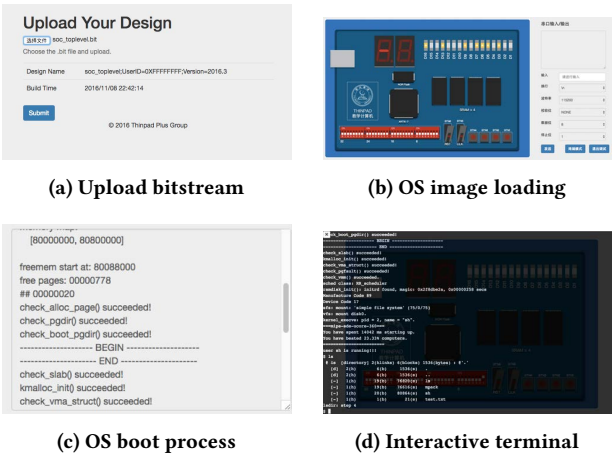
The difficulty of debugging timing problem on memory bus is very common among students. Because of the lack of efficient debug tools, they may spend lots of time figuring out where the problem is. To solve this issue, our platform offers the memory bus analyzer function. The tool helps user check the timing visually, and locate the problem when monitor program runs abnormally.

SRAM read and write functions assist in laboratory assessment. Checking content in memory chips becomes really easy now.

6 DEMONSTRATION

In this section we verify the capabilities of remote lab by running an actual work from students. It is an implementation of 5-stage pipeline MIPS32 processor and its peripherals, which is capable of running uCore operating system and modified Linux kernel. This design utilized 7922K LUT, 7000K FF, and 16 BRAM , which is less than 20% of the available resources of the lab FPGA on our platform.

The steps are shown in Figure 6. Firstly, user logs in and uploads the .bit file to the platform. Then clicking on reset button let processor begin executing the bootloader stored in the block RAM of FPGA. The LED indicates that the bootloader runs. Run the host program of bootloader, which loads image of OS into the SRAM via serial port, and jumps to entry of OS. Later, logs of boot process are printed in the terminal window. After the boot process, user can now interacts with OS in terminal emulator.



(a) Upload bitstream **(b) OS image loading**

(c) OS boot process **(d) Interactive terminal**

Figure 6: Demonstration of remote experiment steps

Memory bus analyzer can be activated at any time while the user-deigned logic running. Then we manually stopped the acquisition after a while. The sampled bus access sequence is displayed in a list. The corresponding signal timing is drawn as waveform when clinking on a specific memory access cycle.

7 CONCLUSION AND FUTURE WORK

We have presented a remote FPGA lab platform designed for computer system curriculum experiments. The platform make it possible for MOOC students to do course experiments with real hardware online. It also helps students debug the hardware logic design issues.

In our future work, the online HDL editing, simulation and compilation functions will be integrated, making the platform more convenient. The debugging function would be enhanced with the supporting of user-defined triggers and internal signal probes. Example of typical experiments will be provided to help beginners quickly understand the usage of the platform.

[5] http://xtermjs.org/
[6] http://wavedrom.com/
[7] http://beaglelogic.github.io/

REFERENCES

[1] Wael M El-Medany. 2008. FPGA remote laboratory for hardware e-learning courses. In *Computational Technologies in Electrical and Electronics Engineering, 2008. SIBIRCON 2008. IEEE Region 8 International Conference on*. IEEE, 106–109.

[2] Vassilis Fotopoulos, Anastasios Fanariotis, Theofanis Orphanoudakis, and Athanassios N Skodras. 2015. Remote FPGA laboratory course development based on an open multimodal laboratory facility. In *Proceedings of the 19th Panhellenic Conference on Informatics*. ACM, 447–452.

[3] Reza Hashemian and Jason Riddley. 2007. FPGA e-Lab, a technique to remote access a laboratory to design and test. In *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*. IEEE, 139–140.

[4] Fearghal Morgan, Seamus Cawley, Frank Callaly, Shane Agnew, Patrick Rocke, Martin O'Halloran, Nina Drozd, Krzysztof Kepa, and Brian Mc Ginley. 2011. Remote FPGA lab with interactive control and visualisation interface. In *2011 21st International Conference on Field Programmable Logic and Applications*. IEEE, 496–499.

[5] Fearghal Morgan, Seamus Cawley, and David Newell. 2012. Remote FPGA lab for enhancing learning of digital systems. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 5, 3 (2012), 18.

[6] Joao Soares, Jorge Lobo, and FCT DEEC. 2011. A remote FPGA laboratory for digital design students. In *7th Portuguese Meeting on Reconfigurable Systems, REC*.

[7] Liu Weidong, Zhang Youhui, Xiang Yong, Wang Shengyuan, and Li Shanshan. 2014. The practice of computer science curriculum design oriented to the cultivation of systematic ability. *China University Teaching* 008 (2014), 48–52.

[8] Ouyang Yang, Dong Yabo, Zhu Miaoliang, Huang Yuewei, Mao Song, and Mao Yunjie. 2005. ECVlab: A web-based virtual laboratory system for electronic circuit simulation. In *International Conference on Computational Science*. Springer, 1027–1034.