

WASHINGTON UNIVERSITY
SEVER INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

FRAMEWORK AND TOOLS FOR SHARED USE OF RECONFIGURABLE HARDWARE
TEST PLATFORMS

by

Jeffrey A. Mitchell, B.S.

Prepared under the direction of Professor John W. Lockwood

A project presented to the Sever Institute of
Washington University in partial fulfillment
of the requirements for the degree of

Master of Science

May, 2005

Saint Louis, Missouri

WASHINGTON UNIVERSITY
SEVER INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ABSTRACT

FRAMEWORK AND TOOLS FOR SHARED USE OF RECONFIGURABLE HARDWARE
TEST PLATFORMS

by Jeffrey A. Mitchell

ADVISOR: Professor John W. Lockwood

May, 2005
Saint Louis, Missouri

A framework has been developed that manages a shared reconfigurable hardware platform. The framework timeshares a hardware test platform among multiple users. It makes the hardware available from anywhere, anytime through a web-based interface. A simple, intuitive interface is presented to users that can serve to control multiple types of hardware. The framework can also be extended to control multiple hardware resources at one time.

Accompanying tools have also been developed that provide useful functionality geared towards testing platforms. These tools analyze and verify data output against blocks of expected output, and allow for a more rapid development cycle.

The framework and tools successfully enabled students in a computer engineering course to program their Field-Programmable Gate Array (FPGA) circuit designs onto a shared test platform.

copyright by
Jeffrey A. Mitchell
2005

To my family and Masha Gelfand for all of their endless encouragement.

Contents

List of Tables	vi
List of Figures	vii
Acknowledgments	viii
1 Introduction	1
1.1 Motivation	1
1.2 Objective and Methodology	2
1.3 Contributions	2
1.4 Outline	3
2 Background and Related Work	4
2.0.1 Co-Verification	4
2.0.2 Multiplexing and Multitasking	5
2.1 Companion Technologies	5
2.1.1 The Washington University Gigabit Switch (WUGS)	5
2.1.2 The Field-Programmable Port Extender (FPX)	6
2.1.3 Networked Configurable Hardware Administrator for Reconfiguration and Governing via End-systems (NCHARGE)	6
2.2 Previous Work by ARL Researchers	6
2.2.1 Project Test Server	7
3 System Description and Design	8
3.1 Framework Description	8
3.2 Control Flow	9
3.3 User Interface and User Management	10
3.4 Connectivity and Running User Commands	10
4 Case Study	11
5 Conclusions and Future Work	14

Appendix A Hardware Sharing and Testing Scripts	15
A.1 reserve.php	16
A.2 restart.php	17
A.2.1 heyu2	17
A.3 restart_script	17
A.3.1 cse566fall04.pl	18
A.3.2 cse566fall04.js	18
A.4 cellcircuit_test.php	18
A.4.1 Search String Files	20
A.4.2 Grade Definition Files	21
A.5 hardware_logout.php	22
Appendix B Installing the Tools	23
Appendix C Necessary Changes for Tool Re-Use	26
Appendix D Building New Tools	28
D.1 Necessities	28
D.2 Useful Code	29
Appendix E Structure of SQL Tables Used	30
Appendix F Controlling Multiple Hardware Platforms	32
References	34

List of Tables

E.1	Structure of <i>resource</i> Table	30
E.2	Structure of <i>resource_journal</i> Table	30
E.2	Structure of <i>resource_journal</i> Table (continued)	31
E.3	Sample Structure of <i>student_grade</i> Table	31

List of Figures

4.1	Photograph of the Network Processing Platform: A WUGS-20, three FPX platform boards, and three Gigabit Ethernet network interfaces	12
4.2	Case Study System Configuration	12

Acknowledgments

Thanks to Dr. Lockwood for all of his support and for supplying some of the figures used in this document, and to Shobana Padmanabhan for helping to proofread this document and provide useful feedback.

Jeffrey A. Mitchell

Washington University in Saint Louis
May 2005

Chapter 1

Introduction

1.1 Motivation

Management of hardware resources, such as hardware test platforms, can pose a significant problem to computer engineers in industry as well as teachers of computer engineering courses. The benefits gained by having more hardware resources available to developers may not scale well compared to the cost of additional units of hardware. More efficient and cost-effective ways to provide multiple developers with needed resources must be found.

If hardware resources are scarce, designs may not be tested thoroughly on real hardware using live data. However, this type of testing is important because it tests the design for faults on much more input. A circuit may simulate correctly for small data sets, but fail when deployed and used with the larger data sets often associated with live data. Functional simulation verifies that the operation of a circuit meets a design specification, but not that the circuit can be synthesized or meet necessary timing requirements. Back-annotated simulation can ensure that a circuit performs correctly after synthesis and place-and-route of the logic elements. Unfortunately, back-annotated simulation is slow and can require several hours or days to process large volumes of data, such as those used for content-processing applications. Testing circuits on actual hardware is significantly faster and ensures that the circuit designs are functional, synthesizable, and operational with large data sets. Failing to test circuits in hardware can lead an engineer to erroneously judge a circuit's design as correct.

Achieving high utilization of hardware resources, such as by providing an efficient method of sharing these resources among multiple developers, is therefore in the best interests of both developers and of those responsible for hardware resource management. However, due to deadlines or budget constraints developers may shortcut testing procedures if the hardware is often unavailable. Accessibility and convenience to shared hardware resources is critical.

Another factor that can lead to insufficient testing of hardware designs is test platform complexity. If the inner workings of a testing platform must be learned in order to make use of it, the process of testing hardware can be unnecessarily time-consuming and frustrating. Ideally, a hardware test platform should be easy to use and contain high-level tools that control and configure

hardware resources, allowing developers to concentrate on their circuits instead of the quirks of the test platform.

1.2 Objective and Methodology

This paper presents a resource-sharing framework and accompanying testing scripts that seek to better utilize hardware for testing and to reduce platform complexity. The overall goal of the work is to enable faster and more complete testing of hardware circuit designs. These tools were designed for use with a hardware test platform as the managed resource, however they can effectively provide resource-sharing functions for some other types of hardware resources as well.

Different types of test platforms can be adopted to work with the framework because there is a standard mechanism to prepare a test platform for usage. In addition, multiple hardware resources can be controlled from a single web server and groupware installation.

The scripts that comprise the toolset along with the resource-sharing framework provide a facility for automating circuit verification. The testing scripts can confirm the correct output of the circuit on input data. They can also be used to automatically assign grades to student circuit designs.

Between the framework and the testing scripts, the toolset streamlines the test-and-debug cycle. It has successfully been used to share a reconfigurable hardware platform among a group of students in a class [1], and its effectiveness has been demonstrated as documented in the case study in Chapter 4.

1.3 Contributions

The contributions to testing methodology in the field of Computer Engineering are numerous:

- Adds hardware resource sharing functionality to the eGroupWare software platform.
- Uses eGroupWare's user and group management capabilities to control access to hardware resources.
- Provides a method for students to test circuit designs on real hardware with little need for understanding the underlying testing platform.
- Can easily be ported to other kinds of hardware resources due to a modular design.
- The software tools have minimal computational demands, allowing the toolset to be run on a server that is also performing other functions.
- Provides powerful and flexible facilities for verifying the correct output of a circuit, and can automatically assign grades based on this output.

1.4 Outline

This document is organized as follows. Chapter 2 contains information about related work and technologies. Chapter 3 presents a high-level view of the structure of the framework and testing scripts. A case study portraying the use of and benefits of the tools is presented in Chapter 4. Conclusions and future work are discussed in Chapter 5.

The appendices will be of most use to those wishing to make use of the toolset. Appendix A contains detailed implementation details for each of the components of the toolset. Appendix B provides information on how to install the toolset. Appendix C has information on the changes necessary to re-use the toolset for different classes and semesters. Appendix D has some basic information on how to extend the functionality of the toolset to apply them to different hardware configurations or different testing needs. Appendix E provides the structure of the SQL tables as used for the case study in Chapter 4. Finally, Appendix F contains information about necessary considerations when extending the toolset to control multiple hardware platforms.

Chapter 2

Background and Related Work

Methodologies have been researched and implemented in the past to increase hardware resource utilization and decrease the time necessary for the test-debug phase of hardware development. Prior methods accelerate design verification or enable multiple designs to run on FPGA circuits concurrently.

2.0.1 Co-Verification

Designed as a replacement for Register Transfer Level (RTL) software simulation, co-verification uses interaction between software and hardware to perform verification faster than RTL. Optimizations can be applied to the algorithms to tradeoff simulation clock accuracy with the speed of the verification system.

In the co-verification system described in [2], a shared register is used to communicate data between the software and hardware components. A system was built that contains multiple FPGAs implemented on a PCI card. A register array called the Shared Communication Register (SCR) is implemented on one of the FPGAs. This FPGA contains control circuits responsible for supporting communication between the hardware and software. Flip-flops on the SCR are made available to the software on the host machine, which can read or write to these through the PCI bus. Device drivers on the PC generate signals to transfer data to and from requested registers.

RTL software simulation programs generally operate at effective speeds of only 10–100 Hz, with the actual frequency depending upon factors such as the complexity of the hardware being simulated and the speed of the host machine running the simulation program. Hardware-accelerated co-verification systems using shared register communication can perform verification at speeds of 0.2–1.1 MHz, providing a speedup between 2,000 and 110,000 compared to software-only RTL simulation.

Co-verification can significantly speed up the simulation of circuits. Since it only performs simulation, however, it is still prone to the shortcomings that make software-only simulation unsuitable for real-world verification.

2.0.2 Multiplexing and Multitasking

Methods have been developed to implement multiplexing [3] or multitasking [4] on a FPGA. These methods seek to time-share hardware between multiple tasks.

One method, proposed in [3], contained eight configurations of a FPGA stored in on-chip memory distributed throughout the die. These configurations can be updated from off-chip while the FPGA is in operation. Logic circuits in the active configuration can also read and write to this memory, giving applications access to a large amount of RAM. The authors propose three modes of use for the time-multiplexing capability: a mode where multiple designs are modeled as a Mealy state machine and cycled through appropriately; a mode where the several independent FPGAs appear to be multiple devices executing multiple problems; and a mode where some or most logic is consistent across various configurations of the FPGA. In this last mode of operation the FPGA, or a part of it, does not appear to be reconfiguring at all.

Necessary requirements for multitasking on FPGAs are explored in [4]. The authors identify some problems with previous implementations (for instance, lack of a task manager can allow long tasks to block execution for a long time). They provide the outline of a design for a multitasking FPGA coprocessor.

An example application of a multitasking FPGA appears in [5], where a real-time video algorithm was mapped onto a context-switched FPGA. The authors used time-based multiplexing to enable the algorithm to fit on a smaller FPGA with high utilization. On a larger FPGA, most of the logic would have been idle.

Although time-sharing hardware still provides some benefit, multiplexing and multitasking had been more important in the past because large designs often did not fit in a single FPGA. Today's FPGAs are much larger, mitigating this motivation. Additionally, the logic required to perform context-switching is complex and wastes space in the FPGA that could be used for circuits.

2.1 Companion Technologies

The framework that has been developed is described in Chapter 3. However, it was built to interact with our existing hardware and software tools. Therefore, an introduction to these existing technologies is warranted.

2.1.1 The Washington University Gigabit Switch (WUGS)

The WUGS switch [6] is a high-speed switching system that forwards data at Gigabit/second rates between input and output ports. It also supports several forms of multicast with full link utilization without large buffers. The WUGS switch operates on Asynchronous Transfer Mode (ATM) cells. ATM provides a common switching and transmission format for network data using fixed-size cells.

Several different types of line cards can be placed in the WUGS ports to allow different types of network packet types to be switched through the WUGS backplane. For instance, line cards exist to connect Gigabit Ethernet and OC-3 to OC-48 fiber optic links to the WUGS. These line cards encapsulate incoming packets into ATM cells. When these cells reach their output ports, the line card reassembles the cells into the appropriate packet format.

2.1.2 The Field-Programmable Port Extender (FPX)

The FPX platform [7],[8] enables rapid prototyping and deployment of packet processing modules in reprogrammable hardware. FPX platforms can be used in a WUGS switch or an industry-produced chassis. Each FPX board includes two data interfaces – one on the bottom and one on top. As a result, FPX platforms can be stacked and can receive data from and transmit data to both the switch connector on the bottom side of the FPX or the line-card connector on the top side of the FPX.

Each FPX board contains two FPGA devices: the Reprogrammable Application Device (RAD) and the Network Interface Device (NID). The RAD contains the customized packet processing functions and connects to Static RAM (SRAM) and Synchronous Dynamic RAM (SDRAM) chips contained on the board. The NID controls how packets are routed to and from the network interfaces and the RAD.

The NID contains a Control Cell Processor (CCP). This lets the NID process control cells that tell the NID to perform various functions. One such function is to change how it routes packets to and from the RAD and the network interfaces. Another is to allow the RAD to be programmed dynamically. When this occurs, the image of the reconfiguration bitstream with the new hardware circuit is uploaded to the FPX board via control cells, and then a control cell is sent to the NID to cause it to initiate the programming of the RAD.

2.1.3 Networked Configurable Hardware Administrator for Reconfiguration and Governing via End-systems (NCHARGE)

NCHARGE is a software application that provides an API for debugging, programming, and configuring an FPX board [9]. The NCHARGE software is used to check the FPX board's status, configure routes on the NID between the RAD and network interfaces, perform memory updates to read values from or write values to the SRAM or SDRAM memory modules, and perform full and partial reconfiguration of the RAD. All of these functions are triggered by control cells sent to the NID or RAD by NCHARGE.

Programs that interface to NCHARGE can be run directly from the command line interactively. NCHARGE runs as a daemon process that listens on a set of network ports. Commands can be sent to an NCHARGE port directly via a companion program called *basic_send* or a user can telnet to the port. The same command syntax is used in either case.

NCHARGE can be controlled via a series of CGI web scripts written in Perl, sometimes called *WebCHARGE*. These scripts are described in [9]. The web provides a much more descriptive and easy-to-use interface than the command-line interface. These scripts take user input, convert them into *basic_send* commands, and show the output on the web page.

2.2 Previous Work by ARL Researchers

Previous work has been performed by researchers within the ARL to allow multiple students to remotely configure FPX hardware and to perform in-circuit testing.

2.2.1 Project Test Server

The Project Test Server (PTS) is described in [10] and provides a web-based interface to allow students to test their placed-and-routed bitfiles. Students upload their bitfiles to the PTS, where it is placed in a queue. When their bitfile moves to the front of the queue, the PTS executes commands to reconfigure the attached FPX hardware with the student's bitfile, creates and injects traffic into the network, records the response traffic from the circuit, and displays a test summary in HTML. The traffic that is injected can be predefined for an assignment or can be specified by the student. There were some desirable features that are not present in the PTS toolset.

First, the PTS toolset does not store sets of results on the server. When multiple versions of a circuit are created, it is useful for the results of previous tests to always be accessible for students, so that they can compare output of changes they make to their bitfiles to see what the effects are. The storage of previous design bitfiles is also useful because it allows students to re-test previous versions of their circuits if they so wish. To support these features, a user management scheme was needed to associate files with each user.

Second, because it is a queue-based system performing batch jobs, interactive testing with the circuits is not possible. For interactive testing, students need to be able to reserve and hold the testing platform for a time period, and the platform must permit interaction with the running circuits.

Finally, the PTS generates and injects specific network traffic into circuits. While this functionality is good for basic testing, it is not robust enough to fully test circuits designed to operate with live Internet traffic. A system is needed that can operate with a real-time Internet connection to satisfy full testing requirements.

Chapter 3

System Description and Design

To meet the needs of our students and our courses, a new toolset was needed. This toolset takes the form of a hardware-sharing framework and a set of accompanying utilities. A description of the new toolset appears in the next section. The later sections describe the high-level design of the tools.

3.1 Framework Description

The toolset that has been developed improves resource utilization by providing a simple yet effective means of time-sharing access and control of a test platform to multiple users or groups of users. Blocks of testing time can be provided to a user during which the user's testing session cannot be interrupted or preempted by another user, preventing tests from conflicting.

The complexity of the test platform is reduced by abstracting communication to the hardware resource and presenting a simple user interface to the developer. The framework makes use of a free, open-source, web-based groupware suite. The test platform can be accessed and controlled from any web browser by connecting to the groupware's web server, making it easy for a developer to test their circuit from anywhere, even if they are not near the physical hardware.

The tools allow time-based sharing of FPX boards in a WUGS-20 switch, although the tools are modular and could be modified to support other types of hardware. Students wishing to program their designs onto the FPX boards first *authenticate* themselves, proving their identity and allowing the system to verify that they should be allowed access to the hardware. They then *reserve* the hardware for a certain period of time, or if the hardware is already in use, they must wait for it to be available. Once they have reserved the hardware, they *control* the hardware by programming their designs onto it and using features of the tools to help analyze the functionality of their design. Finally they *relinquish* the hardware, either by manually giving up control when they are finished, or forcefully losing it if their reserved time passes.

Similar to co-verification, the tools use a hardware-software approach to help students verify their designs. Unlike co-verification, however, circuits run natively in the FPGA hardware at full speed. Rather than simulating their circuits, they are actually executing them. Students can also interact with the running hardware to obtain test results.

Additionally, the manner in which the hardware is shared is similar to time-multiplexing processes on a FPGA as described in [3] and [4], except that this system has both a FPGA large enough to prototype an entire network processing system and a time-slice in terms of minutes, not fractions of a second. There are certain benefits to this. For one, it means no overhead is needed for implementing task-switching circuits in the FPGA, more space is left on the reprogrammable chip for student designs. Also, since the data that students process on the FPX platform is live Internet traffic that may be arriving at any time, it ensures that traffic arriving that needs to be processed by a particular student's circuit will not arrive when another design is currently loaded.

The reservation time was chosen to be large enough to provide a student with adequate time to test a revision of their design without being interrupted. The ability for students to access their designs, the hardware, and the testing tools all from any place and at any time enables efficient utilization of the hardware. This is opposed to hardware that requires physical proximity to access or that is in a lab that is only open during certain hours, where utilization of hardware will likely be lower.

3.2 Control Flow

It is desirable to abstract the interface to the underlying hardware, so that if the hardware resource were changed but the functionality requirements were not, basic functionality will be preserved without a change to the user's interface. To accommodate these objectives, a modular design was created.

The design consists of two computers – the *Server* and the *Controller*. These names were chosen because they succinctly describe the function of each of the computers. The computers are connected via a network connection. Communication between the two consists of a push mechanism, sending commands over the network connection from one to the other. Generally, the Server pushes commands out to the Controller.

The Server provides the interface to the users. The Server runs a collection of other processes and tools that include Apache, MySQL, PHP, and the toolset. The user interfaces with the tools, and depending on the user's directives the Server either executes the commands directly or passes them to the Controller.

The Controller controls the hardware. For the FPX platform, the Controller runs NCHARGE. If the FPX boards are being used with a WUGS-20 switch, the Controller would also be running the Jammer software [11]. The Controller essentially acts as a middleman. It receives generic commands from the Server at the behest of the user and converts them to commands native to the attached hardware. The Controller generally does not send commands back to the Server, although it passes the output of executed commands to the Server for display to the user.

Because of this modular design, the hardware resource connected to the Controller can usually be changed without needing to change the commands sent from the Server to the Controller. For instance, to serve as a controller for a WUGS-20 switch, the Controller must use a specially-patched Linux kernel and a PCI card with a custom-designed network interface card called the APIC. The same functionality can be performed by FPX cards stacked in a GVS-1000 chassis, where only a stock Linux kernel and a commodity Gigabit Ethernet card are required for the Controller. In either

of these cases, the Server sends the same command to the Controller, telling it to run “restart_script.” The actions of this script are specific to the currently-attached hardware – for instance, an IP-based version of NCHARGE would be used with the GVS-1000 chassis as opposed to an ATM-based version that would be necessary with the WUGS-20 chassis.

In this way, the user interface and the user’s high-level commands are separated from the low-level commands the Controller is required to perform. The Server and Controller can run on the same machine, but this is not necessary or recommended.

3.3 User Interface and User Management

User interface and user management is provided by hooking the tools into eGroupWare [12]. eGroupWare is an open-source and free groupware suite written entirely in PHP. eGroupWare provides many benefits: a pleasing and consistent look and feel for web elements such as fonts and buttons; group functions such as a shared calendar and shared files; a personal file manager for each student where they can store their bit files and the results of their tests; and user management and authentication capabilities, allowing users or groups of users to be given or denied access to the hardware. The user interface for the tools is written in HTML dynamically generated using PHP, which allows different pages to be returned to the user if certain conditions were met.

3.4 Connectivity and Running User Commands

One of the benefits to hooking into eGroupWare and using PHP scripts to generate the user interface is that PHP is very versatile. It does much more than just generate web elements. For instance, PHP can pass commands through to the local system. It is in this manner that the tool scripts running on the Server send commands to the Controller to execute.

The Server and the Controller connect to each other over the network via the secure shell program (SSH). SSH is called with an argument after the address, which causes it to run that program on the remote host and return its output to the local host. The tools pass commands to the local system, which executes these commands with the web server’s user context. These commands may run binary files, or may execute other scripts on either the Server or the Controller. As an example, suppose that the Server is at IP address 192.168.50.8 and the Controller is at 192.168.50.9. One of the tools may contain a command to execute such as “ssh 192.168.50.9 script_to_run.sh” which will run the *script_to_run* program on the Controller. The output of this script will be returned to the Server, where it can be displayed on the web page running it.

SSH allows the use of shared keys for authentication between computers. This serves two important functions. First, it allows scripts to perform their functions without requiring a password to connect to the other machine. Second, it prevents these scripts from being run by other computers, even if that computer is impersonating the IP address or hostname of the Server or Controller.

Not all of the scripts used by the toolset are PHP scripts hooked into eGroupWare. Some are Bash shell scripts, and others are Perl scripts. These non-PHP scripts mostly reside on the Controller and perform functions to manipulate the hardware testing platform.

Chapter 4

Case Study

The framework and testing scripts were first successfully used in the *Reconfigurable System on Chip Design* course taught during the Fall 2004 semester at Washington University in St. Louis [1]. Feedback from the students and instructor helped provide an understanding of the necessary capabilities and features needed to make the toolset useful in a production environment.

The Server used for the class was a dual-Athlon system running SUSE LINUX 9.1 with two gigabytes of RAM. Simulation was handled by Modelsim 6.0, and synthesis by Synplify Pro 7.7. Place and route was run through Xilinx ISE 6.2. eGroupWare 1.0.0.006 was served by Apache 2.0.49 on top of MySQL 4.0.18 and PHP 4.3.4. The Controller used for the class was a single-Athlon system running Red Hat Linux 9, controlling a WUGS-20.

Three FPX boards and three Gigabit Ethernet line cards were used in the WUGS-20. A photograph of the system is shown in Figure 4.1. One FPX performed Transmission Control Protocol/Internet Protocol (TCP/IP) processing of data [13], a second FPX implemented the student's content processing circuit, and a third FPX captured received network traffic into the FPX's SRAM chips.

Two of the Gigabit Ethernet line cards were configured to pass bidirectional traffic between ports. One port sent and received traffic to and from the student PC located remotely on the Internet, and the other sent and received data to and from the content host. All traffic that passed between these two cards was multicast to the FPX board running the TCP Processor [15]. The TCP Processor, in turn, forwarded to the student's content processing circuit. The third Gigabit Ethernet line card was used for configuration of the system via Jammer and NCHARGE.

A diagram of this process can be seen in Figure 4.2. Students designed their circuits on their PCs and uploaded their bitfiles to the Server through eGroupWare's File Manager application (step 1). They then used the web interface to reserve the WUGS-20/FPX hardware platform. When they selected a bitfile to program and clicked the "Restart" button (step 2), the tools copied their bitfile to the Controller and invoked the Controller's reset cycle for its attached hardware (step 3).

After the three FPX boards were programmed with the appropriate bitfiles, students sent traffic through their PC and accessed data on the content host (step 4). All traffic that passed between the student PC and the content host was multicast to the TCP Processor, which in turn sent data streams to the student's circuit on the second FPX board. In this manner, students were



Figure 4.1: Photograph of the Network Processing Platform: A WUGS-20, three FPX platform boards, and three Gigabit Ethernet network interfaces

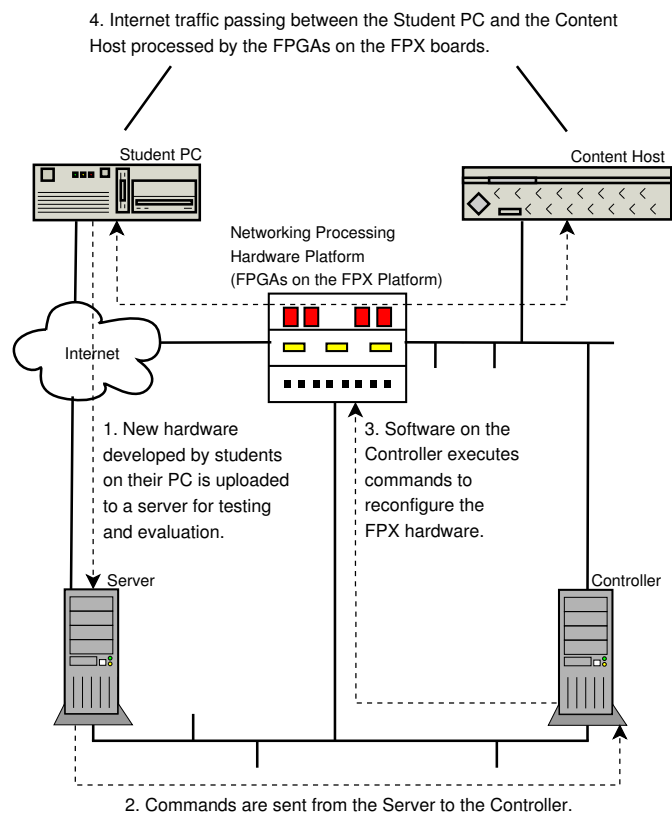


Figure 4.2: Case Study System Configuration

able to run live Internet traffic through their circuits. Statistics about the operation of the circuit as well as the content of input/output packets were stored in memory using the third FPX board.

Students were assigned two machine problems and a final project that made use of the network processing platform. In their first machine problem, students read flows from the TCP processor to implement a statistics circuit. In the second machine problem, students designed and implemented a circuit that performed semantic processing of content from a web server. In particular, they implemented a circuit that counted the frequency of specific words appearing in a set of poems stored on a content server. For their final projects, students implemented circuits that detected email spam, planned collision-free motion for a robot, processed SNORT rules, automatically identified the use of foreign languages, and provided content delivery functions.

The benefit of using the toolset to the students was significant. The interface allowed students to focus their time on the design of their circuits instead of learning details of the hardware test platform. The reservation feature of the framework allowed students to successfully and easily interact with their running circuits. The web-based interface made it convenient for students to test their circuits at their leisure, helping minimize contention for the hardware. Finally, the tools enabled a single hardware platform to be efficiently shared by all 14 members in the class.

Chapter 5

Conclusions and Future Work

The toolset enables successful sharing of hardware resources among multiple developers. Users are able to test their circuits on real hardware with large data sets with only a minimal understanding of the underlying test platform. In addition, users can instantly and automatically verify the correctness of their circuit.

Using the toolset, resource utilization is increased without sacrificing resource availability. Its modular design allows it to be ported to other platforms with minimal effort, and it is readily usable for many purposes.

Future work could be performed to enhance the toolset in some of the following ways:

- *Multiple hardware resources*: Multiple hardware resources could be supported by default. Appendix F details the necessary steps to enable this functionality.
- *Versions for other hardware types*: The toolset could be ported to different hardware resource types. A repository of some sort could be kept, perhaps in CVS, of the different scripts (such as the different restart scripts on the Controller) necessary to support these.
- *Integration with other tools*: Currently integration with the CellCapture circuit and sramdump program allow certain verification methods. Integrating with other testing circuits and software tools would increase the usefulness of the toolset.

Appendix A

Hardware Sharing and Testing Scripts

This chapter contains details of the function and interaction of the toolset's components. The toolset was designed for a WUGS-20 switch containing multiple FPX platform boards, and the information in these sections pertain to that hardware configuration. For information on applying the toolset to other hardware platforms, see the appendices.

The list below provides a brief understanding of the flow of control as a user interacts with the toolset. The items are presented roughly in the order in which they would be accessed or executed. Each item will be explored in greater depth later.

Server

- `reserve.php`: Checks for current valid usage of the hardware. If the hardware is not already reserved and in-use, it is reserved for a seven-minute period for the requesting user. Displays buttons allowing the user to restart the hardware and load a bit file, run tests and receive results, and log out.
- `restart.php`: Runs the `heyu2` program and `restart_script` (described below) with the user-selected bitfile.
- `heyu2`: `heyu2` is a program that controls X10 power modules. It is used to power-cycle the WUGS-20 switch so that it can be completely reset when a new bitfile is loaded.

Controller

- `/home/wwwrun/restart_script`: Runs after `heyu2` power-cycles the WUGS-20 switch. It runs the `do_all.cgi` script, which resets and configures the programs necessary for the Controller to communicate with the WUGS-20 switch, and then runs the `Jammer` script, Perl script, and `scanctl` programs listed below.
- `cse566fall04.js`: The `Jammer` script that sets up the virtual circuit (VC) routing paths necessary for the current testing paradigm.

- `cse566fall04.pl`: A Perl script that runs a series of NCHARGE commands (using the `basic_send` program) to upload bitfiles to one or more FPX boards. It also executes `basic_send` commands to configure NID traffic flows on the FPX boards.

Server

- `cellcircuit_test.php`: `cellcircuit_test.php` calls the `sramdump` program to fetch the memory contents of the CellCapture circuit, using provided parameters. If specified, this script also performs the transfer of HTML files across the WUGS-20 switch and compares the contents of the CellCapture circuit's memory with the defined expected results.
- `logout.php`: Logs the current user out of the hardware testing system, clearing the hardware's use status. Logging out ensures that a user will receive guaranteed uninterrupted time with the hardware next time they reserve it.

All scripts check to ensure they are being run by the user currently in control of the hardware. This prevents another user from manually accessing the scripts via a URL.

The following sections describe each of the scripts listed above in more detail.

A.1 `reserve.php`

When a user accesses the `reserve.php` script, it first logs the access to a table called the *resource journal* as a “trygrab,” meaning that the user tried to grab control of the hardware. Access is granted under one of the following three conditions:

- *No user is in control of the hardware*: The script grants access to the hardware for the requesting user for a guaranteed minimum configurable period, defaulting to seven minutes. This also causes the following actions to be taken: the resource journal is updated to reflect that the current user has taken control of the hardware; the resource table in the database is updated to reflect the current owner of the hardware, as well as the time it was acquired and the expiration time; and several commands are run to perform cleanup from the last user. These final commands ensure that the previous user's scripts ended and are not hung and ensure that the previous user's bitfile is not made available to the new user.
- *A user is in control of the hardware*: The script queries the database to check the expiration time of the previous user. If the previous user's expiration time has not passed, the requesting user receives a message stating at what point the hardware will again be available. If the previous user's time has in fact expired, the requesting user becomes the new owner of the hardware, and the old user is forcefully removed from ownership. The same journal and resource table updates are performed, as well as the cleanup operations.
- *The requesting user is in control of the hardware*: The script allows the user to proceed with using the hardware, although it does not perform the cleanup associated with the user gaining control of the hardware from no user or a previous user, allowing the user to continue their previous session. However, it does not extend the user's guaranteed hardware control time. If

a user wishes to manually extend their time, they must perform a logout and then try to grab the hardware again. Note that if the user attempting to grab the hardware is the same as the previous user of the hardware, and they have manually logged out, they are not allowed access to the hardware again until their initial reservation time has passed. This prevents abuse of the logout feature, so that a user cannot continually log out and back in to keep control of the hardware indefinitely.

Once the user has control of the hardware, the `reserve.php` script causes several options to be displayed. The first option is to restart and configure the WUGS-20 switch and load the selected bitfile. A dropdown (or select) menu box is constructed with the names of all `.bit` files in the home directory of the user's eGroupWare File Manager. This bitfile is then programmed into a FPX board in the switch by the `restart.php` script. The second option is to dump the contents of the CellCapture circuit's SRAM banks, with an option to test it for patterns, and this is performed by the `cellcapture_test.php` file (although the `reserve.php` script does a little more here than has been described, this further functionality will be described in the `cellcapture_test.php` section). The third option is to perform a manual logout from control of the hardware, handled by the `logout.php` script.

A.2 restart.php

`Restart.php` is a simple script with only a few functions. On access, the `restart.php` file will log a value of "programming" into the resource journal. Then it uses the `heyu2` [14] program to turn the switch off and back on. Next it uses `scp` to copy the user's chosen bitfile to the Controller over the network. Finally, it runs the `restart_script` command on the Controller to reset the hardware.

A.2.1 heyu2

`Heyu2` is a program that can control power modules based on certain chipsets. A variety of functions can be performed, one of which is to control whether a connected device receives power. The underlying X10 hardware controlled by the `heyu2` program is slightly unreliable, and the program sometimes does not actually perform a power-cycling, even though there is no failure output.

A.3 restart_script

The `restart_script` program is a bash shell script that performs the functions necessary to set up the hardware platform connected to the Controller. It first kills any `scanctl` processes (used for querying and setting scan strings in the Scan circuit [15]) that are currently open. This is necessary because `scanctl` will remain waiting forever if it does not receive a reply from the Scan circuit due to a problem with setup or the Scan circuit itself.

It then runs the `do_all.cgi` script, which prepares the Controller to talk to the WUGS-20 switch. Next it runs the Perl file described in the next subsection to perform the actual programming of the FPX boards' RAD bitfiles and NID routes. It completes by attempting to ping the Gigabit Ethernet line card on the WUGS-20 that is connected to the private network. This is done to ensure that the ARP table in the line card contains the correct values to talk to the Controller. While

sending a ping is not strictly necessary, it can eliminate problems that can occur when a command seems to fail but has actually worked correctly due to ARP resolution needing to take place. In the particular case of the `scanctl` program, since the `scanctl` program sends one command and waits for a reply, during this ARP negotiation that packet can be dropped, causing `scanctl` to hang indefinitely; pinging the card first eliminates that problem.

A.3.1 `cse566fall04.pl`

This Perl file is used to perform the programming of the FPX boards. It performs a series of commands that uses the `basic_send` program to interface with NCHARGE. Bitfiles are loaded onto the RADs, and the NID VC routes are programmed.

Two things are of note. First, each print command is run twice – once with quotes and once with backticks. The first causes the quoted text to be printed; the second causes the command to be run and its output printed. Second, at the end of the Perl script an associated Jammer script (described in the next subsection) is executed. It is necessary to pair this script and a Jammer script together, since the VCs on the switch must be tuned for the particular setup on the FPX boards. It follows that any change in one of these two files should cause the other file to be evaluated to ensure that it is still correct.

A.3.2 `cse566fall04.js`

This Jammer script sets up the VCs on the WUGS-20 switch to set up traffic flows between the FPX boards and the Gigabit Ethernet line cards.

A.4 `cellcircuit_test.php`

This script has two functions: it downloads data from the CellCapture circuit and it enables testing of data. To do this, it works hand-in-hand with the `sramdump` utility (written by David Schuehler). The `sramdump` utility interfaces with the Control Cell Processor contained in various FPX RAD modules to retrieve the contents of the SRAM memories contained on FPX platform boards.

A discussion of the `cellcircuit_test.php` script should first begin by explaining the further features of the `reserve.php` script. In order for the `cellcircuit_test.php` script to work, it needs the parameters to pass to the `sramdump` utility. Values of these parameters are obtained by the `reserve.php` script.

The `reserve.php` script displays multiple select boxes and text boxes in which the user can enter the format of the data dump, the address to begin dumping, which SRAM bank to use, and how many words to dump. If the user chooses solely to dump data from the CellCapture circuit, this is all the information used. However, another select box is also created in case the user chooses to run tests on that data as well, with the entries in this select box drawn from three separate sources: a default source in case no others are chosen, any `.ssf` (Search String File, explained in detail in a following subsection) file contained in the user's home directory in the File Manager, and a file named `gradedefs.gdf` (Grade Definition File, also explained in detail later) located in the eGroupWare base directory.

The `gradedefs.gdf` file is used during automatic grading of student submissions, and contains a list of parameters for not only testing the circuit but optionally for the cell dump itself. The `reserve.php` script uses the entries contained in this file to construct the values for the select boxes that will be passed to the `cellcircuit_test.php` script. The difference in the format of the constructed values between the user's `.ssf` files and those in the Grade Definition File allow autodetection by the `cellcircuit_test.php` script of whether the output should be graded or not: if the value of the select box returned to the `cellcircuit_test.php` script contains a string with `".ssf"` as the last four characters, such as `"my_strings.ssf,"` it is assumed to be a student's Search String File; if the value of the select box consists of a series of dashes separating strings – for instance, `"mp1.ssf-student_grade-mp1-mp1_bitfile_hash-50-testsites.txt-zero-zero-1000,"` – it knows that the value contains the parameters for a graded test.

Moving onto the `cellcircuit_test.php` script itself, its first function is to update the resource journal to record the `sramdump`. Next it tests to see whether it is running a test and dump or a dump only. If testing is being performed, it checks to see which Search String File to use (the Default option is always available, so there cannot be a case where no file is selected). If the Search String File was defined by the Grade Definition File, it deconstructs the testing parameters that were passed in into variables for later use, and then opens the specified Search String File. If the file was defined by the user, it simply opens it without any other processing.

The defined Search String File is filtered for comments and blank lines, and then separated into the specified search blocks, with each block an index to an array. Each of these blocks is then separated into another array by lines, so that the final structure holding the search strings is a two-dimensional array. While not necessarily the fastest structure that could be used, it works well for a scripting language processing limited amounts of data. In addition, using some other structures would make wildcards very difficult to handle; for instance, using a tree-type structure would mean that when a wildcard was encountered each leaf would have to be traversed to look for the next character.

The values of the parameters to use for the `sramdump` program are then read (if they have not already been set by a Grade Definition File). The output file name is then constructed from the name of the bitfile used, the action taken, the data format, and a timestamp. If testing has been selected, the script displays to the user whether or not values for the `sramdump` have been selected for them, and, if applicable, runs any defined commands prior to testing. Finally, the actual `sramdump` itself takes place.

Following this, the file containing the memory dump is copied to the user's home directory in the eGroupWare File Manager. Some entries are made into eGroupWare's database tables that are necessary for the File Manager to recognize the file, and the file is then parsed and displayed on the web page for immediate viewing by the student. If testing has not been selected, this script is finished.

If testing has been selected, the contents of the dump file are compared to the search strings. Because this is done on a line-by-line basis, and the entire file is gone through for each search block, for very large dumps and large numbers of search blocks this can conceivably take a long time. When the number of matches has been found, the totals are compared with the expected counts. If they are the same, the output is listed as being correct; if they are different is is listed as being incorrect.

If the testing is being controlled by a Grade Definition File, it will update an associated grade in the database.

A.4.1 Search String Files

Search String Files are used by the `cellcircuit_test.php` script to test for values contained within dumped data. If the files are to be autodetected in a user's File Manager, they must be in the user's home directory and have a ".ssf" extension. The file format consists of blocks that are separated with the line "[Output]" with each output block constructed of eight hexadecimal characters per line. This is purposefully the same format as `sramdump`'s output with the `-t C` option, which outputs in a "Cell" format that can be used as input for ModelSim.

Each block is accompanied by two extra lines. The first of these lines is a number representing the length of the previous output block, in lines. The second line is a number denoting the number of times that particular data block is expected to be seen in the dump file.

Newlines are acceptable anywhere in a Search String File and are ignored. Comments are denoted by a hash mark (`#`) at the beginning of a line. Wildcards are supported and are denoted by asterisks. Wildcards will match against any character in the source file, and a line containing all asterisks is valid and will match any line (which is useful for information that may change from run to run, such as a timestamp). A line consisting of "new_cell" is a valid line for comparison since it could match the source input. Note that partial matches are not counted; for instance, if the first two lines of a four-line test block match the last two lines of a dump file, this is not counted as a match.

Finally, while the length of the block must never be less than one, it is supported to search for zero occurrences of a block. This essentially ensures that the specified block *never* appears in the output. However, when searching for zero blocks, the word "zero" must be used in place of the number zero. This has to do with the semantics of PHP; a string consisting of only the number zero is considered a null string and will be filtered out elsewhere in the software.

Following is an example format of a valid search string file:

```
[Output]
20938EA2
*****
00000000
new_cell
03**29*1
00E01932
6
zero
```

```
[Output]
8339AACD
...
```

This will tell the testing program to search for the specified block in the six lines below the [Output] tag and that it should find nine copies of the block in the dump file. The second line of the block will be ignored completely, and three of the characters in the fifth line can be any value.

A.4.2 Grade Definition Files

Grade definition files have an extension of “.gdf” and are used to define the conditions under which a student project is graded. Currently the software looks for a file named “gradedefs.gdf” in the /egroupware directory.

Similarly to Search String Files, each entry in the Grade Definition File consists of a header followed by several lines of information. Comments are denoted by a hashmark (#) and are ignored, as are blank lines. Also similarly to Search String Files, a value of zero must be denoted by the word “zero” and not by the number zero.

The structure of a Grade Definition File is somewhat more complex than a Search String File. Each entry details an association between a Search String File (which must be located in the /egroupware directory), the number of points the problem is worth, the database table and fields to use, and what commands should be performed prior to the sramdump and grading being performed. It can optionally include three lines at the end detailing the memory bank, starting address, and number of words to dump that should be passed to sramdump. If any of these values are defined, all three must be defined, and if these are defined they will override any user-chosen settings on the reserve.php page.

The file containing commands to be performed prior to the sramdump can contain any commands; for this reason the gradedefs.gdf file is located in the eGroupWare base directory (and should have appropriate permissions). Any commands defined in these files will be run with the web server’s permissions, so students should not have write access to either the gradedefs.gdf file or the command files. A command file *must* be defined and must exist for each entry. If no commands are to be run, the file should exist but be empty.

The structure of the Grade Definition File is shown below:

```
[Gradedef]
name of the entry for select box on reserve.php
name of the corresponding .ssf file
name of the table in the database
name of field in table for point scoring
name of field in table for hash of bitfile
number of points for success
file containing commands to run
```

Optional lines that can follow the above:

```
memory bank (1 or 0)
starting address
number of words
```

An example entry is shown below:

```
[Gradedef]
#Comments are OK, as are empty lines
Machine Problem 2
mp2.ssf
#Database
student_grade
mp2
mp2_bitfile_hash
50
testsites.txt
#Going to use the optional values
zero
zero
1000

[Gradedef]
Machine Problem 3
mp3.ssf
...
```

A.5 hardware_logout.php

This simple script has only a few functions: it manually changes the status of the hardware in the resource table to “empty” to make it immediately available for other users; adds a “logout” entry for the current user into the resource journal; it kills any necessary programs that may be still be running from the previous user.

Appendix B

Installing the Tools

A reference on properly installing and configuring eGroupWare is out of the scope of this document. Good references exist, however. The most important is the “Howto Install and Secure eGroupWare” guide written by the lead developer of eGroupWare, at [16].

One “gotcha” that should be noted, however, is that some versions of MySQL, including version 4.1, use a different password scheme that is incompatible with previous versions. To use eGroupWare properly, the MySQL server must use the old password scheme. This is most easily accomplished by passing in the `-old-passwords` directive to `mysqld` on startup, although this has other security implications. See [16] for more in-depth information.

Once eGroupWare has been installed and a domain has been properly set up, follow the list below to get the tools installed. This is a generalized list that does not cover the specifics of any particular implementation of the testing platform.

Note: You should read Appendix C before following these steps. There is important information contained in Appendix C that must be considered when designing your database tables. *In addition, if you are going to control multiple hardware platforms from one Server, you should read Appendix F.*

The first few items should be performed on the Server:

- *Ensure Home is Working:* Ensure that the Home application is installed and enabled for the domain (it is by default).
- *Set up the Symlink:* Make a symlink from `/egroupware` to the eGroupWare base directory. For example, if the base directory is `/srv/www/htdocs/egroupware`, the following command will produce the correct symlink: `cd /; ln -s /srv/www/htdocs/egroupware egroupware`
- *Copy the Tools:* Copy the tool scripts to the `/egroupware` directory. Make sure to set the permissions on these scripts to match the rest of the files in the `/egroupware` directory).
- *Modify Home:* Modify `/egroupware/home.php` to contain a link to your tools. See Appendix D, bullet point “Linking from Home,” for more information.
- *Set up Database:* Create the necessary tables in your database using the command line interface or a tool such as phpMyAdmin. See Appendix E for example tables.

The next set of items should be performed on the Controller:

- *Set up User Account:* Make a user account with the same name as the web server user on the Server. This might be `www`, `wwwrun`, `apache`, `http`, `nobody`, or something else. If possible, make sure that if there is a web server running on the Controller as well (for instance, to act as a content host) that the web servers on the Server and Controller do not share the same user name. The name that Apache is using can be both found and changed from Apache's configuration file. The location of this file is system-dependent. It is possible to use a user account with a different name as the web server user on the Server. However, this would require modifying the scripts on the Server so that any commands passed via `ssh` also contain the appropriate username.
- *Copy the Tools:* Any tool scripts required to be run on the Controller should be copied to the appropriate places. Usually this will be the home directory of the user account created in the step above.

The next item should be performed on both the Server and the Controller:

- *Set Up Communication Between Server and Controller:* Use `ssh-keygen` to generate a public-private key pair on both machines. This is a system-dependent operation; for instance, it depends on whether you are using the OpenSSH or SSH.com implementation of SSH, so consult references on the Internet if you need help with this step.

Apache is generally denied access to a login shell for security reasons, so to run the `ssh-keygen` command on the Server, you will have to be root and use `su` with the `-l` and `-c` switches to run the command in the login context of the web server user. Another, less secure method for doing this is to modify `/etc/passwd` to change the login shell of the web server user from `/bin/false` to `/bin/sh` and then using `su -l` to obtain an interactive login shell. Since the Controller should be using a normal user account for communication, you should be able to run the command directly. (Sometimes this does not immediately allow you to obtain an interactive login shell with `su -l`. Running the command `kill -HUP 1` or rebooting should fix that.)

The public key generated by the Server will need to be copied to the Controller, and vice-versa. The flags necessary to `ssh-keygen` and the locations that the public keys should be copied to, as well as the permissions necessary on the keys and the directories containing them, are highly dependent on both the implementation of SSH as well as the version of the implementation. Therefore, it is not covered here, but should problems arise a Web search should provide all necessary information.

Be sure to test communication between the machines by using `ssh`. You should be able to `ssh` from the Server to the Controller using a command such as:

```
su -l -c "ssh <controlleruser>@<controlleraddress>"
```

and gain a login shell without using a password. To test communication from the Controller to the Server, you can use a command such as:

```
ssh <apacheuser>@<serveraddress> date
```

This should return the date and time of the Server to the local console without requiring a password.

Once these steps are performed, the tools should be fully set-up and ready for use.

Appendix C

Necessary Changes for Tool Re-Use

In this chapter, the changes that would be necessary to reuse the existing tools in different situations are detailed. A few pointers are also given as to how to modify the scripts to support different setups, such as multiple groups of users. Most changes are fairly minimal.

- *Database and Connection:* The database name will likely change from year to year or course to course. Remember that the appropriate tables must exist in each database used, but each domain in eGroupWare has its own database. Therefore, if new classes use different eGroupWare domains, they will be using different databases, and the database information and connection information to the database will have to be updated in the scripts.

A way to get around this limitation is to store the tables for the hardware-sharing scripts in a common database outside of one being used by eGroupWare to store its domain tables. Not only does this allow sharing of hardware by multiple domains of users at once, but so long as that database is used the connection information in the scripts does not need to be changed. This approach is currently being used by the Spring 2005 Digital Systems Laboratory (CSE 465) [17] course, so that the course can share the hardware they are using with other research groups that may have need for it. One note of caution, however, is that the eGroupWare APIs do not allow scripts to retrieve the name of the domain that the user executing the scripts belongs to; therefore to avoid potential problems where two users with the same username are accessing the scripts, ensure that usernames are unique between any eGroupWare domains that use the hardware.

Class-specific information, such as the grades table, could still be stored within that domain's database, and could be accessed using eGroupWare's API (via the "db" object – see eGroupWare developer documentation for more information). This would use the correct database for the current user's domain.

- *eGroupWare Directory:* You need to make a symbolic link named "egroupware" located in the filesystem root and pointing to wherever the eGroupWare directory exists. Presuming that the eGroupWare files are located at /var/www/html/egroupware, this could be accomplished by running the following command in the / (filesystem root) directory:

```
ln -s /var/www/html/egroupware /egroupware
```

This symbolic link should always exist on any Server that this software is set up on, and files placed here are accessible to any domain. Remember that this means that entries in the Grade Definition File, and the default Search String File, are available to all users, and all domains with users that use these features should have the appropriate tables in their databases. As mentioned above, there is currently no way to access the name of a user's domain from within the eGroupWare API; if there was, it would make it trivial to restrict access to certain "common" Search String Files or Grade Definition Files to specific domains.

- *Home Directory:* Because of a long-standing bug in eGroupWare, the API call to return a user's private directory is broken. To get around this bug, for each domain that you have, go to the home directory of that domain (as specified in the eGroupWare's domain setup accessed from, for example, <http://aqua4.arl.wustl.edu/egroupware/setup>). Make a symbolic link named "users" that points to "home" by running the following command in the domain home directory:

```
ln -s users home
```

- *Connection to the Controller:* The address of "192.168.50.9" is hard-coded as the address for the Controller. This would have to be changed if the Controller's address on the network has changed.

Remember that the Controller and the Server communicate by passing commands with SSH. Therefore public keys must be shared between the Server and the Controller, and the username of the webserver process on the Server will have to match the username on the Controller that is being used to execute the commands, unless the *ssh* commands used by the scripts on the Server are modified to compensate.

Keeping the above information in mind, it should be trivial to ready the scripts for different groups of users, even to allow them to be shared by multiple domains of users at once.

Appendix D

Building New Tools

This chapter highlights some of the basic commands that must be in your PHP files to hook them into eGroupWare and into the rest of these tools. Also presented is some code to perform common tasks.

D.1 Necessities

- *Placing Your Tools:* Place your scripts within the base eGroupWare directory (symlinked by /egroupware). If you wish to place them in another directory, you will have to ensure that the paths of all links are correct and that the paths are accessible by the web server.
- *Linking from Home:* Since the tools link into the Home application of eGroupWare (which must be installed and enabled, of course), simply place a link (which can be in the form of a simple HTML hypertext link, an image, or a button) to your PHP file (such as reserve.php) within the main section of eGroupWare's home.php file. A good place to do this is probably directly before the second-to-last line of the file, which calls the phpgw_footer() method. This is a good idea for two reasons. First, while you could link to your PHP file from within any application, the Home application is particularly uncluttered compared to most of the other common ones. Second, placing your link near the bottom of the file instead of in the middle ensures that it does not interfere with any other functions of the Home application.

As an example, the following line of code, inserted into home.php before the phpgw_footer() statement, will generate a link to your reserve.php file from the bottom of the Home application:

```
print '<a href=reserve.php>Reserve</a>';
```

- *Setting the Application:* Your tool *must* include the following two lines, preferably as the first two lines of PHP code and after the opening HTML body tag. These lines allow the script to interact with the current eGroupWare section and display the eGroupWare header, respectively. If your scripts are not located in the base eGroupWare directory, ensure that the second line has the correct path and that the path is accessible by the web server.

```
$GLOBALS['phpgw_info']['flags']['currentapp'] = 'home';
```

```
include('./header.inc.php');
```

- *Providing Appropriate Hardware Control Functions:* Remember that the Server expects to run the “restart_script” command on the Controller, and have the Controller worry about setting up the hardware. This means that you will have to have the restart_script in turn execute other scripts and perform functions as appropriate. In general, most or all of the programs and scripts described in Chapter 5 that are contained on the Controller will not be useful for any other hardware configuration and will need to be replaced.

D.2 Useful Code

Here are some useful code snippets to perform various tasks. The source code itself has code to perform many functions and may be useful as a reference.

- *Getting the eGroupWare User ID:*

```
$UserId = $GLOBALS['phpgw_info']['user']['userid'];
```

- *Getting the User's First Name:*

```
$Username = $GLOBALS['phpgw_info']['user']['firstname'];
```

- *Return a Query from MySQL with the Current Time In Place of the Stored Timestamp:*

```
$sqlstatus = "SELECT now() as timeenow FROM resource";
```

- *Setting a Timestamp Field in MySQL to the Current Time:*

```
$sqlupdate = "UPDATE resource SET LastUserExp=now()";
```

- *Setting a Timestamp Field in MySQL to a Future Time x Minutes Away:*

```
$sqlupdate = "UPDATE resource SET LastUserExp=date_add(now(), interval x MINUTE)";
```

- *Run a Command on the Server (Runs as the Web Server's User):*

```
passthru("perl /egroupware/my_script.pl")
```

- *Open a Directory to Access its files; Access the Files Individually; Print the Filenames:*

```
$bitfiledir = opendir($path)
```

```
while(( $file = readdir($bitfiledir)) != null){ echo $file;}
```

Appendix E

Structure of SQL Tables Used

The structure of the tables used are below. The notes for each table should be read before the table is created.

Table E.1: Structure of *resource* Table

Field	Type	Null	Default	Attributes
ID	smallint(6)	No	(leave blank)	PRIMARY, auto_increment
RName	varchar(30)	No	(see note below)	UNIQUE
Description	text	No	(description)	
Status	varchar(18)	No	empty	INDEX
LastUserID	varchar(50)	No	none	
LastUserExp	datetime	No	0000-00-00 00:00:00	
LastUserPath	varchar(250)	No	none	
LastUserFile	varchar(100)	No	none	

Notes on table “resource”: Entries for this table are not automatically created, and the RName for an entry must be manually filled in. Using the hostname or FQDN of the Controller for that hardware is a good idea for RName, since it allows using the value of the RName returned from a query as a variable in scripts. If you are going to control multiple hardware platforms, see Appendix F for more information on your options.

Table E.2: Structure of *resource_journal* Table

Field	Type	Null	Default	Attributes
ID	bigint(20)	No	(leave blank)	PRIMARY, auto_increment
Status	varchar(18)	No		
UserID	varchar(50)	No		
UserExp	datetime	Yes	NULL	
UserFile	varchar(250)	No		
ActionDone	varchar(38)	No		

Table E.2: Structure of *resource_journal* Table (continued)

Field	Type	Null	Default	Attributes
ActionDateTime	datetime	No	0000-00-00 00:00:00	

Notes on table “resource_journal”: Unlike the “resource” table, all entries in this table are automatically created.

Table E.3: Sample Structure of *student_grade* Table

Field	Type	Null	Default	Attributes
login	varchar(25)	No		PRIMARY
hw1	smallint(6)	Yes	NULL	
hw2	smallint(6)	Yes	NULL	
hw3	smallint(6)	Yes	NULL	
mp0	smallint(6)	Yes	NULL	
mp1	smallint(6)	Yes	NULL	
mp1_bitfile_hash	varchar(40)	Yes	NULL	
mp2	smallint(6)	Yes	NULL	
mp2_bitfile_hash	varchar(40)	Yes	NULL	
midterm	smallint(6)	Yes	NULL	
final	smallint(6)	Yes	NULL	
total	smallint(6)	Yes	NULL	

Notes on table “student_grade”: This table is only an example. The login column should be created initially, and other columns can be added as assignments are added. Also notice the “hash” columns which can be used to store hashes of bitfiles, to ensure that students are not using the same ones for their assignments.

Appendix F

Controlling Multiple Hardware Platforms

The tools were written for a single hardware platform. As shown in Appendix C, it is fairly easy to allow multiple domains of users to share this hardware platform. However, it is also fairly easy to enable multiple hardware platforms to be shared by a single Server. Following are details and considerations to take into account.

- While the “resource” table contains a RName field that should contain a unique identifier for each piece of hardware, the scripts do not actually take this value into consideration. There are two ways, therefore, that multiple-hardware functionality could be added.

Use different databases or tables: One way is to have physically separate databases or database tables for the different hardware resources. Then, either use separate scripts (such as reserve-hardware1.php and reserve-hardware2.php) with different database connection information stored in them, or have the user select which hardware they want to connect to in a drop-down box on the Home application and use the selected value to choose the database connection.

Use the RName field: The other way is to use multiple entries in the “resource” table and differentiate between them by the unique value of the RName field. The SELECT and UPDATE statements can be modified to access specific hardware resources based on the RName value (using standard SQL statements such as SELECT x FROM resource WHERE RName = $'y'$). The RName value could be passed in from the Home application using a drop-down box as described above.

- The “resource_journal” table does not contain a RName column. If it is desired to know which hardware a user tried to grab or program, a RName column should be added and the SQL updates in the script should set this value appropriately.

Both methods for dealing with the “resource” table would benefit from passing in the value of which hardware resource to use from the Home application as opposed to using multiple copies of a script. This not only permits greater flexibility by allowing conditional statements controlling

the behavior of the tools to be based on the this value, but it also provides for greater ease of administration than keeping multiple copies of a script updated.

References

- [1] J. Lockwood, "Reconfigurable System-On-Chip Design." <http://www.arl.wustl.edu/~lockwood/class/cs566-f04/>, Sept. 2004.
- [2] Y. Nakamura, K. Hosokawa, I. Kuroda, K. Yoshikawa, and T. Yoshimura, "A fast hardware/software co-verification method for system-on-a-chip by using a C/C++ simulator and FPGA Emulator with shared register communication," in *Design Automation Conference*, (San Diego, CA), pp. 299–304, June 2004.
- [3] S. Trimberger, D. Carberry, A. Johnson, and J. Wong, "A time-multiplexed FPGA," in *Proceedings of the 5th IEEE Symposium on FPGA-Based Custom Computing Machines*, (Napa Valley, CA), pp. 22–28, April 1997.
- [4] H. Simmler, L. Levinson, and R. Manner, "Multitasking on FPGA coprocessors," *Unpublished*, 2002.
- [5] S. Kelem, "Mapping a real-time video algorithm to a context-switched FPGA," in *Proceedings of the 5th IEEE Symposium on FPGA-Based Custom Computing Machines*, (Napa Valley, CA), pp. 136–237, April 1997.
- [6] T. Chaney, J. A. Fingerhut, M. Flucke, and J. S. Turner, "Design of a gigabit ATM switch," Tech. Rep. WU-CS-96-07, Washington University in Saint Louis, 1996.
- [7] J. W. Lockwood, "Evolvable Internet hardware platforms," in *The Third NASA/DoD Workshop on Evolvable Hardware (EH'2001)*, pp. 271–279, July 2001.
- [8]
- [9] T. Sproull, J. W. Lockwood, and D. E. Taylor, "Control and configuration software for a reconfigurable networking hardware platform," in *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, (Napa, CA), Apr. 2002.
- [10] C. E. Neely, C. K. Zuver, and J. W. Lockwood, "Internet-based tool for system-on-chip project testing and grading," in *International Conference on Microelectronic Systems Education (MSE)*, (Anaheim, CA), June 2003.
- [11] O. M. Beal, "Jammer language description: A script language for gigabit switch testing." ARL Working Note 96-01, 1996.

- [12] R. Jung, “The egroupware project.” Available via Internet at <http://www.egroupware.org>.
- [13] D. V. Schuehler, J. Moscola, and J. W. Lockwood, “Architecture for a hardware-based, TCP/IP content-processing system,” *IEEE Micro*, vol. 24, pp. 62–69, Jan. 2004.
- [14] D. Schuehler and J. Lockwood, “A modular system for FPGA-based TCP flow processing in high-speed networks,” in *14th International Conference on Field Programmable Logic and Applications (FPL)*, (Antwerp, Belgium), pp. 301–310, Aug. 2004.
- [15] D. Suthers and C. Sullivan, “The heyu2 program.” Available via Internet at <http://heyu.tanj.com/heyu2>.
- [16] R. Jung, “How to install and secure egroupware.” Available via Internet at <http://sourceforge.net/projects/egwsec>, 2004.
- [17] J. Lockwood, “Digital Systems Laboratory.” <http://www.arl.wustl.edu/~lockwood/class/cs465-s05/>, Dec. 2004.