A Low-Cost Full-Featured Extensible Laboratory For Online Hardware Engineering

http://dx.doi.org/10.3991/ijoe.v10i3.3517

T.R. Pearson
Raptor Engineering, Belvidere, United States

***Abstract*—This paper describes the uLab, a new method and framework for remote hardware design laboratories, which uses Linux and FOSS to provide real-time design and debug services to students over standard RDP channels. A secure, encrypted, plugin-based remote laboratory framework allows customization of programming and debug/test services to match physical laboratory resources. Industry standard technologies such as LDAP and Kerberos are utilized to ensure scalability, security, and ease of management. Emphasis is placed on direct access to real hardware, with the normal array of simulation tools and design software also being provided. In contrast with many of the remote laboratories currently in existence, this system places strong emphasis on direct, long-duration access to real, physical hardware for non-trivial design and evaluation tasks. In order to achieve this goal, secure, network-enabled hardware "pods" were created from inexpensive COTS components, and a blend of new and existing open-source software was used to connect with the overall laboratory framework. Hardware-design software and tools, including the software for physical hardware access, are preloaded and made available within the desktop session, allowing students to log in and start working almost immediately.**

***Index Terms*—Client-server system, cost effective, engineering education, hardware-access pods, hardware design, online engineering, uLab, Universal Laboratory**

I. INTRODUCTION

A typical hardware design laboratory consists of several workstations and associated hardware in an access-controlled room with rigidly scheduled laboratory dates and times. This laboratory model inherently presents several drawbacks. One of the typically overlooked issues with this type of laboratory is the low average utilization ratio; there normally are large portions of each day when the laboratory is nearly or completely idle with no student access permitted. Another drawback is a relatively short window for laboratory sessions, during which the students are more focused on completing particular assignments within the allotted time frame than they are on learning vital concepts via semi-structured, hands-on interaction with design tools and hardware. Additionally, in many institutions, there are insufficient resources available to handle simultaneous usage by all students within a particular laboratory period; this forces multiple students to be assigned to a given workstation and further removes each student from hands-on interaction with the design software and physical hardware. Remote laboratories, such as the uLab system described in this paper, not only alleviate many of the drawbacks listed above, but also provide exciting new opportunities for students to interact with the laboratory hardware in a non-traditional manner.

In contrast to many of the remote laboratories currently in existence, the uLab places strong emphasis on direct, long-duration access to real, physical hardware for non-trivial design and evaluation tasks. In addition, most of the existing laboratories offering access to real, physical hardware, such as MIT's iLab[1] or the VISIR system[2][3], require expensive, proprietary software packages, such as LabView, in order to function. By contrast, the new uLab laboratory system, which derives its name from the goal of providing a "Universal Laboratory," is not only open-source itself, but also is built entirely upon open-source software and, where possible, open hardware. This frees institutions from the requirement of purchasing expensive software licenses for each new hardware workspace, and allows them to, instead, focus on providing the best possible experience for their students. This characteristic also enables institutions to modify the uLab system to meet their particular needs rather than adjusting their curriculum to work around any limitations present in existing, closed-source software.

The uLab system is comprised of three main components: infrastructure, terminal services, and hardware-access workspaces. The infrastructure component provides relatively mundane but essential services, such as Kerberos authentication, to the other two main components; the infrastructure component, therefore, plays a critical role in the provision of a unified laboratory experience. The terminal services component provides a full-featured remote desktop environment, complete with hardware design and simulation tools, to the end user over a standard Remote Desktop Protocol (RDP) link. These terminal services leverage the provided Kerberos infrastructure to enable single sign-on functionality across all uLab components, presenting a more unified environment to the end user. Finally, the hardware-access workspaces component provides the end user with access to real, physical hardware. This component also leverages the Kerberos infrastructure for authentication and encryption of both client-server and server-server connections, thereby maintaining the security of all hardware-access components. A diagram illustrating the relationships between major components in a typical uLab system is shown in Fig. 1.

II. INFRASTRUCTURE

The infrastructure component is comprised of a Kerberos realm controller or controllers; a large, central disk array; networking hardware; a router/firewall; and various related services. The Heimdal Kerberos realm controller utilized in the uLab system uses OpenLDAP as its directory backend, while OpenLDAP uses Kerberos as its primary authentication system. Due to the high level of difficulty involved in manually configuring Heimdal Kerberos,

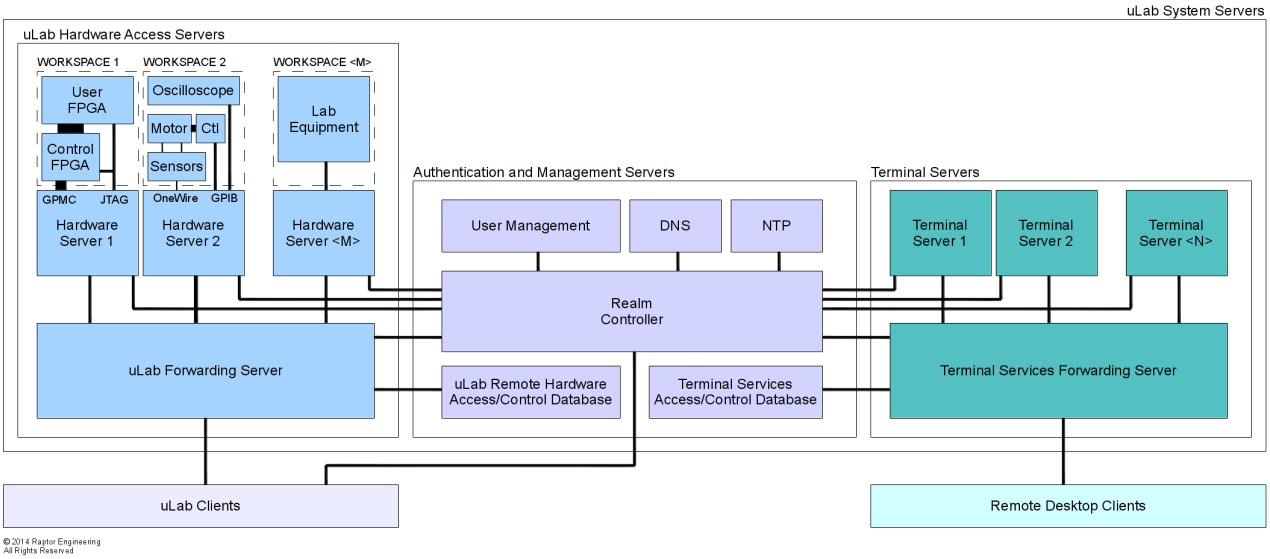

Figure 1. Major components of a generic uLab system

OpenLDAP, SASL, and any necessary cryptographic components, new graphical and command line tools were created to provide easier set up and maintenance of the realm controllers. Central storage is provided by a 500GB RAID 1 disk array, which is then exported to the uLab servers using the Network File System version 3 (NFS v3) protocol. Several independent internal networks are utilized, with a 1Gbps Ethernet switch handling most non-storage traffic, and a 10Gbps Infiniband switch handling disk array access over NFS v3; these networks are fully isolated from any external networks, including the Internet. Internet service is provided though the use of a pfSense router and firewall, which provides appropriately filtered Internet service to dedicated Ethernet ports on the servers though the use of Network Address Translation (NAT). In the current uLab system, a single master server contains the disk array and also provides NFS, Domain Name Service (DNS), Dynamic Host Configuration Protocol (DHCP), Trivial File Transfer Protocol (TFTP), Network Time Protocol (NTP), and MySQL database services to the entire internal network.

The terminal servers and workspace servers, the operation of which will be detailed below, contain no disks; they store their system files on the central disk array and boot via NFS. This configuration allows servers to be easily and quickly replaced in the event of hardware failure; instead of reinstalling and reconfiguring software, the MAC address configured in the DHCP server simply is updated to reflect the MAC address of the replacement hardware. An exception to this rule is found in the workspace servers residing outside the central uLab cluster; these workspace servers use local disk and/or Flash-based storage as appropriate. Each terminal server, hereinafter referred to as a "node," is configured to boot using its built-in Preboot Execution Environment (PXE) client; the PXE client on each server then acquires an Internet Protocol (IP) address from the DHCP server. Since each server has its MAC address preassigned to a specific IP address within the DHCP server's configuration files, each server reliably boots with a known IP address. Once an IP address has been acquired by the PXE client on a particular node, the PXE client proceeds to download the Linux kernel, the initial RAM disk (initrd), and the parameter list for that IP address from the TFTP server into RAM. After download, the Linux kernel is booted; the kernel then mounts the configured NFS root directory to the local root directory and allows the node to finish booting via its standard System V Init system.

III. TERMINAL SERVICES

Unlike most competing remote laboratory solutions, the uLab system does not use a Web-based interface for access to any of its services; instead, it provides a traditional, feature-rich, WIMP-based GUI in order to provide an experience that is as close to real world as possible. This requires a full desktop environment in which the design tools and remote laboratory GUI can be utilized effectively and efficiently. For this reason, terminal services are provided to the students over Microsoft's industry-standard Remote Desktop Protocol (RDP)[4]. Although normal terminal services over RDP--such as those provided by Microsoft or the open-source FreeRDP project[5]--are not easily scalable due to the fact that each client must connect to a given server, providing additional terminal servers to alleviate overcrowding on existing servers brings significant challenges in the form of session consistency, server management, and active session resumption after disconnection.

To provide redundancy and scalability to uLab's terminal services, the xrdp server from the FreeRDP project was modified to support the concept of a central forwarding server. This forwarding server is responsible for initial authentication and selection of an appropriate RDP backend server; each backend server is then responsible for hosting individual terminal sessions. This architecture alleviates the scalability and reliability concerns typically associated with a single terminal server by allowing multiple RDP backend servers to be utilized transparently; in this configuration the system only presents a single terminal services address to its users. Cross-session consistency is ensured through the use of the central disk array, which stores all user data including each users' desktop configuration files. Additionally, each login is recorded in a dedicated database with all of the pertinent information required to reestablish a connection to the backend server if needed. This information includes the username, backend

server, X11 display number, and Process Identifier (PID) of the window manager in control of the session. Upon termination of the window manager, the associated session information is removed from the database, and all daemons and processes related to that session are terminated.

Communication between the RDP forwarding server and the RDP backend servers is accomplished over three separate channels. Control of the RDP backend servers is handled via passwordless secure shell (SSH) commands; the commands originate on the forwarding server in response to login and termination requests, and are sent to the appropriate backend server during session startup and teardown. RDP video and input device actions are handled via an xrdp session stream, with the forwarding server acting as a simple router, ensuring that each stream is routed from the appropriate RDP backend server to its attached remote client. Audio is handled in a third PulseAudio stream. Each terminal session executes an independent PulseAudio server; this server then captures all sounds generated within the active session, and transmits the resultant audio stream to the RDP forwarding daemon for subsequent transmission to the client. While portions of this architecture were present as legacy code within the xrdp project, the original intent of that code appears to have been separation of a single forwarder and single backend server. A significant amount of work on the xrdp codebase was required to make the aforementioned design work reliably and to incorporate much-needed features, such as remotely commanded termination of active sessions and tracking of active-connected versus active-disconnected sessions. The final design of this component is shown in Fig. 2.

Since Linux provides a wide variety of desktop environments to choose from, selection of an appropriate environment for the uLab that both enables complex engineering tasks and functions well over RDP is critical. The two most widely available desktops, as of this writing, are KDE[6] v4.x and Gnome[7] v3.x; however, neither of these is suitable for use over RDP for a number of reasons. Both rely heavily on OpenGL and raw CPU power for "eye candy" (graphical effects that primarily serve to entertain the user); because the terminal servers do not contain graphics cards to enable accelerated OpenGL, severe performance penalties would be incurred by OpenGL's continual use. Although KDE contains an OpenGL-free compatibility mode, it is built on top of the Qt v4.x toolkit, which does not perform well over remote desktop links. Furthermore, both desktops are built around a concept generally referred to as the "semantic desktop"; this concept primarily is designed for personal information management and retrieval. The search and indexing tools on which the semantic desktop is built also require a fair amount of CPU power and memory to function properly, and the desktop environment itself will not function correctly without them. Supporting the maximum number of users on a given terminal services node requires that all sources of replicated bloat, that is, any unneeded CPU and memory usage caused by a single user session, must be carefully controlled. In the author's opinion, neither KDE nor Gnome is a suitable candidate for use with the new remote laboratory system, since both inherently use non-trivial amounts of CPU and memory per session for functions that do not enhance the overall laboratory experience.

Fortunately, there are several less popular, non-semantic, WIMP desktops from which to choose, including LXDE[8], XFCE[9], Cinnamon[10], and TDE[11]. All are reasonable choices for inclusion in a remote laboratory, however TDE was chosen due to both its unique feature set and the author's familiarity with this desktop environment. In the author's opinion, LXDE and XFCE are too light on features to be chosen if another reasonable alternative is present; also, they, together with Cinnamon, suffer from a somewhat complex, confusing programming style and set of Application Programming Interfaces (APIs). Because all three desktops are based on the Gimp Toolkit (GTK)[12], programs written in the competing Qt[13] toolkit do not integrate well with any of these desktop environments, and vice versa. Providing a consistent user experience requires the use of the native programming toolkit of the desktop environment in use; where workarounds do exist for other desktops, they generally consist of complex and largely unsupported pieces of software. By contrast, TDE is built on the older Qt 3.x toolkit, which, despite its age, not only works efficiently over RDP and XDMCP links but also presents a reasonably powerful programming style and set of APIs to the developer, thus making it the best choice for the uLab system. The feasibility of using the uLab desktop even on

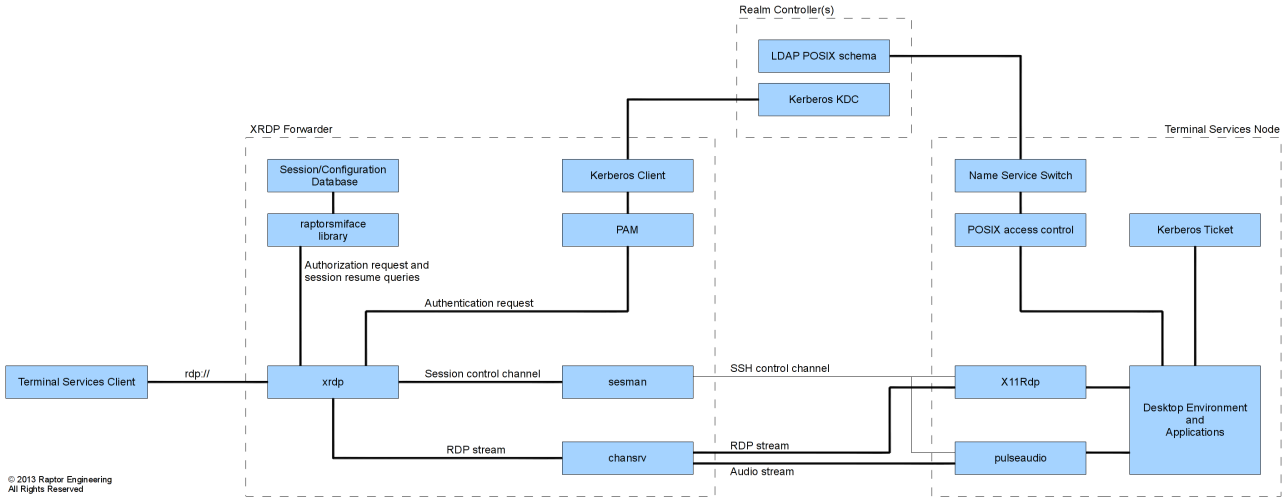

Figure 1. Architecture of the terminal services component

completely closed systems such as the newer Apple iPhone and iPad devices has been verified by the author, although either one will require a keyboard and the iPhone screen generally is too small for comfortable use in this application. The fact that these types of devices are usable means that a student who has access to a tablet type device and compatible keyboard has a third option with which to complete his or her assignments instead of being limited to a more expensive laptop or desktop computer. It is expected that Android devices, which typically enforce fewer software restrictions than comparable Apple devices, will work just as well with the uLab system.

The desktop environment in use is only one consideration in providing a full-featured laboratory workspace for the student. Careful attention also must be given to the applications that are made available to the user; for a computer engineering laboratory to be successful, a wide variety of design tools must be installed. Broad categories include: software development tools, hardware development tools, hardware simulation tools, office and graphics tools, and remote-hardware access tools. In the uLab system, the first category is populated with FOSS programs, such as KDevelop[14], Eclipse[15], gcc[16], and similar utilities, with no need to resort to proprietary or closed-source tools of any type. Hardware development tools are a different matter; in particular, FPGA design tools, such as Xilinx's ISE, are only available as closed-source bundles from the FPGA's manufacturer. However, a few notable exceptions to this general rule exist, such as gEDA[17] and LibreCAD[18]. The new remote laboratory also includes closed-source freeware, such as LASI, an integrated circuit design package. Hardware simulation tools include KPicoSim, a FOSS Picoblaze simulator, and LTSpice, an excellent closed-source, SPICE-based circuit simulator. Office and graphical tools are provided through the comprehensive FOSS LibreOffice suite, with advanced graphics handled through the inclusion of GIMP[19], another excellent FOSS program. Finally, a means of accessing the hardware of the remote laboratory must be provided; in the uLab system, this is handled via the inclusion of the uLab remote client, as detailed below.

IV. HARDWARE-ACCESS WORKSPACES

Because students will be using the hardware-access workspaces as a substitute for direct, physical access to laboratory hardware, this component of the uLab system must expose the functionality of the remote laboratory hardware in an easy-to-use manner. Furthermore, since the hardware-access workspaces cannot be combined into one server, as the terminal services can, it is important to keep the cost of each hardware-access workspace to a minimum. To accomplish this goal, hardware-access "pods" were created using the BeagleBone Black[20] ARM v7 single-board computer (SBC). The pods in the reference uLab implementation are each comprised of a single Beaglebone and two attached Xilinx FPGAs, with the BeagleBone acting as both JTAG programmer and debug interface.

As with the other uLab components, the hardware-access workspaces are integrated with Kerberos to enable transparent authentication and encryption; this not only ensures security between client and server, but it also allows hardware-access pods to be placed at remote locations and accessed safely over public networks, such as the Internet. A central arbiter daemon provides a uLab Kerberos service, and each client (including hardware-interface servers) must present a valid Kerberos ticket on initial connection. Upon successful Kerberos authentication, the communications channel immediately is switched to encrypted mode for security purposes. The lowest levels of this functionality are broken out into a new library (tdekrb), which provides an easy-to-use, frame-based data transfer method to higher-level applications, while transparently handling Kerberos-based authentication and channel encryption. The use of Kerberos tickets in this application ensures that the hardware-access client can utilize the credentials provided on initial login, thus avoiding the need for the user to re-enter login credentials when starting the hardware-access client.

The central arbiter utilizes a MySQL database to store its configuration information. The central arbiter checks each authenticated, incoming service-access request against the permissions database to ensure that only authenticated users are allowed to access the hardware-access daemons for which they have permission. Upon detection of a request to access a disallowed resource, an authorization failure message is sent and the connection is terminated immediately; this happens without establishment of a connection to the requested hardware access daemon. This process effectively prevents an anonymous Distributed Denial of Service (DDoS) attack against the hardware-access servers themselves.

Hardware-access servers utilize the same authentication methods and encrypted links as the hardware-access client; therefore, the hardware-access servers can be placed safely away from the remote laboratory cluster if desired. The central arbiter utilizes a persistent, non-expiring Kerberos ticket to identify itself to each hardware-access server; this prevents a rogue or malicious arbiter from utilizing hardware resources to which it has not been granted access. If a hardware-access server becomes compromised, the damage would be limited to any directly connected hardware and/or the hardware-access server itself, presenting, in the worst case, an effective Denial of Service (DoS) attack against a single laboratory workspace. This design allows inexpensive hardware-interface servers to be placed directly on site, where bulky or sensitive equipment is present. This also could allow laboratory hardware located on the other side of the world to be safely and securely accessed by users of a given remote laboratory cluster.

The hardware-access client is primarily a container application into which GUI client "parts" can be inserted. The container provides a Multiple Document Interface (MDI) container window and status bar; the latter may be changed by the active client part to present informative status messages to the user. Additionally, toolbars and menus are provided from which installed GUI parts may be launched as desired. These toolbar buttons and menu items automatically change, based on the type of remote workspace the user has selected; for example, an FPGA development workspace might show the FPGA programmer and FPGA viewer parts, while a process-control workspace might show sensor plotter and PLC programmer parts. Each part communicates with the central arbiter, utilizing a rigidly defined protocol. If the client part is authorized to use the requested hardware on initial connection, then the central arbiter contacts the appropriate hardware-access server and routes the client connection to that server. This architecture allows multiple, identical

workspaces to be provided within a given laboratory with the system, not the user, deciding which particular workspace will be utilized for a given connection; this type of architecture provides redundancy and masks from the end user any potential hardware-access server failures. A screen shot of a typical FPGA development session, illustrating the use of the FPGA Viewer, FPGA Programmer, and Serial Console is shown in Fig. 3.

The client MDI container handles initial connection to the central arbiter, and on initial connection receives a list of available workspace types to which the user has been granted access. It presents this list to the user; then, after the user selects a workspace type, the client MDI container requests a workspace reservation, matching the selected type, from the central arbiter. If all workstations of that type are in use, the central arbiter will respond with a busy code, and the client will prompt the user to try again later. Otherwise, a reservation for a specific workspace of the selected type is entered into the system, and will remain valid until the initial connection to the arbiter has been terminated, either through a client disconnection or through the action of a laboratory manager. The central arbiter keeps track of these reservations and, on establishment of a connection by a client part, will ensure that the reserved workspace receives the client part's connection request. The client MDI container is designed to take a single command line parameter that specifies the DNS address of the central arbiter of the cluster; this can be used to hide the implementation details from the user and present an "instant-on" hardware-access interface when used with existing Kerberos tickets from the initial login. If this command line parameter is missing, the client will prompt for the address of the central arbiter to which it should connect and authenticate. All of these details are hidden from the client parts, simplifying development of new client parts and ensuring that system security is continuously maintained.

Each client part connects to a specific hardware-access daemon that runs on a hardware-access server. In turn, each hardware-access daemon is assigned a specific port on a given host; this architecture allows one workspace to be assembled from either one server running one or more hardware-access daemons, or from multiple hardware-access servers, each running one or more access daemons. Each access daemon opens a Kerberized server socket and expects connections to be made from the central arbiter; if the provided Kerberos credentials do not match the known arbiter credentials, the connection is immediately terminated. As with the central arbiter, each hardware-access daemon follows a strict protocol for identification and connection setup; after this process is completed, arbitrary data may be transmitted between the client part and the hardware access daemon until the connection is terminated.

The hardware-access system currently includes several client parts and hardware-access daemons that will find use in most electrical engineering laboratories. A GPIB-based oscilloscope and spectrum analyzer part is provided, along with an I2C sensor plotter part. All three parts allow export of captured data to external files for later analysis, and import of external data files for viewing. FPGA access is included via two parts: one for programming the FPGA and one for interacting with the design after it has been loaded into the laboratory hardware. A serial console part also is provided for low-speed communication with the user's design. On the management end, two parts have been provided: one to manage authenticated user access permissions, and one to view and control both active terminal service and active workspace users. The latter part, in particular, allows a laboratory manager to set session timeouts and even disconnect users who have, for example, been idle for too long or who might not be using laboratory resources in accordance with an institution-specific acceptable use policy. Following UNIX tradition, each part has been designed to do one thing and to do it well. This separation of duties not only allows the maintainer of each part to stay within his or her areas of expertise, but also has the effect of dividing loosely related functions into separate GUI windows. For example, the FPGA programmer is separate from the FPGA viewer, allowing the FPGA programmer to be minimized or obscured when testing the FPGA, and vice versa. Alternatively, on large screens, both may be visible at the same

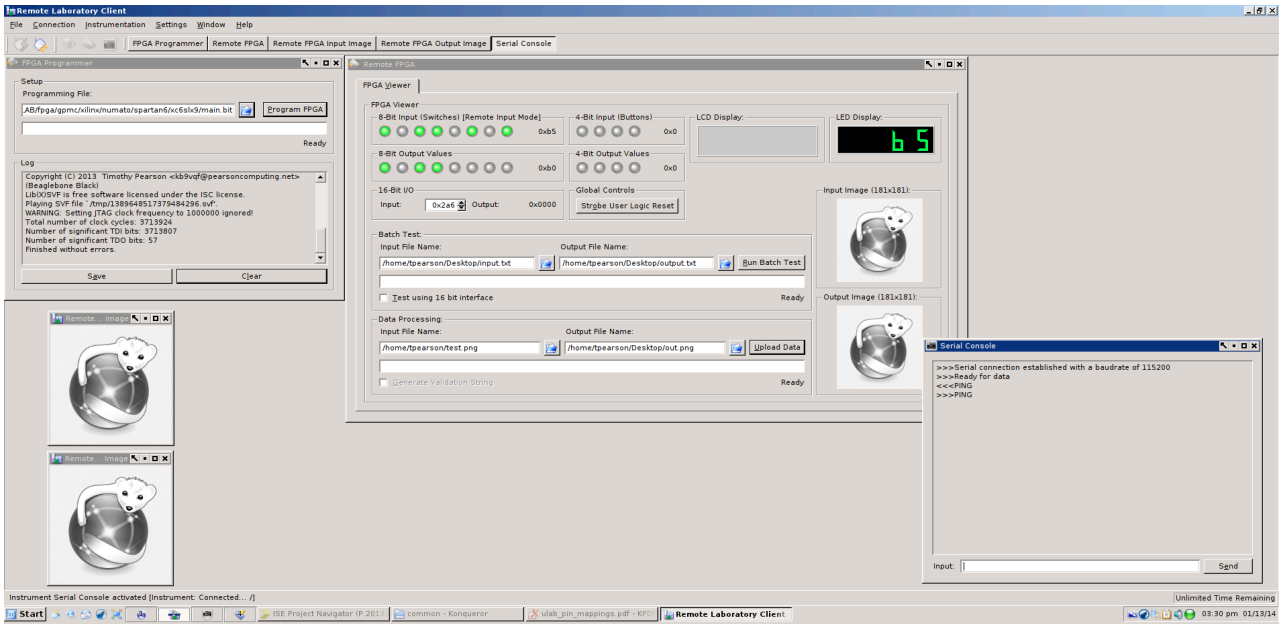

Figure 2. A typical FPGA development session illustrating use of several different "parts"

time, with the user choosing where each part should be located on his or her screen for maximum usability and efficiency. By following this UNIX tradition, the user is granted more control over how he or she sets up and uses his or her workspace.

The GPIB interface part, unlike its distant predecessor utilized in the RemoteFPGA system[21], handles all display and processing of the raw instrument data on the client end. This increases responsiveness and enables real-time operation; unlike the older system, which captured raw screen shots of the instrument displays, the uLab system essentially comprises a complete instrumentation front end, similar to the interfaces that have been integrated with stand-alone test equipment since the beginning of the digital test equipment era. This allows the backend test equipment in use to be fully abstracted from the user; aside from various hardware-driven specifications, such as number of traces and bandwidth, each major type of test equipment added to the system will present the same generic graphical interface to the user. This also enables the possibility of using "headless" test equipment, such as some of the more recent PC-based oscilloscopes that do not contain a display or physical keypad, in the laboratory to reduce overall cost. In addition, it also enables the repurposing of specialized hardware for more general purposes without the added complexity this often brings; for example, the spectrum analyzer server included in the uLab software package interfaces with an Agilent CDMA test set, allowing use of the spectrum analyzer functionality buried within that specialized equipment without requiring the user to first understand how to operate the basic functions of the test set.

It should be noted that the general principles discussed herein are not only applicable to GPIB; the principles documented above are applicable to newer interfaces as well. As long as the manufacturer provides a programming reference manual that is not protected by a non-disclosure agreement (NDA) or similar legal instrument, new backend server daemons can be written to interface with test equipment over almost any hardware interface. Special effort was made to ensure that the test equipment client parts use a generic protocol that should be applicable to all test equipment within a particular class; therefore, for example, the oscilloscope client part and protocol should be usable with any type of oscilloscope backend interface daemon. GPIB interfaces were implemented primarily because of the low cost of GPIB-enabled test equipment, the fact that a GPIB interface is all that typically is required to obtain adequate functionality from oscilloscopes and other signal analyzers, and the fact that most GPIB-enabled equipment manufacturers freely provide protocol documentation for those instruments. The author recommends that only GPIB, VXI-11, or similarly fully open and documented equipment be utilized in new remote access laboratories because this will ensure that support for the resultant laboratory can be maintained even if the original test equipment interface becomes unusable.

Because the reference uLab installation at Raptor Engineering initially will be used for FPGA design, the architecture of the FPGA viewer part will be examined. The new FPGA viewer is a full rewrite with improvements of the original FPGA remote access solution created by the author and deployed in 2009 at Northern Illinois University (NIU)[22]. The new system uses a hardware interface module, contained within a dedicated control FPGA, that interacts with a given set of signals connected to the user FPGA. This interface module then transmits current signal levels to the hardware-access server via a high-speed, memory-mapped interface while allowing a second set of signals to be controlled by the hardware-access server via the same memory-mapped interface. The hardware-access server translates the raw signal values to a basic command-oriented protocol, then sends the translated data across the network to the client part, which interprets the data and displays the current status of the virtual lights, switches, and displays. The interface module also includes the ability to read and write to either internal block RAM or external RAM; when interfaced with both the client part and the user FPGA, this allows block-data-based algorithms, such as image processing, to be implemented on the user FPGA, then to be tested easily using the FPGA viewer client part. A final feature, implemented within the client part itself, is the ability to run a batch test and record the results. This batch-test feature takes a list of inputs from a simple text file, sequentially applies them to the 8-bit data bus, and records the results in a second text file.

V. CURRENT STATUS AND FUTURE WORK

Work is ongoing to make the uLab even better by adding new features and expanding the number of supported devices. Solicitation of student feedback has led to the split of the control and user FPGAs into two separate devices; this allows a student to retain full control of the DUT, including generation of an appropriate implementation constraints file specific to his or her design. This new architecture also will enable the rapid integration of other types of devices, such as microprocessors, to the uLab system. The user FPGA is now forcibly reset immediately upon client disconnection in order to better avoid the slight but non-negligible chance of hardware damage by a malicious student, and an additional user logic reset signal is provided for student use. The FPGA devices currently in use at Raptor Engineering are fully open hardware designs, produced by Numato, in keeping with the stated goals of the uLab project. Users now have access to a large, dedicated DDR RAM device for advanced data processing in addition to the emulated SRAM device provided by the control FPGA. As of this writing, a built-in 64-channel logic analyzer is in the final stages of development; it is hoped that this instrument will help to demystify high-speed digital interfaces for intermediate hardware design students. By making the uLab more instrumentation-rich than the small development kits typically used by engineering students--while keeping costs to a minimum though open design--the degree of learning attainable in a typical hardware design course should increase while the cost of the same course per student decreases.

Because the uLab makes extensive use of encrypted network communication, it is possible to safely and seamlessly create workspaces comprised of hardware devices located in different rooms, buildings, or even countries. This offers an excellent opportunity for worldwide collaboration on a single project; for example, one institution offers hardware resources while another offers intellectual property and/or experience with the technology being developed. Similarly, development hardware in one country can be made available to students in another where the hardware in question may not be affordable or widely available. This concept also is extensible to the terminal

services themselves; provided that sufficiently fast network connections are available between the disparate physical locations, there is no requirement that all the equipment be installed in a single location. This functionality is particularly useful when the DUT is located in an area that cannot support a standard development computer due to harsh environmental conditions or a lack of physical space.

When deploying a large system such as the uLab, it is important to identify and address potential bottlenecks and single points of failure. The main bottleneck in a typical uLab setup is the limited bandwidth of the Internet connection between the master server(s) and the remote desktop clients. It is important, therefore, to secure an Internet connection of the highest speed possible for the master server(s), as well as to selectively fall back to executing the remote hardware access client on local PCs if a suitably fast connection is not available to one or more users. The central disk array and master server both present single points of failure that must be addressed. The disk array uses btrfs and can make use of redundant storage pools spread across multiple machines in a Storage Area Network (SAN). Similarly, the master server can be duplicated across multiple machines in a fail-over configuration, reducing the impact of a potential master server failure--from complete inaccessibility of the uLab cluster to a nuisance disconnection of users on a failed server. If multiple physical server locations are in use, performance can be enhanced by selectively routing incoming connections to the closest available master server.

VI. CONCLUSION

In this paper, a new method and framework for a full featured, cost-effective, extensible remote laboratory has been introduced. Several current remote access laboratories were identified, along with their major limitations. The three main components of the new uLab system were discussed, and the concept of self-contained, network-attached hardware-access "pods" was introduced. A modified open-source terminal services framework and its usage was described, along with a new, secure, hardware-access workspace architecture. Requisite laboratory infrastructure was discussed, and novel uses of this new architecture and framework were highlighted. Finally, several specific hardware-access implementations were described from both client and server perspectives.

Because a major goal of this research not only is to prove that a fully open-source remote laboratory can be built, but also to ensure the sustainability of such an environment for multiple institutions to use, all source code written for the uLab system has been released in a set of Git trees. At the time of this writing, the uLab system uses donated space in the TDE project's infrastructure for bug tracking and patch submission. Project code repositories also are available at http://ulab.trinitydesktop.org; authoritative protocol documentation for developers is included in a text file within the hardware-access package Git tree. In addition, a highly condensed set of installation instructions for a lab using the design detailed herein is available at the same location. These instructions are designed to be read and understood by a Linux system administrator or similarly qualified individual, and assume familiarity with UNIX-like systems, the Linux command line, and various configuration files for several software packages. Finally, the aforementioned Kerberos realm setup and management tools have been integrated into the TDE project at http://www.trinitydesktop.org, and are, therefore, available in both source and binary form for TDE R14.0.0 and above.

REFERENCES

[1] V. J. Harward, J. A. del Alamo, S. R. Lerman, P. H. Bailey, J. Carpenter, K. DeLong, C. Felknor, J. Hardison, B. Harrison, I. Jabbour, P. D. Long, Tingting Mao, L. Naamani, J. Northridge, M. Schulz, D. Talavera, C. Varadharajan, Shaomin Wang, K. Yehia, R. Zbib, and D. Zych, "The iLab Shared Architecture: A Web Services Infrastructure to Build Communities of Internet Accessible Laboratories," Proceedings of the IEEE, vol. 96, no. 6, pp. 931–950, Jun. 2008. http://dx.doi.org/10.1109/JPROC.2008.921607

[2] D. G. Zutin, M. E. Auer, and I. Gustavsson, "A VISIR lab server for the iLab Shared Architecture," presented at the Global Engineering Education Conference, 2011.

[3] M. Tawfik, E. Sancristobal, S. Martin, C. Gil, A. Pesquera, P. Losada, G. Diaz, J. Peire, M. Castro, J. Garcia-Zubia, U. Hernandez, P. Orduna, I. Angulo, M. C. C. Lobo, M. A. Marques, M. C. Viegas, and G. R. Alves, "VISIR deployment in undergraduate engineering practices," presented at the Frontiers in Education Conference, 2011.

[4] "Understanding the Remote Desktop Protocol (RDP)." [Online]. Available: http://support.microsoft.com/kb/186607.

[5] "FreeRDP." [Online]. Available: http://www.freerdp.com/.

[6] "KDE - Experience Freedom!" [Online]. Available: http://www.kde.org/.

[7] "GNOME." [Online]. Available: http://www.gnome.org/.

[8] "LXDE.org | Lightweight X11 Desktop Environment." [Online]. Available: http://lxde.org/.

[9] "Xfce Desktop Environment." [Online]. Available: http://www.xfce.org/.

[10] "Cinnamon." [Online]. Available: http://cinnamon.linuxmint.com/.

[11] "Trinity Desktop Environment." [Online]. Available: http://www.trinitydesktop.org/.

[12] "The GTK+ Project." [Online]. Available: http://www.gtk.org/.

[13] "Qt Project." [Online]. Available: http://qt-project.org/.

[14] "Welcome to KDevelop.org | KDevelop." .

[15] "Eclipse - The Eclipse Foundation open source community website." [Online]. Available: http://www.eclipse.org/.

[16] "GCC, the GNU Compiler Collection." [Online]. Available: http://gcc.gnu.org/.

[17] "gplEDA: All things GPL and EDA." [Online]. Available: http://www.gpleda.org/.

[18] "Home of LibreCAD, 2D-CAD." [Online]. Available: http://librecad.org/cms/home.html.

[19] "GIMP - The GNU Image Manipulation Program." [Online]. Available: http://www.gimp.org/.

[20] "BeagleBoard.org - BeagleBone Black." [Online]. Available: http://beagleboard.org/Products/BeagleBone+Black.

[21] R. Hashemian and T. Pearson, "Teaching Hardware Design with Online Laboratories," in Internet Accessible Remote Laboratories, A. K. M. Azad, M. E. Auer, and V. J. Harward, Eds. IGI Global, 2011.

[22] R. Hashemian and T. R. Pearson, "A low-cost server-client methodology for remote laboratory access for hardware design," presented at the Frontiers in Education Conference, 2009.

AUTHOR

T. R. Pearson is the founder and owner of Raptor Engineering, Belvidere, IL 61008 USA (e-mail: tpearson@raptorengineeringinc.com). He has been working with open source software and online engineering laboratories for over 5 years, and has extensive experience with FPGAs and high speed digital/analog systems. He is also an active member of the open-source community, and currently leads the Trinity Desktop Environment and uLab projects.

Submitted 15 January 2014. Published as re-submitted by the author 28 April 2014.