Final Project Report: Atmel Webserver

prepared for Professor Bruce Land
School of Electrical and Computer Engineering

by Eric Mesa (erm26)
School of Electrical and Computer Engineering
and
Richard West (rw88)
School of Electrical and Computer Engineering

submitted on 28 April 2005

Table of Contents

Introduction

Sound Bite

Our project is a fully web-standards compliant webserver running on an Atmel Mega32.

Summary

Using code from graduate student Jeremy Tzeming Tan, we improved upon his webserver source code by making it more fully compliant with Internet standards. Additionally, we modularized his code for easier maintenance, documentation, and implentation. By splitting up the code, instead of keeping all 2000 lines in one file, someone who wishes to maintain a specific function wouldn't have to hunt for it throughout the file. Documentation of each of the files is made easier by splitting them up because a lot more documentation can exist at the top of each of the files. Finally, we have, therefore, built up a library of Internet protocols. Someone else may take our library files and implent only those protocols which are important to that specific project instead of having to implement a full-blown webserver.

High Level Design

Rationale and Source of Project Idea

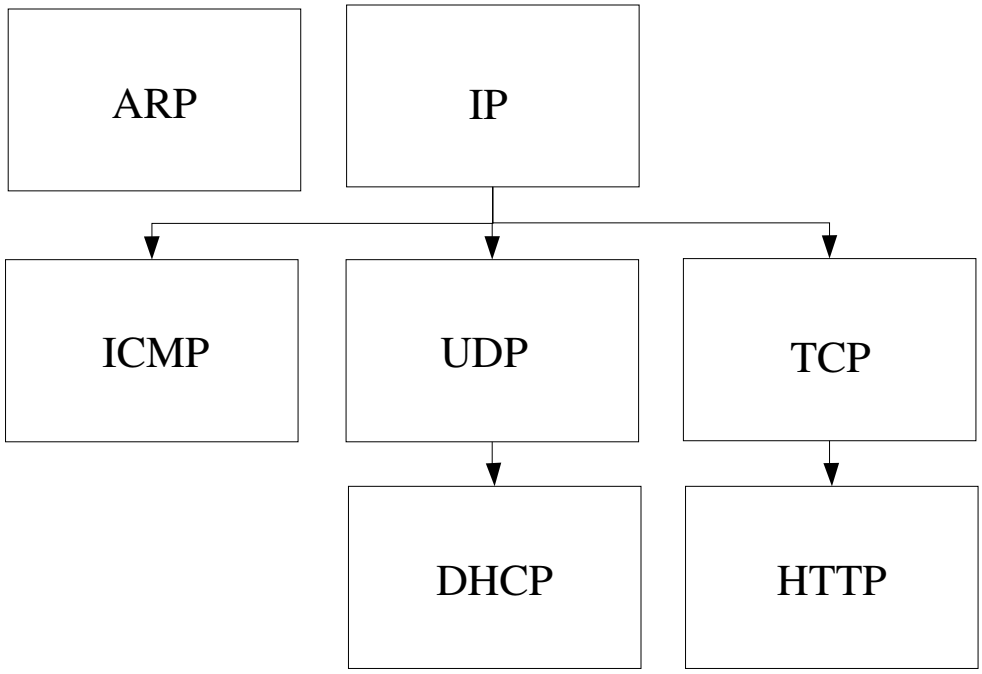
When blade servers exist, capable of serving thousands of complicated web pages, why bother building a webserver on an Atmel Mega32 16 megahertz microcontroller? We each saw a different vision as to how this webserver might serve a useful purpose. In more of a traditional microcontroller approach, Richard saw the project as being useful as an interface to control household appliances or systems. In other words, by using Internet protocols, a home owner could interface with his household systems and control the lights, security system, or certain smart appliances. While our project would require more development to serve this purpose, we have certainly laid the groundwork for such a system.

Eric saw the project as potentially serving another purpose. Since the final server was just around $50 while a blade server is in the tens of thousands of dollars, Eric saw the webserver as a potential playing field flattener for developing countries. The last decade and a half has seen an increase in the number of developing countries desiring to obtain an internet presence as soon as possible. They know that an internet presence is more important than their industrial capabilities. However, the cost of a

professional grade server makes it out of their reach. With the Mega32, they would have the ability to serve up web pages for the small price of $50 for the parts and whatever a connection to the Internet costs in their country.

Logical Structure

Internet Protocol Hierarchy



Hardware and Sotware Tradeoffs

Due to the nature of our project, there were no hardware and software tradeoffs. The software could not perform the actions of the packet wacker and the packet wacker could not perform the requirements of the Internet Protocols.

Standards Governing Project

All internet standards are documented in a series of RFCs (available from http://www.rfc-editor.org/rfcxx00.html) dating as far back as the early days of DARPA. Each RFC outlines the standard for a specific internet protocol. The RFCs that apply to this webserver are: RFC768 (UDP), RFC791 (IP), RFC792 (ICMP), RFC793 (TCP), RFC826 (ARP), RFC2131 (DHCP), and RFC1945 (HTTP/1.0).

Existing Copyrights, Patents and Trademarks

As the Internet runs on open source protocols, there are no patents or trademarks affecting the protocols we are using for our webserver. As a published piece of software Jeremy Tzeming Tan does own the copyright to his source code, from which we built our source code. We have assumed an implied right to use this source code as it was provided freely on the Internet without explicit copyright notice or explicit restrictions upon its use.

Program and Hardware Design

Program Details

As mentioned above, we took Jeremy's design and split it up into a few libraries of functions. We also changed the HTML parsing code to make it more compliant as well as able to serve up multiple files. Although we feel it would have been very hard to have written the code from scratch, using Jeremy's code without knowing the thought process behind most of his decisions made working with his code very tricky.

The first tricky aspect was figuring out why the page would only load in Internet Explorer. Greg Roth suggested diagnosing the problem by using a telnet connection instead of trying to pull up the page in a web browser. This turned out to be our strategy for testing throughout most of the rest of our modifications to the code. Telnet, as the reader may know, displays the raw HTML and HTTP code of the page without any formatting. We were able to see that part of the page's header was not being transmitted. The first part of the first line was " OK" instead of "HTTP/1.0 200 OK". A whole twelve characters were missing! Therefore, the reason the other web broswers weren't working is because the page being transmitted did not meet the standards requirements. Internet Explorer, in Microsoft tradition, was ignoring the fact that the page was not compliant. After many hours of exploration we

figured out what the problem was. Jeremy's original code was putting the page into the part of the TCP protocol reserved for options, instead of the data section. Once we modified this we were finally getting the beginning of the page. However, the page was now terminating prematurely. Although we were unable to figure out the root cause, by simply adding twelve to one of our counters, we were able to consistently serve up the entire page.

In order to make the server even more completely compliant, we added in error messages for users who asked for a nonexistant page or attempted to use a non-supported protocol.

Our next tricky aspect was adding in support for multiple pages. The code worked perfectly independent of the microcontroller. However, when we added it in and tested it in Firefox it refused to work. Now it was time to use the Ethereal packet sniffer program to determine the cause of the errors. It turned out that the browser was requesting favicon.ico, the icon shown in the bookmarks page of Internet Explorer and in near the URL in Firefox, Opera, and Netscape. We quickly put together an icon to use for the server. We used a hex editor to read the hex values of the contents of the file. We typed this into our structure, but it didn't display the icon we had designed. However, it was good enough for the browser to stop complaining.

Hardware Details

We connected our STK500 to an EDTP packet wacker we had sitting on a bread board. The specific connections are mentioned in the code. Also see Jeremy's original report for a detailed connection schematic. The link can be found in the following section.

Base Source Code

Our base code came, as mentioned above, from code written by Jeremy Tzeming Tan. We maintained the basic structure he had set up, but have rewritten key parts of the code for greater compliance with Internet protocols. His original code may be found at http://instruct1.cit.cornell.edu//courses/eceprojectsland/STUDENTPROJ/2003to2004/tt82/index.html .

What did not work

Despite our best efforts, we were unable to get the server to continue working after a few minutes of operation. It would simply stop responding to requests for packets after a certain amount of time had elapsed. We searched through every timing or timeout related function and line without coming across the true reason for the timeouts.

Results of the Design

Speed of Execution

When the server is correctly operating, it serves up pages as fast as static pages are loaded from a normal high-end server. The pages basically appear instantaneously. The biggest wait time comes when the server is first reset or flashed. The server needs about two to five seconds to request and obtain an IP address from the router.

Accuracy

Unfortunately, we were unable to get the server to operate in an accurate manner. Sometimes the server would be usable immediately after being flashed and sometimes it would need to be reset a number of times before it would work. Once the server was working, the user could go click on different links to pages within the server for a random amount of times before ceasing to work. For simplicity we had index.html link to about.html in both directions. We would click on the link to about.html from index.html, wait a few seconds and then click on the linked to index.html. At times we could do this about twenty times before the server would stop responding. Other times it would stop working after three clickthroughs. In either case, if the server was left for a while without use it would also stop taking requests.

We were not able to determine the causes of these errors, but we have a few postulations. With respect to the clickthrough problems, we feel that the buffer of the packet wacker was getting trashed. Since the server was implemented to either send a packet or receive one, but never both at once, the buffer ended up with garbage and incomplete code. Implementing the capability to do this was beyond the scope of our project within the constraints of this semester. We feel it would have required a recoding of nearly 90% of the source.

We feel the reason for the the server ceasing to function after a certain amount of time has to do with a timeout or timing scheme in Jeremy's code which we could not find despite many attempts. It is possible that there are many root causes which combine to make this effect. Again, we were not able to fix this before the end of the semester.

Enforcement of Safety

Safety was not a true concern with our project. It has no moving parts or sharp edges other than the edges of the STK500 and bread boards.

Interferance with other people's designs

Our project did not interfere with anyone else's designs. We ran our project on a closed local area network involving Eric's laptop and router and the webserver. We did not have any kind of radio frequency transmision. The only interferance our design introduced was the inevitable interferance produced by the microcontroller and packet wacker being turned on. However, it was the responsibility of the manufacturers of these chips to ensure they complied with FCC standards.

Usability

Our final project was usable by us very easily as we both understood the functionality of the server. Other outside our group would be able to use the server easily as well as long as they understood some of the intricacies. For example, upon turning on or resetting the server, it takes approximately fifteen seconds before the server can be used. Also, it seems to be most reliable after a reset, when compared to turning it on or flashing it. Additionally, in order to use the server as a dynamic server with ever-changing pages, the user would have to have access to a computer and releated equipment for flashing the server.

Conclusions

Analysis

Our design met its specifications in many of the ways we intended for it to do so. First of all, the code is now cleaner as a result of the migration of the source code into library. Second, one of our major goals was to ensure that our server interfaced properly with broswers other than Internet Exploror, which we were able to achieve. Finally, the code is now much more compliant with Internet Protocols, one of the most important changes we made to Jeremy's code.

Conformation to Standards

The webserver meets minimum standards as outlined by all the RFCs except for ICMP and TCP. ICMP is being disabled by many network administrators due to security vunerablities, and so the only use of ICMP on the webserver is for pings; this hasn't changed from the supplied code. TCP is designed to handle multiple connections through use of a TCP stack, but the webserver does not have the memory capacity to implement a complete stack; again, this hasn't changed from the supplied code. However, the webserver has had its TCP implementation changed slightly in order to bring it within the minimum spec and to reduce errors on the client-side.

The webserver's HTTP implementation has been improved in order to accomodate all possible HTTP requests. Since the server only accepts a GET request, a '501 Not Implemented' error is returned to client if a correct, non-supported request is received. If the request is malformed, the server returns a '400 Bad Request' error in compliance with the RFC. A '404 Not Found' error was implemented once the HTTP implementation was upgraded in order to accommodate mulitple pages. Numerous other error codes are listed within the HTTP RFC that were not implemented by the webserver, but these error codes apply to specific events the webserver will never encounter (nor would any general purpose webserver).

Intellectual Property Considerations

So that others may benefit from what Rich and I have learned and assembled, we will be licensing our source code under the GNU Public License version 2 (GPLv2). This license enables other to take our code and improve upon it as long as they provide credit to Jeremy, Richard, and I as the originators of the version which will be made public. Any licensees must also agree to provide any improvements

they make under the GPLv2 as well. This ensures that any improvments will find themselves back into the public domain. Finally, by using this license we indemnify ourselves from any liabilities others may encounter from using or modifying our code.

Ethical Considerations

By releasing this report under the Creative Commons License and releasing the code under the GPLv2, we are acting within code of ethics item number five, by improving the understanding of technology. This will also increase the welfare of of the public by bringing server technology to those who normally wouldn't have access to it, fulfilling item number one. We were also very realistic with the claims of our technological limits, item number three. We mentioned all of the problems our server has in dealing with too many packet requests. During our development phase we sought out criticism from Greg and others who knew about the protocols we were using, item number seven. We also rejected bribery in all its forms while working on this project, item number four.

Legal Considerations

Anyone who uses our server must be sure they only use the server on a network for which they have the right to do so. The server must not be used if the ISP's terms of service explicitly states that no servers may be run out of a residential location.

Appendix A – Commented Code

ack.c

ack.h

arp.c

arp.h

atmelwebserver.c

dhcp.c

dhcp.h

echo.c

echo.h

icmp.c

icmp.h

ipad.c

ipad.h

nic.c

nic.h

ringbuff.c

ringbuff.h

ring.c

ring.h

tcp.c

tcp.h

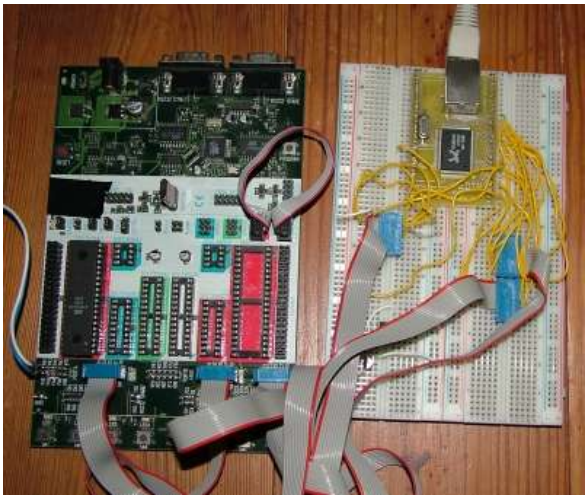
tcpsend.c

tcpsend.h

udp.c

udp.h

Appendix B – Schematics



Appendix C – Cost Details

STK500 $15

Mega32 $8

Custom PC board $5 (for future improvements)

Packetwacker $0 (part of prototype)

Protoboards (2x) $0 (part of prototype)

Total: $28

Appendix D – Specific Task Details

Eric Mesa

- copy entire original source code from Jeremy's PDF into C editor (15 total hours!)
- move all of the functions into library files
- run the ethereal tests on the network
- write 80% report
- debug

Richard West

- redo HTML sending code files for compliance
- redo TCP send fucntion for greater compliance
- create compliante HTTP request parser
- write 20% report, including creating diagrams and schematics where necessary
- debug

We both worked very hard on this code and no one person slacked off. We were both in lab in every

available lab period testing and debugging our code together.

References

Our references included Jeremy's code, RFC standards web page and the Wikipedia entry for Internet Protocol.