

## Project Final Report

### Embedded Web Server with SD/MMC File Storage

24 April 2007

Andrew J. King <andrewjk@gwu.edu>

CSCI 297 – Real-Time Embedded Systems – Prof. Eisenreich  
George Washington University, Spring 2007

#### Project Abstract

The goal of this project was to convert the Z8 Encore platform into an embedded web server. For this proof of concept, simple text-only HTML files were stored on an attached SD/MMC card and served over a network connection via the HyperText Transfer Protocol (HTTP). The network interface was provided by the addition of a NICHolas network interface board from EDTP. The SD/MMC card interface was provided by the addition of an SD/MMC breakout board from SparkFun. Once assigned a static IP address, the embedded web server was able to continuously serve the HTML web pages provided on the SD card to any host on the network. For this project, not only was a web server program developed, but driver support for both the network interface card and the SD/MMC specifications were added to the Z8 platform. Communication between the Z8 Encore and the EDTP NICHolas was accomplished through publicly available driver source code that was modified to provide a TCP connection server on port 80 for HTTP. Communication between the Z8 Encore and the SD/MMC card was accomplished by using the Z8's Serial Peripheral Interface (SPI) support, and by utilization of not only previously written SPI library code for the class, but also publicly available library code for interaction with a FAT16 formatted filesystem.

#### Project Status

The embedded web server project was completed and deemed a successful proof-of-concept, but with some lessons learned. In order to connect both peripheral devices simultaneously to the Z8 Encore, a port connection for the network card had to be relocated to an unused GPIO port, because the port used by default in the network card driver code is the only GPIO port that can be used alternately for SPI communication. Hence, since SPI was a requirement for SD/MMC card communication, the network driver code needed to be altered to use another GPIO port for its data bus. This change was easily implemented due to the heavy use of macros in the network driver code, and only required changing several lines.

Since the network driver code was written with support for an elementary TCP connection server, changes to it were known to be necessary to implement a HTTP server. Namely, these changes involved removing code for supporting the "Telnet-like" server present, and the addition of code relevant to the support of HTTP requests and responses. To resolve memory issues later discovered during debugging, constant data strings were relocated to ROM to free up RAM for buffers used during execution. A limitation in the TCP packet handling procedures in the network driver code was discovered, and prevented the transmission or serving of HTML resources greater than the packet buffer size (512 bytes). This limitation was due to the packet handling procedures being written only to respond to TCP packets received, and no ability to either initiate or terminate connections or send additional packets in already established TCP connections. This prevented the ability of the

HTTP server portion of the code to send resources larger than a single packet (maximum 512 bytes), since it could not send multiple packets. The inability for the server to terminate the network connection also was visible in the testing web browser as a “busy” state.

These limitations could be resolved to develop a more robust HTTP server, but would involve significant reimplementing of the network driver code, and was considered outside the scope of this project. So while both the external devices, the network interface and the SD/MMC card slot, were implemented and worked in tandem, support of both forced artificial limitations on the performance and memory utilization of the Z8 Encore platform.

## Specifications

### Materials List

- Hardware
  - Zilog Z8F6403 Encore Development Board
  - EDTP NICHolas Ethernet Interface
  - SparkFun SD/MMC Breakout Board
  - SD/MMC Card, formatted with a FAT16 filesystem
  - Additional wires for connections
  - Test Network Equipment
    - Ethernet Hub
    - Laptop (MacBook Pro) with web browser (Firefox)
    - Two Ethernet Cables
- Software
  - Zilog Developer Studio II (ZDS II) IDE v4.10.1
  - EDTP Z8 ASIX Code
  - SparkFun Logomatic Code

### Hardware

The microcontroller platform selected, the Z8 Encore, was selected based on the fact that it was provided for the class. However, since it was a development platform, it contained more than enough supporting peripherals to meet the requirements of the project. The SD/MMC card only has the requirement of a Serial Peripheral Interface (SPI) bus, and the Z8 can act as an SPI bus master with a small software library. To operate the EDTP NICHolas network interface, enough GPIO pins were necessary to support a data bus, and addressing bus, and several other individual GPIO pins for input/output control, device resets, and catching interrupts. There were more than enough GPIO pins on the Z8 Encore development board for this purpose.

The EDTP NICHolas network interface was selected based on its availability for the class, and also its usage in prior class projects. Previously, the EDTP NICHolas card was used by a student and their wiring configuration was generously donated to future students of the class. This, in conjunction with their documentation provided an expedited wiring hookup of the interface. Network driver code for the Z8 is available on the EDTP website. This device is moderately priced at \$30 and can be obtained through the EDTP website store.

The SparkFun SD/MMC Breakout board was selected primarily for its low cost and ability to provide pin access to the SD/MMC card easily. SparkFun sells the card for \$14.95, but are sometimes sold out due to their popularity. A pin leader or individual wires can be soldered onto the breakout board to provide the wiring connections. Only a total of six pins are necessary to interact with the SD/MMC card over an SPI bus.

## Software

Since the hardware peripherals were being used in fairly standard ways in this project, sample code was available to be used to verify the functionality of the equipment, and eventually tailored to the specific end goal of the project. Overall, development time was shortened because of this code reuse, however there was some initial time required to be taken to fully understand the APIs and modules provided.

The EDTP NICHolas network driver code was an extraordinarily fortunate find, because it was written for the specific embedded development platform selected (Z8). This was a positive because the driver code would not have to be ported to the Z8 platform. The network driver code relies heavily on macros (both its own and those of the Z8 IDE) and as such has stayed portable across IDE versions. It was originally written over three years ago, but it was found to cause the network device to function properly on the latest IDE version (4.10.1). The only portion of the network driver code that did not function properly was support for printing debugging output to the serial console. This was considered a minor setback, and did not affect the outcome of the project, but only hindered debugging ability. This was worked around by using the IDE step-through debugger and debugger watches.

The EDTP NICHolas network driver code provided full support for several network protocols: ARP, ICMP Ping, and UDP Echo (port 7). It also provided support for a TCP "Telnet-like" server, that also provided an echo service. However, this did not function properly, due to inability to account for TCP Options in received TCP packets. It did provide an API handler function called `application_code()` that could be filled in with a handler routine for incoming data packets. This is where incoming HTTP requests to the embedded web server would be processed and a response would be generated and stored in the packet buffer to be sent to the requesting host. Once the "Telnet-like" functionality was removed from the network driver code, the web server functionality could be added. While the ICMP Ping support and the UDP Echo server support could be removed from the driver, the ARP support cannot be removed from the driver. The ICMP and UDP support does not interfere with the web server operation, so they were ignored and included in the network driver code used in the project.

The SparkFun SD/MMC Breakout board did not come with any sample code, but a similar product sold by SparkFun, the Logomatic Widget (a device that logs serial data to a file on an SD/MMC card) includes some driver code that supported SD/MMC card device SPI communication and a library for read/write FAT16 filesystem support. While not for the Z8 architecture, the C code served as an easy starting point for porting the functionality to the Z8. From this code base, and from an application note for the LPC2000 architecture about the same functionality, a generic SPI communication support module was written. Using the generic SPI module, a SD/MMC Card communications module was written according to the sample code, and referencing the SD/MMC Specification documentation. With this SD/MMC module, the source for the FAT16 module from the Logomatic source code required only small alterations to reference the API for the SD/MMC module. This FAT16 module provided file read and write access from the SD/MMC Card, though only file read access was necessary for the embedded web server project, so write access was disabled.

With network connection support and SD/MMC file reading support added to the project, the only support module that needed to be written was the HTTP request and response module. This module contained the true operating logic of the web server, and parsed HTTP requests, loaded requested resources (or recognized non-existent requested resources), and returned the resource (or an HTML error page) in a response. Since this module required a number of hard coded constant strings, these strings were better stored in read-only memory (ROM), to alleviate memory usage in RAM.

## Implementation

Connecting the EDTP NICHolas network interface to the Z8 Encore development board was relatively straightforward thanks to previous student projects. Eight pins of data bus need to be connected to a Z8 GPIO port, and this is more easily done if the pins are contiguously grouped. The EDTP network driver code uses Port C on the Z8, but it was found that this conflicted with the SD/MMC card because of the four pins located in Port C that have alternate functionalities for Serial Peripheral Interface (SPI). So the eight wire data bus (SD0 through SD7) was relocated to GPIO Port B (pins 0 to 7 respectively). Five of the ten addressing lines were required to be connected to GPIO (pins SA0 through SA4), and connected to Port G (pins 0 to 4 respectively). The other five addressing lines (pins SA5 through SA9) were hardwired (SA9 to VDD, the rest to Ground) to force a base address in the configuration of the network interface. The Channel Select line (/CS) was hardwired to Ground because the EDTP NICHolas was being wired as a single device slave. The input/output lines, reset line, and interrupt line were connected to individual GPIO port pins as necessary (refer to Schematics), and as determined by the EDTP network driver code.

Connecting the SD/MMC Breakout board to the Z8 Encore development board is much less complicated than connecting the network interface. For simple interaction with a single card, only six of the available pins are necessary to connect to the microcontroller. The two pins labeled VCC and GND should be connected to +3.3V voltage and ground respectively. The pins labeled DO and DI are the Data Out and Data In pins, and are used for the data lines of our four-wire SPI bus. They should be connected to the respective SPI data pins on the microcontroller. In the case of the Z8, DO should be connected to Master-In/Slave-Out (MISO/PC5), and DI should be connected to Master-Out/Slave-In (MOSI/PC4). Both data lines are used because communication over SPI will involve reading and writing data to and from the SD/MMC card. The other two necessary wires to connect to the breakout board are from the CS pin to the SPI Slave Select line, and from the CLK pin to the SPI Master Clock line. On the Z8, these two lines are located at SS/PC2 and SCK/PC3 respectively. Note that there are only four wires required to support the SPI bus in the wiring diagram, or six including the wiring for voltage and ground.

## Implementation Milestones

- Equipment Acquisition and Verification.
- NICHolas Network Interface Driver Completed.
- Web Server Prototype Completed (Static Pages).
- SD/MMC Communication Component Completed.
- FAT16/32 Compatibility Component Completed.
- Final Web Server Completed.
- Final Testing and Validation using Laptop on Network.

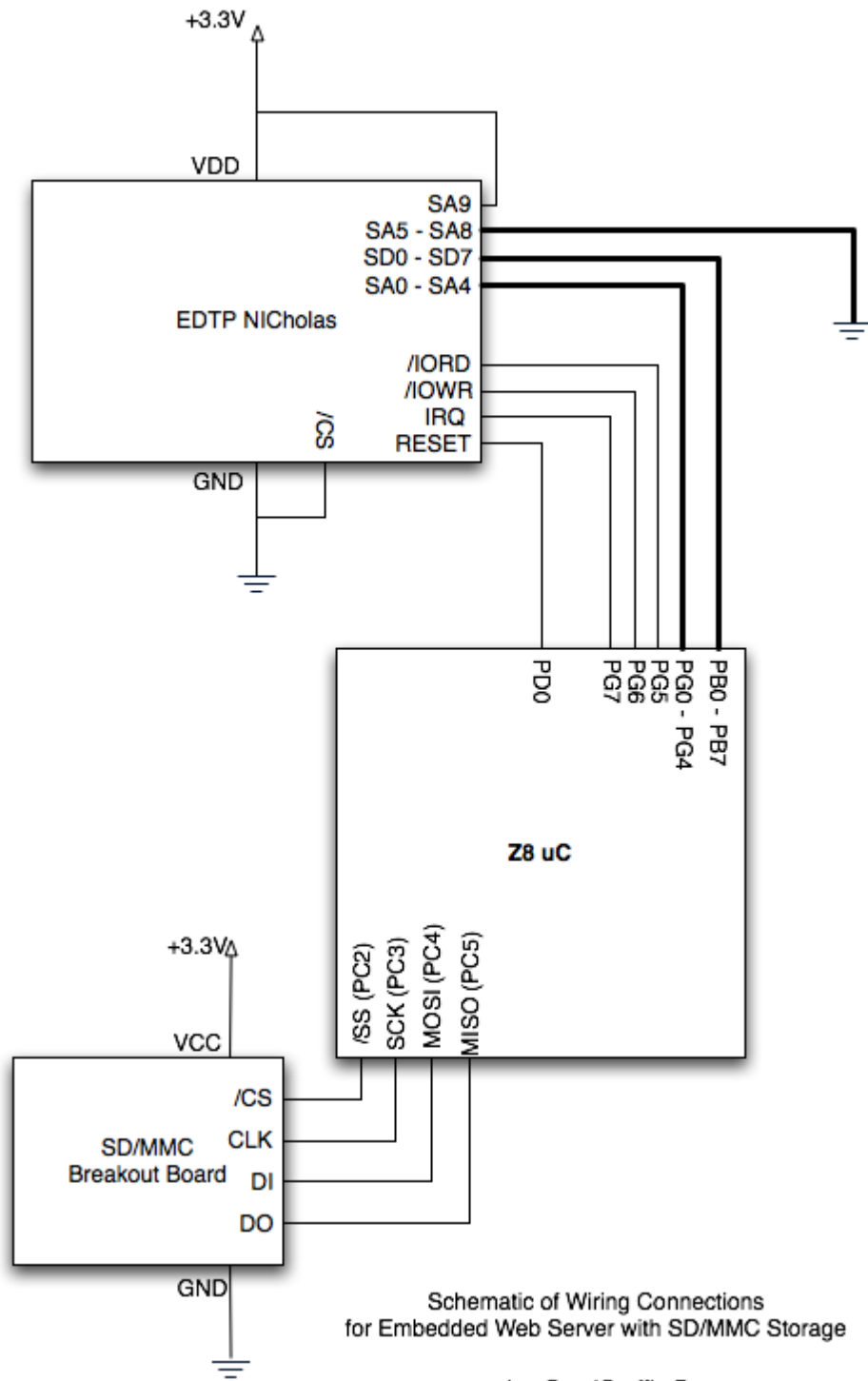


Illustration 1.: Wiring Schematics of Project.



*Illustration 2.: Detail of the EDTP NICHolas network interface.*

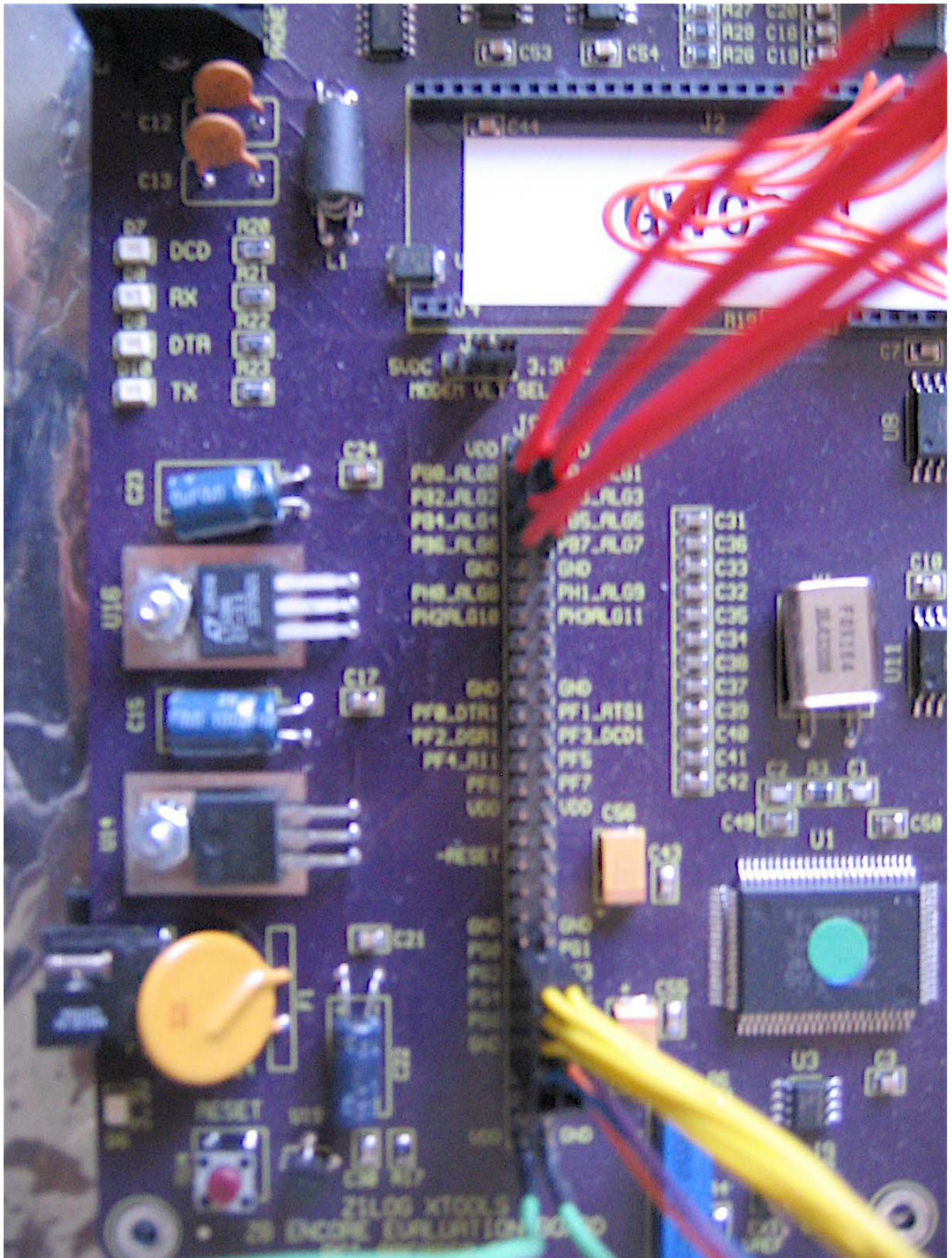
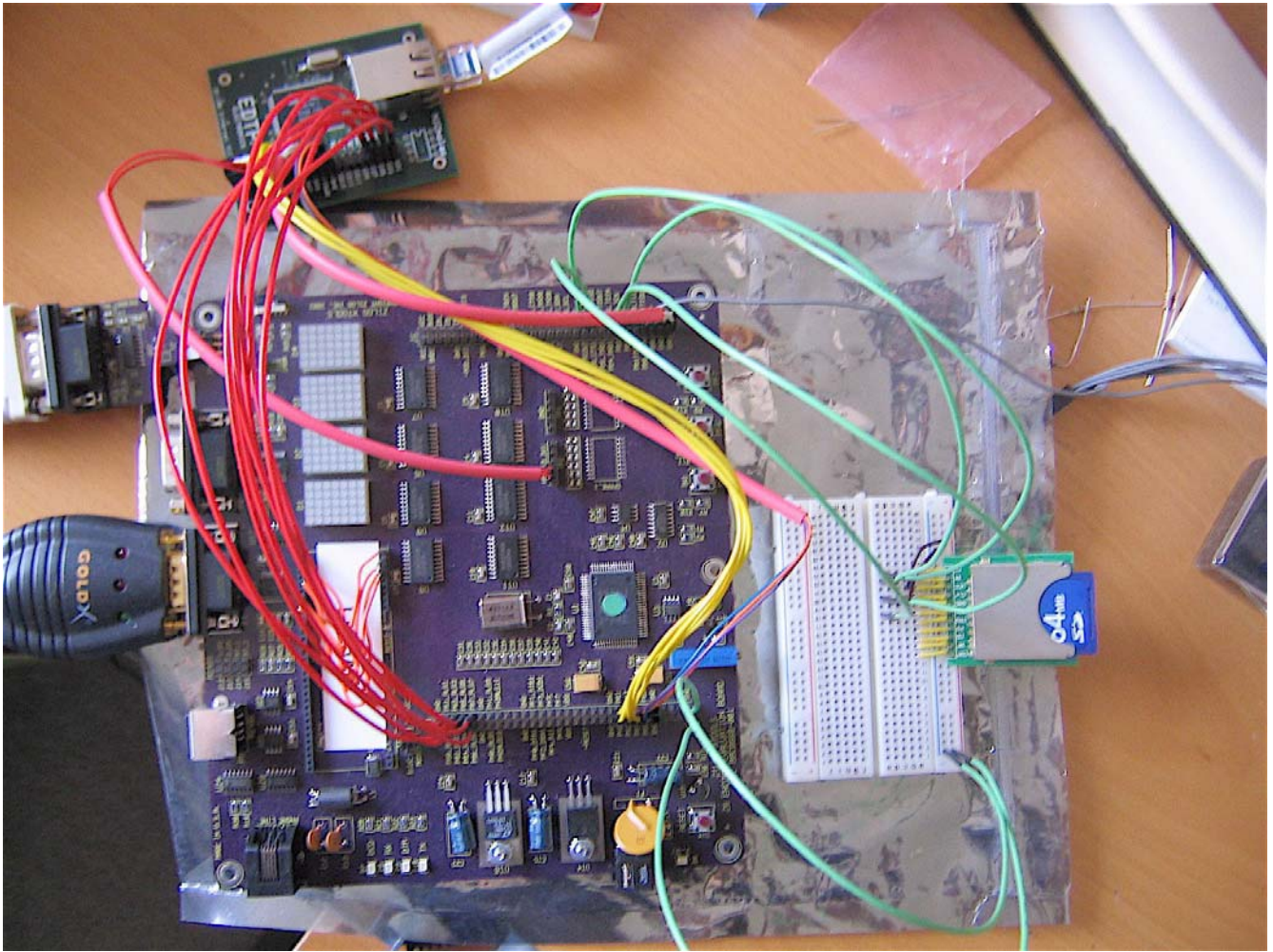


Illustration 3.: Detail of GPIO Port B and GPIO Port G on the Z8 Microcontroller.



*Illustration 4.: View of entire Project.*

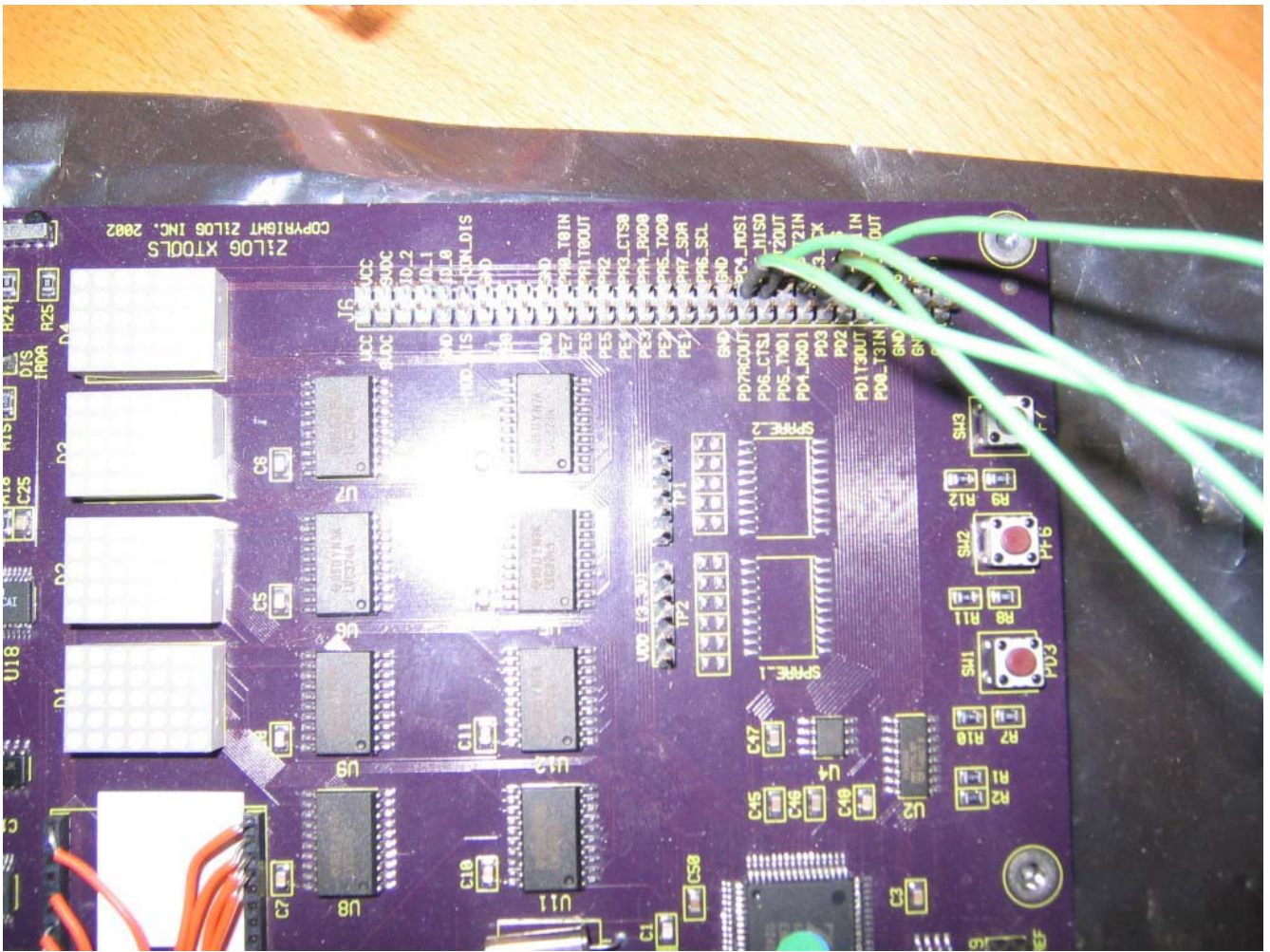


Illustration 5.: Wiring of SD/MMC Breakout Board to Z8 Microcontroller.

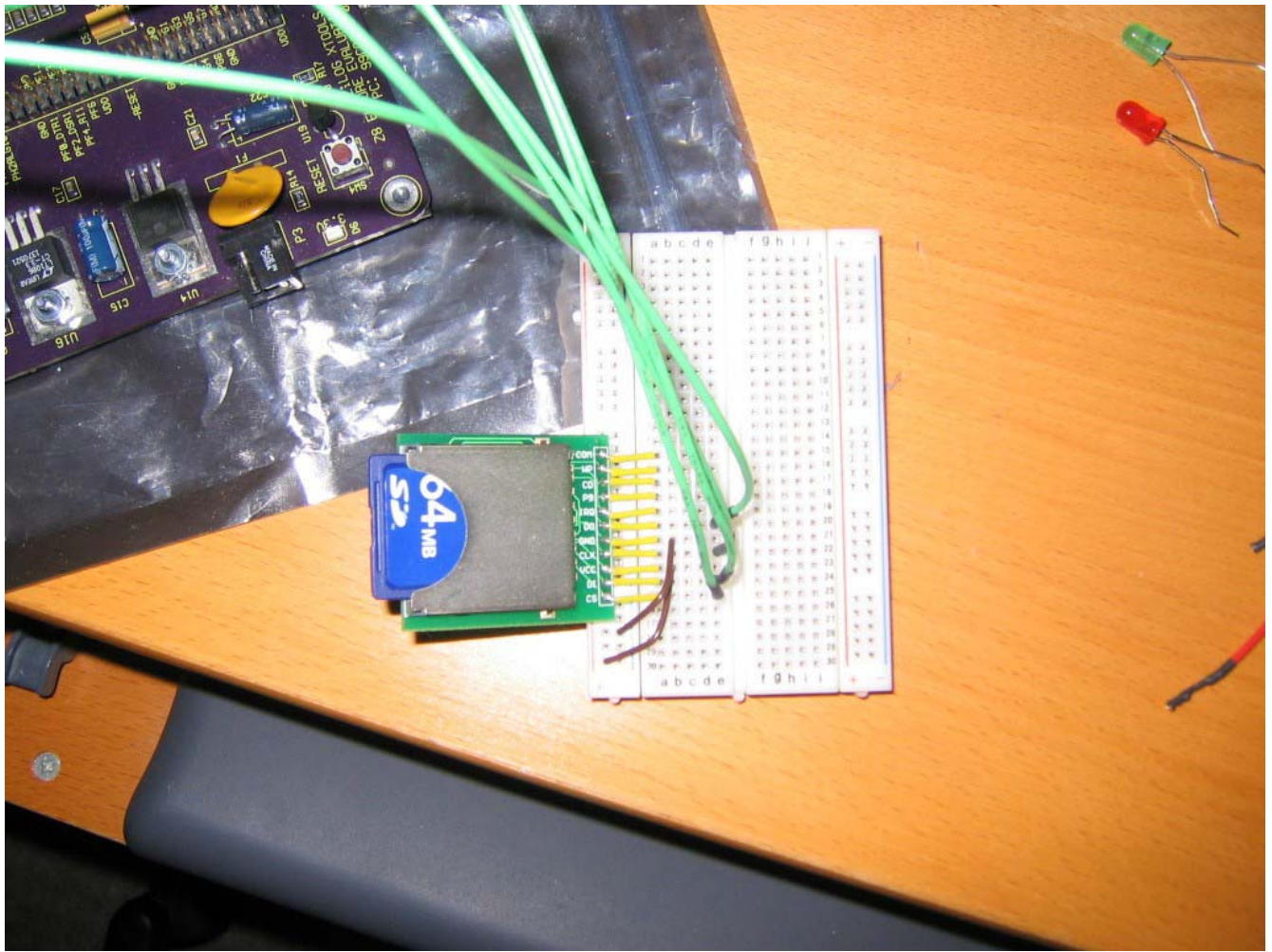
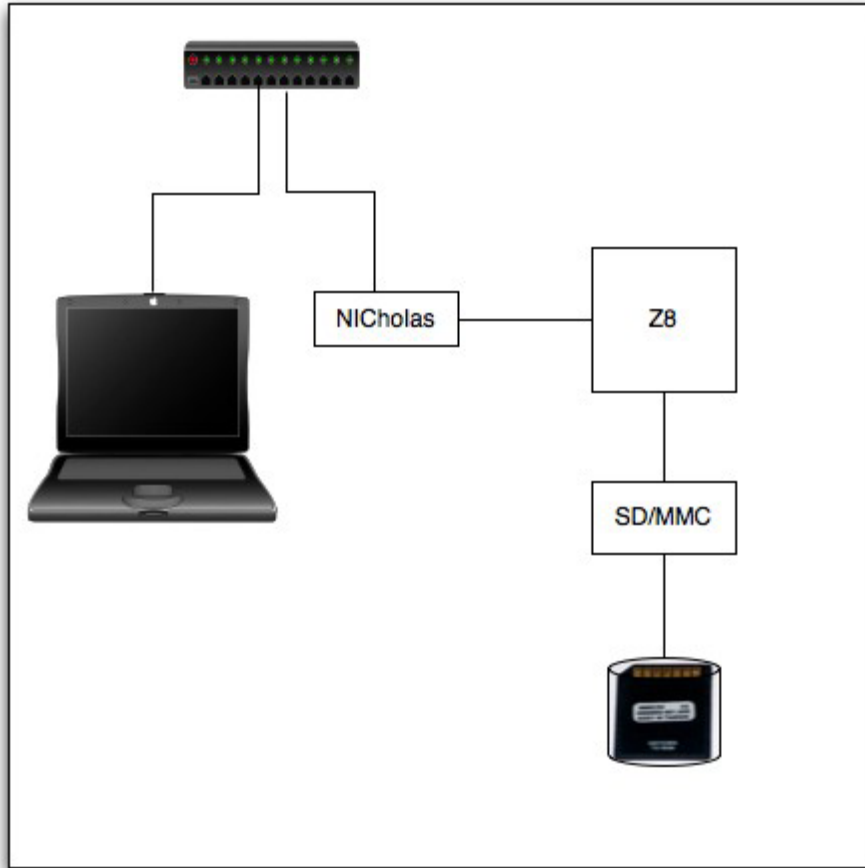
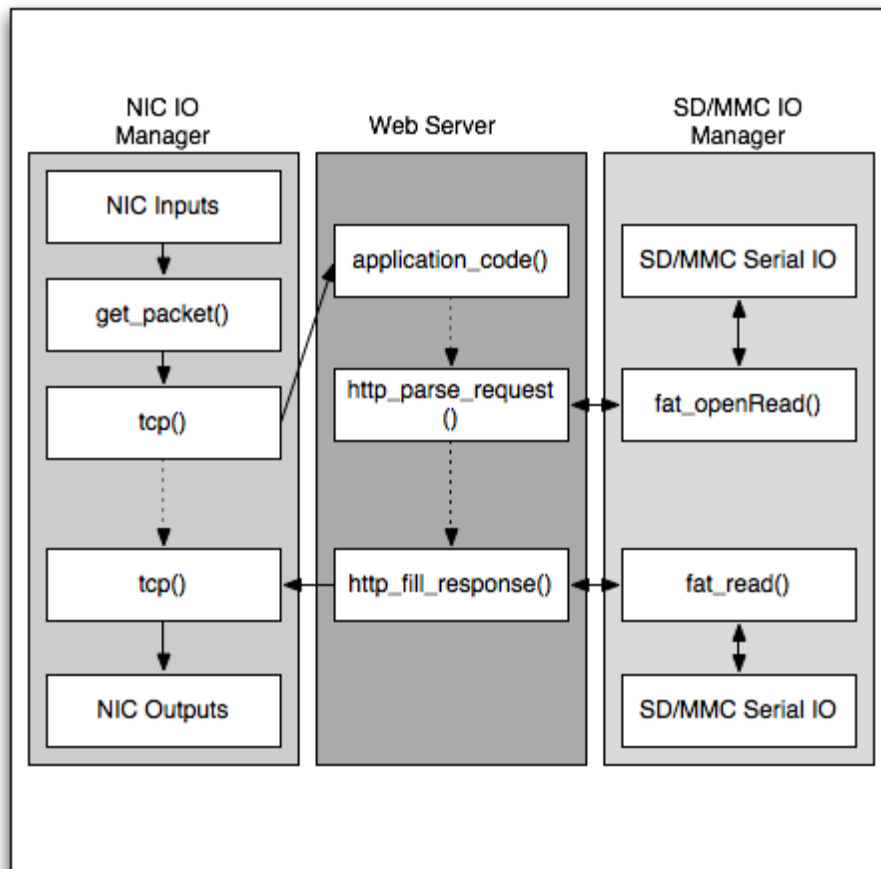


Illustration 6.: Wiring of SD/MMC Breakout Board detail.



Final Hardware Block Diagram

*Illustration 7.: Final Hardware Block Diagram*



Final Software Block Diagram

Illustration 8.: Final Software Block Diagram

## Retrospective

All in all, though the project was considered successful, there are portions that could be improved. The implementation of packet handling in the network driver was sufficient for the project, but was ultimately flawed. This portion of the driver should be rewritten to support the sending of packets from the server side, perhaps through the implementation of a full TCP/IP stack. Memory usage was a crucial resource for the server application, and the packet handling implementation imposed an artificial restriction on the size of resources sent by the web server. If this portion were rewritten, the ability to serve HTML pages larger than 512 bytes, and potentially graphic files, could be implemented. The rest of the modules could potentially be cleaned up and organized slightly better, but all in all, the design for those portions would likely remain the same in a reconstruction of this project.

## References

Accessing SD/MMC Card using SPI on LPC2000

[http://www.nxp.com/acrobat\\_download/applicationnotes/AN10406\\_3.pdf](http://www.nxp.com/acrobat_download/applicationnotes/AN10406_3.pdf)

Understanding FAT32 Filesystems

<http://www.pjrc.com/tech/8051/ide/fat32.html>

Breakout Board for SD-MMC Cards

[http://www.sparkfun.com/commerce/product\\_info.php?products\\_id=204](http://www.sparkfun.com/commerce/product_info.php?products_id=204)

Logomatic Serial Data Logger

[http://www.sparkfun.com/commerce/product\\_info.php?products\\_id=752](http://www.sparkfun.com/commerce/product_info.php?products_id=752)

SD Physical Layer Specification

[http://www.sandisk.com/Assets/File/OEM/Manuals/SD\\_Physical\\_specs\\_v101.pdf](http://www.sandisk.com/Assets/File/OEM/Manuals/SD_Physical_specs_v101.pdf)

Secure Digital Input/Output (SDIO) Card Specification

[http://www.sandisk.com/Assets/File/OEM/Manuals/SD\\_SDIO\\_specs\\_v1.pdf](http://www.sandisk.com/Assets/File/OEM/Manuals/SD_SDIO_specs_v1.pdf)

SanDisk MultiMedia Card and Reduced Size MultiMedia Card Product Manual

<http://www.sandisk.com/Assets/File/OEM/Manuals/ProdManRS-MMCv1.3.pdf>

EDTP Downloads – Z8 ASIX Code Package

[http://www.edtp.com/download/z8/z8\\_asix\\_code.zip](http://www.edtp.com/download/z8/z8_asix_code.zip)

EDTP Downloads – Easy Ethernet/NICHolas Schematic Package

[http://www.edtp.com/download/easy\\_schematics/easy\\_schematics.zip](http://www.edtp.com/download/easy_schematics/easy_schematics.zip)

EDTP NICHolas

[http://www.edtp.com/nicholas\\_page.htm](http://www.edtp.com/nicholas_page.htm)

Dean Linsalata – Zilog Ethernet Port Controller

[http://www.atomicrhubarb.com/embeddedsystems/2006/FinalProjects/Linsalada\\_FinalReport.pdf](http://www.atomicrhubarb.com/embeddedsystems/2006/FinalProjects/Linsalada_FinalReport.pdf)

Jeffery Karrels – EDTP NICHolas Implementation on a Z8 Encore

[http://www.atomicrhubarb.com/embeddedsystems/2006/Presentations/Karrels\\_Summary.pdf](http://www.atomicrhubarb.com/embeddedsystems/2006/Presentations/Karrels_Summary.pdf)