E-Quarium
A fully automated, remotely-accessible aquarium controller

Mark Robinton and Brandon Balkind
Senior Design Project
Tufts University Department of
Electrical and Computer Engineering
April 29, 2005

Table of Contents

1. Overview

This is the comprehensive design and operational documentation for the E-Quarium product.

1.1 Design Team

Brandon Balkind (Tufts EN '05, Comp. Engineer) — *Project Lead*
Mark Robinton (Tufts EN '05, Elect. Engineer) — *Project Lead*
Prof. C. H. Chang, Tufts University Dept. Electrical and Computer Engineering — *Advisor*
Prof. Ron Lasser — *Project Management Advisor*

1.2 Scope, Context

The design of the E-Quarium was conducted as part of the undergraduate engineering curriculum requirement in the Tufts University Department of Electrical and Computer Engineering. In the fall of 2004, the project team was formed and the project framework was developed under Professor Ron Lasser. In the spring of 2005, the design and implementation were carried out under Prof. C. H. Chang's advisement.

The product design is for the E-Quarium, a commercially viable prototype of an automated, remotely controllable aquarium. Intellectual property rights for the prototype are reserved to the students unless explicitly stated otherwise.

1.3 Problem Statement

It is common for inexperienced caretakers to accidentally poison, starve, overheat, or overstress aquatic life, despite efforts to the contrary. Common problems include, but are not limited to: feeding frequency/portions, adequate filtration, ammonia/nitrate buildup, water salinity/hardness, and maintaining proper heat and light cycles. All of these issues can be addressed with automated control systems via sensor feedback and scheduling. A self-sustaining aquatic environment may be difficult to achieve, but progress can be made by controlling several environmental variables.

Integrating existing technology to accomplish this goal is the greatest challenge. Commercial-off-the-shelf products are available to address single-need environment quality issues, such as: thermometers, heaters, automatic feeders, and lighting systems. The goal of this project will be to incorporate such sensors and stimuli mechanisms into a digitally integrated network environment suitable for automation.

The exploration of this problem will be of particular commercial interest to companies in the pet product industry and to those who work in aquatic biology. For researchers, the issue of tight environmental control must be addressed in any serious study. For pet owners, a less labor intensive way of caring for a home aquarium is appealing. In either case, those who are concerned with aquatic environment management will always seek to make their work easier via automation.

1.4 Contact Information

Mark Robinton	Brandon Balkind
Mark.Robinton@alumni.tufts.edu	Brandon.Balkind@alumni.tufts.edu

Tufts University Department of Electrical and Computer Engineering
161 College Avenue
Medford, MA 02155

2. Design and Implementation

This section details the concept and design procedures.

2.1 Performance Requirements

There are several key performance factors which must be met to satisfy the problem statement. While the E-Quarium needs not be entirely "hands-free" in its operation, it must automate several critical tasks in a user-friendly manner. The goal is to simplify aquarium maintenance for the novice (or experienced) aquarium user, not necessarily for the novice electronics user. Thus, understanding of RJ-45 Ethernet connections and web browser use are reasonable expectations of client.

2.1.1 Life Support Systems

The E-Quarium must implement and (to whatever extent possible) integrate the basic functions of an aquarium. The systems and their functional requirements are listed in Table 2–1.

Life Support System Performance Requirements

Function	Requirements
Feeding	Must be able to deliver adjustable portions of food in regularly scheduled intervals. The food can be stored in small reservoir for up to several days supply.
Lighting	Min. 25 W Fluorescent lights must be controllable for feeding cycles and day/night cycles.
Filtration	Normal independent filtration system (many varieties exist). Must be able to filter waste products in 10 gallon tank. Nitrite and ammonia will be consumed in biological nitrate cycle.
Chemical	*This feature reserved for future implementation.*
Air Supply	Maintain positive pressure through under gravel air stone. Must have valve to prevent moisture returning into air pump.
Temperature	Manage temperature within 3 degree F range of user defined setting. Range 40-104 degrees F.

Table 2–1

2.1.2 Hardware Interface

The hardware interface systems and their functional requirements are listed in Table 2–2.

Hardware Interface System Performance Requirements

Function	Requirements
Control Panel	16x2 Character LCD with 6 momentary buttons Button Group 1: Up/Down, OK, Back These buttons drive the menu along with LCD. Button Group 2: Feed (now), Light On/Off Clear labeling of control hardware
Ethernet	RJ-45 Connector, clearly labeled
Power	Multiple AC plugs and adapters are permissible for prototype. On production model, these should be integrated into a single plug. In the production model only, a backup battery might be used to prevent failure of computer systems and data loss. A small battery would allow for graceful shutdown in power failure. There entire system should draw under 10 Amperes.

Table 2–2

2.1.3 Software interface

The software interface systems and their functional requirements are listed in Table 2–3.

Software Interface System Performance Requirements

Function	Requirements
Panel Menu	Easy to understand button-driven menu system with OK, Back, Up, Down options. Should be capable of viewing IP address manually for initial access, as well as controlling all tank features like temp, feeding Can display temp and next feeding time
Web Interface	Able to use HTML web form to change lamp status, feed fish, schedule daily feeding, and set desired tank temperature.

Table 2–3

2.1.4 Cost

The functional prototype had a budget of no more than $500, with some supplemental reimbursement from the EECS department. The approximate spending guidelines are listed below in Table 2–4.

Expenditures

Part	Qty	Budget
TS-3300 Single Board Computer	1	200.00
Liquid Crystal 2x16 Display w/ RS-232 Input	1	20.00
5v DC, 125 VAC 2A SPDT Relays	3	9.00
Enclosure	1	10.00
256 MB Compact Flash	1	40.00
Momentary Buttons	4	5.00
Power Supply, TS-330	1	20.00
MOSFETs	3	9.00
Total		*313.00*

Table 2–4

2.2 Design Methods

2.2.1 Initial Design Approach

The project had flexibility as to which subsystems were implemented, but the central control and integration of multiple systems remained essential to the product design. Here were the initial objectives:

1. Combine and alter COTS components to provide digital response/feedback. This will at the least involve overriding the control of a light source, a feeder, and an aquatic heater. At the sensor level, digital feedback will be drawn from a thermometer, and potentially a pH sensor. Additional sensor/control components are contingent on available time and budget constraints.
2. Establish common interface and communication among devices (centralized control system).
3. Develop custom automation systems on an embedded microcomputer system. This will include either an HC12 or Intel-based architecture and a running OS which will provide Ethernet connectivity to the aquatic environment's caretaker.

As is noted in the conclusion section, some of the initial design goals were reviewed and altered, but the overall spirit of the three objectives was fulfilled entirely.

2.2.2 In-Progress Design Methods

The most significant time investment in the project design was fully understanding the operation and capabilities of all the COTS parts. This might have been a project on its own (understanding the HC12 evaluation board and the TS-3300 single board computer for example).

General system parameters and expectations were clearly established before building the prototype, but the medium-level decisions about how to make the systems communicate or perform were in constant flux due to lack of understanding. After gaining experience with the components in a laboratory setting over the semester, it is expected a more optimal comprehensive design might be achievable at a future date.

The challenges faced by the steep learning curve are discussed in more detail in the conclusion section.

2.3 Systems

The control and aquarium systems are defined as follows in the system outline of figure 2–5. This representation defines the necessary components, but does not clearly define how the pieces operate with one another. As mentioned in the design approach section, the means of system interaction were subject to significant deliberation in this design, as the designer became more familiar with the COTS components used. The best way to understand the interaction of the components is to examine the connections between the control devices in the diagram at the end of section 2.

Simple System Outline for E-Quarium

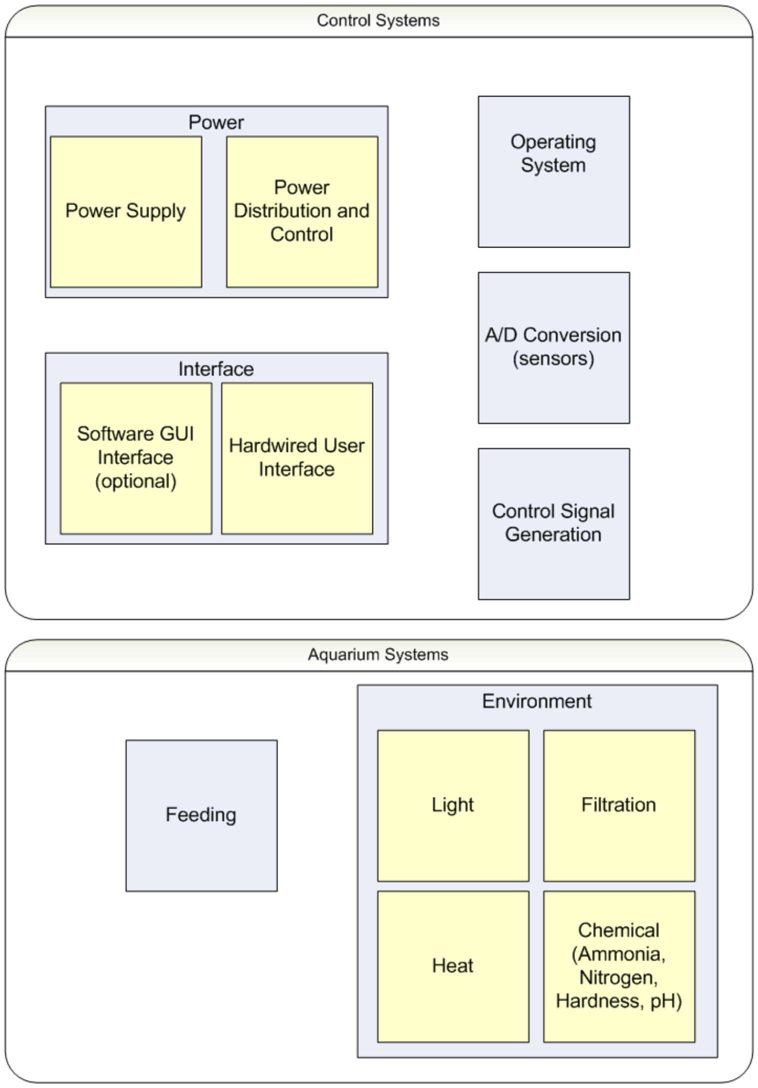


Figure 2–5

2.3.1 Control Systems

2.3.1.1 Operating System

The operating system in this design encompasses the general purpose hardware and software at the heart of the control system. TS Linux was used as the base software in this design, running on a TS-3300 single board computer from Technologic Systems. TS Linux is like Redhat Linux and is extremely space effective (fits into 32MB if necessary). Table 2–6a shows a brief list of the SBC characteristics. Table 2–7 describes TS Linux.

TS 3300 Single Board Computer
(Technologic Systems)

Parameter	Description
General	PC compatible Single Board Computer with 33 MHz Intel 386EX cpu
Memory	8 MB SDRAM 1 MB Integrated Flash Disk 1 IDE Compact Flash Socket
Peripheral I/O	2 Serial COM ports (RS-232) 1 RJ-45 Ethernet port with controller 40 Digital I/O pins PC 104 expansion bus

Table 2–6a

TS 3300 Single Board Computer
Picture

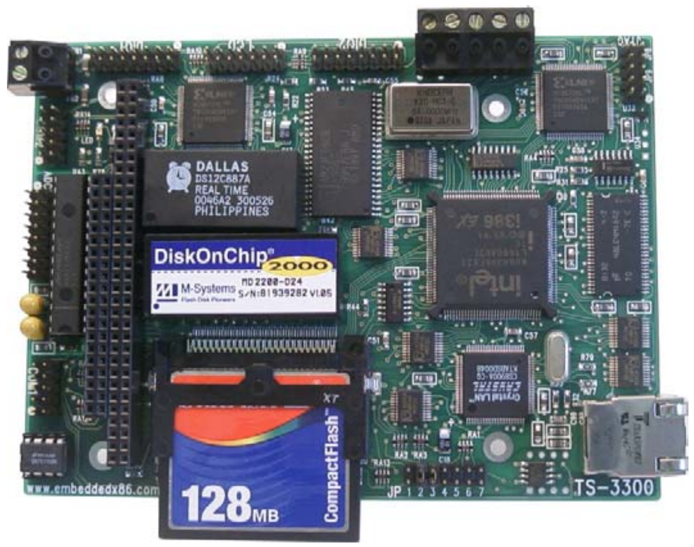


Figure 2–6b

TS Linux
(Technologic Systems)

Parameter	Description
General	Many applications implemented by the “BusyBox” memory efficient Unix suite
License	Public Domain

Table 2–7

The following procedure was used to prepare the TS-3300 Single Board Computer from receiving to completion (disregarding integration and assembly):

Linux Boot Image Phase
Using the 256 MB Compact Flash card in SanDisk media reader on host Fedora Linux system, download unzip and untar the file system image from the Technologic Systems website (there is a specific image for a 256 MB card). Format the partitions as appropriate.

Mount the various partitions as sda1 (boot, FAT16), sda2 (OS, ext2) on the host computer and expand the partition images into the appropriate drives.

Install the compact flash card in the IDE socket on the TS-3300 SBC. Mount serial COM header on COM2 terminal, with JP2 installed. Open minicom with 9600 8N1 on host Linux machine.

Apply power to the +5v and Gnd terminals to boot the system. Press Ctrl-C in minicom connection (the console redirection for BIOS startup should be visible) to enter CMOS setup mode. In CMOS setup, use the TAB, Ctrl-e, and Ctrl-X keys to navigate the menus.

In CMOS setup, change drive assignments to make drive C the IDE0 device. Change boot order to make drive C boot first. In the IDE detection settings, choose IDE0 as an “autoconfig physical” device.

Save CMOS settings and exit; the system will reboot into Linux.

Linux Configuration Phase
A file was added in the /etc/init.d/ folder called “custom.script” which launches all of the background services and initialization processes. Here is how that shell script reads:

```
#!/bin/sh

cd /dev
ln –s /dev/ttyS0 lcd # creates symbolic link for LCD
cd /
/var/www/bin/httpd –X & # start apache in single ps mode
/usr/local/bin/perl /bgps.pl & # runs monitor process
/usr/local/bin/perl /menu_driver.pl & #runs menu driver
```

The /etc/init.d/rc3.d/ folder was then amended to include a symbolic link to the /init.d/custom.script. This ensured automatic execution of the commands in run level 3 of the init process. It should also be noted that the symbolic link of the Apache startup script was removed from the rc3.d folder to allow for a custom startup in single process mode. Single process mode is important in this design because of limited memory availability.

Other notes on Linux configuration
The system should be set to DHCP for address configuration on interface eth0 and the system time should be set via the "date" command (and preferably maintained).

System software
There are three main processes which are integral to the function of the control unit (in addition to normal operating system programs).

1. The maintenance background process (bgps.pl)
   A Perl script responsible for polling the current temperature every 25 seconds, checking if the heater should be on, and checking if it is time for a daily feeding. The current temperature is obtained from the HC12 via another Perl script "copy_s19.pl" which transfers an s19 bytecode program to sample the chips ADC (attached to a thermistor). This program then executes the code on the HC12. The sub-process "copy_s19.pl" reads the 8-bit raw result over a serial connection and calls a C program to convert the value to a Fahrenheit temperate (as shown in section 2.3.2.4). This conversion program is called "tempconv" and is included in the appendix along with all other user-software. The current temperate is stored in a file "/curTemp" which is used by many processes. The desired temperature lies in /var/www/cgi-bin/ and can only be set via the web interface (along with feeding times). Daily feeding is managed by a timer comparison and a flag file. The flag is reset at 1am and is set after the daily feeding completes.
2. The webserver (httpd)
   This is an Apache webserver which must be booted in "-X" or single-process mode to conserve memory. Otherwise, the processes which it executes do not have enough memory to run properly. Permissions are generously set with "sticky" bits to make sure the webserver is allowed to access the DIO memory space. This is potentially insecure and better methods should be developed in a future design.

   The actual webserver CGI is described in a section dedicated to the remote interface.
3. The menu driver (menu_driver.pl)
   The menu driver polls for button input and sends data to the 2x16 LCD display via COM2 on the SBC (this is /dev/ttyS1). This is described more fully in the section on the manual interface.

Much of the system software interacts with the DIO1 port of the TS-3300. This is done through memory mapped I/O which requires super-user privileges. Specifically, memory address 0x7A controls the direction and operation of the DIO1 pins, while 0x7B and 0x7C reflect the output and input data itself. This memory map is compatible with the x86 architecture, but these addresses only have meaning on the TS-3300.

2.3.1.2 Control Signal Generation

Digital I/O ports on the Technologic Systems TS-3300 Single Board Computer act as logic drivers for secondary control signal circuits in this design.

DIO Output Electrical/Logical Detail

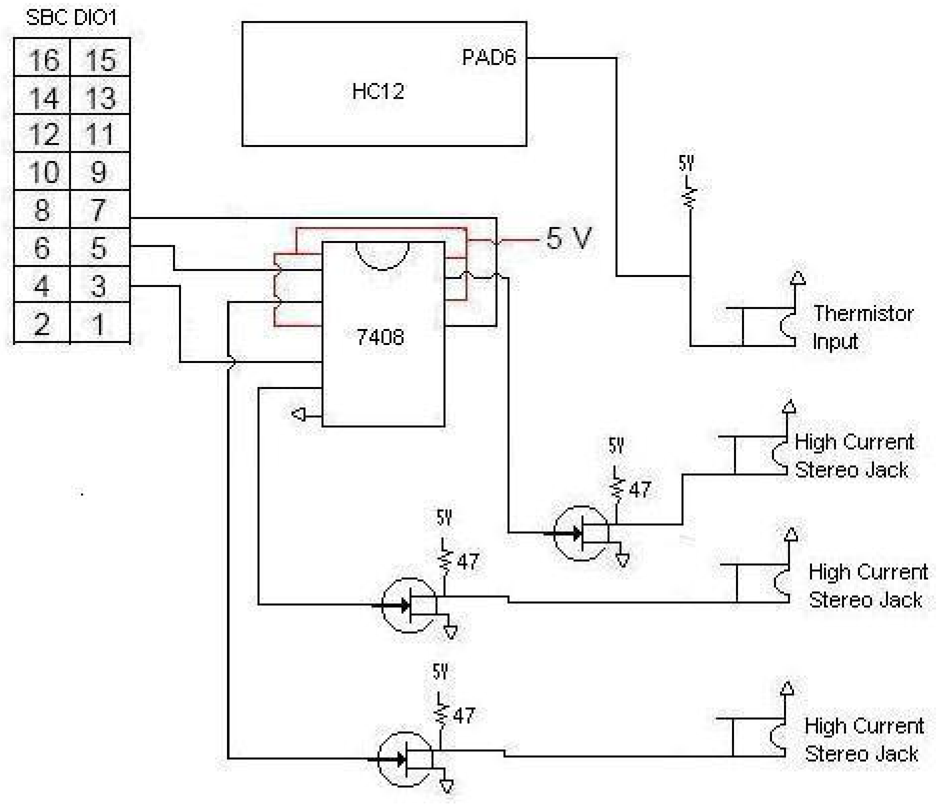


Figure 2-8

SBC DIO1

Pins 3, 5, and 7 of DIO1 on the TS-3300 are used to control external aquarium elements. Each of these is connected to a 7408 2input AND gate in order to increase the output voltage level from 3.3—0v to 5—0v (the driving voltage for the relays). These amplified signals feed into MOSFETs which amplify current. The DIO port itself can only source 4mA.

MOSFET Circuitry

Three N-channel MOSFETs are used as current amplifiers in the circuit shown in figure 2-8. These MOSFETs are configured as inverters with a resistor tied to power on the drain and the source grounded. This was done because an N-channel MOSFET (the only kind available for this project) can transmit a “0” better than a “1”. With this inverter setup, the MOSFET need only transmit “0”s and not “1”s. This inversion is accounted for in the DIO software running on the SBC.

Motorola HC12
The HC12 depicted in Figure 2-8 is used only for its Analog to Digital Conversion properties. A voltage divider between the 10K thermistor and 10K resistor is created when the thermistor is plugged into the stereo jack. Pin PAD6 of the HC12 is then used to sample this voltage in 8-bit mode and it is output the byte over a serial link to the SBC. On the SBC a program is running to convert the measured voltage into a Fahrenheit temperature that can be displayed or used in temperature relation calculations. The code for these operations can be found in the appendix.

I/O Stereo Jacks
As shown in figure 2-8, the control box contains 4 stereo jacks used for centralized input/output to the system. Facing the front of the box, the right box jack is used to connect the external thermistor to Analog Digital Converter of the HC12 inside the control box. The other 3 jacks are identical high current outputs. These can be used interchangeable to control the light, feeder, or heater and each uses a SPDT relay that requires up to 90ma to switch.

2.3.1.3 Manual Control, Access
For manual control of the aquarium, an LCD menu driven system with push buttons is necessary. The menus are arranged as follows:

HOME:

1. Info
    1. IP Address = display current IP address on eth0
    2. Temp = get current temp and desired temp
    3. Feed = display next feeding time
    4. Time = display current date and time
2. Actions
    1. Toggle light = toggle lamp on/off
    2. Feed = feed the fish now
    3. Shutdown = shutdown the board (soft)
    4. Reserved = saved for future use

The left red button acts as an ‘OK’ button, while the right red button acts as a ‘Back’ button in menu navigation. The user can move up and down the menus via the vertical imposed black buttons.

The buttons are separately connected to ground at their output terminal via 470 ohm resistors. Their input terminals are +5v, with a series 47 ohm resistor to the 4 of them. This design allows a correct voltage reading at the DIO1 input pins as shown in the routing diagram at the end of section 2.

As mentioned in section “2.3.1.1 Operating System”, under the header “System Software”, the LCD menu is driven by a background process called “menu_driver.pl”. This is a Perl script which polls the user buttons approximately every tenth of a second for input, and acts accordingly to drive the menus.

2.3.1.4 Remote Control, Automation

The hardware for this system is the RJ-45 Ethernet port driven by Crystal control logic. The operating system runs an Apache web server and “dropbear” ssh daemon over IP interface eth0. The webserver is considered the standard user interface, while ssh is available for command line access. This should not be necessary for the user of a production model, but it would be very useful for field assistance and debugging.

The webserver runs a Perl CGI in the /var/www/cgi-bin/ directory implementing CGI.pm. The script is called “default.pl” and can be accessed via http://IPADDR/cgi-bin/default.pl Perl however, is memory intensive and extremely slow. A future design should consider replacing this implementation with a standard compiled C program. The website allows the user to turn the light on and off, feed immediately or schedule a feeding, and also allows viewing and setting the desired temperate of the tank. These are accomplish via memory-mapped i/o programs or simple shell scripts.

2.3.1.5 Debug

The RS-232 COM2 port of the TS-3300 is shared between the LCD and the external DB-9 debug port. The debug port was included to allow easier diagnosis of file-system and crash-related problems on the TS-3300. It is a feature which would not be necessary on a production implementation, but is very useful for the prototype. The LCD and the debug port display the same information, but at boot the LCD cannot be used to fully understand the errors and warnings. This is where a host PC with minicom or hyperterm is useful.

2.3.2 Electromechanical, Sensor Systems

2.3.2.1 Feeding

Original Item

Peen Plax Daily Double Automatic Fish Feeder

Original Operation

Device feeds twice daily using internal oscillator circuit and gear train to spin a barrel of fish food.

Modifications

Original oscillator circuit removed.
3V DC motor attached to central gear assembly to spin food barrel.
5V SPST relay added
Plastic casing cut to accommodate motor and relay assembly.

Feeding Circuit Diagram

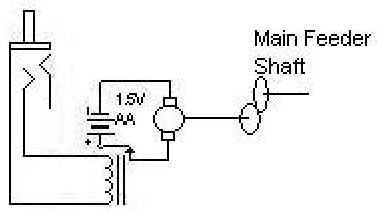


Figure 2-9

Picture of Feeder

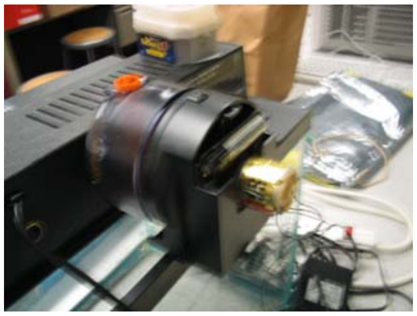


Figure 2-10

2.3.2.2 Light

Original Item

10 gallon "Perfect a Light" hood assembly and dual 25W incandescent light fixture.

Original Operation

Light toggled by means of a button on the back of the assembly

Modifications

Original toggle button removed and replaced by short circuit.
5V relay placed into the power cord.

Light Circuit Diagram

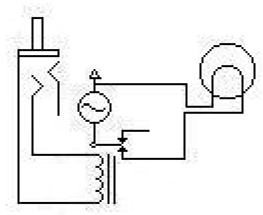


Figure 2-10

Picture of Lamp

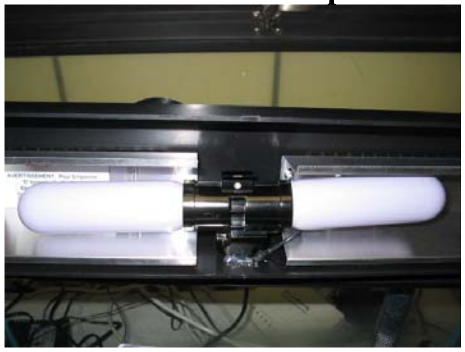


Figure 2-11

2.3.2.3 Chemical

Not included in this prototype.

2.3.2.4 Temperature Measurement

Original Items

10K Thermistor, Resistor
Motorola HC12 Evaluation Board

Original Operation

Voltage divider circuit built using the thermistor and the resistor. The output of this circuit was then fed into the Analog to Digital conversion input of the HC12 board. A program is run on the HC12 to sample the ADC port, compute the temperature, and output it over the serial port to the Single Board Computer.

Thermistor Circuit Diagram

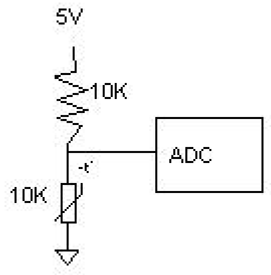


Figure 2-12

Modifications

The thermistor was attached to longer lead wires so it could reach from the inside of the tank to the control until. The thermistor was also dipped several times in clear nail polish to waterproof it and the solder joints on it.

$$Voltage = \frac{ADC}{51} \qquad Resistance = \frac{2000 * Voltage}{1 - \frac{Voltage}{5}}$$

Picture of Thermistor

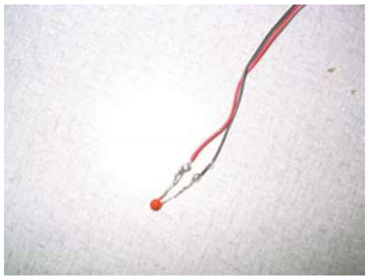


Figure 2-13

2.3.2.5 Filtration

Standard Whisper power filter unit with active carbon will be used.

2.3.2.6 Heater

Original Items

75W 8" TOPFIN Aquarium Heater

Original Operation

Internal thermostat would turn heater on and off when a malleable piece of metal open and closed a circuit. The "Set point" was controlled by a knob on top of the device that added spring pressure to the metal to facilitate or impede movement.

Modifications

A wire was soldered between the malleable metal strip and the contact point, so as to always close the circuit.
A 5V 1A relay was added to the power chord so the heater could be turned on and off at will. (When given power, it's always heating vs. the old self regulation)

Heater Circuit Diagram

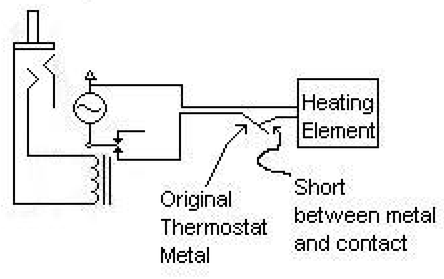


Figure 2-14

Picture of Heater

Figure 2-15

2.4 Form Factor

The control unit is built into a relatively plain black box which was adapted to the system's needs. Holes were drilled for LCD mounting, stereo jacks, buttons, an Ethernet port, power connections, and a debug DB-9 port. The front view of the control unit is drawn in figure 2-16. This was chosen as a prototype form factor because it provided (barely) adequate room for the components while shielding them from water and the environment. The box is 8" x 6" x 3". (it is 3 inches deep)

Basic Front View of Control Unit

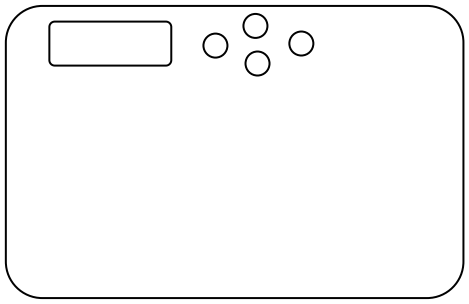


Figure 2-16

2.5 Production Considerations

In order for this prototype to be realized as a product, there must be several phases of design revision. They are listed according to their scope below.

Reproduction/Simplification Phase

The next phase of the design would essentially redraft the current functions and systems as simply and efficiently as possible. The original prototype would seem very unreliable and weary when compared to prototype using the same design built after the first. Instead of "hacking" components brutally to get them to work, a second prototype could be made from a recipe of only needed parts and precision assembly. This prototype would not necessarily address the potential need for platform changes (like moving from x86 to ARM). It would just be a way to demonstrate a reliable product to a potential investor. For example, instead of many loose wire connections and custom-modified cables, a second prototype would use PCB's for auxillary circuitry and secure, manufactured connectors and cables.

Feature Development / Platform Phase

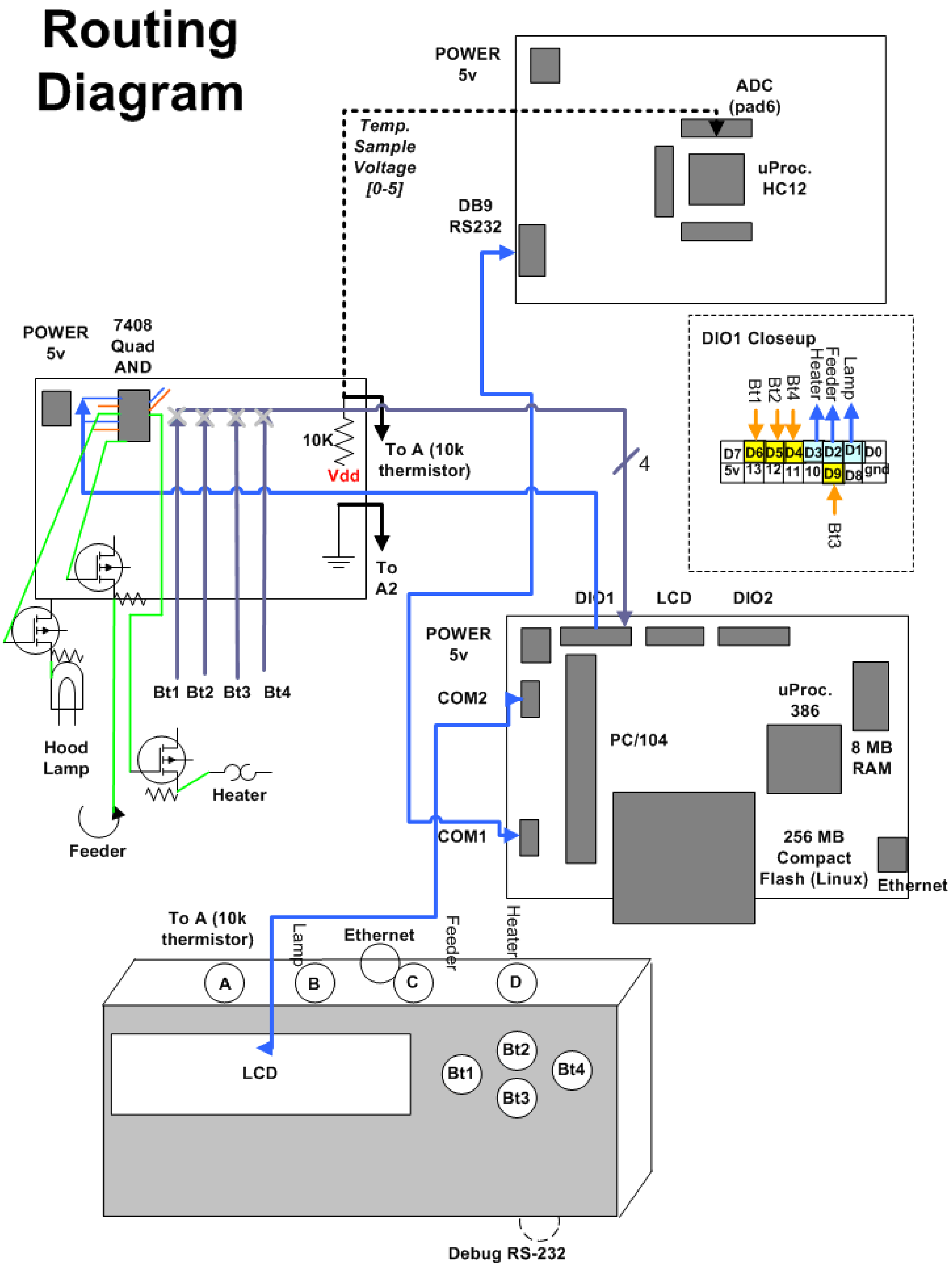
After demonstrating a prototype confidently to potential investors or customers, a list of additional features for the final product would be developed, as well as performance criteria. To assist in meeting these criteria in a cost effective manner, a new platform would be chosen. This would likely be the ARM platform. The software must move away from runtime compiled languages like Perl, and into space and speed efficient languages like C. The operating system would have to be reevaluated and optimized for production. Some functions might be implemented in hardware (most likely FPGA's). All hardware would be miniaturized and designed onto a single board (ADC, microprocessor, memory, flash storage, relay drivers, transformers, power supply, etc). This is in contrast with the current system where many COTS parts as simply wired together (HC12 for ADC, SBC for DIO and uProc, separate relay drivers, etc). Production cost evaluation in this phase would be crucial. The design must also consider the eventual integration plan for the electronic controls into the electromechanical systems. (there should be a better solution than stereo jack connections). Wireless internet should be considered.

Integration Phase

The control platform would be built into an aquarium directly. Peripheral sensors and devices would need to have reliable and easy to understand method of interfacing with the control unit. The control unit would become a standard part of the aquarium much as a computer is now standard in most automobiles.

2.6 Current Hardware Design

The diagram below represents the integration of the COTS control systems in the current design, so that another designer could reproduce the hardware systems.



3. Operation/Results

This section details the operational performance and user documentation of the E-Quarium product.

3.1 Design Performance Successes/Failures

This section gives a brief overview of how the final design met the initial requirements. Detailed analysis of these results is provided in section 4.

Life Support System Performance

Function	Requirements	Results
Feeding	Must be able to deliver adjustable portions of food in regularly scheduled intervals. The food can be stored in small reservoir for up to several days supply.	**SUCCESS**
Lighting	Min. 25 W Fluorescent lights must be controllable for feeding cycles and day/night cycles.	**PARTIAL SUCCESS** Lights not on schedule, but are controllable
Filtration	Normal independent filtration system (many varieties exist). Must be able to filter waste products in 10 gallon tank. Nitrite and ammonia will be consumed in biological nitrate cycle.	**SUCCESS**
Chemical	*This feature reserved for future implementation.*	**N/A**
Air Supply	Maintain positive pressure through under gravel air stone. Must have valve to prevent moisture returning into air pump.	**SUCCESS**
Temperature	Manage temperature within 3 degree F range of user defined setting. Range 40-104 degrees F.	**PARTIAL SUCCESS** Temperature measurement software is unreliable

Table 3–1

Hardware Interface System Performance

Function	Requirements	Results
Control Panel	16x2 Character LCD with 6 momentary buttons Button Group 1: Up/Down, OK, Back These buttons drive the menu along with LCD. Button Group 2: Feed (now), Light On/Off Clear labeling of control hardware	**PARTIAL SUCCESS** Only four buttons implemented, but all functions are still possible via the menu (see software requirements)
Ethernet	RJ-45 Connector, clearly labeled	**SUCCESS**
Power	Multiple AC plugs and adapters are permissible for prototype. On production model, these should be integrated into a single plug. In the production model only, a backup battery might be used to prevent failure of computer systems and data loss. A small battery would allow for graceful shutdown in power failure. There entire system should draw under 10 Amperes.	**SUCCESS**

Table 3–2

Software Interface System Performance Requirements

Function	Requirements	Results
Panel Menu	Easy to understand button-driven menu system with OK, Back, Up, Down options. Should be capable of viewing IP address manually for initial access, as well as controlling all tank features like temp, feeding Can display temp and next feeding time	**PARTIAL SUCCESS** Not always easy to use: sometimes errors are reported to LCD and user can lose menu. Temperature display not always accurate
Web Interface	Able to use HTML web form to change lamp status, feed fish, schedule daily feeding, and set desired tank temperature.	**PARTIAL SUCCESS** Very slow

Table 3–3

3.2 Other Important Results

Overall, the board performs all the basic functions that were essential to solving the initial problem. The reliability of its performance however, leaves something to be desired. There are still software errors related to memory use, multiple process I/O issues, and occasional hardware mishaps (like loose alligator clips on power cord etc). The prototype is only that, and cannot compete with a production-quality device.

4. Conclusions

This section will discuss lessons learned from the design and implementation.

The greatest lesson learned in this design project was the appreciation for the value of experimentation. Most facets of the design process were specifically detailed in the project proposal of fall 2004. The experimentation process was nearly entirely neglected in this proposal however. While time was allocated for "research" and "implementation," the majority of the time was actually spent in between the two phases, trying to determine how the physical materials we obtained fit into the design we originally researched. If structured experiments had been planned ahead of time with elements like the LCD, we would have learned that cheap displays often correlate to faulty displays. Because we were largely unfamiliar with the operation of the LCD's, we spent a great deal of time trying to debug multiple systems which used them. The LCD issue was eventually resolved by using higher quality parts, only after significant time had been spent with trying to understand how the actual device worked. This was not necessarily time wasted, but it was time that was not considered in the original plan, and is something that should be considered for all future work.

The need for experimentation was evident in all aspects of the prototype. The undocumented voltage levels of the DIO ports of the TS-3300 for example, became a significant surprise after they were expected to operate in conjunction with other systems. Only after having stepped back and experimented with the DIO pins in isolation, were we able to learn how to solve the problem. The modified components and relays also presented unique learning experiences. For example, Reed relays, though they were initially capable of operating the peripheral devices, miraculously and irreversibly stopped working half way through the project. They weren't well constructed for the needs of the design. In mid-stream, the Reed relays were abandoned in favor of electromechanical SPDT switches. Since then, they have operated fine.

Indeed, the design of a simple web-enabled control system is not very difficult. Having learned about the operation of low-current DIO, relay driving circuits, relays, LCD's, ADC sampling, RS-232, and small resource computing, we might honestly claim that a digital web-enabled control system for any series of 110 VAC or DC-powered devices could be designed and assembled in a week (providing all parts were available). The design patterns and software structure would be readily understandable for many devices—now that the basic elements are familiar.

The E-Quarium design should be recognized in a broader context. Though it has been applied to a rather light-hearted scenario of managing fish, the control systems demonstrated in this project, as mentioned above, could be used to manage just about any system. While the ability of the controller to quickly integrate with peripheral 110 VAC devices does not compete with products like X10 (which has controllable wall outlets), the X10 system also does not provide the level of control that can be achieved with an embedded computing and sensing device.

That being said, the value and marketability of the E-Quarium product idea deserves serious consideration. We feel the features developed in the prototype only scratch the surface of what might be in need. In biological laboratories for example, there is a large demand for tedious labor in maintaining multiple experimental variables. A controller like what is used in the E-Quarium prototype would be able to use chemical, thermal, and other sensors in conjunction with control mechanisms to fully automate the process. It could profile and graph temperature variations, pH fluctuations, and even light intensity very easily on a 24-hour schedule. This would ordinarily be a significant undertaking for laboratory technicians.

As for the actual performance of the prototype created in this design, the results were deemed an overall success by the group. The evaluation criteria were clearly laid out in terms of user functionality in section 2. These criteria were then compared with the results in section 3. It is evident that many key objectives were met with "partial success."

The use of four buttons instead of six does not represent a major violation of the hardware interface criteria, because the functions of the additional buttons were redundant when one considers the purpose of the software menu. The user can feed the fish and toggle the hood light via the menu, and special, separate buttons are not entirely necessary.

The LCD menu itself however, is the least completely debugged feature of the prototype, because it is the top layer in a stack of dependent systems. The menu depends on the Perl menu driver, the operating system's driver for ttyS1, the background management process, the HC12, the thermistor circuit, the ADC, the driver for ttyS0, the webserver, and just about every system in the entire device down to the memory and CPU itself. This chain of dependency is responsible for increasing the complexity of debugging attempts. The biggest problem is the reliability of the period driver which loads the HC12 and samples the thermistor. For some reason, the HC12 periodically fails and the current temperature is corrupted. Temporary fixes to this problem do not address the real cause of the problem, which as of April 2005, is unknown.

The only other feature deemed "partial" in its success that does not relate to the temperature measurement systems is the lamp system. The scheduling of the light according to day/night rhythms was not implemented in time for the first demonstration. The light is controllable via the web or the control panel however, which fulfills some, but not all of the design criteria.

With design improvements largely suggested in the section entitled "Production Considerations," (2.5), it is important at this point to recognize that this project was not just an exercise in electrical or computer engineering, but also an endeavor in systems engineering and project management. Pursuing the E-Quarium project resulted in success not only from the technical standpoint, but from the perspective of continuing education.

Appendix

bgps.pl PERL SCRIPT
Background process to maintain temp, heat, check for feeding time

```
#!/usr/bin/perl -w
use strict;

sub checkFeed{
    my @rawdat;
    my $curfeed=0;
    my $nextfeed;
    my $feedSemaphore;
    my @curtime;
    my @tempvar;

    open(mtempfile,"/var/www/cgi-bin/feedTime");
    @rawdat = <mtempfile>;
    $nextfeed = $rawdat[0];
    close(mtempfile);

    open(mtempfile,"/feedsemaphore");
    @rawdat = <mtempfile>;
    $feedSemaphore = $rawdat[0];
    close(mtempfile);


    #print "trying";
    @curtime = split(/ /,`date`);
    @tempvar = split(/:/, $curtime[3]);
    #if($curtime[5] eq "PM"){
    #    $curfeed =  $tempvar[0] + 12;
    #}
      #else{
    #if($tempvar[0] < 12){
    $curfeed =  $tempvar[0];
    #}else{
    # $curfeed = 0;
    #    }
    #}

    if($feedSemaphore == 0){
      if($curfeed > $nextfeed){
          open(mtempfile,">/feedsemaphore");
          print mtempfile "1";
          close(mtempfile);

          `./dio_mask_off 8`;
          sleep(1.5); #1.5 rotations
          `./dio_mask_on 8`;

      }
    }
    if($curfeed == 1){
      open(mtempfile,">/feedsemaphore");
      print mtempfile "0";
      close(mtempfile);
    }
}

sub updateTarTemp{
    # grab it from the website directory
    `cp /var/www/cgi-bin/tarTemp /`;
}

sub updateCurTemp{
    my $curTemp;
    open(mtempfile,">/curTemp");
    print mtempfile `/usr/local/bin/perl /copy_s19.pl`;
    close(mtempfile);
}

sub checkTemp{
    my @rawdat;
    my $curtemp;
    my $tartemp;
    open(mtempfile,"/curTemp");
    @rawdat = <mtempfile>;
    $curtemp = $rawdat[0];
    close(mtempfile);
    open(mtempfile,"/tarTemp");
    @rawdat = <mtempfile>;
    $tartemp = $rawdat[0];
    close(mtempfile);

    if($curtemp < ($tartemp - 1)){
      `/dio_mask_off 4`;
    }
    if($curtemp > ($tartemp + 1)){
      `/dio_mask_on 4`;
    }
}

while(1){
    updateCurTemp();
    updateTarTemp();
    checkTemp();
    checkFeed();
    sleep(25);
}
```

menu_driver.pl PERL SCRIPT
Drives the LCD menu and controls many tank functions

```
#!/usr/bin/perl -w

open(LCD,">/dev/lcd")||die "ERROR: cannot write to /dev/lcd\n";

sub init_display{
   print LCD chr(0xfe), chr(0x01); # clear the display
   sleep(0.1);
   print LCD chr(0xfe), chr(0x02);
   sleep(0.1);
   print LCD chr(0xfe), chr(13);
}

sub goto_line2{
   print LCD chr(0xfe), chr(192);
}

sub goto_line1{
    print LCD chr(0xfe), chr(128);
}

sub poll_buttons{
    my $button_reg = `./dio_input`;
    my $button_pushed = 0;
    if(ord(pack("H2",substr($button_reg,1,2))) & 0x04){
      $button_pushed = 1;
    }
    if(ord(pack("H2",substr($button_reg,1,2))) & 0x02){
      $button_pushed = 2;
    }
    if(ord(pack("H2",substr($button_reg,1,2))) & 0x01){
      $button_pushed = 4;
    }
    if(ord(pack("H2",substr($button_reg,1,2))) & 0x20){
      init_display();
      $button_pushed = 3;
    }
    return $button_pushed;
}

sub info_menu
{
    #init_display();
    #goto_line1();

    my $buttonp=0;
    my $curline_main=1;

    while($buttonp != 3){
      init_display();
      if(($curline_main==1)||($curline_main==2)){

          goto_line1();
          print LCD " 1. IP Address";
          goto_line2();
          print LCD " 2. Temp";
          if($curline_main==1){
            goto_line1();
          }
          else{
            goto_line2();
```

```
        }
    }
    if(($curline_main==3)||($curline_main==4)){

        goto_line1();
        print LCD " 3. Feeding";
        goto_line2();
        print LCD " 4. Time";
        if($curline_main==3){
          goto_line1();
        }
        else{
          goto_line2();
        }
    }

    $buttonp = poll_buttons();
    if($buttonp == 1){
        # 'OK' BUTTON
        if($curline_main==1){
          # print the IP address

          init_display();
          goto_line1();
          my @comResult = `ifconfig eth0`;
          foreach(@comResult){
              if(/inet addr:([\d.]+)/){
                print LCD "$1";
              }
          }
          while(poll_buttons()!=3){
              sleep(0.1);
          }
        }elsif($curline_main==2){
          # print temperature
          my $curTemp;
          my $tarTemp;
          my  @rawdat;
          open(tempfile,"/curTemp");
          @rawdat = <tempfile>;
          close(tempfile);
          $curTemp = $rawdat[0];
          open(tempfile,"/tarTemp");
          @rawdat = <tempfile>;
          close(tempfile);
          $tarTemp = $rawdat[0];

          init_display();
          goto_line1();

          print LCD "Cur: ".$curTemp;
          goto_line2();
          print LCD "Target:".$tarTemp;


          while(poll_buttons()!=3){
              sleep(0.1);
          }
        }elsif($curline_main==3){
          # print feeding time
          init_display();
          goto_line1();
          my @nextfeed = `cat /var/www/cgi-bin/feedTime`;
          if($nextfeed[0] eq "10"){
                print LCD "10:00 AM";
            }
            else{
                print LCD "2:00 PM";
            }
            while(poll_buttons()!=3){
                sleep(0.1);
            }
          }elsif($curline_main==4){
            init_display();
            goto_line1();
            print `date`;
            while(poll_buttons()!=3){
                sleep(0.1);
            }
          }
      }
      elsif($buttonp == 2){
          # 'UP' BUTTON
          goto_line1();
          $curline_main-=1;
      }
      elsif($buttonp == 4){
          # 'DOWN' BUTTON
          goto_line2();
          $curline_main+=1;
      }
      elsif($buttonp == 3){
          # 'BACK' BUTTON
          return;
      }

      if($curline_main>4){
          $curline_main=4;
      }
      if($curline_main<1){
          $curline_main=1;
      }
      sleep(0.1);
    }

}

sub action_menu
{
    #init_display();
    #goto_line1();

    my $buttonp=0;
    my $curline_main=1;

    while($buttonp != 3){
      init_display();
      if(($curline_main==1)||($curline_main==2)){

          goto_line1();
          print LCD " 1. Toggle Light";
          goto_line2();
          print LCD " 2. Feed";
          if($curline_main==1){
            goto_line1();
          }
          else{
            goto_line2();
          }
      }
      if(($curline_main==3)||($curline_main==4)){

          goto_line1();
          print LCD " 3. Shutdown";
          goto_line2();
          print LCD " 4. Reserved";
          if($curline_main==3){
            goto_line1();
          }
          else{
            goto_line2();
          }
      }

      $buttonp = poll_buttons();
      if($buttonp == 1){
          # 'OK' BUTTON
          if($curline_main==1){
            #
            init_display();
            goto_line1();
            if(ord(pack("H2",substr(`/dio_input`,2,2))) & 0x02){
                `/dio_mask_off 2`;
                print LCD "Light is now ON";
            }else{
                `/dio_mask_on 2`;
                print LCD "Light is now OFF";
            }
            while(poll_buttons()!=3){
                sleep(0.1);
            }
          }elsif($curline_main==2){
            `./dio_mask_off 8`;
            sleep(1.5); #1.5 rotations
            `./dio_mask_on 8`;
            init_display();
            goto_line1();
            print LCD "Done Feeding";
            while(poll_buttons()!=3){
                sleep(0.1);
            }
          }elsif($curline_main==3){
            # shutdown
            `shutdown 0 -h`;
          }elsif($curline_main==4){
            # nothing yet
            while(poll_buttons()!=3){
                sleep(0.1);
            }
          }
      }
      elsif($buttonp == 2){
          # 'UP' BUTTON
          goto_line1();
          $curline_main-=1;
      }
      elsif($buttonp == 4){
          # 'DOWN' BUTTON
          goto_line2();
          $curline_main+=1;
      }
      elsif($buttonp == 3){
```

```
          # 'BACK' BUTTON
          return;
      }

      if($curline_main>4){
          $curline_main=4;
      }
      if($curline_main<1){
          $curline_main=1;
      }
      sleep(0.1);
    }

}

# MAIN ****************************************

init_display();
goto_line1();
print LCD "E-Quarium";
sleep(5);
goto_line2();

while(1){
    my $buttonp=0;
    my $curline_main=1;
    init_display();
    goto_line1();
    print LCD " 1. Info";
    goto_line2();
    print LCD " 2. Actions";
    goto_line1();
    while($buttonp != 1){
      $buttonp = poll_buttons();
      if($buttonp == 1){
          # 'OK' BUTTON
          if($curline_main==1){
            info_menu();
          }
          else{
            action_menu();
          }
      }
      elsif($buttonp == 2){
          # 'UP' BUTTON
          goto_line1();
          $curline_main=1;
      }
      elsif($buttonp == 4){
          # 'DOWN' BUTTON
          goto_line2();
          $curline_main=2;
      }
      elsif($buttonp == 3){
          # 'BACK' BUTTON
      }
    }
}

close LCD;
```

default.pl PERL SCRIPT
Web page code

```
#!/usr/local/bin/perl -w
use lib '/usr/local/lib/perl5/5.8.0/';
use CGI::Carp qw(fatalsToBrowser);
use CGI qw(standard);
use strict;
package eqaweb;

print "Content-type: text/html\r\n\r\n";
print "<html>";

print "<head>";
print "<title>Control Panel</title>";
print "</head>";

print "<body>";
print "<form action=\"default.pl\" method=\"GET\">";
print "<table>";
print "<tr><td><b>E-Quarium Web Control Panel</b></td></tr>";

# form processing actions

print "<tr><td><b>";

if(CGI::param('toggle1off') eq "off"){
    print "Light turned OFF";
}
if(CGI::param('toggle1on') eq "on"){
    print "Light turned ON";
}
if(CGI::param('feed') eq "Feed"){
    print "Fish have been fed";
}
if(CGI::param('changeTemp') eq "Change"){
    open(tempfile,">tarTemp") || print "Permissions problem";
    print tempfile CGI::param('desiredTemp');
    close(tempfile);
    print "Changed desired temp";
}
if(CGI::param('setFeed') eq "Set"){
    print "Changed desired feed time";
}
print "</b></td></tr>";

# form

print "<tr><td colspan=\"2\">Aquarium Light:</td></tr>";
print "<tr><td><input name=\"toggle1on\" type=\"submit\"
value=\"on\"></td>";
print "<td><input name=\"toggle1off\" type=\"submit\"
value=\"off\"></td></tr>";
my $curTemp;
my $tarTemp;
my $feedTime;
my @rawdat;
my @rawdat2;
open(tempfile,"tarTemp");
open(tempfile2,"/curTemp");
@rawdat = <tempfile>;
@rawdat2 = <tempfile2>;
close(tempfile);
close(tempfile2);
$tarTemp = $rawdat[0];
$curTemp = $rawdat2[0];

open(tempfile,"feedTime");
@rawdat = <tempfile>;
$feedTime = $rawdat[0];
close(tempfile);

print "<tr><td colspan=\"2\">Feeding:</td></tr>";
print "<tr><td>Currently set to feed at: ".$feedTime."-hour</td></tr>";
print "<tr><td><select name=\"feedTime\">";
print "<option value=\"10\">10am";
print "<option value=\"14\">2pm";
print "</select></td></tr>";
print "<tr><td><input name=\"setFeed\" type=\"submit\"
value=\"Set\"></td></tr>";
print "<tr><td><input name=\"feed\" type=\"submit\"
value=\"Feed\"></td></tr>";


print "<tr><td colspan=\"2\">Temperature:</td></tr>";
print "<tr><td colspan=\"2\">Cur: ".$curTemp."<br>Target:<input
width=\"5\" name=\"desiredTemp\" value=\"".$tarTemp."\"></td>";
print "<td><input name=\"changeTemp\" type=\"submit\"
value=\"Change\"></td></tr>";


print "</table>";
print "</form>";
print "</body>";
print "</html>";

#close STDIN;
#close STDOUT;
#close STDERR;

if(CGI::param('toggle1off') eq "off"){
    `./dio_mask_on 2`;
}
if(CGI::param('toggle1on') eq "on"){
    `./dio_mask_off 2`;
}
if(CGI::param('feed') eq "Feed"){
    `./dio_mask_off 8`;
    sleep(1.5); #1.5 rotations
    `./dio_mask_on 8`;
}
if(CGI::param('setFeed') eq "Set"){
    open(tempfile,">feedTime") || print "Permissions problem";
    print tempfile CGI::param('feedTime');
    close(tempfile);

}
if(CGI::param('changeTemp') eq "Change"){

}
```

copy_s19.pl PERL SCRIPT
Load the HC12 via ttys0 with ADC sampling program, get result, save
This program is used to sample the current tank temperature

```
#!/usr/bin/perl -w
# Brandon Balkind 4-12-05
use strict;
my $mfhandle;
`stty 9600 -F /dev/ttyS0`;
open($mfhandle, "/adc.s19") or
    die("Error");
my $line;
open(ttyS0,">/dev/ttyS0")||die "ERROR: can not write to /dev/ttyS0\n";
print ttyS0 "load\r";
close ttyS0;
open(ttyS0,">/dev/ttyS0")||die "ERROR: can not write to /dev/ttyS0\n";
# copy the s19 file
while ($line=<$mfhandle>){
    chomp($line);
    #print "$line\n";
    print ttyS0 "$line\n";
}
close ttyS0;


open(ttyS0,">/dev/ttyS0")||die "ERROR: can not write to /dev/ttyS0\n";
print ttyS0 "g 800\n";
close ttyS0;

open(ttyS0,">/dev/ttyS0")||die "ERROR: can not write to /dev/ttyS0\n";
print ttyS0 "\r"; # this was found to be the only
                  # character which activated the output sequence
close ttyS0;

open(ttyS0,"+</dev/ttyS0") || die "ERROR: can not read from
/dev/ttyS0\n";

#while($line=<ttyS0>){
$line=<ttyS0>;
    chomp($line);
    #$line =~ s/\r//g;
    if($line ne ""){
      #print "ADC Says: $line";
      $line = ord(pack("H2",$line));
      # $line = "0x".$line;
      #open(resultFile,">curTemp") || die "ERROR: ";
      print `./tempconv $line`;
      #print resultFile `./tempconv $line`;
      #close(resultFile);
    }
    $line="";
#}

close ttyS0;
close($mfhandle);
```

convert_temp.c COMPILED C
Takes raw HC12 ADC sampling data and turns It into Fahrenheit temperature according to mathematical approximation

```
#include <math.h>
#include <stdio.h>

int convert_temp(int adc);

int main(int argc, char** argv) {
  if(argc > 0){
    printf("%d",convert_temp(atoi(argv[1])));
  }
  return 0;
  //debug
  //printf("The temp is %d\n", convert_temp(127));
  //printf("The temp is %d\n", convert_temp(200));
}



int convert_temp(int adc) {
  float voltage;
  float resistance;
  double temp;

  voltage = (float)adc/51; //this may need +0.15 V
  resistance = (2000.0 * voltage) / (1.0-voltage/5.0);

  // debug
  // exp = log((double)voltage);
  // exp = cos((double)voltage);
  //printf("voltage is %2.2f\n", voltage);
  //printf("resistance is %2.2f\n", resistance);
  temp = log(resistance/10000.0);
  temp = temp/4100.0;
  temp = temp + ((double)1/(double)298);
  temp = 1.0/temp;
  temp = temp - 273.0;
  //convert to fahrenheit 2 versions
  //temp = (temp*((float)9/(float)5)) + 32;
  temp = (temp*1.8) + 32.0;
  return((int)temp);
}
```

dio_init.c COMPILED C
Sets DIO1 i/o directions and sets initial port values

```
#include <stdio.h>
#include <sys/io.h>
#include <unistd.h>

#define bank_control1 0x7A
#define io_bank1 0x7B
#define io_bank2 0x7C

int  main (int argc, char *argv[])
{
  /* Get access to the ports */

  iopl(3);
  // sets DIO-0 to DIO-3 to out
  // DIO-4 to DIO-7 to out
  // DIO-8 to DIO-11 to IN
  outb(0x01, bank_control1);
  // reverse logic
  outb(0x0f, io_bank1);
  return 0;
}
```

dio_input.c COMPILED C
Gets value of port (DIO1) used frequently for button polling

```
#include <stdio.h>
#include <sys/io.h>
#include <unistd.h>

#define bank_control1 0x7A
#define io_bank1 0x7B
#define io_bank2 0x7C


int  main (int argc, char *argv[])
{
  /* Get access to the ports */

  iopl(3);
  printf("%02.2x%02.2x", inb(io_bank2), inb(io_bank1));

  return 0;
}
```

dio_mask_on.c COMPILED C
Sets the bits of arg[0] in the DIO1 port

```
#include <stdio.h>
#include <sys/io.h>
#include <unistd.h>

#define bank_control1 0x7A
#define io_bank1 0x7B
#define io_bank2 0x7C

int  main (int argc, char *argv[])
{
  /* Get access to the ports */

  iopl(3);
  outb((inb(bank_control1) | 0x01), bank_control1);

  if(argc > 1){
    if(!(((inb(io_bank1))==0x00)&&(atoi(argv[1])==0))){
      outb((inb(io_bank1) | atoi(argv[1])), io_bank1);
    }
  }
  return 0;
}
```

dio_mask_off.c COMPILED C
Clears the bits of arg[0] in the DIO1 port

```
#include <stdio.h>
#include <sys/io.h>
#include <unistd.h>

#define bank_control1 0x7A
#define io_bank1 0x7B
#define io_bank2 0x7C

int  main (int argc, char *argv[])
{
  /* Get access to the ports */

  iopl(3);
  outb((inb(bank_control1) | 0x01), bank_control1);

  if(argc > 1){
    if(!(((inb(io_bank1))==0x00)&&(atoi(argv[1])==0))){
      outb((inb(io_bank1) & (~atoi(argv[1]))), io_bank1);
    }
  }
  return 0;
}
```