

# **Digi-Key / Freescale Design Contest Entry**

## ***Automated Carnivorous Fish Feeding System***

***Contest Entry by Brad Hanken***

### ***Design Concept***

#### **The Problem**

Keepers of carnivorous fish currently lack a technology other fish owners have long had available: an automated way of feeding their pets, either as a convenience while they are away, or as a means to simplify the task of consistently meeting the nutritional needs of multiple tanks of fish. While many products currently on the market provide simple automated feeding systems for small quantities of dry flake food, feeding systems for carnivorous fish has been conspicuously lacking, due in large part to their complex feeding needs.

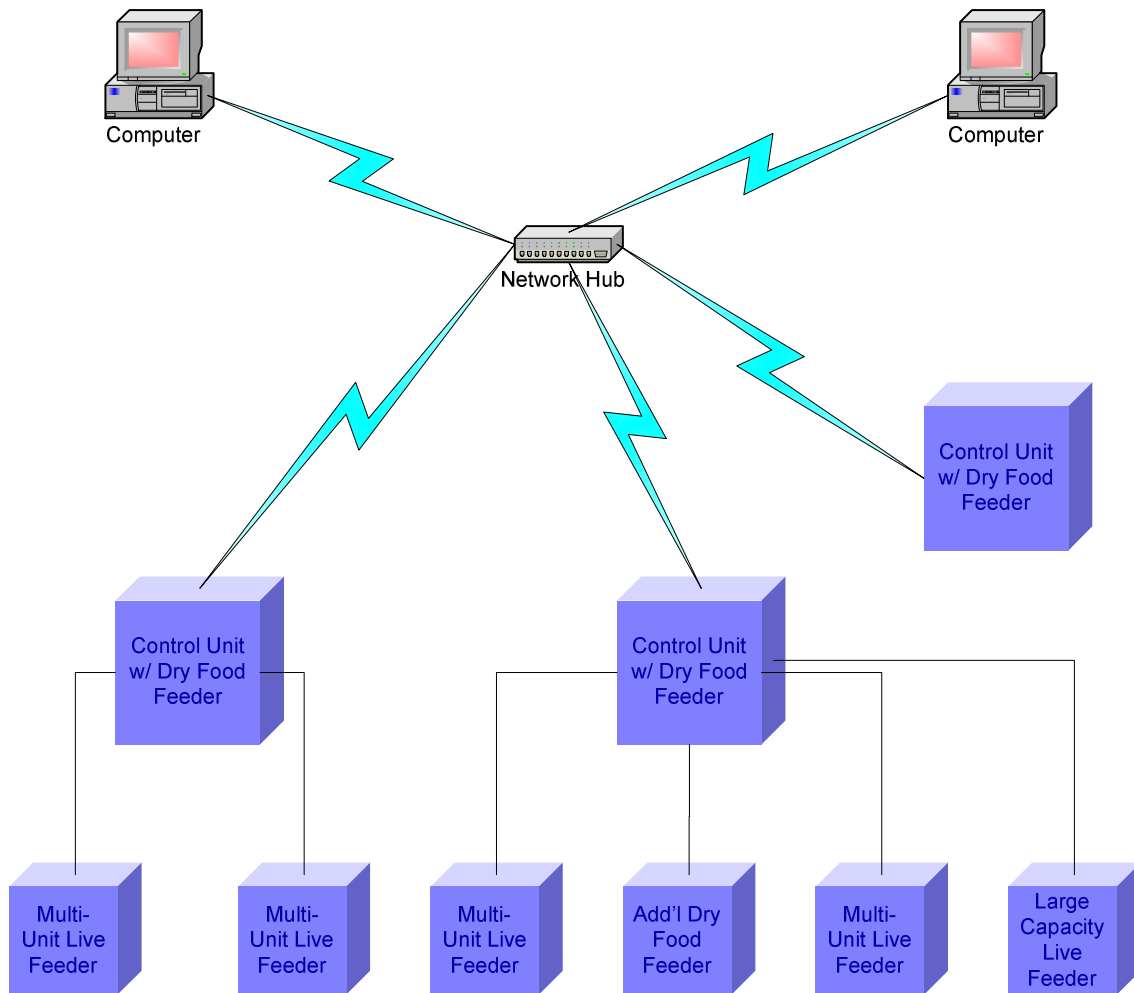
A carnivorous fish feeding system needs to be able to accomplish the following:

- Dispense live food (generally feeder goldfish or guppies) in measured quantities on a predetermined schedule.
- Segregate live food, while keeping them alive and healthy.
  - Food containment areas should be made of materials which allow water to circulate through the containment areas.
  - The live food needs to be automatically fed until the time of release. (Typically, dry flake fish food.)
- Dispense dry food as well, typically large pellet type foods. (Live food is often given as a supplement in combination with a daily schedule of dry food even to carnivorous fish.)
- Allow for the coordination of feeding schedules among multiple feeding systems, either in the same, large tank, or among several, or many, tanks – all with varied nutritional needs.

#### **The Solution – Design Overview**

My entry uses the MC9S12NE64 at the core of a flexible, modular, adaptable feeding solution that would meet the needs of any keeper of large or carnivorous fish. My design calls for a main control unit that houses the MC9S12NE64 microcontroller and a simple character LCD module based UI for initial TCP/IP setup when DHCP is unavailable, as well as for manual feeder configuration and status display. The control unit also houses a large-capacity dry pellet food feeding system to meet the dry food feeding requirement. Scheduling, RTC setting, and monitoring is provided by an application the user runs on any networked desktop. This GUI application allows for easy maintenance of complex feeding schedules across multiple units. EEPROMs in each control unit would serve as storage for the feeding schedules; the desktop GUI would serve only as a scheduling aid, not as a required element for operation.

The above described control unit and software, in and of itself, forms an adequate feeding system for many large fish that don't require automated live food release. However, for aquarium keepers with more complex needs, the control unit also contains an array of connectors designed to inter-connect with live-food feeding modules or additional dry-food feeding modules. Each connection provides SPI communication, as well as regulated power. During operation, the control unit polls any external modules for device capabilities. The modules then respond with their capabilities and status. This modular approach allows a single system architecture to accommodate a variety of system requirements. A user could begin with only the control unit, and enjoy the convenience of an automated dry food feeder. Later, the system could be tailored to the aquarium's expanding needs by adding additional dry food feeding modules, multi-compartment live food modules, large-capacity single-compartment live food modules, or even additional control units, each with an optional combination of additional feeding modules. The user would only have to plug in another module and instantly enjoy the expanded capacity of the system. (See Figure 1)

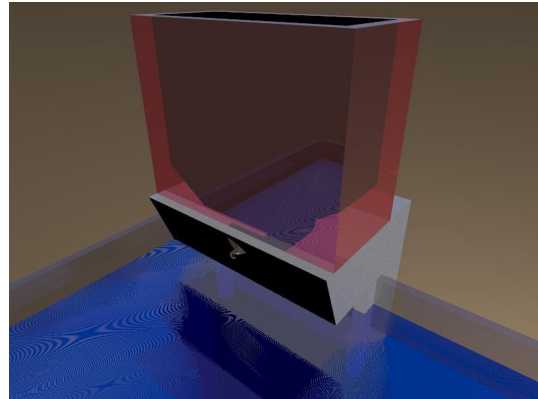


**Figure 1 – A networked, modular approach provides the user with maximum flexibility.**

## Design Implementation

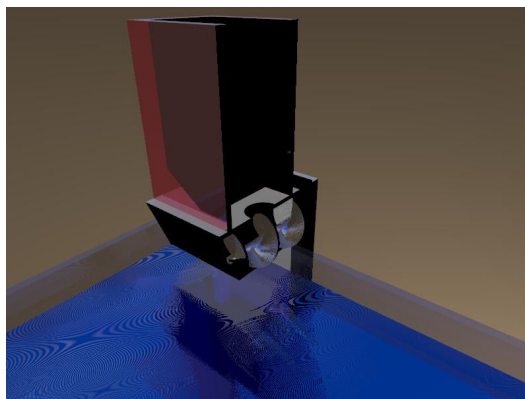
### Hardware Design – Control Unit

The control unit (see illustration, right) consists of four main parts: the PCB housing the MC9S12NE64 and associated circuitry, the front panel housing our LCD, LEDs, and buttons, the electromechanical subsystem for releasing dry food, and the external connections for power and communications. Figure 2 provides a basic block diagram of the unit. We still have ample I/O available for integrating additional dry-food or live food feeding support directly into the control unit. However, for the purpose of demonstrating what I believe would be a viable, popular consumer device, a basic control unit combined with a high level of expandability seems the more appropriate choice.



The main purpose of the control unit is to dispense dry food and coordinate the dispensation of dry and live food via other feeding modules. In order to accomplish this, it also needs to keep time, store feeding schedules, and interact with users through both a simple LCD and pushbutton interface, and through the network in a predefined TCP/IP based protocol the desktop UI can access. These tasks, and the manner in which I will implement them, are outlined in the table below.

Task	Implementation
Dispense dry food	Connect general purpose I/O from the MC9S12NE64 to a stepper motor driver. This will drive a stepper motor connected to an auger, which will be capable of dispensing large dry food pellets. (See cutaway view below.)
Keep Time	Poll an RTC, such as the DS1305, via SPI.
Store Feeding Schedules	Utilize an SPI EEPROM, such as the AT25HP512.
Front panel UI	Connect to any typical parallel-interface character LCD module with GP I/O. Provide additional visual cues of operation to user via LEDs connected to GP I/O. Provide simple menu selection keys using GP I/O.
Communicate with desktop UI	Use the MC9S12NE64's built-in ethernet controller.



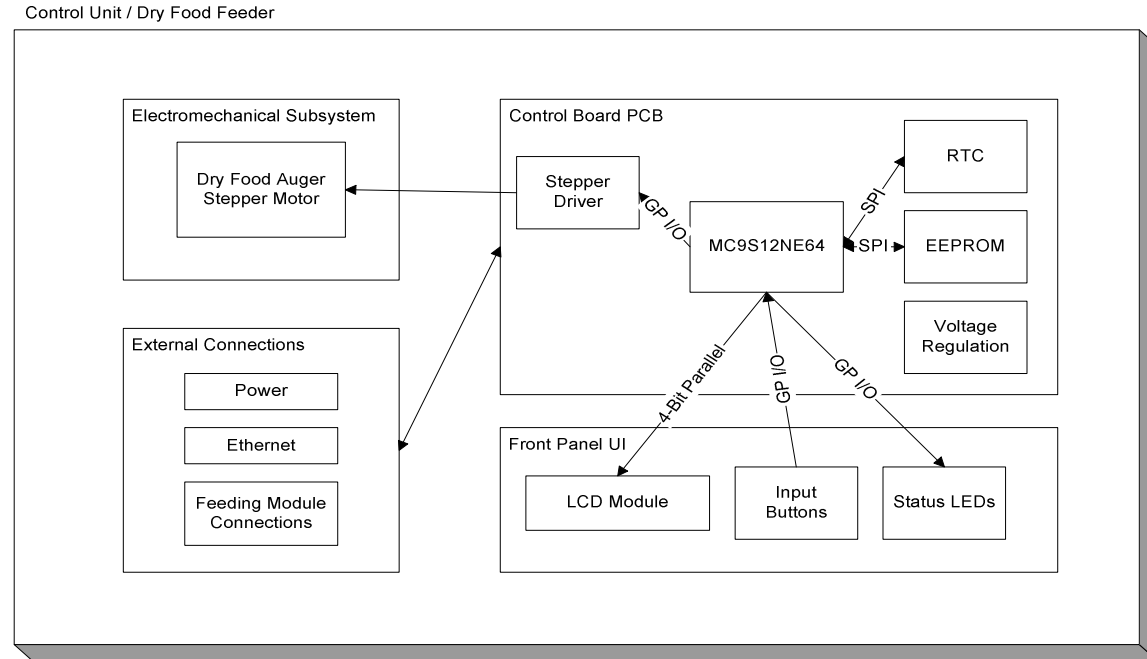


Figure 2 – Control Unit Block Diagram

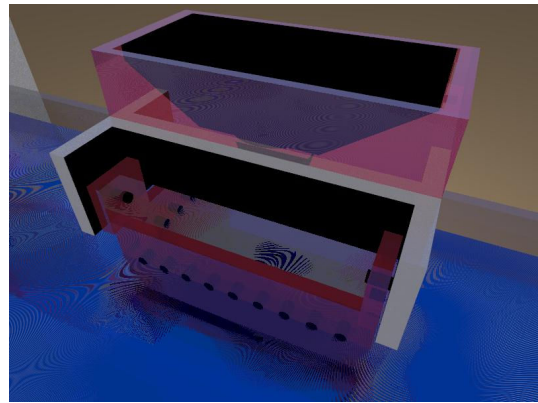
### Software Design – Control Unit

Upon initialization, the microcontroller will first check for stored settings in the EEPROM. If none are found, the system will first set all settings to the default values of no food schedule, and DHCP for IP setup. Once settings have been loaded, control is passed to the supplier of the bulk of the software for this project – the IP stack supplied by the OpenTCP project. Once the TCP stack has been initialized, no business logic is actually run until one of several interrupt vectors are called. They are summarized below:

Interrupt	Logic
Data Received from IP Stack	Several commands will be supported: <ul style="list-style-type: none"> <li>• Set RTC (send time / date)</li> <li>• Clear schedule (per feeder output)</li> <li>• Add scheduled feed (send time and output)</li> <li>• Query capabilities (receive output list)</li> <li>• Query status (receive filled and position status)</li> </ul>
Periodic Timer Interrupt 1	Poll RTC. Check to see if a scheduled feed is past due. If so, commence feed either locally or via SPI and store time processed in EEPROM.
Periodic Timer Interrupt 2	Increment stepper position if currently dispensing dry food locally.
Keyboard Interrupt (Generated by basic buttons for interacting with LCD UI – up, down, left, right, and select.)	Perform action determined by current state. Operations to be supported: <ul style="list-style-type: none"> <li>• Update network settings. (DHCP on/off, set static IP &amp; mask)</li> <li>• Basic setup tasks. (RTC set, basic scheduling)</li> </ul>

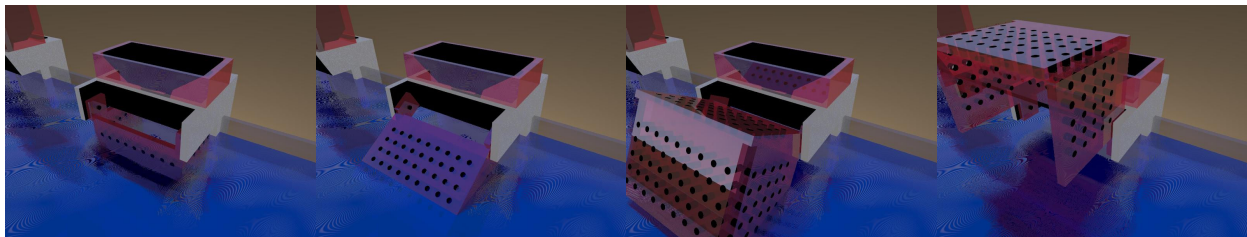
## Hardware Design – Feeding Module

The feeding modules consist of an SPI capable microcontroller, the required electromechanical equipment required to dispense the food, and an extremely basic UI. Because of the limited general purpose I/O requirements, combined with the desirability of a hardware-based SPI interface, I believe Freescale's new addition to the Nitron line, the MC68HC908QB4, would be ideally suited. The combination of low cost, low pin-count and hardware communications peripheral support make it an ideal choice.



The feeding module consists of four distinct sections, as summarized in Figure 3. Because of the limited requirements of a feeding module and the presence of regulated power from the control module, the PCB is quite simple, consisting only of the microcontroller and a few passive components. The front panel UI is likewise simple, consisting only of a button and LED indicator to toggle the presence of food in the feeder, as well as a manual release button to manually request that the system opens or closes the feeding bay.

As discussed in the design overview, an individual feeding module can have a variety of configurations. The actual number and type of feeding capabilities is reported to the control unit in a standard format, so that the system can be easily expanded upon. For the purposes of this contest, however, I intend to build a simple, single-bay, large-capacity live food feeding module. As illustrated above, right, the physical makeup of the module consists mainly of a live food bay that normally remains closed and submerged, and a small flake food hopper and dispenser unit for keeping the live food fed. While the bay is occupied, the feeder fish remain alive and healthy by sharing the aquarium water through a perforated or porous bay. The feeder fish are fed on a regular, configurable schedule from the flake food hopper above. Because flake food is considerably easier to dispense than large, hard, dry food pellets, an auger-type dispenser is unnecessarily complex. A simple rod with a hole drilled into the side will be rotated between the supply hopper and an opening to the water below to dispense the food. A simple PWM driven servo will suffice for this job. At feeding time, another servo connected via an actuating rod to the bay will rotate the entire bay out of the water, releasing the fish. The maximum lift height could be adjustable to accommodate the space limitations of individual aquariums. See sequence, below.



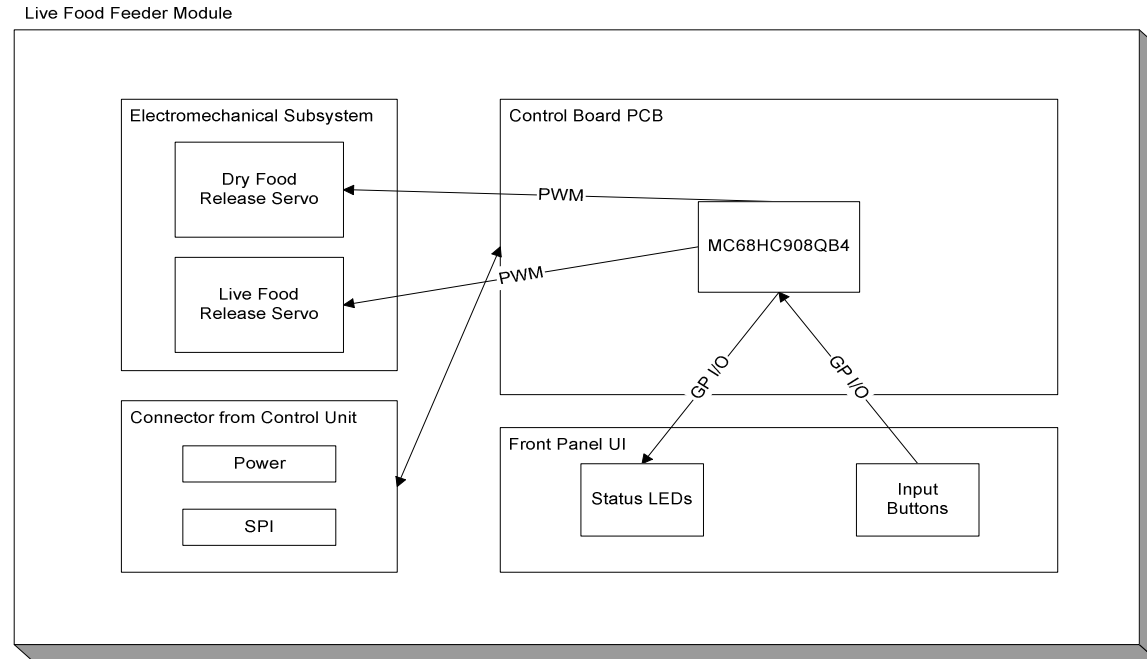


Figure 3 – Feeding Module Block Diagram

### Software Design – Feeding Module

Upon initialization, the microcontroller will wait for a command to be received from the control unit. The control unit stores state information for all feeding modules in non-volatile memory, and so will be responsible for sending initialization commands to each feeding module. Once initialized, the microcontroller will await one of the following interrupts:

Interrupt	Logic
Data Received from SPI Interface	Several commands will be supported: <ul style="list-style-type: none"> <li>• Open / Close live food bay</li> <li>• Actuate flake food release servo</li> <li>• Set occupied status</li> <li>• Query occupied status</li> <li>• Query capabilities</li> </ul>
Periodic Timer Interrupt 1	Actuate flake food release servo if required by current state.
Periodic Timer Interrupt 2	Service servo PWM signaling
Open/Close Button Keyboard Interrupt	Open bay if closed, close bay if opened. Toggle status LED to reflect state.
Occupied Status Button Keyboard Interrupt	Toggle status LED to reflect state. Store status in memory and report if requested by control unit.

## Software Design – Scheduling Software

The scheduling software for this system will be written in Java, using the SWT GUI library. This combination has the benefit of allowing the software to be easily run on all major desktop platforms, with the look and feel of the platform it is being run on. It can also still be compiled using GCJ, for ease of distribution and installation for several platforms, including Windows.

On startup, the user will be presented with a familiar, windows explorer like interface. On the left side of the screen the user's control units and modules will be displayed in a Tree control. A root element would contain all the known control units. A simple dialog would be available for adding recognized control units. The user would have the option of automatically connecting with all recognized control units upon launching the program, or connecting to each control unit manually by clicking on it. Each control unit element could then be expanded to show its connected modules. Once selected, the control units would be queried, and the right side of the screen would display any active schedules. If the root element were selected, all schedules for all control units would be displayed. If the user selected a control unit, or a single feeding module, the schedule list on the right would be paired down accordingly.

While the program is open, the user could add or delete feeding schedules by manipulating the scheduling window using a familiar, intuitive interface. When these operations are selected, the program would immediately issue a TCP/IP command to the connected control unit to update its schedule, making the new schedule active immediately. If the user desires, an immediate feed command could also be issued, causing an immediate release of food.

## Conclusion

The MC9S12NE64 microcontroller opens up the world of network connectivity to reasonably priced consumer devices that have not traditionally enjoyed that benefit. There is a wealth of consumer devices out there that could benefit from being network connected but have not due to cost limitations. The emergence of low-cost network connectivity will no doubt accelerate the "wired home" so often envisioned but never quite grasped. The device outlined in this contest entry is an example of a step towards that vision.

