DEVELOPMENT OF A MICROCONTROLLER SYSTEM FOR A SENSOR NETWORK USED TO STUDY BEHAVIOR OF *TACHYCINETA BICOLOR*

A Design Project Report

Presented to the School of Electrical and Computer Engineering of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering, Electrical and Computer Engineering

Submitted by

Yichi Zhang

MEng Field Advisor: Bruce Land

MEng Outside Advisor: David Winkler

Degree Date: May, 2013

Abstract

Master of Engineer Program

School of Electrical and Computer Engineering

Cornell University

Design Project Report

Project Title: Development of a microcontroller system for a sensor network used to study behavior of *Tachycineta bicolor*

Author: Yichi Zhang

Abstract:

This report describes the development and implementation of a microcontroller system that is used in a sensor network that is used to study the behavior of tree swallows (*Tachycineta bicolor*). The microcontroller system, traditionally referred to as a “webplug”, is used to measure temperature at nestboxes, control a Peltier device to simulate artificial climate change at the nestboxes, and provide access by HTTP to control and access webplug data. Attrition of original webplugs from past field seasons requires new webplugs to be made; many of the crucial components in the original webplug being obsolete required the design of a new webplug. A new webplug was developed using a TS-7553 single board computer (SBC) from Technologic Systems. A separate printed circuit board (PCB) was designed and constructed to complement the SBC to provide functions not found on the SBC, including Peltier control and temperature measurement. The new webplug implements all current webplug functions, include measure nestbox temperature, control Peltier to simulate climate change, and host web server to provide HTTP access. In addition to original webplug functions, the new webplug could also access RFID reader data used in the network to provide real time RFID data. A total of 15 new webplugs were constructed and will be deployed in the 2013 field season.

EXECUTIVE SUMMARY

This report describes the development and implementation of a microcontroller used to simulate climate change in artificial nestboxes of tree swallows (*Tachycineta bicolor*) and to collect data that reflect changes in their behavior. The microcontroller is to be used as a part of a sensor network that also includes the use of cameras and RFID readers to study how climate change affect tree swallow behavior during their breeding season. This project stems from a larger project that aims to develop a sensor node network that collects data on the behavior of tree swallows to study how simulated climate change could affect their behavior. A microcontroller system, dubbed a "webplug" is used in the network to implement climate change and collect data at each nest. A combination of webplug attrition and its parts becoming obsolete requires the design of a new webplug.

The new webplug was implemented using a single board computer, the TS-7553, from Technologic Systems. The software aspects of the webplug were implemented in a combination of C, Python, and PHP, as appropriate to function. A separate PCB will be made to be attached to the TS-7553 to accommodate functions not offered in the TS-7553, such as an H-bridge to control Peltier, and thermocouple amplifiers and digitizers.

The new webplug implements all current webplug functions, including controlling a Peltier device, collecting data and interfacing with a central server for webplug status update and data transfer. The new webplug can also interface with the RFID reader to collect RFID data. Other considerations for the new webplug includes its backward compatibility with current webplug to avoid confusion of using two different devices during field season during transition years when the current webplugs are being phased out.

A total of 15 new webplugs were produced and will be used in conjunction with the original webplugs in the 2013 field season.

Table of Contents

INTRODUCTION

The purpose of this project is to design and develop a microcontroller system that is to manage functions of nodes in a sensor node network used to study the behavior of tree swallows. The goal of this document is do describe the process and consideration involved in the microcontroller system design.

Background

This project stems from a larger project that aims to develop a sensor node network that collects data on the behavior of tree swallows (*t. bicolor*) to study how artificially induced climate change could affect their behavior. Each node consists of temperature sensors, a camera, and a Peltier device to simulate climate change by adjusting the nest temperature relative to ambient temperature.

Currently, the nodes are connected to two central servers in an expanded star formation and are managed by a microcontroller, historically referred to as a webplug (WP), that controls the Peltier device, collect data from the temperature sensors, and transmit it back to the central servers (see Figure 1). The cameras used are security cameras with FTP capabilities that transfer images from the nests via FTP directly to the central servers and operate independently of the webplug. The temperature sensors used are a pair of thermocouples. Data and images collected from each individual node are sent to a hub via Ethernet cable and are sent back to the central server through Wi-Fi via a wireless bridge.

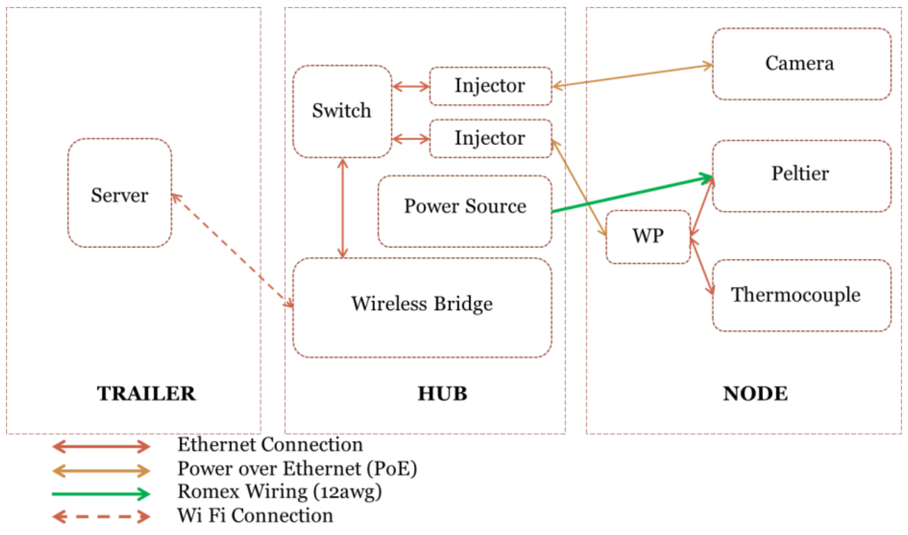


Figure 1: Schematic of network setup

The current webplug is based on an experimental web-based embedded system interface developed by HP and is based on IEEE 1451.2. The project was developed in the late 90's but was cancelled before a final product was produced. The sponsors of the sensor node project managed to salvage a number of the prototypes and developed them to the webplugs currently used in the network. Each webplug currently consists of an H-bridge to control the Peltier, thermocouple amplifiers/cold junction compensators/analogue to digital converter (ADC) to read data from the thermocouples. Provisions were also made to accommodate RFID reading capabilities via an ATA2270-EK1 RFID kit.

The current version of webplug has been used for 4 field seasons. Over the past field seasons there have been significant attrition of webplugs due to damages that were not repairable. As of right now there are only 8 webplugs that are functional. While it is sufficient to establish a network to conduct experiment over the course of a season, the inevitable breakage means that there needs to be backups ready, and to continue the experiments in the future more webplugs would be needed. Multiple components from the current webplugs, such as the H-bridge, and most importantly, the web-interface device, however, are now obsolete. A new webplug design is needed. The focus of this project is to design and develop a new webplug that has basic webplug functionality and serve as a platform for further expansion and development.

DESIGN CONSIDERATIONS

The new webplug must be capable of the functions of the current webplug. The functions include:

- Measuring temperature from type J, T and K thermocouples
- Controlling a Peltier by changing current polarity and turning it on/off as needed
- Access an external RFID reader
- Interface with central servers to upload thermocouple readings to servers and for the servers to update Peltier setting.

In providing these functions, the webplug must be able to operate within the same software and hardware settings as the current webplugs. As the new webplugs are designed to function to supplement a lack of current version of webplugs, they should conform to the settings of the current versions of the webplugs. Otherwise effectively two separate networks are needed to have the same number of nodes operational. That would be both time and cost inefficient. It is also hoped that the new webplug would be able to facilitate the development and implementation of new functions in the future.

Software considerations

The current version of the webplugs is accessed and controlled through HTTP. The central servers use a collection of bash and python scripts to manage the webplugs. The scripts are customized to work exclusively with the current version of webplugs used in the sensor node project. Since it would be years before all current version webplugs are phased out, the new webplugs must respond to the scripts on the central servers in the same format as the current version of webplugs.

In addition to simply responding to the scripts, the data output format of the new webplug must also match that of the current version webplug. As over the course of a season large amounts of data are generated from the network, majority of data processing for in-season as well as post season analysis relies on automated methods and consistent data format is crucial to efficient data processing.

Hardware considerations

The current version of webplugs is connected to hubs through Ethernet, which connect to the central servers via Wi-Fi access points. The new webplugs must be able to either connect to the hubs via Ethernet or Wi-Fi.

Power in the network is drawn from 110V consumer power. The hubs, in addition to serving to connecting the central servers to each node, also provide power to the nodes. At each hub, power is drawn from the grid and converted to appropriate voltage levels by either injectors, which powers the cameras and current versions of webplugs via power over Ethernet (PoE) or a DC power source, which powers the Peltiers (see Figure 1). The current version of webplug is capable of drawing power either via PoE or from the DC power source that powers the Peltiers. The new version of webplugs must be able to convert the output from either the DC voltage source or the injector via PoE to a voltage level that it can use.

An external printed circuit board (PCB) is needed to provide the functions that are not natively on the TS-7553. That includes an ADC and amplifer for the thermocouples, a controller to manage the Peltiers, and voltage regulator to convert power from the hubs to provide power for the TS-7553.

HARDWARE DESIGN AND IMPLEMENTATION

Microcontroller

Microcontroller selection is the most crucial decision in the webplug design. The microcontroller’s capabilities will determine how the necessary functions will be implemented and what hardware to choose for those functions. It will also determine what additional features can be implemented. One of the hopes for the new webplug is to be able to eventually provide caching for image and temperature data that cannot be transmitted when a node loses connection to the server. The focus has thus been on microcontrollers with either large internal memory, fast clock speed for acceptable external memory access, or both. Several PIC microcontrollers were considered, as they included significant internal storage as well as capable of single cycle operations. The PIC microcontrollers selected can also be programmed with the PIC programmer, which means shortened prototyping and production time.

The final microcontroller choice was to use the TS-7553, a single board computer (SBC) developed by Technologic Systems. TS-7553 is based on an ARM 9 processor. It is clocked at 250MHz, with 64MB RAM, and comes preinstalled with full version of Debian with a 2.6.24 kernel. The TS-7553 uses a 2GB MicroSD card for storage, which provides ample space for caching data. It supports multiple communications protocols including Ethernet, WiFi, SPI, I2C and UART, which makes facilitating its communication with attached devices and the rest of the network a straight forward task. The ARM processor and memory provide ample processing power to support the need of current webplug functions and any developments in the future. Being able to work in a Linux environment means easy access to other hardware while being able to work in a well-structured and documented environment. The combination of the factors above makes the TS-7553 an ideal platform for this project.

Peltier controller

Using an H-bridge to control the Peltier seems like a natural choice. The H-bridge used in the current version of the webplug was the first choice. It is, however, obsolete. Effort was then focused on designing a webplug. Given the wide variety of robotics projects online and H-bridge is an integral part of robotic motor control, the endeavor seemed well supported.

The Peltier device draws about 5A at 14VDC. The most important concern in the H-bridge design is to be able to control the Peltier with the microcontroller, which has digital input/output (DIO) pins capable of only 3.3 V at 0.01A. Four MOSFETS were used, two P channel and two N channels (see Figure 2). The MOSFETS are positioned so that full H-bridge control can be achieved with two input pins. It is important to make sure that both leg of the H-bridge are not closed at the same time, which would potentially damage the H-bridge. It is also important that when the webplug boots up, whatever output control (high) from the webplug does not turn on the H-bridge. By placing the MOSFETS as seen in Figure 2, both functions are achieved.

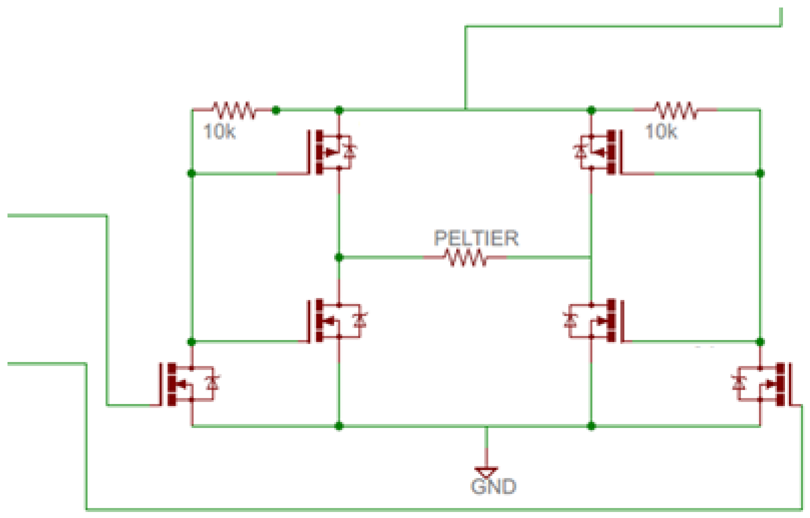


Figure 2: H-bridge design.

In implementing the H-bridge design, the N channel MOSFET used was SI4944, a dual MOSFET in a SOIC-8 packaging, the P channel MOSFET used was IRF9358, a dual MOSFET also in a SOIC-8 packaging. A SI4944 and an IRF9358 was used for all four legs of the H-bridge; an additional SI4944 was used to toggle the H-bridge from the TS-7553.

Thermocouples

The use of thermocouples stems from the reason that all studies done related to tree swallows, at least in the Ithaca field site, are done by thermocouples and it is important that consistent temperature measurement tools are used. The current version of the webplug uses AD595, which is a type K thermocouple amplifier and cold junction compensator. In the author's experience type K thermocouples were never actually used; the thermocouples used tend to be type J or T. The voltage output for type T thermocouple, however, is very comparable to that of type K at the temperature range of interest (0^oC to 40^oC). For consistency a type K thermocouple amplifier and analogue to digital converter was selected (MAX6675). MAX6675 has 12 bit, 0.25^oC resolution and supports serial-peripheral interface (SPI).

Voltage regulation

It is hoped that the new webplug would operate as a wireless unit, thus providing it with PoE does not seem particularly important. The main source of power for the webplug would thus be either the power source for the Peltier, which provides 13.8VDC, or potentially batteries, for which 12VDC would be a reasonable expectation. It is thus necessary to be able to power the device using a power source of approximately 12VDC. The Peltier uses 12VDC and does not need voltage conversion, all other components, including the TS-7553 and MAX6675 would need 5VDC.

A DC/DC converter is used to provide a 5VDC power source. The particular one used in the project is the LMZ12001 by National Semiconductors. It is a buck converter that can output 5VDC and up to 1A.

To establish 5V output, the setup as suggested in the LMZ12001 datasheet was used.

Indicators

To add indication of whether the device is actually working properly, two LEDs were added. One LED is linked to the output of the DC/DC converter and lights up when power is on, to indicate that the device is getting power. A bipolar LED is connected in parallel to the output of the H-bridge, so that it would light up when the H-bridge is on and would change colors when the polarity of the H-bridge changes. It was learned from previous experience that being able to debug webplug in the field without the need for tools such as multimeters reduces the time required to debug, which could minimize human interference at active nests and thus minimize impact on the experiment.

PCB design

In designing the board, consideration was given to ease of mass production. In the first year, it was expected that the author will be the only person to produce the boards. It is also hoped that in the future more copies of the board can be made by people with minimal soldering experiences. Thus all component selection was selected with keeping soldering simple in mind. Thus all passive components such as resistors and capacitors used were through-hole components. It is part of the motivation in designing an h-Bridge instead of buying one premade and in selecting the MOSFETs.

The layout of all components for the external PCB is shown in the appendix. In order to connect to the TS-7553, simple 1” pin connectors are used, since it is available on the TS-7553. Terminal blocks were used to connect other sources, including the power supply to the board (from Peltier power source), power source to the TS-7553 and Peltier, as well as thermocouples (see Figure 3 and Figure 4).

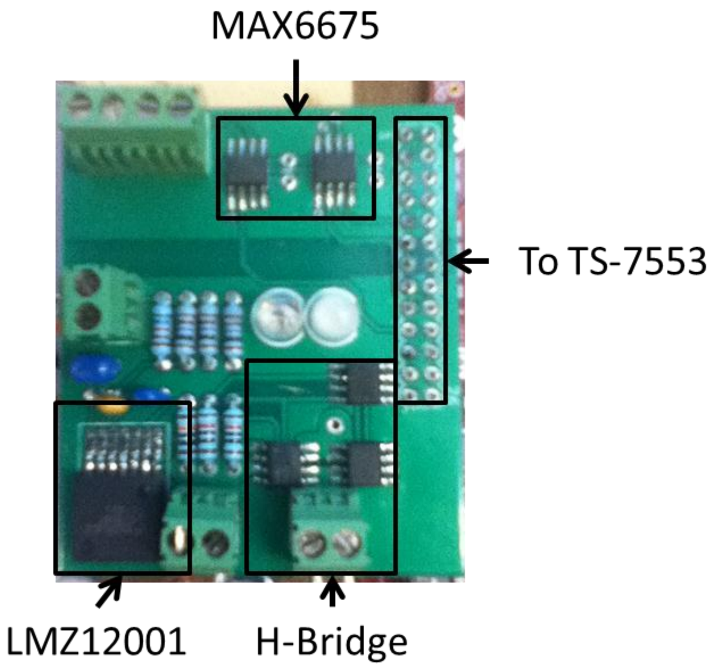


Figure 3: The populated PCB attachment.

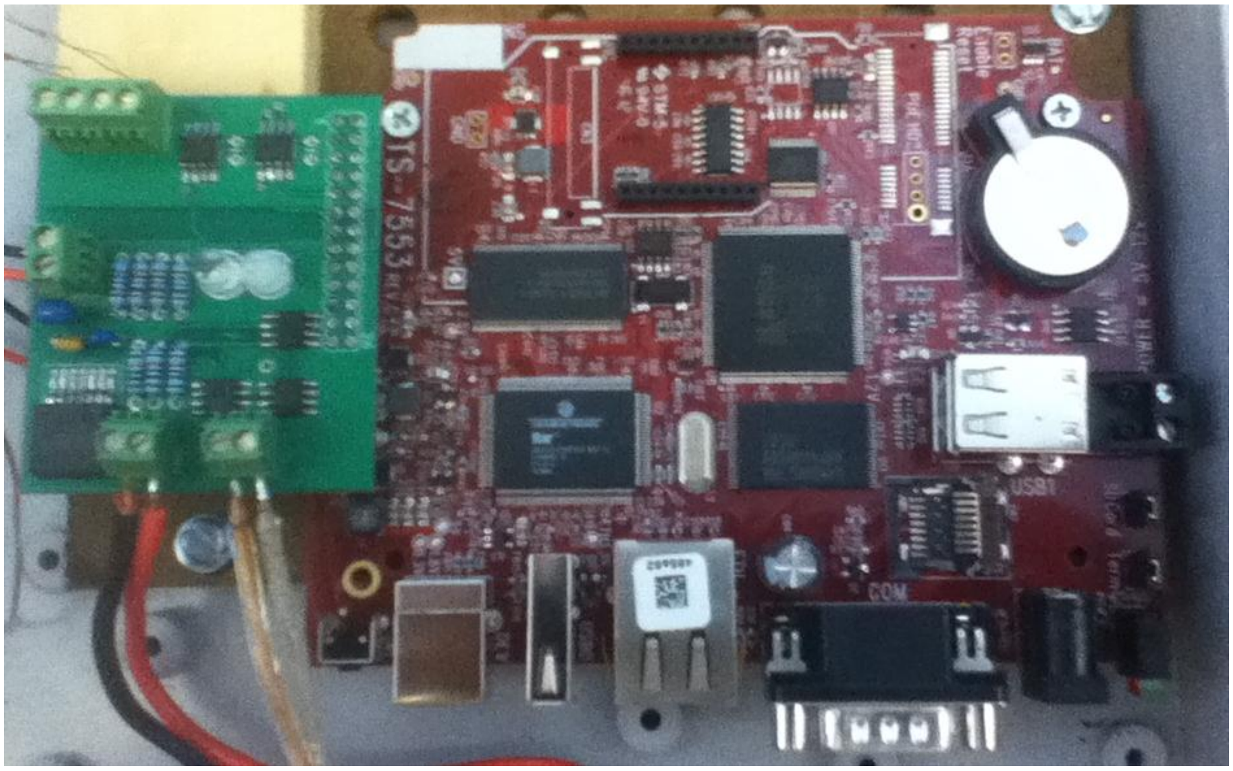


Figure 4: The new webplug. The TS-7553 is on the right (red board), the PCB attachment is mounted onto the TS-7553 (green board).

The following pins from the TS-7553 are used.

Pin	Function
3	Ground
9	SPI MISO
10	3.3 VDC power supply
12	SPI MOSI
14	SPI Clock
18	SPI chip select
19	SPI chip select
23	H-bridge control
25	H-bridge control

Most of the pin selection in connecting between the PCB and the TS-7553 was dictated by the TS-7553

The boards were designed using EAGLE, since there were plenty of tutorials and libraries available. It also offered variety in choosing a board manufacturer, which would not be possible with software like ExpressPCB.

In the choice of board manufacturer there were four choices: Sunstone, Gold Phoenix, Seeed Studios, and OSHPark It ultimately came down to a tradeoff between cost and lead time. Sunstone offered the least lead time, but the highest price, Seeed Studios offered the lowest price (10 2 layer boards for $10) but had a lead time of approximately 20 days, Gold Phoenix offered a reasonable tradeoff between lead time and price, but has a $100 purchase minimum. During prototyping stages, boards were ordered from OSHPark; for the final production of the webplug the boards were ordered from Gold Phoenix. The final PCB ordered used FR4-TG130, 1oz. copper and was a 2 layer board.

External housing

The webplug needs to function outdoors. So a protective case is needed. The current version of the webplug is housed in a plastic case (see Figure 5). The same case is used for the new webplug. To mount the TS-7553 into the case, a peg board was used as an adaptor for the screw mounts in the box and the TS-7553 was held onto the peg board using self-tapping screws. A small piece of wood was glued onto the peg board to provide support for the PCB attached to the TS-7553 (see Figure 6).

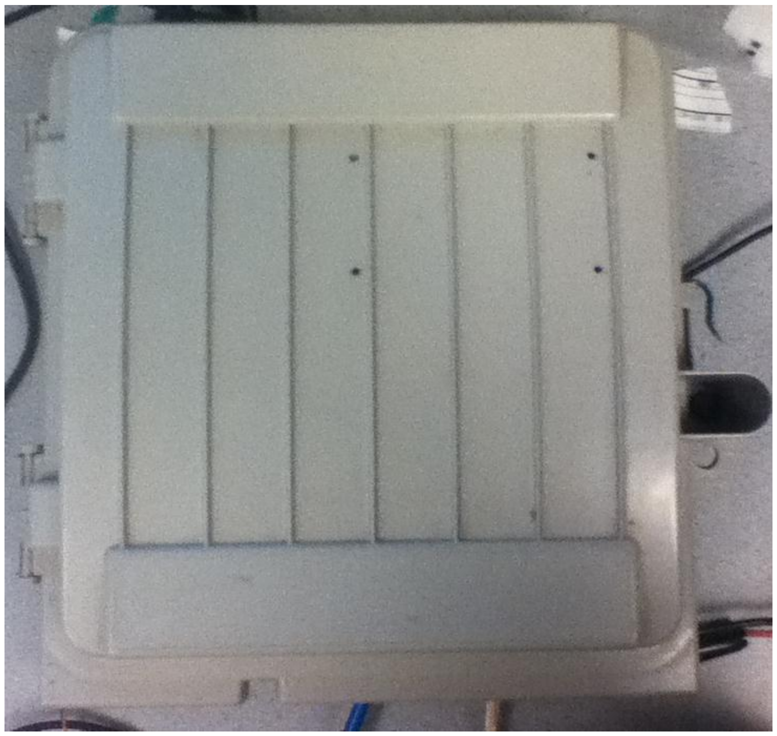


Figure 5: Plastic case used to house the webplug

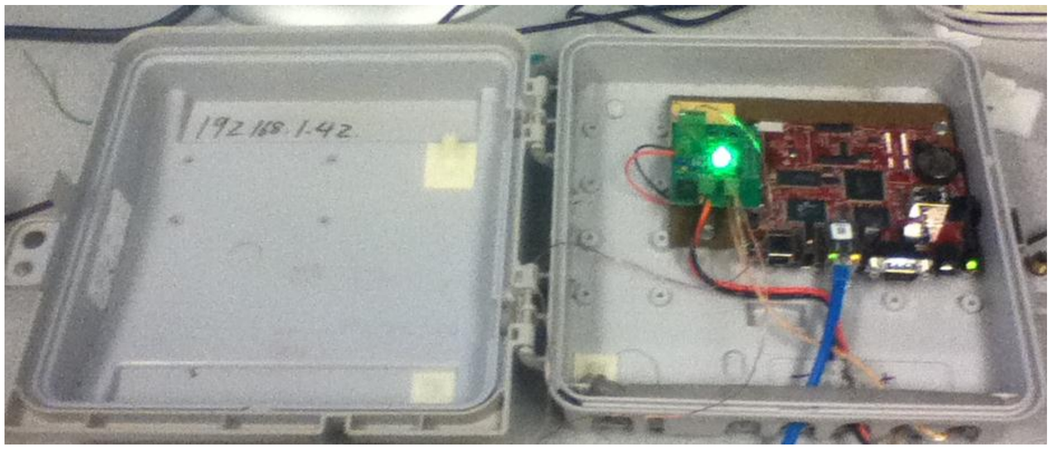


Figure 6: Mounting of the TS-7553 and the PCB attachment in the case using the peg board and wood support for the PCB attachment

SOFTWARE DESIGN AND IMPLEMENTATION

The software in the webplug was implemented using a combination of C, Python, Bash, PHP, and HTTP. In designing the software components of the webplug, the following components are considered:

- Using the TS-7553
- Thermocouple controller
- Peltier controller
- RFID controller
- Task Scheduling
- Webplug settings
- Providing HTTP access to webplug
- User interface

The interactions of the above components are shown in Figure 7.

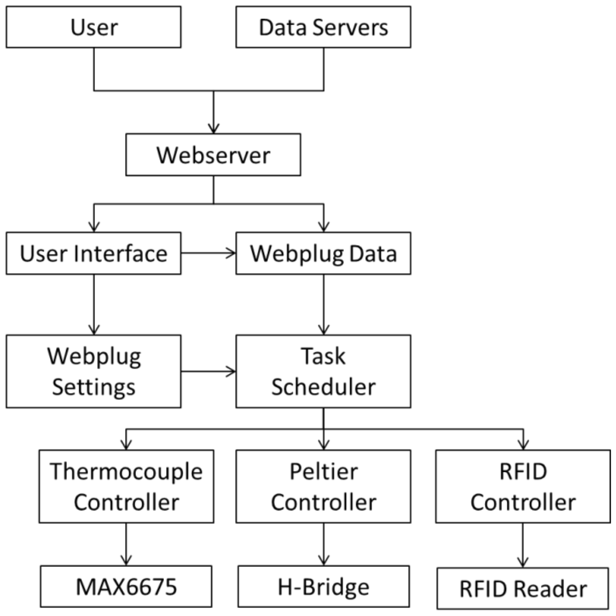


Figure 7: Structure of software used in the webplug.

The components will be discussed in that order. All the code written are listed in the Appendix.

It was initially planned to implement final versions of all software in C and use Python to create an initial prototype. After implementation of the Python prototype it was realized that it was sufficiently fast to handle the needs of the webplug so the C implementation was not pursued.

Using the TS-7553

The TS-7553 can be booted from a microSD card. Technologic Systems has provided an image of Debian Lenny for a 2GB microSD card. That image is used. Upon boot, the provided image boots to a busybox shell. Exiting the shell provides access to the full Debian system. The shell boots up in seconds compared to about 2 minutes for the full Debian system. It, however, does not have sufficient function for the webplug and boot time is not a concern. It was circumvented by removing its boot script so that the system boots to full Debian on startup.

The TS-7553 can be connected to using serial or SSH. To SSH, the busybox shell needs to be disabled. Also, in the original Debian image, the RSA key was removed and need to be regenerated.

The serial connection to the TS-7553 is to the header pins upon boot up. According to Technologic Systems, if the reset button is pressed for 3 seconds the system reboots and changes the serial connection to the serial port on the back of the TS-7553. On the module used for development, it did not happen. Instead, a development connector board made by Technologic Systems, a TS-9448, was used to connect a serial cable to the header pins.

During the course of programming the TS-7553, the root account was used. Several programs had to be installed, including sudo and PHP. The repository for Debian Lenny has been moved to archive and a path to the archive and backports need to be provided. The path to the archive is:

```
deb http://archive.debian.org/debian lenny main
```

The path to the backport is:

```
deb http://archive.debian.org/debian-backports lenny-backports main contrib non-free
```

It needs to be included in the file: `/etc/apt/sources.list`

In the image used, the path was given and simply needed to be uncommented.

Thermocouple collection controller

In the current version of the webplug, the webplug status, including thermocouple temperatures, set point for the webplug, and the Peltier status, was hosted via HTTP. The server processing the data expected the data to be arranged in a HTML table in a certain order. To maintain backward compatibility, a function was written to collect the data needed and store them in a format that matched the format used in the current version of the webplug. `manageData.py` is used to sample thermocouple data, the set point for the Peltier, and the Peltier state, and print all the data in a comma separated file in the format used by the current version of the webplug.

`manageData` is a class. Upon initialization the clock frequency for SPI was defined. It was thought early during development that the clock frequency might need to change, so the option was built it. It includes the following functions

readTC:

This function reads the thermocouple data from MAX6675 via SPI. The TS-7553 has a function called `ts7500ctl` that is used, among other things, to access the pins on the TS-7553. That function is used to toggle the pin select for one of the two MAX6675s. The TS-7553 provided a function called `spictl` to access SPI on the TS-7553. In readTC, Popen from subprocess was used to call `spictl` to access the MAX6675.

The process in reading thermocouple data included:

Chip select:

```
ts7553ctl –poke16=0xbe14 or ts7553ctl –poke16=0xbf14
```

Read 2 bytes from SPI:

```
spictl –readstream=2
```

The result read using spictl was a hexadecimal number as a string. The class `binascii` was used to convert that to a hexadecimal number. The TS-7553 supports Python 2.5, which does not convert hexadecimal numbers to other formats. A dictionary was thus written to convert the hexadecimal numbers to binary numbers. The binary number is made available to other functions in the class as `binnum`. Of the 2 bytes of data read, only the first 12 bits are temperature; those bits were selected and cast to a float. The converted number was then divided by 4 to convert the data to a temperature in Celsius.

CheckTC:

In the two bytes of data read from the MAX6675, bit 14 indicated whether the thermocouple junction was open. If it is 1, then the thermocouple junction is open [check]. It can be used to check whether the thermocouple has become unattached. CheckTC checks bit 14 of `binnum`.

binTC:

This function prints `binnum`. It is used to debug.

checkPeltier:

This function is used to check on the current state of the Peltier. It checks the pins that are used to control the H-bridge, which dictates Peltier state. The function ts7553ctl is again used to read the state of pins 23 and 25.

The command used was:

```
ts7500ctl --peek16  --address=0x68
```

Due to the bus structure of the ts-7553, data are always accessed two bytes at a time. Two bytes of data are read from the register regions that contained pins 23 and 25 at offset 0x68. To simplify processing the data was simply read as hexadecimals and the last 2 hex digits were checked. The hexadecimal and the corresponding states for pins 23 and 25 are listed below.

Hex	**Pin 23**	**Pin 25**
0x2b	0	0
0x3b	0	1
0x2f	1	0
0x3f	1	1

When both pins 23 and 25 are high or low, the Peltier is off. When pin 23 is high and pin 25 is low, the Peltier state was cooling, and when 23 is low and 25 is high, the Peltier was heating.

getSetPoint:

This function read the set point that was assigned to the TS-7553 by the server. The set point is written to a configuration file. The function `getSetPoint` reads the configuration file and parses the set point.

write2file:

This function writes the measured thermocouple temperatures, the Peltier state, the set point and the sample time to a CSV file. The time expected is in Unix Epoch format. The Peltier state is indicated by an integer: 0 for off, 1 for cool, and 2 for heat. The format for the CSV file is as follows:

```
Time, Thermocouple 1, Thermocouple 2, Set Point, Peltier State
```

A separate Python script, `dataLogger.py`, was written to run `manageData` at scheduled intervals. To track the running of the function, the PID of `dataLogger.py` is written to `/home/dataLogger.pid`. `dataLogger.py` consists of two functions, `checkInterval` and `print2file`.

The function `checkInterval` is used to check how often the thermocouple should be sampled. The sampling interval is written to a configuration file, `sample.ini`, at `/var/www`. A custom Python class found online, `SimleConfigParser`, was used to parse the file for the sampling interval.

The function `print2file` creates an object of `manageData` and calls its `write2file` function to log thermocouple data. It uses Timer in threading to call the `write2file` function at intervals set by `checkInterval`. The function is set to run continuously.

Peltier controller

A class, `managePeltier`, was written in Python to implement control of the Peltier. Upon initialization, it defines an object of class `managePeltier` with an offset value. The offset value defines the hysteresis for temperature control. Below are the functions of `managePeltier`.

setTemp:

This function is used to set the Peltier to cool, heat or off. It takes in the set point and the nest temperature. If the nest temperature is greater than the set point by more than the offset, the Peltier is set to cool, if the nest temperature is less than the set point by more than the offset, the Peltier is set to heat. Otherwise the Peltier is set to turn off. The offset typically used is 1^{o}C.

Technologic Systems provided a C function, `pinio.c`, that allows access to individual pin values. For convenience, a compiled version of pinio was saved in /home/python, where `managePeltier` is saved. To set the Peltier to a certain state, Popen from subprocess is used to call `pinio.c` to set pins controlling the H-bridge individually:

```
/home/python/pinio set [pin] [1/0]
```

As described above, if pin 23 is set to 0 and pin 25 is set to 1, the Peltier cooled, if pin 23 was set to 1 and pin 25 was set to 0, the Peltier heated. If both pins were set to 1 or 0, the Peltier was off.

A separate Python script, `peltierControl.py`, was written to run `managePeltier` at scheduled intervals. To track the running of the function, the PID of `peltierControl.py` is written to `/home/peltierControl.pid`. `peltierControl.py` consists of two functions, `checkInterval` and `print2file`.

The function `checkInterval` is used to check how often the Peltier should be set. The sampling interval is written to a configuration file, `sample.ini`, at `/var/www`. A custom Python class found online, `SimleConfigParser`, was used to parse the file for the sampling interval.

The function `print2file` creates an object of `manageData` and calls its `write2file` function to log thermocouple data. It uses Timer in threading to call the `write2file` function at intervals set by `checkInterval`. The function is set to run continuously.

RFID controller

Provisions were made for the TS-7553 to interface with a custom-designed RFID tag reader. The RFID tag reader is attached on the entrance of nestboxes and is used to track RFID tags that were attached to certain birds so as to track when and which birds visited the nestbox. The RFID reader constantly provided an output through UART of the tag read. If no tags are present, it outputs "FFFFFFFFFF".

It was also thought of during the development of the RFID controller that it might be of interest to be able to toggle the camera that is installed on some nestboxes to capture images of the nest based on RFID. Since the camera does not take any external input, a method was devised to use the motion capture function of the camera to monitor a corner of the nest, where an infrared LED would be placed. When the RFID reader registers entrance of a bird, the TS-7553 would toggle the LED to have the camera capture images.

A class, `manageRFID`, was written in Python to read the UART output of the RFID reader. Functions include the following.

setLED:

This takes in an integer as argument. If "1", it calls `pinio` to turn pin 20 high to turn the LED on. If "0" it calls `pinio` to turn pin 20 low to turn the LED off.

toggleLED:

This function calls `setLED` to turn the LED off if it is on and turn the LED on if the LED is off.

write2file:

This function takes in data as argument and writes it to a CSV file.

readRFID:

This function parses the output of the RIFD reader to get the RFID tag value. It also attempts to determine the activity of the bird. It was thought that it would be helpful to track what the birds associated with a tag is doing relative to RFID occurrence data. Since the RFID reader triggers as long as the tag is within vicinity of the reader, the exact activity of the bird (i.e. entering nest, leaving nest, perched) relative to RFID data could not be determined. Instead, a finite state machine was implemented to give an approximation of what the bird could be doing based on the RFID activity (see Figure 8).

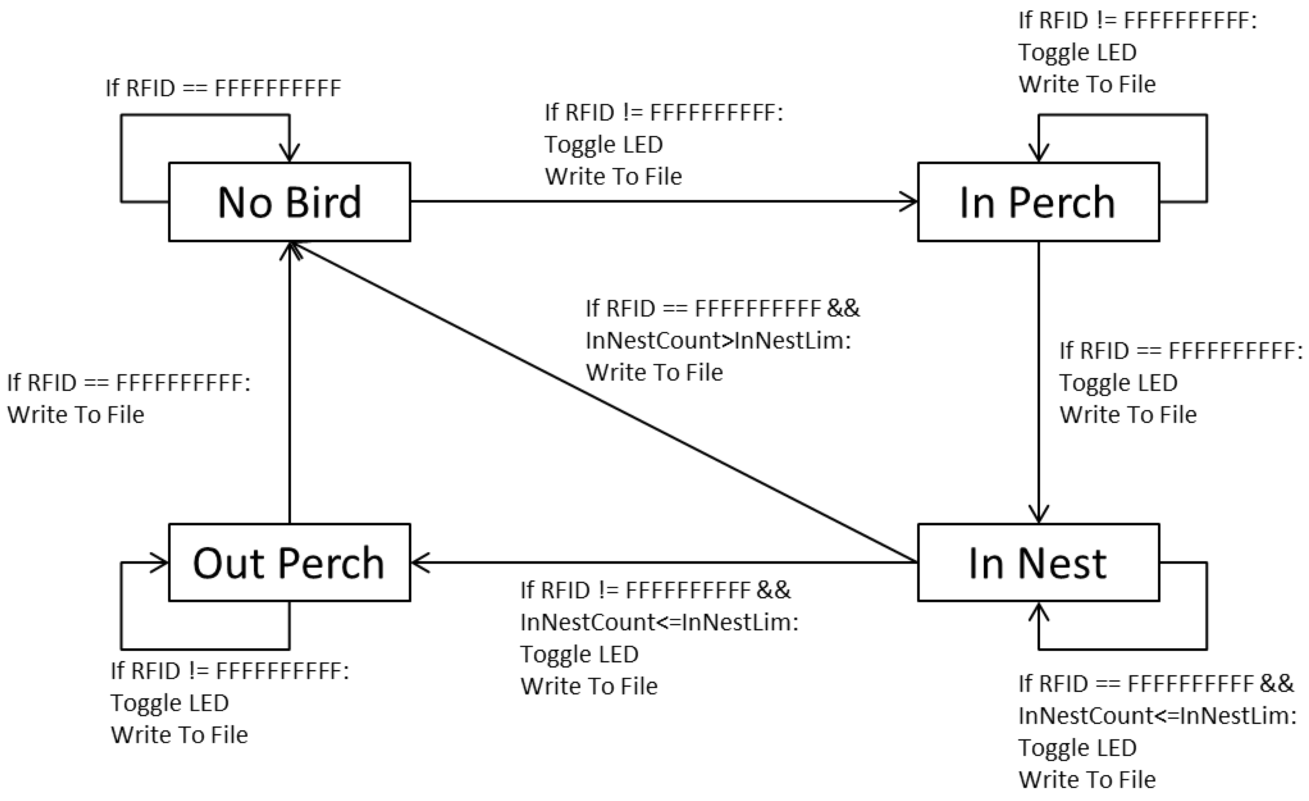


Figure 8: State machine used to estimate state of the bird based on RFID activity**.**

Since the definite states of the birds' activities relative to RFID output cannot be determined, some assumptions were made. Error correction is included in the event the assumption does not hold true.

It is assumed that the state machine will always start with no birds in the nest. The installation of the RFID reader will always scare the birds away from the nest, so this holds well. As long as the RFID reader does not detect a tag, the state machine stays in the "No Bird" state. It is assumed that the first time the RFID reader detects a tag a bird has perched on the entrance to the nest. The state machine goes to "In Perch" state, the LED is toggled and the RFID tag and state machine state is recorded. As long as the tag is present, the state machine stays in "In Perch" state. If the tag disappears, it is assumed that the bird entered the nest. The state machine goes to "In Nest" state. While in "In Nest" state a counter, `InNestCount`, is incremented for every RFID tag check that shows up as no tag. If `InNestCount` is greater than a limit as defined by `InNestLim`, then it is assumed that the bird did not enter the nest at all and the state machine goes back to "No Bird" state. The value for `InNestLim` is set to 5 minutes worth of RFID check. This value needs to optimized by actual field testing. While in "In Nest" state, if a tag appears, it is assumed that the bird is leaving the nest, and the state machine goes to "Out Perch" state. During the entirety of "In Nest" State, the LED is toggled and data is logged. At the entirety of "Out Perch" state, LED is also toggled and data logged. As long as a RFID tag registers, it is assumed that the bird is perched. If the tag disappears, it is assumed that the bird left and the state machine goes to "No Bird" state. It is possible that the bird went back into the nest, but in the author's experience, it has yet to occur.

The file is written to `/var/www/api/rfidout.csv`.

A python script, `RFIDReader.py`, was written to run `manageRFID`. It creates an object of class `manageRFID`. It uses Timer from threading to call `readRFID` function every 10ms to write RFID data to file.

Task scheduling

A Bash script, init7553.sh was written to call `dataLogger.py`, `RFIDReader.py` and `peltierControl.py`. The function of init753.sh is to provide one function that can be called to initiate all functions needed of the TS-7553. To allow the script to be run from the website, both were called with nohup. To make sure neither takes up too much system resources, both were run with nice value of 15.

The TS-7553 provided a function to establish a XUART session using the command `xuartctl`. Before running RFID functions with `manageRFID.py`, that needs to called as follows:

```
xuartctl --server --port=0 --speed = 9600
```

This initiates an XUART session at speed 9600Hz and mounts it to `/dev/pts/0`.

Webplug settings

Settings need to be made per webplug includes the interval to sample thermocouples, interval to adjust Peltier state, the IP address of the webplug, and the nestbox to which the webplug is associated. All settings are written to an .ini file, `sample.ini` and saved at `/var/www`. As each function needs different values, the sample.ini is accessed. Since the content of sample.ini can only be modified manually by the user, there is no concern of read/write protection.

Providing HTTP access to webplug

To provide HTTP access to the webplugs, Apache server was installed on the TS-7553. The Debian image provided for the TS-7553 by Technologic Systems has Apache2.2 preinstalled. To provide all needed functionality all that was needed was to install PHP. PHP5.5 was installed.

To interface with the original webplug, several scripts were written in a combination of Bash and Python to access several functions of the webplug's API. The following functions are used and should produce the exact output as the original API so as to avoid rewriting the entire server side operation of the network.

The following functions were implemented using PHP on the TS-7553 and are separated as individual files in appropriate directories. Since the server scripts called to API functions and not so much different pages, .htaccess was used to allow the pages to be called without the .php extension.

Read:

This function is really only used by the user to get the latest status of the webplug, including the temperatures, Peltier setting, and Peltier set point. The call to it is:

```
http://[IP address]/api/read
```

In the TS-7553 it was implemented by using the tail function to get the last line in the CSV file that stored webplug data.

loggerData:

This function is used to display all data stored on the webplug. The original webplug stored 2048 lines of data. The server script uses this to collect data from the webplug. It collects five minutes worth of data every time it is called and filters out redundant entries based on time. A function was written in PHP to access the last 6 minutes of data, based on the assumption that thermocouple data is sampled at 15 second intervals (that is based on the limitation of other equipment that is used in combination with the webplug and it is highly unlikely that it will change. If it does, it is only a matter of changing the number of lines printed accordingly). This function is accessed through:

```
http://[IP address]/api/loggerData
```

Time:

This function is used to set the system time on the webplug to ensure that the time on the servers and all the webplugs are consistent. With the original webplug the server needs to check the webplug time every so often to make sure everything's synchronized. On the TS-7553 NTP could be used to synchronize time, but the server script managing webplugs will not run unless it successfully performs a time check using this function. To maintain backward compatibility with the original webplugs, this was implemented in the TS-7553 as well. The format the time setting function is accessed is:

```
http://[IP address]/bin/node/time?yyMmDd=[time]
```

Where the time is entered in the following format:

yyyyMMddhhmmss

Where:

yyyy: year in 4 digits

MM: month in 2 digits

dd: day in 2 digits

mm: minutes in2 digits

ss: seconds in 2 digits

Initially the time setting was simply accepted and nothing was done with it, since the TS-7553 kept system time sufficiently precise given initial manual synchronization with the server time. Later time setting function to actually change system time to be that set by the server was implemented using `shell_exec` to run `date` as follows:

```
sudo date --set [Hour]:[Minute]:[Second]
```

Settings related to changing system parameters such as system time and IP address required sudo privilege. By default PHP runs commands using the “www-data” user, which does not have sudo privilege. The user “www-data” was added to the sudoers file and the sudo configuration file was changed to allow “www-data” to use sudo without a password. The Debian version that was used did not have sudo installed. Sudo had to be installed.

Provision to set the day was considered, but since it never seems to be off it was not added for simplicity.

setPoint:

This function is used to set the set point for the Peltier. It writes the set point to a configuration file, `wp.conf`, saved at `/var/www`. It is called by:

```
http://[IP address]/api/setPoint?newValue=[new set point]
```

User interface

A user interface (UI) was implemented in HTTP and PHP so common webplug operations can be done without needing to use command line interface.

All of the display was done using HTTP and organized using tables. All functionalities are provided using PHP.

The user interface consisted of the following pages: main page, data settings, time settings, read, loggerData, RFID, and Download.

Main Page:

The main page provided an overview of the webplug status, including nestbox number, IP address, intervals for checking thermocouple and setting Peltier. It also allowed the user to start webplug.

Data Settings:

The data settings page allowed the user to set the IP address, the nestbox number, the interval to sample the thermocouple and the interval for setting the Peltier state. It also included an interface to manually set the Peltier state for testing purposes that is made accessible when the webplug is not collecting data.

To set the IP address, PHP was used to call ifconfig:

```
sudo ifconfig eth0 [new IP address]
```

To manually set the Peltier, a Python script, `setPeltier.py`, was written to set the H-bridge pins and was called using PHP. `setPeltier.py` took in one argument: the Peltier state to set the Peltier. Following convention from before, the state was defined as an integer: 1 was for cool, 2 was for heat, and 0 was to turn off. The user input was done using radio buttons. To make sure that the radio button state reflected the current state of the Peltier, `ts7500ctl` was accessed through PHP to get the pin settings for the H-bridge control pins. To make sure that temperature regulation was not occurring while the Peltier was set manually, existence of the PID file for dataLogger was checked. If the file exists, it means that temperature regulation was occurring and the option of manually setting the Peltier state was disabled by disabling the selection buttons.

The settings for IP address, thermocouple sampling interval, Peltier setting interval, and nestbox number were written to sample.ini.

Time Settings:

This page allowed the user to set the system time on the webplug. It was done separately from other settings to maintain backward compatibility with the original webplug, since in the original webplug the time settings were done separate. The page is really used by the server to define system time, but the UI included a verbose function where a user accessible interface can be used to manually set the system time.

Read:

In the current version of the webplug, one of the functions offered was to get the latest status of the webplug, including the temperatures, Peltier setting, and Peltier set point. The page was made accessible through the UI for convenience.

loggerData:

Same as Read, a link to loggerData is simply provided for convenience.

RFID:

This displays the logged RFID data. Unlike temperature data, this has no precedence and did not have to conform to a prior format. The page uses PHP to display the last n number of RFID tags read as selected by the user. The number of lines read can be either selected through the form or passed in as part of a GET statement as:

```
http://[IP address]/api/rfid?nlines=[number of lines]
```

Download Data:

To facilitate transfer of large amount of data from the TS-7553, a link to the CSV files for both temperature data and RFID data was provided.

Prototype and testing

A prototype was built and tested to make sure everything worked as planned. The fabricated PCB and the TS-7553 was able to measure temperature of the 2 attached thermocouples and the PCB was able to reliably provide a 5VDC power source for the TS-7553.

Getting proper function of the H-bridge, however, was not as straight forward. An initial problem was the improper poring of the ground layer that resulted in the MOSFETs pins being not grounded. During testing, rework wires were used to provide ground. Three separate prototypes were made to address the issue. On the third iteration, a properly functioning PCB was made.

During initial testing, the H-bridge was used to toggle the indicator LED. It worked as expected. When the Peltier power supply was connected to try to toggle an actual Peltier, however, the MOSFETs sizzled and burned. A current limited power supply was used to test the H-bridge design and the voltage drops across all MOSFETs were measured. The failures were due to defect in the MOSFETs that were most likely due to heat damage during soldering. The symptom was the H-bridge shorting. Depending on how the MOSFETs were damaged, it has occurred when power was turned on or when the H-bridge was toggled between states. When properly soldered, the H-bridge was able to toggle the Peltier properly.

Measuring with a multimeter showed that when the input voltage was 13.5VDC, the output on the H-bridge to the Peltier was consistently around 13.4VDC, suggesting that the H-bridge dissipated approximately 0.1V and it is 99% efficient.

The DC-DC converter output was measured before it was used to supply power to the TS-7553. It consistently output 5VDC.

The thermocouple reading was verified by pinching the thermocouple between thumb and forefinger and measuring the temperature for approximately 5 minutes and then simply measuring room temperature for about 5 minutes. The idea is that if the thermocouple reading was accurate, the temperature output when the thermocouple was held should go up to around 36^{o}C, body temperature, and then should return to about 23^{o}C, room temperature. Both did.

Peltier regulation was checked by taping the thermocouples to the operating side of the Peltier and setting the set temperature for the webplug to be above or below ambient temperature. Ideally the Peltier would start cooling if the set temperature is lower than ambient and would start heating if the set temperature is higher than ambient. Both should stop if temperature measured is within 1^{o}C of the set point. The webplug performed as expected.

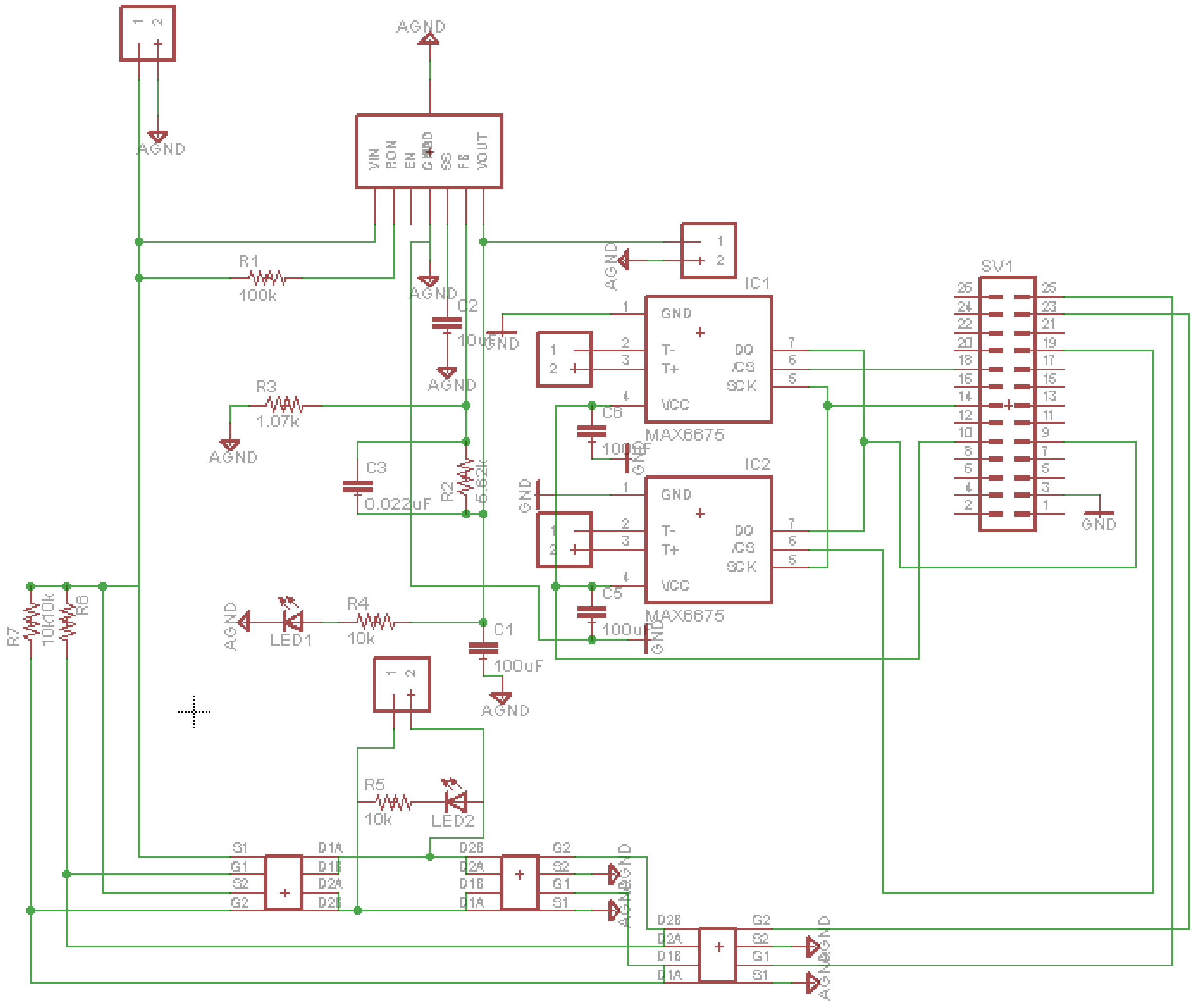
MASS PRODUCTION

To reduce cost of the webplug, all 15 PCBs were made by the author. During the manufacturing process, all boards were tested as they were constructed. Given the problem with the H-bridges during prototyping, care was taken to ensure that the H-bridge worked properly. A 12VDC, 1.5A power adaptor was used as a current limited power supply for initial testing. The board was tested when the MOSFETs, the pull-up resistors and a terminal block for power input was placed. If there were no short, then the input voltage should be very close to the voltage measured on the higher legs of the H-bridge. Once that was assured, the rest of the board placement continued. When the entire board was assembled, the current limited power source was again used. If the H-bridge performed as expected, the regular Peltier power supply was then used to test Peltier control.

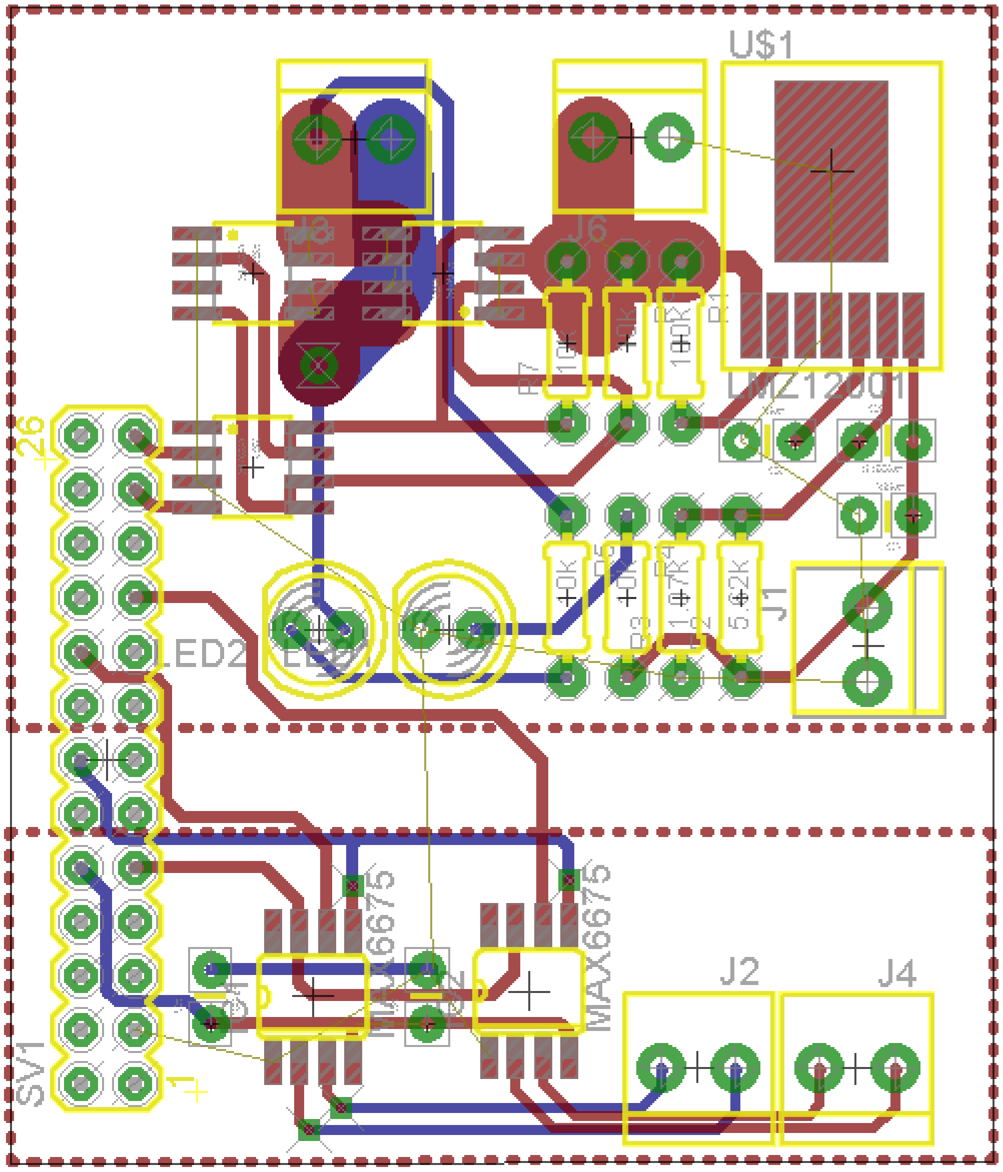
Since the entire system operated from a microSD card, an image of the card was made and written to new 2GB microSD cards using the `dd` command.

APPENDIX

Schematic of attachment PCB to TS-7553



Board layout of attachment PCB



Python Code

managePata.py

```
#!/usr/bin/python2.5
import os
import time
import calendar
from subprocess import Popen, PIPE
import binascii

class manageData(Popen):
    def __init__(self,freq):
        self.clk = freq
        self._child_created = False #Using Popen as a class creates an
exception; this avoids that.
    def readTC(self,lun):

        b = "--readstream=2"
        s = "spictl"
        if lun == 2:
            r40 = "--poke16=0xbe14"
        else:
            r40 = "--poke16=0xbf14"
        adr = "--address=0x40"
        t = "ts7500ctl"
        Popen([t,adr,r40])
        time.sleep(0.001)
        self.spiread = Popen([s,b],stdout=PIPE).communicate()[0]
        bins = {"0":"0000","1":"0001","2":"0010","3":"0011",
            "4":"0100","5":"0101","6":"0110","7":"0111",
            "8":"1000","9":"1001","a":"1010","b":"1011",
            "c":"1100","d":"1101","e":"1110","f":"1111"}
        hexin = binascii.hexlify(self.spiread)
        h0 = hexin[0]
        h1 = hexin[1]
        h2 = hexin[2]
        h3 = hexin[3]
        self.binnum = bins[h2]+bins[h3]+bins[h0]+bins[h1]
        spiout = float(int(self.binnum[1:13],2))/4.00
        return spiout

    def checkTC(self):
        noTC = self.binnum[14]
        return noTC

    def binTC(self):
        return self.binnum

    def checkPeltier(self):
        adr = "--address=0x68"
        t = "ts7500ctl"
        c = "--peek16"
        out = Popen([t,adr,c],stdout=PIPE).communicate()[0]
        if out[2:4] == "2b" or out[2:4] == "3f":
            peltierState = "0"
        # heat
        # 23: 1, 25: 0
        elif out[2:4] == "2f":
            peltierState = "1"
        # cool
        # 23: 0, 25: 1
        elif out[2:4] == "3b":
            peltierState = "2"
        else:
            peltierState = "derp"
        return peltierState

    def getSetPoint(self):
        fconf = open("/var/www/api/wp.conf","r")
        line = fconf.readline()
        setTime = line.split(',')[0]
        setTemp = float(line.split(',')[1])
        fconf.close()
        return [setTime, setTemp]

    def write2file(self):
        unixtime = str(int(time.mktime(time.localtime())))
        setpt = self.getSetPoint()[1]
        sett = self.getSetPoint()[0]

tofile=unixtime+","+str(self.readTC(2))+","+str(self.readTC(3))+","+str(setpt
)+","+self.checkPeltier()+"\n"
        fout = open("/var/www/api/output.csv","a")
        fout.write(tofile)
        fout.close()
        return

# Main Function, for running manageData instead of importing as a class
if __name__ == '__main__':
    print "Starting manageData..."
    print "This is largely written for debugging purposes."
    d = manageData(2500000)
    eggT = d.readTC(3)
    nestT = d.readTC(2)
    setT = d.getSetPoint()[1]
    setTTime = d.getSetPoint()[0]
    pelt = d.checkPeltier()
    print "Egg Temperature: "+str(eggT)
    print "Nest Temperature: "+str(nestT)
    print "Set Point: "+str(setT)
    print "Set Time: "+str(setTTime)
    print "Peltier State: "+pelt
    print "Writing to file ..."
    d.write2file()
```

managePeltier.py

```
from subprocess import Popen, PIPE

# The current scheme uses pin 23 and 25 for Peltier control.
# Assumes that [23=0 25=1] = heat [23=1 25=0] = cool

class managePeltier(Popen):

    def __init__(self,offset):
        self.offset = offset
        self._child_created = False
    def setTemp(self,setTemp,nestTemp):
        #Using Popen as a class creates an exception; this avoids that.
        self._child_created = False
        if setTemp - nestTemp>=self.offset:
            # if temperature difference greater than offset, heat
            Popen(["/home/python/pinio","set", "23","0"],stdout=PIPE)
            Popen(["/home/python/pinio","set", "25","1"],stdout=PIPE)
            setPeltier = 2
            return setPeltier
        elif setTemp - nestTemp<=-self.offset:
            # if temperature difference greater than offset, cool
            Popen(["/home/python/pinio","set", "23","1"],stdout=PIPE)
            Popen(["/home/python/pinio","set", "25","0"],stdout=PIPE)
            setPeltier = 1
            return setPeltier
        else:
        # if temperature doesn't vary by less than offset, turn off
            Popen(["/home/python/pinio","set", "23","0"],stdout=PIPE)
            Popen(["/home/python/pinio","set", "25","0"],stdout=PIPE)
            setPeltier = 0
            return setPeltier
    def getSetting(self):
        return setPeltier
```

manageRFID.py

```
#!/usr/bin/python
from subprocess import Popen, PIPE, STDOUT, call
import pty
import os
import time

class manageRFID():

    def __init__(self):
        # initialize xuart to port 0 at speed = 9600
        # call(["xuartctl --server --port=0 --speed=9600"],stdout=PIPE)
        # by default this is where port 0 is mounted, if that's not the case, 
change this
        cmd = 'cat /dev/pts/0'
        master, slave = pty.openpty()
        p = Popen(cmd, shell=True, stdin=PIPE, stdout=slave, stderr=slave, 
close_fds=True)
        self.stdout = os.fdopen(master)
        self.ledstate = 0
        self.birdstate = 0
        call(["/home/python/pinio","set", "20","0"],stdout = PIPE)

        # InNestLim defines how many RFID counts a bird can be in the nest 
before calling it
        # a false positive.
        # 10 is not that good; it just makes debugging less boring
        self.InNestLim = 10
        self.InNestCount = 1;

    def setLED(self,io):
        if io == 0:
            call(["/home/python/pinio","set", "20","0"],stdout = PIPE)
        else:
            call(["/home/python/pinio","set", "20","1"],stdout = PIPE)
        self.ledstate = io

    def toggleLED(self):
        if self.ledstate == 0:
            self.setLED(1)
        else:
            self.setLED(0)

    # writes rfid state, plus time, to file
    # by default the file is saved to /var/www/api/rfid
    def write2file(self,data):
        fp = "/var/www/api/rfidout.csv"
        fh = open(fp,"a")
        data = str(time.ctime())+","+data+"\n"
        fh.write(data)
        fh.close()

    def readRFID(self):
        # this parses the whole line from the RFID board to get at the RFID
number
        # The RFID number is a 10 digit hex number, it is read in as a
string.
        # When there are no RFID tags in there, FFFFFFFFFF is the default
value

        # gets the line from stdout, remove excess characters, and split it
into a list
        # the RFID tag should always be the last item on the list
        strlist = self.stdout.readline().rstrip().split()
        # readline doesn't always get a full line, sometimes only the RFID
value is read
        # doing things this way allows RFID to be parsed no matter what
portion of line is read
        rfidout = strlist[len(strlist)-1]
        # state 0
        if self.birdstate == 0:
            if rfidout == 'FFFFFFFFFF':
                print "No Bird"
                self.setLED(0)
            else:
                self.toggleLED()
                self.birdstate = 1
                bird = "Arrived," + rfidout
                self.write2file(bird)
                print bird
        # state 1
        elif self.birdstate == 1:
            if rfidout == 'FFFFFFFFFF':
                self.toggleLED()
                bird = "In nest,"+ " "
                self.write2file(bird)
                print bird
                self.birdstate = 2
            else:
                self.toggleLED()
                bird = "Perch," + rfidout
                print bird
        # state 2
        elif self.birdstate == 2:
            if (rfidout == 'FFFFFFFFFF'):
                # if the count <= limit, keep going
                if self.InNestCount <= self.InNestLim:
                    self.InNestCount = self.InNestCount+1
                    self.toggleLED()
                    bird = "In nest," + " "
                    self.write2file(bird)
                    print bird
                    self.birdstate =2
                # otherwise assume a false positive and stop recording and
reset
                else:
                    bird = "false positive,"+ " "
                    self.write2file(bird)
                    print bird
                    self.birdstate = 0
                    self.InNestCount = 0
            else:
                self.toggleLED()
                bird = "Perch," + rfidout
                self.write2file(bird)
                print bird
                self.InNestCount = 0
                self.birdstate = 3
        # state 3
        else:
            if rfidout == 'FFFFFFFFFF':
                bird = "Bird out,"+ " "
                self.write2file(bird)
                print bird
                self.birdstate =0
            else:
                self.toggleLED()
                bird = "Perch," + rfidout
                self.write2file(bird)
                print bird
                self.birdstate = 3
# Main Function, for running manageRFID instead of importing as a class
if __name__ == '__main__':
    r = manageRFID()

    while(1):
        r.readRFID()
```

PeltierControl.py

```
import os
from manageData import manageData
from managePeltier import managePeltier
from SimpleConfigParser import SimpleConfigParser
import threading

cp = SimpleConfigParser()
tc = manageData(2500000)
peltier = managePeltier(1)

# defines file path and get the pid
pidfile = "/home/peltierControl.pid"
pid = str(os.getpid())
```

```
# if the pid file already exists, then remove it
if os.path.exists(pidfile):
    os.remove(pidfile)

# store pid to file
pidfh = open(pidfile,"w+")
pidfh.write(pid)
pidfh.close()

def checkInterval():
    filename = '/var/www/sample.ini'
    cp.read(filename)
    timeint = int(cp.getoption("peltierInterval").replace('\"',''))
    return timeint

def peltierctl():
    threading.daemon=True
    waitTime = 5
    threading.Timer(waitTime,peltierctl).start()
    setPoint = tc.getSetPoint()[1]
    nestTemp = tc.readTC(2)
    peltier.setTemp(setPoint,nestTemp)
    waitTime = checkInterval()

peltierctl()
```

RFIDReader.py

```
# this file should be initiated from the web script

import os
import threading
from manageRFID import manageRFID

r = manageRFID()

def print2file():
    threading.daemon=True
    waitTime = 0.01
    threading.Timer(waitTime,print2file).start()
    rfidout = r.readRFID()

print2file()
```

SetPeltier.py

```
# manually control Peltier

import sys
from subprocess import Popen,PIPE

if (len(sys.argv) > 1):
    peltstate = sys.argv[1]

if (peltstate == "1"):
    Popen(["./home/python/pinio","set", "23","1"],stdout=PIPE)
    Popen(["./home/python/pinio","set", "25","0"],stdout=PIPE)

elif (peltstate == "2"):
    Popen(["./home/python/pinio","set", "23","0"],stdout=PIPE)
    Popen(["./home/python/pinio","set", "25","1"],stdout=PIPE)

else:
    Popen(["./home/python/pinio","set", "23","1"],stdout=PIPE)
    Popen(["./home/python/pinio","set", "25","1"],stdout=PIPE)

# bilingual.py
"""
Please excuse my highschool german! :~)
"""
import sys

if (len(sys.argv) > 1):
    sys.argv[1] == 'english' ):
        print "Hello World!"
    elif( sys.argv[1] == 'german' ):
        print "Guten Tag Welt!"
    else:
        print "Which language?"
        print "Welche sprache?"
else:
    print "Which language do you want?"
    print "Welch sprache wisst du?"
```

Web server Code

Index.php

```
<html>
<body>

<?php

function ctl7553($onoff){
if ($onoff == 'on'){system("sudo bash /home/python/init7553.sh &");}
else{system("sudo killall python");}
sleep(5);
}

(isset($_POST["Start"]))?(ctl7553('on')):(sleep(0));
(isset($_POST["Stop"]))?(ctl7553('off')):(sleep(0));

$params = parse_ini_file("sample.ini", true);

$ip = $_SERVER['SERVER_ADDR'];
$nb = $params['nestbox'];
$sampleInt = $params['sampleInterval'];
$peltInt = $params['peltierInterval'];

$pidpath = "/home/dataLogger.pid";
$fh = fopen($pidpath, "r");
```

```
$pid = fgets($fh);
$procpath = "/proc/".$pid;

if (file_exists($procpath)){
$runState = "Running";
}else{
$runState = "Not Running";
}

// check Peltier state by reading pins that control the H-bridge
$peltcmd = 'sudo ts7500ctl --peek16 --address=0x68';
$hexPeltState = shell_exec($peltcmd);
if ($hexPeltState == 0x2f){
$peltState = 'Cooling';
}elseif($hexPeltState == 0x3b){
$peltState = 'Heating';
}else{
$peltState = 'Off';
}

?>

<table align=center border="0">
<tr>
    <td colspan="2" style="background-color:#99CCFF;"><h1>Nestbox <?php echo
$nb; ?><br/><h2>Main Page</h2></h1></td>
</tr>

    <td style="background-color:#99CCFF;width:200px;text-align:center;">
    <table align=center border="0">
        <tr><td><b>Main Page</b></td></tr>
        <tr><td><a href="<?php echo
'http://'.$_SERVER['SERVER_ADDR'].'/bin/node/time.php'; ?>">Time
Settings</a></td></tr>
        <tr><td><a href="<?php echo
'http://'.$_SERVER['SERVER_ADDR'].'/dataSettings.php'; ?>">Data
Settings</a></td></tr>
        <tr><td><a href="<?php echo
'http://'.$_SERVER['SERVER_ADDR'].'/api/rfid.php'; ?>">RFID</a></td></tr>
        <tr><td><a href="<?php echo
'http://'.$_SERVER['SERVER_ADDR'].'/api/read'; ?>">Read</a></td></tr>
        <tr><td><a href="<?php echo
'http://'.$_SERVER['SERVER_ADDR'].'/api/loggerData';
?>">loggerData</a></td></tr>
        <tr><td><a href="<?php echo
'http://'.$_SERVER['SERVER_ADDR'].'/dataDownload.php'; ?>">Download
Data</a></td></tr>
    </table>
    </td>

    <td style="background-color:#FFFFFF;width:1200px;text-align:top;">
        <table align=center width="300" border="1" style="font-
family:verdana;font-size:90%;color:black">
        <tr><td><b>IP Address</b></td><td><?php echo $ip; ?></td></tr>
        <tr><td><b>Sample Interval</b></td><td><?php echo $sampleInt;
?></td></tr>
```

```
        <tr><td><b>Peltier Interval</b></td><td><?php echo $peltInt;
?></td></tr>
        <tr><td><b>Peltier State</b></td><td><?php echo $peltState;
?></td></tr>
        <tr><td><b>Device Status</b></td><td><?php echo $runState;
?></td></tr>
        <tr>
        <form method="post" action="<?php echo $PHP_SELF;?>">
        <?php
        if ($runState=="Not Running")
        {echo '<td></td><td><input type="submit" value="Start"
name="Start"></td>'; }
        else{echo '<td></td><td><input type="submit" value="Stop"
name="Stop"></td>';}
        ?>
        </form>
        </tr>

        </table>
    </td>
</tr>


</table>

</body>
</html>
```

dataSettings.php

```
<?php

// check Peltier state by reading pins that control the H-bridge
$peltcmd = 'sudo ts7500ctl --peek16 --address=0x68';
$hexPeltState = shell_exec($peltcmd);
if ($hexPeltState == 0x2f){
$peltS = '1';
}elseif($hexPeltState == 0x3b){
$peltS = '2';
}else{
$peltS = '0';
}

// loads current settings from ini file
$params = parse_ini_file("sample.ini", true);

$pidpath = "/home/dataLogger.pid";
$fh = fopen($pidpath, "r");
$pid = fgets($fh);
$procpath = "/proc/".$pid;

$PeltRadio = '';

if (file_exists($procpath)){
    $PeltRadio = 'disabled';
}else{
    $PeltRadio = '';
}

// this function is used to write an array to a .ini file
function write_ini_file($assoc_arr, $path, $has_sections=FALSE) {
    $content = "";
    if ($has_sections) {
        foreach ($assoc_arr as $key=>$elem) {
            $content .= "[".$key."]\n";
            foreach ($elem as $key2=>$elem2) {
                if(is_array($elem2))
                {
                    for($i=0;$i<count($elem2);$i++)
                    {
                        $content .= $key2."[] = \"".$elem2[$i]."\"\n";
                    }
                }
                else if($elem2=="") $content .= $key2." = \n";
                else $content .= $key2." = \"".$elem2."\"\n";
            }
        }
    }
    else {
        foreach ($assoc_arr as $key=>$elem) {
            if(is_array($elem))
            {
                for($i=0;$i<count($elem);$i++)
                {
                    $content .= $key."[] = \"".$elem[$i]."\"\n";
                }
            }
            else if($elem=="") $content .= $key." = \n";
            else $content .= $key." = \"".$elem."\"\n";
        }
    }

    if (!$handle = fopen($path, 'w')) {
        return false;
    }
    if (!fwrite($handle, $content)) {
        return false;
    }
    fclose($handle);
    return true;
}

// check if there are any inputs, if there are, store them, otherwise, use
previous values from ini file
$ip =
((isset($_POST["ip"]))?htmlspecialchars($_POST["ip"]):$_SERVER['SERVER_ADDR']
);
$nb =
((isset($_POST["nb"]))?htmlspecialchars($_POST["nb"]):$params['nestbox']);
$sampleInt =
((isset($_POST["sampleint"]))?htmlspecialchars($_POST["sampleint"]):$params['
sampleInterval']);
```

```
$peltInt =
((isset($_POST["peltierint"]))?htmlspecialchars($_POST["peltierint"]):$params
['peltierInterval']);
$peltSet =
((isset($_POST["peltState"]))?htmlspecialchars($_POST["peltState"]):$peltS);

system('sudo ifconfig eth0 '.$ip);

if ($PeltRadio !== 'disabled'){
switch ($peltSet) {
    case "0":
        $offcheck = "checked";
        $coolcheck = "";
        $heatcheck = "";
    $setpelt = "bash /home/python/setPeltier.sh 0";
    break;
    case "1":
        $offcheck = "";
        $coolcheck = "checked";
        $heatcheck = "";
    $setpelt = "bash /home/python/setPeltier.sh 1";
        break;
    case "2":$params['peltierInterval'] = $peltInt;
        $offcheck = "";
        $coolcheck = "";
        $heatcheck = "checked";
    $setpelt = "bash /home/python/setPeltier.sh 2";
        break;
}

$out = shell_exec('sudo '.$setpelt);
}

// updates array value
$params['ipaddr'] = $ip;
$params['nestbox'] = $nb;
$params['sampleInterval'] = $sampleInt;
$params['peltierInterval'] = $peltInt;

// defines path to ini file
$path = "/var/www/sample.ini";

// write array to ini file
write_ini_file($params, $path, false);

// the html stuff largely prints the borders and tables and stuff, since
commenting in html is annoying, its here.
?>
<html>
<body>

<table align=center border="0">
<tr>
    <td colspan="2" style="background-color:#99CCFF;"><h1>Nestbox <?php echo
$nb; ?><br/><h2>Device Settings</h2></h1></td>
</tr>
```

```
<tr valign="top">

    <td style="background-color:#99CCFF;width:150px;text-align:center;">
    <table align=center border="0">
        <tr><td><a href="<?php echo
'http://'.$_SERVER['SERVER_ADDR'].'/index.php'; ?>">Index</a></td></tr>
        <tr><td><a href="<?php echo
'http://'.$_SERVER['SERVER_ADDR'].'/bin/node/time.php'; ?>">Time
Settings</a></td></tr>
        <tr><td><b>Data Settings</b></td></tr>
        <tr><td><a href="<?php echo
'http://'.$_SERVER['SERVER_ADDR'].'/api/rfid.php'; ?>">RFID</a></td></tr>
        <tr><td><a href="<?php echo
'http://'.$_SERVER['SERVER_ADDR'].'/api/read'; ?>">Read</a></td></tr>
        <tr><td><a href="<?php echo
'http://'.$_SERVER['SERVER_ADDR'].'/api/loggerData';
?>">loggerData</a></td></tr>
        <tr><td><a href="<?php echo
'http://'.$_SERVER['SERVER_ADDR'].'/dataDownload.php'; ?>">Download
Data</a></td></tr>
    </table>
    </td>

    <td style="background-color:#EEEEEE;width:1200px;text-align:top;">
        <form method="post" action="<?php echo $PHP_SELF;?>">
        <table align=center width="500" border="0" style="font-
family:verdana;font-size:90%;color:black">
        <tr>
        <td>Nestbox</td>
        <td><input type="text" size="14" maxlength="3" name="nb" value =
"<?php echo $params['nestbox']; ?>" ></td>
        </tr>
        <tr>
        <td>IP Address</td>
        <td><input type="text" size="14" maxlength="15" name="ip" value =
"<?php echo $params['ipaddr']; ?>" ></td>
        </tr>
        <tr>
        <td>Thermocouple Sample Interval</td>
        <td><input type="text" size="14" maxlength="2" name="sampleint" value
= "<?php echo $params['sampleInterval']; ?>" ></td>
        </tr>
        <tr>
        <td>Peltier Control Interval</td>
        <td><input type="text" size="14" maxlength="2" name="peltierint"
value = "<?php echo $params['peltierInterval']; ?>"  ></td>
        </tr>
        <tr>
        <td>Peltier State <?php echo $PeltRadio; ?></td>
        <td>
        <input type="radio" name="peltState" value="0" <?php echo $offcheck.'
'.$PeltRadio; ?> /> Off<br />
        <input type="radio" name="peltState" value="1" <?php echo
$coolcheck.' '.$PeltRadio; ?> /> Cool<br />
        <input type="radio" name="peltState" value="2" <?php echo
$heatcheck.' '.$PeltRadio; ?> /> Heat<br />
        </td>
        </tr>
        <tr>
        <td></td><td><input type="submit" value="submit" name="submit"></td>
        </tr>
        </table>
        </form>
    </td>
</tr>
</table>

</body>
</html>
```

Time.php

```
<?php
 $yyMmDd = $_POST["yyMmDd"];
 $verbose = ((isset($_GET["verbose"]))?htmlspecialchars($_GET["verbose"]):1);
 $currTime = time();
 $currTimeDisp = date(DATE_RFC822,$currTime);

 $yr = substr($yyMmDd, 0,4);
 $mon = substr($yyMmDd, 4,2);
 $d = substr($yyMmDd, 6,2);
 $h = substr($yyMmDd, 8,2);
 $m = substr($yyMmDd, 10,2);
 $s = substr($yyMmDd, 12,2);

 //$date2set = "sudo date --set ".$yr."-".$mon."-".$d;
 $time2set = "sudo date --set ".$h.":".$m.":".$s;

 $params = parse_ini_file("/var/www/sample.ini", true);
 $nb = $params['nestbox'];

 if ($verbose == 1) {
?>

<html>
<table align=center border="0">

<tr>
    <td colspan="2" style="background-color:#99CCFF;"><h1>Nestbox <?php echo
$nb; ?><br/><h2>Time Settings</h2></h1></td>
</tr>

<tr valign="center">

    <td style="background-color:#99CCFF;width:150px;text-align:center;">
    <table align=center border="0">
        <tr><td><a href="<?php echo
'http://'.$_SERVER['SERVER_ADDR'].'/index.php'; ?>">Main Page</a></td></tr>
        <tr><td><b>Time Settings</b></td></tr>
        <tr><td><a href="<?php echo
'http://'.$_SERVER['SERVER_ADDR'].'/dataSettings.php'; ?>">Data
Settings</a></td></tr>
```

```
        <tr><td><a href="<?php echo
'http://'.$_SERVER['SERVER_ADDR'].'/api/rfid.php'; ?>">RFID</a></td></tr>
        <tr><td><a href="<?php echo
'http://'.$_SERVER['SERVER_ADDR'].'/api/read'; ?>">Read</a></td></tr>
        <tr><td><a href="<?php echo
'http://'.$_SERVER['SERVER_ADDR'].'/api/loggerData';
?>">loggerData</a></td></tr>
        <tr><td><a href="<?php echo
'http://'.$_SERVER['SERVER_ADDR'].'/dataDownload.php'; ?>">Download
Data</a></td></tr>
    </table>
    </td>

<td style="background-color:#EEEEEE;width:1200px;text-align:left;">
 Current Time is (Unix): <?php echo $currTime;?> <br/><br/>
 Current Time is (RFC822): <?php echo $currTimeDisp;?> <br/><br/>
 <form method="post" action="<?php echo $PHP_SELF;?>">
 Time:<input type="text" size="14" maxlength="14" name="yyMmDd">
 <input type="submit" value="submit" name="submit">
 </form>
</td>
</tr>

</table>

</html>

<?
 }else{
    echo $currTime;
 }
 if (isset($_POST['submit'])) { // if page is not submitted to itself echo
the form
 echo "You have set time to ".$yyMmDd."<br />";
 echo $date2set."<br/>";
 echo $time2set."<br/>";

 //shell_exec($date2set);
 shell_exec($time2set);

 }
 ?>
```

Read.php

```
<?php
$pidpath = "/home/dataLogger.pid";
$fh = fopen($pidpath, "r");
$pid = fgets($fh);
$procpath = "/proc/".$pid;

if (file_exists($procpath)){
    echo "<table><tr><td>time</td><td>box temp</td><td>egg temp</td><td>set
point</td><td>Peltier control</td></tr>";
    $file = "/var/www/api/output.csv";
    $file = escapeshellarg($file);
    $line = `tail -n 1 $file`;
    $pieces = explode(",",$line);
    echo
"<tr><td>".$pieces[0]."</td><td>".$pieces[1]."</td><td>".$pieces[2]."</td><td
>".$pieces[3]."</td><td>".$pieces[4]."</td></tr></table>\n";
}else{
echo "Device is not running; no new data to display";
}
?>
```

loggerData.php

```
<?php

//$lines = ((isset($_GET["Lines"]))?($_GET["Lines"]):1);

// this assumes 15s sample interval and that server reads it every 5 minutes.
// actuall 6 minutes are provided. the data collecting scripts can deal with
redundant data,
// so the extra minute is to make sure all data's covered.
$lines = 360;

// full path to text file
define("TEXT_FILE", "/var/www/api/output.csv");
// number of lines to read from the end of file
define("LINES_COUNT", $lines);


function read_file($file, $lines) {
    //global $fsize;
    $handle = fopen($file, "r");
    $linecounter = $lines;
    $pos = -2;
    $beginning = false;
    $text = array();
    while ($linecounter > 0) {
        $t = " ";
        while ($t != "\n") {
            if(fseek($handle, $pos, SEEK_END) == -1) {
                $beginning = true;
                break;
            }
            $t = fgetc($handle);
            $pos --;
        }
        $linecounter --;
        if ($beginning) {
            rewind($handle);
        }
        $text[$lines-$linecounter-1] = fgets($handle);
        if ($beginning) break;
    }
    fclose ($handle);
    return array_reverse($text);
}
echo "<table>\n";
echo "<tr><td>time</td><td>box temp</td><td>egg temp</td><td>set
point</td><td>Peltier control</td></tr>\n";
$lines = read_file(TEXT_FILE, LINES_COUNT);
foreach ($lines as $line) {
    $pieces = explode(",",$line);
    echo
"<tr><td>".$pieces[0]."</td><td>".$pieces[1]."</td><td>".$pieces[2]."</td><td
>".$pieces[3]."</td><td>".$pieces[4]."</td></tr>\n";
}
echo "</table>";
?>
```

rfid.php

```
<?php
$params = parse_ini_file("/var/www/sample.ini", true);
$nlines = ((isset($_POST["nlines"]))?htmlspecialchars($_POST["nlines"]):5);

// full path to text file
define("TEXT_FILE", "/var/www/api/rfidout.csv");
// number of lines to read from the end of file
define("LINES_COUNT", $nlines);

function read_file($file, $lines) {
    //global $fsize;
    $handle = fopen($file, "r");
    $linecounter = $lines;
    $pos = -2;
    $beginning = false;
    $text = array();
    while ($linecounter > 0) {
        $t = " ";
        while ($t != "\n") {
            if(fseek($handle, $pos, SEEK_END) == -1) {
                $beginning = true;
                break;
            }
            $t = fgetc($handle);
            $pos --;
        }
        $linecounter --;
        if ($beginning) {
            rewind($handle);
        }
        $text[$lines-$linecounter-1] = fgets($handle);
        if ($beginning) break;
    }
    fclose ($handle);
    return array_reverse($text);
}
$lines = read_file(TEXT_FILE, LINES_COUNT);

?>

<html>
<body>

<table align=center border="0">
<tr>
    <td colspan="2" style="background-color:#99CCFF;"><h1>Nestbox <?php echo
$params['nestbox']; ?><br/><h2>RFID</h2></h1></td>
</tr>
```

```
<tr valign="top">

    <td style="background-color:#99CCFF;width:200px;text-align:center;">
    <table align=center border="0">
        <tr><td><a href="<?php echo
'http://'.$_SERVER['SERVER_ADDR'].'/index.php'; ?>">Main Page</a></td></tr>
        <tr><td><a href="<?php echo
'http://'.$_SERVER['SERVER_ADDR'].'/bin/node/time.php'; ?>">Time
Settings</a></td></tr>
        <tr><td><a href="<?php echo
'http://'.$_SERVER['SERVER_ADDR'].'/dataSettings.php'; ?>">Data
Settings</a></td></tr>
        <tr><td><b>RFID</b></td></tr>
        <tr><td><a href="<?php echo
'http://'.$_SERVER['SERVER_ADDR'].'/api/read'; ?>">Read</a></td></tr>
        <tr><td><a href="<?php echo
'http://'.$_SERVER['SERVER_ADDR'].'/api/loggerData';
?>">loggerData</a></td></tr>
        <tr><td><a href="<?php echo
'http://'.$_SERVER['SERVER_ADDR'].'/dataDownload.php'; ?>">Download
Data</a></td></tr>
    </table>
    </td>

    <td style="background-color:#EEEEEE;width:1200px;text-align:center;">
        <br/><br/>
        <table align=center border="0" style="font-family:verdana;font-
size:90%;color:black">
        <tr>
        <td>Number of entries to read</td>
        <form method="post" action="<?php echo $PHP_SELF;?>">
        <td><input type="text" size="14" maxlength="2" name="nlines" value =
<?php echo $nlines; ?> ></td>
        <td></td><td><input type="submit" value="submit" name="submit"></td>
        </form>
        </tr>
        </table>
        <br/><br/>
        <table align=center width="600" border="1" bgcolor="#FFFFFF"
style="font-family:verdana;font-size:90%;color:black">
        <tr><td align="center"><b>Time</b></td><td align="center"><b>Bird
State</b></td><td align="center"><b>RFID Tag</b></td></tr>
        <?php
        foreach ($lines as $line) {
            $pieces = explode(",",$line);
            echo "<tr><td align='center'>".$pieces[0]."</td><td
align='center'>".$pieces[1]."</td><td
align='center'>".$pieces[2]."</td></tr>";
        }
        echo "</table>";
        ?>

    </td>
</tr>
</table>
```

```
</body>
</html>
```

DataDownload.php

```
<?php
$params = parse_ini_file("sample.ini", true);

?>

<html>
<body>
<table align=center border="0">
<tr>
    <td colspan="2" style="background-color:#99CCFF;"><h1>Nestbox <?php echo 
$params['nestbox']; ?><br/><h2>Download Data</h2></h1></td>
</tr>

<tr valign="top">

    <td style="background-color:#99CCFF;width:200px;text-align:center;">
    <table align=center border="0">
        <tr><td><a href="<?php echo 
'http://'.$_SERVER['SERVER_ADDR'].'/index.php'; ?>">Main Page</a></td></tr>
        <tr><td><a href="<?php echo 
'http://'.$_SERVER['SERVER_ADDR'].'/bin/node/time.php'; ?>">Time 
Settings</a></td></tr>
        <tr><td><a href="<?php echo 
'http://'.$_SERVER['SERVER_ADDR'].'/dataSettings.php'; ?>">Data 
Settings</a></td></tr>
        <tr><td><a href="<?php echo 
'http://'.$_SERVER['SERVER_ADDR'].'/api/rfid.php'; ?>">RFID</a></td></tr>
        <tr><td><a href="<?php echo 
'http://'.$_SERVER['SERVER_ADDR'].'/api/read'; ?>">Read</a></td></tr>
        <tr><td><a href="<?php echo 
'http://'.$_SERVER['SERVER_ADDR'].'/api/loggerData'; 
?>">loggerData</a></td></tr>
        <tr><td><b>Download Data</b></td></tr>
    </table>
    </td>
    <td style="background-color:#EEEEEE;width:1200px;text-align:top;">
        <br/><br/>
        <table align=center bgcolor = "EEEEEE" border="0">
        <tr>
            <td><a href="api/output.csv">Download Thermocouple Data</td>
        </tr>
        <tr></tr>
        <tr>
            <td><a href="api/rfidout.csv">Download RFID Data</td>
        </tr>
    </td>

</tr>
</table>

</body>
</html>
```