Design, Construction and Analysis of an LED Strobe Controller

by

Nicholas E. Villalva

B.S., Massachusetts Institute of Technology (2011)

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Masters of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2012

Author ....
Department of Electrical Engineering and Computer Science
August 10, 2012

Certified by ....
Dr. James W. Bales
Assistant Director, Edgerton Center
Thesis Supervisor

Accepted by ....
Prof. Dennis M. Freeman
Chairman, Masters of Engineering Thesis Committee

Design, Construction and Analysis of an LED Strobe Controller

by

Nicholas E. Villalva

Submitted to the Department of Electrical Engineering and Computer Science
on August 10, 2012, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Computer Science and Engineering

Abstract

In this thesis, I describe an LED strobe controller that I designed and built. With the advent of much brighter LEDs, it is possible to create stroboscopes with both high intensity output and high maximum flash rates. Additionally, the proliferation of Wi-Fi enabled devices provokes the idea of a next-generation interface for tools in Strobe Lab. The initial goal was to produce an LED-based strobe that has sufficient light to replace existing flash tube stroboscopes. The final product combines three high powered LEDs with a microcontroller and Wi-Fi card. Users can adjust settings through a web based interface accessible from both laptops and mobile devices. In addition to the simple user interface, the controller is compatible with other equipment in Strobe Lab as it can be triggered by the commonly used methods. Established techniques, such as sync and delay, and newer techniques, such as color-encoded multi-flash images, are possible with this device.

Thesis Supervisor: Dr. James W. Bales
Title: Assistant Director, Edgerton Center

Acknowledgments

I would like to thank Dr. James W. Bales for giving me the opportunity to work on this project and for his constant support and guidance throughout. I am very grateful for the amount of time and effort he has invested over the past two years to answer questions, discuss design and foster both learning and understanding.

I would also like to thank my parents for providing me with vital support every step of the way. From the early emphasis on education and the pursuit of dreams to the kind words of encouragement in the final stretch your love and help have been with me at all times. Without that spark and continued urging, this would not have been possible.

Finally, I would like to thank my friends for making my life over the past five years at MIT fun, caring support and wonderful memories. I would not have been able to do this without you.

Contents

1 Introduction **13**

1.1 Vision 13

1.2 Background & Motivations 13

1.3 The Solution 15

2 Components and Sub-Assemblies **17**

2.1 Hardware On Hand 17

2.1.1 LEDs 17

2.1.2 Power Boards 17

2.1.3 Power Supply 18

2.2 Hardware Decisions 18

2.2.1 Microcontroller 19

2.2.2 Wi-Fi Card 20

2.2.3 Serial Display 21

2.2.4 Final Parts List 21

3 From Parts to Product **23**

3.1 Goals 23

3.2 Hardware 23

3.3 Software 24

3.3.1 Timing 24

3.3.2 WiFly Shield 25

3.3.3 UI Templating Engine 26

3.4 Ease of Updating 26

3.4.1 Waveform Generation and Triggering 26

3.4.2 User Interface 26

4 Results and Analysis **29**

4.1 Overview 29

4.2 LEDs 29

4.3 Timing 30

4.3.1 Delay 30

4.3.2 Flash Duraton 30

4.3.3 Flash Frequency 31

4.4 Shortcomings 31

5 Contributions **33**

5.1 What's been done 33

5.2 Future Work 33

5.2.1 Consolidate Multiple Controllers 33

5.2.2 LEDs 34

5.2.3 Microcontroller 34

5.2.4 User Interface 35

A Appendix A: Arduino C++ Code **37**

B Appendix B: User Interface and Templating **45**

B.1 Example Page Template 45

B.2 Script to convert templates to a C++ Pages class 48

List of Figures

1-1 Color-encoded time progression of a towel being whipped (Lee, Allen and Smith, 1993)[8] 14

1-2 Block diagram depicting an overview of the interaction between components in the strobe controller 15

2-1 Teensy++ 2.0 20

2-2 WiFly Shield [5] 21

2-3 Serial LCD screen 21

3-1 Diagram of overall timing 24

4-1 Comparison of typical intensity curves of flash tubes and LEDs 30

List of Tables

2.1 Key Specifications for Luminus LEDs 17

2.2 Key Specifications for V-Tech Power Board 18

2.3 Key Specifications for Arduino UNO, Arduino Mega and Teensy++ 2.0 19

3.1 Hardware I/O Methods 24

Chapter 1

Introduction

1.1 Vision

High speed photography, pioneered and popularized by Harold Edgerton in the early 20th century, uses bright, short pulses of light to freeze an image in time. Technology has come a long way since then, but most flashes and stroboscopes used are still electronic flash tubes. With the increased output and efficiency of LEDs and digital hardware, I wanted to see if these improved technologies could be brought to bear in the fields of stroboscopy and high speed photography. In addition the proliferation of Wi-Fi allows for a flexible, low-cost user interface. The demonstration of timing and other functions with a single microcontroller is another priority in this thesis.

1.2 Background & Motivations

Among Edgerton's famous applications of strobes to photography is the technique of sync and delay. In sync and delay, a trigger is used to synchronize a known instant of an action with the start of a delay timer. Once the previously programmed delay has expired, the timer sets off the strobe, capturing an image. This technique is used to capture images such as bullets shredding cards and milk drops splashing. With a pulse of light measuring as short as 500ns in duration, these complex movements that happen in the blink of an eye can be frozen for the camera. With such a short period of illumination, the brightness of a flash tube must be immense for enough photons to strike the film or digital camera sensor. As such, the light output of typical strobes used in high speed photography is measured in beam-candlepower-seconds (BCPS). A typical strobe used in MIT's Strobe Lab has a BCPS reading of 1-40 [4].

High output LEDs designed for use in projectors and televisions suggest a possible transition to the use of LEDs in other high intensity applications. Frequently these LEDs come in red, green and blue rather than a single bright white light. This offers an interesting application in which colors can be used individually or combined for a single white flash. While photographs produced by a single, white flash tube can be seen on the walls of the Edgerton Center, multicolored flash photographs are rarer. At North Carolina School of Science and Mathematics, interesting images have been created using multiple multi-flash strobes and colored filters to encode time progression. An image exemplifying this technique can be seen in Figure 1-1 below. This image was created with a series of 12 flash units equipped with colored filters. During the motion, the flash units were triggered in succession, creating the multi-colored photograph below. While these techniques produce amazing photographs, the technique and setup required to apply filters of varying colors with traditional strobe lights is complicated, time consuming and requires multiple expensive equipment items to perform. A simpler LED strobe with three programmable color channels would provide an increased ease of use and decreased time and monetary cost.

Figure 1-1: Color-encoded time progression of a towel being whipped (Lee, Allen and Smith, 1993)[8]

1.3 The Solution

The device produced in this thesis serves as a proof of concept for the use of both LEDs and Wi-Fi in future devices for Strobe Lab. The current ecosystem of controllers in Strobe Lab consists of one main controller capable of accepting sensor inputs and issuing signals to the many strobe lights in the lab. Each of the larger strobes has dials for settings such as flash duration, delay and number of flashes. In the future, it would be advantageous to have a single controller capable of handling all of the various settings required in lab with an interface that no longer requires students to fiddle with settings on every piece of equipment they are using. A block diagram of the overall design and interaction can be seen below in Figure 1-2.

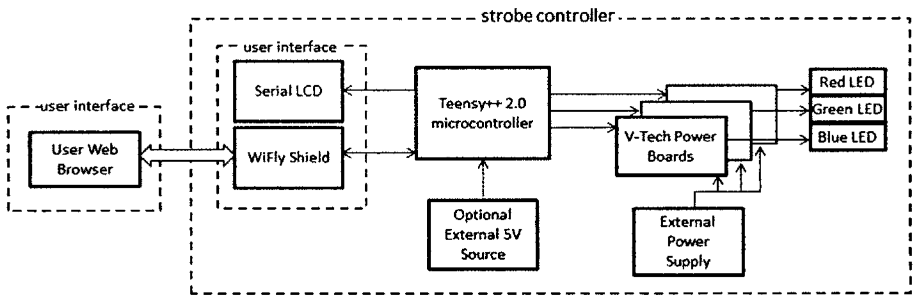

Figure 1-2: Block diagram depicting an overview of the interaction between components in the strobe controller

The actual device will consist of the components within the dashed "strobe controller" box – a serial LCD screen, a Wi-Fi card, a microcontroller, three V-Tech driver boards, three high current (30A) LEDs and a 12V power supply. The Serial LCD and Wi-Fi connection will combine to serve as a user interface. This requires the user to procure a device with a web browser to interact with the controller, which frees the user from having to physically touch the controller to alter most settings. The three pairs of power boards and LEDs are the electronics responsible for reliably flashing when given an input signal. The microcontroller will coordinate all of the actions – serving webpages of settings over Wi-Fi and controlling the timing of the strobes.

Chapter 2

Components and Sub-Assemblies

2.1 Hardware On Hand

One of the first requirements decided upon was the LED and power board hardware. The Edgerton Center was in possession of three Luminus Devices PhlatLight® PT120s and three V-Tech High-Current LED Driver Boards.

2.1.1 LEDs

The PhlatLight® PT120s produced by Luminus Devices are designed for use in projectors and DLP televisions. As such, they have a single 12mm^2 emitting area and come in typical red, green and blue colors. Each LED has a diffraction grating to eliminate the light bleed that usually occurs at the edges of an LED. Finally, their light output is higher than most other LEDs on the market. A quick overview of the important specifications follows. BCPS ratings are not readily available from the manufacturer, but measured values are given in Section 4.2.

Table 2.1: Key Specifications for Luminus LEDs

Specification	Red	Green	Blue
Wavelength [nm]	623	525	460
Lumens	2225	4300	970
Drive Current[A]	30	30	30
Max Duty Cycle [%]	25	50	25

2.1.2 Power Boards

The V-Tech power boards are current sourcing boards that take a 12-18V input. An inverted TTL signal controls the output, with up to a 51A current supplied to the LED. Key specifications for the board follow.

Table 2.2: Key Specifications for V-Tech Power Board

Specification	Value Range
Input Voltage	12V to 18V
Output Voltage	2.2V to 6.1V
Output Current	12A to 51A (adjustable)
Min pulse	400ns
Max trigger rate	5kHz

2.1.3 Power Supply

Initially, a power supply with three 12V rails was present in lab. It failed during the first few weeks of development and was replaced with another equivalent power supply. The power supply is rated to take 115V at 7A input while providing a continous output of 12V at 25A. This meshes well with the power boards, allowing them to approach the LEDs maximum 30A current.

2.2 Hardware Decisions

With the LEDs and power boards already supplied, the next step is to select a power supply, microcontroller and Wi-Fi card. These parts needed to satisfy a few major requirements:

- **Off the shelf compatibility -** As a proof of concept device using current technology, it is ideal that the hardware be easy to both acquire and work with. Attributes that contribute to this are the existance of open source libraries and utilization of standard interfaces such as UART or SPI. Another consideration is that off-the-shelf parts are more inexpensive and allow for a greater focus on the overall product.

- **Reasonable power requirements -** One of the largest benefits of using LEDs instead of flash tubes is the reduced power requirements of the LEDs. The other hardware should be able to run off of a regular AC to USB adapter which generally provides about 5W maximum.

- **Serviceability -** Eventually all electronics fail, and serviceability is key in these situations. If this device is to be used in Strobe Lab, then it should be both durable and have parts that can be replaced or upgraded easily in the future.

2.2.1 Microcontroller

Based on the requirements, the initial choice of microcontroller was an Arduino UNO [2]. The Arduino provides a well known and relatively standard featureset while allowing for easy compatibility with all kinds of peripherals through the use of shields. However, while evaluating other specifications for our ideal microcontroller, two desired improvements over the Arduino UNO were:

- **More memory** - To embed a webserver and the pages it was to serve up, additional flash memory space would allow me to pursue a slightly more sophisticated interface while leaving room for expandable control features. More storage also means more javascript to create an interactive experience.

- **A 16-bit timer with 3 output compare registers** - In preliminary testing, the best waveforms for the V-Tech power boards were generated by using a 16-bit timer and one output compare register per LED. Unfortunately, the Arduino UNO only has one 16-bit timer with two output compare registers.

After some research, the choices became apparent – there was the Arduino Mega 2560 [1] or an Arduino-compatible alternatively, like the Teensy++ 2.0 [9]. Key specifications of the two alternatives compared with the Arduino UNO can be seen below.

Table 2.3: Key Specifications for Arduino UNO, Arduino Mega and Teensy++ 2.0

Specification	Arduino UNO	Arduino Mega 2560	Teensy++ 2.0
Processor	ATmega328	ATmega2560	AT90USB1286
Flash Memory	32K	256K	128K
RAM Memory	2K	8K	8K
EEPROM	1K	4K	4K
16-bit Timers	1	4	2
Cost	$30	$59	$24

Furthermore, the benefit of having a plug-and-play compatible shield is mildly compromised once moving away from an Arduino UNO as wire jumpers must be used to connect the SPI pins on the shield to the corresponding pins on the microcontroller as their positions on the breakout boards do not match. This, combined with the information in Table 2.3, resulted in the Teensy++ 2.0 being selected for its balance of meeting our desires and minimizing cost.

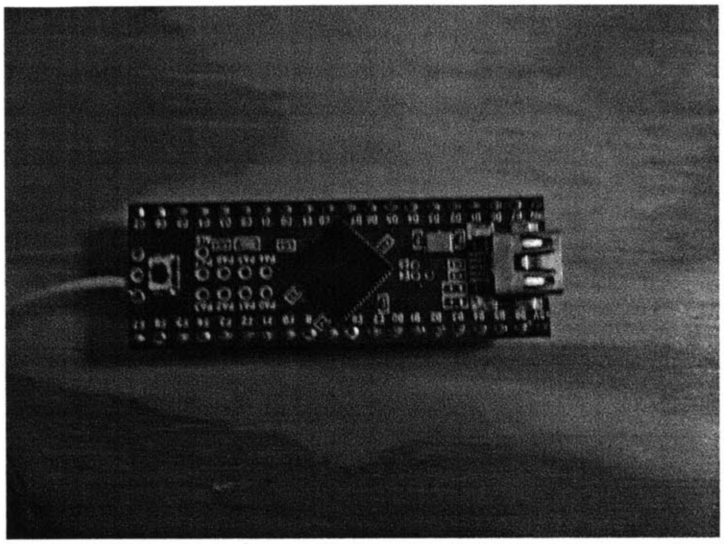

Figure 2-1: Teensy++ 2.0

2.2.2 Wi-Fi Card

The desired means of interaction, a Wi-Fi card, came with a few requirements. It needed to have a straightforward interface with the microcontroller. Fortunately, Wi-Fi breakout boards such as the WiFly Shield, have been created by SparkFun Electronics. The Wi-Fi card communicates with the Teensy through the SPI interface. Additionally, we preferred that the Wi-Fi card have an open source library already made to perform most of the setup and connection handling. Once again, the WiFly shield produced by SparkFun satisfied this requirement with an open source library hosted on GitHub [6].

Figure 2-2: WiFly Shield [5]

2.2.3 Serial Display

Finally, a 16x2 character serial LCD was selected to allow for visual feedback on the device in addition to whatever user interface will be seen over Wi-Fi. Additionally, a physical display allows for compatibility with using DHCP for IP address acquisition as the controller can display its IP address for users to connect to. During development, the serial display was also incredibly useful as a debugging tool.

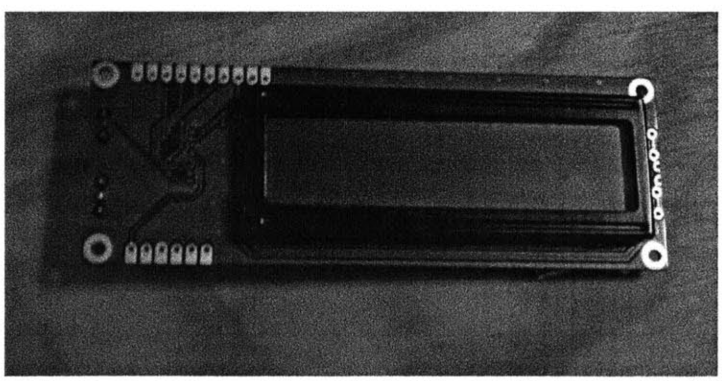

Figure 2-3: Serial LCD screen

2.2.4 Final Parts List

3 Luminus Devices PT120 LEDs (Red, Green, Blue)

3 V-Tech High-Current LED Driver Board

1 12V 25A power supply (HengFu model HF300W-S-12)

1 Teensy++ 2.0

1 WiFly shield (SparkFun p/n: WRL-09954)

1 16x2 Serial LCD (SparkFun p/n: LCD-09393)

Chapter 3

From Parts to Product

3.1 Goals

Moving beyond the vision and hardware, a set of more tangible goals were necessary for the final device. To successfully call this a working proof of concept, it needed to meet three key requirements:

- **Reliable triggering system for sync & delay-** The device must have both a means of receiving a signal from external sensors and a method of inserting a delay between a signal being received and the strobe firing.
- **Three programmable channels for stroboscopy -** The power requirements of the LEDs dictate that each should have its own V-Tech power board. With red, green and blue LEDs, the controller should have an output channel for each light to maximize the versatility of the device. This allows for both mutli-colored flashes and, when all three are used at once, white flashes.
- **Intuitive and accessible user interface -** For this device to be used in lab or in a display, it must have a user interface that is easy to use. It should also be something that can provide a means of viewing and changing all settings and be extended to future devices.

3.2 Hardware

Once gathered, combining the hardware was straightforward. The SPI pins on the WiFly shield were connected to the SPI pins broken out on the Teensy++ 2.0. The V-Tech power boards were connected to three output pins on the Teensy++ 2.0 and the power supply, while the LEDs were each connected to a power board. The serial LCD was wired to a pin on the Teensy++ 2.0 for use with a software serial library as the other serial interface was occupied by the WiFly shield. All components were then connected to the same ground and 5V supply. A summary of connections and the communication methods can be found in Table 3.1.

Table 3.1: Hardware I/O Methods

Device	I/O to Teensy
WiFly Shield	bi-directional SPI
V-Tech LED Driver	3 Digital IO pins
LCD	Software Serial

3.3 Software

Having been successful with acquiring mostly plug-and-play hardware, the bulk of the work was on the software side of things. The LEDs needed to each have a user-adjustable waveform for drive them, the WiFly shield library had to be tweaked to better respond to HTTP requests, and the overall workflow and user interface had to be created.

3.3.1 Timing

Timing will consist of three key components – the triggering system will observe a programmable delay before the flashes begin, then each LED will flash at its own programmable frequency for a programmable length of time. The combination of delay and three programmable frequencies can best be visualized by the diagram in Figure 3-1. Each tick along the lines representing the red, green and blue LEDs indicates a flash. The duration of the flashes can be set independently for each channel.

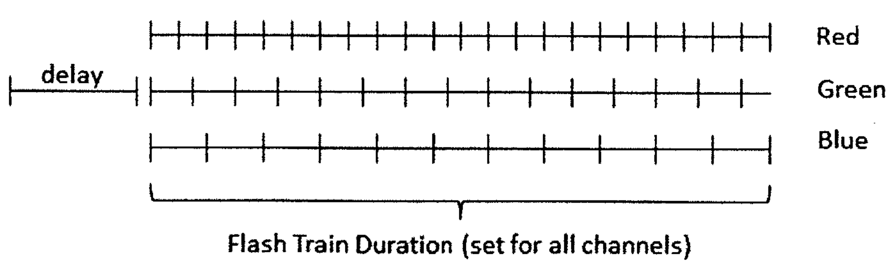

Figure 3-1: Diagram of overall timing

Triggering & Delay

Triggering the controller is controlled by the input capture pin on Timer 1 as it allows for the use of the microcontroller's noise cancellation at the expense of a four clock cycle delay (250ns). This begins the wait of the user-prescribed delay. The delay is also handled with Timer 1 and an output compare interrupt – the interrupt handler enables Timer 3 and starts the generation of red, green and blue waveforms.

Setting up the Waveforms

For each of the three LED channels, the user can set the flash rate and flash duration. The user can also set the time over which the channels will flash (the flash train duration) which is the same for all channels. The user can choose how frequently they want their scene illuminated and how long of a flash is required to properly capture an image more intuitively than other combinations such as on-time and off-time or period and duty cycle.

For timing such as this, it is perferable to use the hardware timers on the microcontroller rather than software polling a system clock or using delay statements. I used Timer3 on the AT90USB which has three output compare registers. Initial attempts at utilizing these parameters ran into the issue of addressing both timer overflows and integer overflow on the output compare registers. In keeping with Occam's Razor, the simplest solution of using an overflow counter to effectively extend the 16-bit timer to 48-bits worked wonderfully. With a 16MHz clock, a 16-bit timer overflows every 4ms while a 48-bit timer will last over 200 days – more than long enough for our purposes. With this structure in place, the number of clock cycles until the next change in the logic level of the waveform was easy to calculate. The output compare register for the corresponding channel would be incremented by the lower 16-bits of this value while the upper 32-bits and any overflow from the incrementation would be added to an unsigned long holding the most significant bits of the next change point.

3.3.2 WiFly Shield

The WiFly shield is shipped with an alpha version of an Arduino compatible library. This library is effective when it comes to communicating with the Wi-Fi module over SPI, though the detection and handling of connections required some tweaking to properly handle. Additionally some revisions were required to enable automatic connecting to a saved Wi-Fi network.

The addition of a struct representing URLs aided in creating a more straightforward mutli-endpoint handler capable of serving up the various webpages required for the user interface. The URL struct also simplified the process of collecting and parsing incoming data into relevant settings.

3.3.3 UI Templating Engine

While the WiFly library provided a means of writing data in response to incoming requests, this data had to be stored in some sort of C or C++ data structure. A natural choice is as a type of character array or string. When creating a webpage, it is easiest to work with a full-fledged HTML document that can be tested in local browsers, not editing parts of a long cstring. To simplify this process, I created a script that compiles a set of specified HTML files into a C++ class while detecting developer-specified variables and storing them within the class. When this Pages class is instantiated, it not only manages the data that is each page but also maintains all server side variables such as frequency and flash duration. The current value of these variables is then able to be presented in the UI and utilized by the timing code.

3.4 Ease of Updating

As this is a proof of concept, flexibility for future updates is important. Many of the updates enumerated in Section 5.2 can be easily implemented.

3.4.1 Waveform Generation and Triggering

The software was written to work with the Arduino IDE so as to make getting started with revisions as easy as possible. Where practical, the abstracted Arduino method of doings things is used. The one area where this is not true is in the triggering and waveform generation code. As this code is specialized for the AVR on which it is running, future updates to these modules would be easiest for someone experienced in using AVR timers and interrupts.

3.4.2 User Interface

The templating engine described in Section 3.3.3 was designed to make updating the user interface both quick and simple. To get an updated version of any of the views to the controller takes only a few steps. First, the maintainer runs the python script that creates the Pages class. The Arduino code is then recompiled and this class is automatically included. Finally, the new hex file is uploaded to the controller and the updated view is now live.

Chapter 4

Results and Analysis

4.1 Overview

Overall the construction of the proof of concept strobe controller was successful. It turns on, serves up a web based user interface, has adjustable settings and, when triggered, flashes appropriately.

4.2 LEDs

We evaluated the efficacy of the LEDs used in this project by measuring their BCPS rating. BCPS, or beam-candlepower-seconds, is calculated by analyzing the output of a photocell. The photocell's load resistor, was set to 10kΩ. The equation to determine BCPS is

$$BCPS = \frac{K V_{MAX} D^2_{photocell}}{R_L} \times \Delta t_{1/3} \tag{4.1}$$

where K is a photocell constant (3.73×10^6), V_{MAX} is the highest voltage output by the photocell, $D_{photocell}$ is the distance between the photocell and LED in feet, and $\Delta t_{1/3}$ is the total time where output is greater than $\frac{1}{3}V_{MAX}$[3]. A V_{MAX} of 276mV was measured, with a $\Delta t_{1/3}$ equal to the programmed duration, or 100μs for a 100μs flash.

The combination of V-Tech power board and PT120 LEDs used is exceptional in that the LEDs are kept at peak output for the entire duration of the flash. Flash tube strobes used in Strobe Lab, such as Spot, have a more gradual rise and a sharper drop-off in output intensity. Due to the maintanance of peak intensity and the ability to have a longer flash, the BCPS calculations for these LEDs become a linear function of flash duration. While Spot provides a 500ns flash and has a BCPS rating of 8, the green LEDs used for this device were measured at 0.1 BCPS for a 100μs flash. A highlight of this difference between the two intensity curves of flash tubes and LEDs can be seen in Figure 4-1.

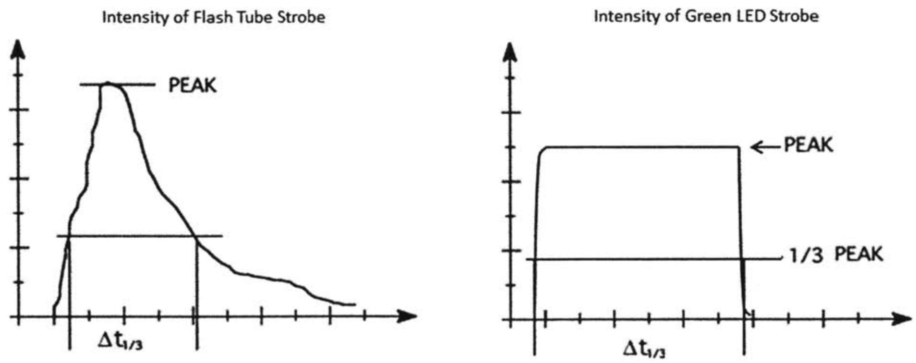

Figure 4-1: Comparison of typical intensity curves of flash tubes and LEDs

4.3 Timing

The timing of the system, in the initial delay, the flash duration and the flash frequency, were found to be accurate enough for our purposes.

4.3.1 Delay

The delay was measured from the input signal to the first flash. Sample measurements showed a margin of error of $\pm 0.05\mu s$. As most delays are on the order of milliseconds, this is more than adequate. The upper limit on the delay is the maximum value of a 48-bit timer operating at 16MHz, or approximately 200 days. The minimum delay sucessfully tested is $25\mu s$.

4.3.2 Flash Duraton

While measuring the BCPS of the LEDs, I also measured the accuracy of flash duration settings. For each setting tested, the flash was at peak output for the programmed time $\pm 0.1\mu s$. An additional time of $1\mu s$ is partially illuminated as the LED sharply rises and falls to and from the peak intensity. In most applications, the LEDs will be active on the order of $100\mu s$ in order to properly illuminate the scene, meaning an error of $1 \pm .1\mu s$ is on the order of 1%. The lower limit on flash duration is likely to be set by the level of illumination required rather than device operating limits, while the upper limit is the operating limit of the LED, either a 25% or 50% duty cycle, as shown in Table 2.1.

4.3.3 Flash Frequency

Flash frequencies between 10-280Hz were tested. Most foreseeable applications for this strobe controller would be within this range. Errors of up to 0.02Hz were found, but there was no relation between programmed frequency and the error. The maximum frequency of flashes is limited by the V-Tech power board at 4kHz. The minimum is governed by the timer's effective overflow rate, and is theoretically one flash every 200 days.

4.4 Shortcomings

As successful as the prototype is, it is not without its shortcomings. As a consequence of being a singly threaded environment, the Wi-Fi connection is only capable of handling one connection at a time. This means that if the device is receiving a request or sending a response to a client other incoming connections will be ignored.

Chapter 5

Contributions

5.1 What's been done

Throughout this thesis I have worked to integrate new technology into a proof of concept for a new piece of Strobe Lab equipment while maintaining the ability to perform future upgrades. More specifically, I have:

- Combined off-the-shelf hardware and high output LEDs into a single working prototype of an LED strobe controller
- Embedded a web server into the strobe controller capable of serving webpages as a user interface
- Created a framework for future user interface updates to be easily compiled into Arduino code and uploaded to the controller
- Developed a scheme for defining, reading, updating and displaying server-side variables in a natural and straightforward manner
- Proposed a set of next-generation developments that would improve the overall user experience in the use of strobes in Strobe Lab

5.2 Future Work

5.2.1 Consolidate Multiple Controllers

The use of a web-based interface means that we could try to consolidate more controllers and their interfaces, usually a combination of buttons and a screen, into a single, smart controller. This would allow for the purchase of other "dumb" components and for a unified interface for students and users to learn. This could simplify the workflow and give one place to change between setups for several photographic techniques with the push of a button.

5.2.2 LEDs

As technology continues to improve, brighter LEDs can be used to further increase the BCPS rating of an LED strobe. Additionally, as lights become more efficient, it may be possible to reduce the power supply requirements and further improve upon form factor or cost.

5.2.3 Microcontroller

Using other hardware, even a small computer, such as the recently released Raspberry Pi, could open up many new possibilities and overcome a few current limitations [7].

Multi-client Webserver

The current hardware is limited to a servicing a single client request at a time. With a small, inexpensive device such as the Raspberry Pi, existing mutli-client webservers such as Apache could easily be used. Alternatively, a framework such as Flask or Django could also be used to create highly interactive web pages. With currently available kernels, this hardware could use a cheaper Wi-Fi dongle and communicate with a microcontroller to pass along variable values over an interface such as SPI. A real time kernel would allow for all operations to take place on the Raspberry Pi.

More Accurate Timing

Either a different microcontroller or a small linux implementation with a real time kernel with a higher clock speed would allow for more fine-grained timing. When flashes are supposed to either toggle on or off at the same time, there is always a split-second delay and changes do not occur at exactly the same time. In the current situation, this delay is already minimal, but the higher the clock speed the smaller and less significant this delay becomes.

Cost

Rather than a relatively inexpensive microcontroller and a costly Wi-Fi card, hardware like the Raspberry Pi can use relatively inexpensive USB Wi-Fi dongles. The combined cost of the controller and Wi-Fi connectivity is lower in the case of the Raspberry Pi. Additionally, these two configurations use approximately the same space and have wire connections.

Another option to reduce the cost is to create a custom board for both the microcontroller and Wi-Fi module while selecting a microcontroller with specifications that are closer to the baseline requirements and have less room for future growth. This would make it more difficult or impossible to increase the number of devices controlled by the device, but would successfully reduce the overall cost.

Increased Storage

Other devices are easily able to store information on an SD card and access the filesystem. While this is possible with the current microcontroller, it makes more sense on a more computer-like system. This would allow for archiving of previous setting configurations and the removal of the template conversion layer described in Section 3.3.3.

5.2.4 User Interface

A more responsive interface could be made. Ideally it would use more AJAX calls to update settings transparently while allowing the periodic "preview" of their effect on the device's output. Anything that would speed up the iterative process of testing delays or flash rates on a new laboratory setup would be useful.

Appendix A

Appendix A: Arduino C++ Code

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <avr/pgmspace.h>
#include <stdint.h>
#include <stdbool.h>
#include <WiFly.h>
#include <SPI.h>
#include <SoftwareSerial.h>
#include <serLCD.h>
#include "bitmagic.h"
#include "timemagic.h"
#include "webpages.h"
#include "strobe_arduino.h"

float p[] = {100, 100, 100}; // pulse duration in us
float f[] = {59.0, 60.0, 61.0}; // pulse frequency in Hz
float tot_duration = 3;
unsigned long int t_delay = 0;
bool extTrigger = true;


unsigned long int overflows = 0;
unsigned long int ovf_targ[3] = {0,0,0};

unsigned long int cycles[3][2] = {{0,0},{0,0},{0,0}};
unsigned long int ovfs[3][2] = {{0,0},{0,0},{0,0}};
int ovf_cnt[3] = {0, 0, 0};
int state[3] = {1,1,1};

unsigned long int delayOvfs = 0;
unsigned long int targDelayOvfs = 0;
unsigned long int delayCycles = 0;

int deadcount[3] = {0,0,0};

int OP_STAGE = 0;

WiFlyServer server(80);
```

```
serLCD lcd(26);
Pages pages;

void setup()
{
  CPU_PRESCALE(0x00);
  DDRB |= (1<<7);
  DDRD |= (1<<0);
  DDRD |= (1<<1);
  DDRD |= (1<<6);
  SETBIT(PORTB, 7);
  SETBIT(PORTD, 0);
  SETBIT(PORTD, 1);
  delay(100);

  Serial.begin(38400);
  pinMode(26, OUTPUT);

  delay(100);

  lcd.setBrightness(20);
  lcd.clear(); delay(100);
  lcd.selectLine(1); lcd.print(F("3-LED Strobe:"));
  lcd.selectLine(2); delay(100);

  WiFly.begin();
  int failcount = 0;
  lcd.print(".");
  while(strstr("0.0.0.0", WiFly.ip()))
  {
    if(failcount == 10){
      lcd.clear(); delay(100);
      WiFly.createAdHocNetwork("strobeWiFiSetup");
      lcd.selectLine(1); lcd.print(F("strobeWiFiSetup"));
      lcd.selectLine(2); lcd.print(WiFly.ip());
      break;
    }
    lcd.print(".");
    delay(500);
    failcount++;
  }

  // Connected to a network, even if ad hoc
  OP_STAGE = 1;
  Serial.print("IP: "); Serial.println(WiFly.ip());
  lcd.clear(); delay(100);
  lcd.selectLine(1); lcd.print(F("Connect on")); delay(100);
  lcd.selectLine(2); lcd.print(WiFly.ip()); delay(100);

  pages.changeSetting("MAX_F_VAL", 120);
  pages.changeSetting("MAX_P_VAL", 200);
  pages.changeSetting("MAX_T_VAL", 5);
  pages.changeSetting("F0", 60);
  pages.changeSetting("F1", 60);
  pages.changeSetting("F2", 60);
  pages.changeSetting("P0", 100);
  pages.changeSetting("P1", 200);
  pages.changeSetting("P2", 300);
  pages.changeSetting("T", 2.5);
```

```
  pages.changeSetting("TRIG", 0.0);
  pages.changeSetting("TDELAY", 10.0);
  pages.changeSetting("MAX_TDELAY", 10000.0);

  sei(); //Global interrupt enable
  Serial.println(F("Setup complete"));
}

void loop()
{
  if(OP_STAGE == 1){ // in server stage
    server.begin();
    WiFlyClient client = server.run();
    if (client) {
      lcd.clear(); delay(100);
      lcd.selectLine(1); lcd.print(F("Connected"));
      // an http request ends with a blank line
      boolean current_line_is_blank = true;
      unsigned long tstart = millis();
      while (client.connected()) {
        if (client.available()) {
          char c = client.read();
          // if we've gotten to the end of the line (received a newline
          // character) and the line is blank, the http request has ended,
          // so we can send a reply
          if (c == '\n' && current_line_is_blank) {
            URI u = client.getURI();
            Serial.print("URI: "); Serial.print(u.uri);
            Serial.print(" || "); Serial.println(u.params);
            // send a standard http response header
            client.println(F("HTTP/1.1 200 OK"));
            client.println(F("Content-Type: text/html"));
            client.println();
            if(strcmp(u.uri, "/wifi") == 0){
              lcd.selectLine(2); lcd.print(F("WiFi Setup"));
              pages.writeWiFi(&client);
            }else if(strcmp(u.uri, "/time") == 0){
              lcd.selectLine(2); lcd.print(F("Timing"));
              if(strlen(u.params) > 5){
                setTimeSettings(u.params);
              }
              pages.writeTime(&client);
            }else if(strcmp(u.uri, "/gensettings") == 0){
              lcd.selectLine(2); lcd.print(F("Settings Home"));
              if(strlen(u.params) > 0){
                pages.reloadSettings(u.params);
              }
              pages.writeGenSettings(&client);
            }else if(strcmp(u.uri, "/settings") == 0){
              lcd.selectLine(2); lcd.print(F("Settings"));
              pages.writeSettings(&client);
            }else if(strcmp(u.uri, "/about") == 0){
              lcd.selectLine(2); lcd.print(F("About"));
              pages.writeAbout(&client);
            }else if(strcmp(u.uri, "/start") == 0){
              OP_STAGE = 2;
              pages.writeStandby(&client);
              client.stop();
              return;
```

```
        }else if(strcmp(u.uri, "/") == 0){
          lcd.selectLine(2); lcd.print(F("Home"));
          pages.writeHome(&client);
        }else{
          client.println();
          client.stop();
          break;
        }
        client.println();
        break;
      }
      if (c == '\n') {
        // we're starting a new line
        current_line_is_blank = true;
      } else if (c != '\r') {
        // we've gotten a character on the current line
        current_line_is_blank = false;
      }
    }
  }
  // give the web browser time to receive the data
  delay(750);
  client.stop();
  lcd.clear(); delay(100);
  lcd.selectLine(1); lcd.print("Connect on");
  lcd.selectLine(2); lcd.print(WiFly.ip());
}
}else if(OP_STAGE == 2){ // arming
  cli();

  lcd.clear();
  lcd.selectLine(1); lcd.print(F("Preparing"));
  lcd.selectLine(2); delay(100);

  Serial.println(F("Loading all settings"));
  loadSettings(); lcd.print(F(".")); delay(100);

  Serial.println(F("Updating clock calculations"));
  update_clock_calcs(); lcd.print(F(".")); delay(100);

  Serial.println(F("Setting up timers"));
  // Input capture/output compare for triggering
  TCCR1A = 0;
  TCCR1B = 0;
  TCCR1B |= (1 << ICNC1 | 1 << ICES1 | 1 << CS10);
  SETBIT(TIMSK1, ICIE1); lcd.print(F(".")); delay(100);

  // Timer 3 output compare and ovf for pulsing
  TCCR3A = 0;
  TCCR3B = 0;
  TCCR3B |= (1 << CS30);
  SETBIT(TIMSK3, TOIE3); lcd.print(F(".")); delay(100);
  SETBIT(TIMSK3, OCIE3A); lcd.print(F(".")); delay(100);
  SETBIT(TIMSK3, OCIE3B); lcd.print(F(".")); delay(100);
  SETBIT(TIMSK3, OCIE3C); lcd.print(F(".")); delay(100);

  lcd.clear(); delay(100);
  lcd.selectLine(1); lcd.print("Ready!");
```

```
    OCR1A = delayCycles;
    CLEARBIT(PRR0, PRTIM1); // Timer1
    sei();
    if(!extTrigger){ //Start timer3 pulsing now
      Serial.println("Not using external trigger. Going now");
      cli();
      if(t_delay > 16000){
        delay(t_delay/1000);
        delayMicroseconds(t_delay % 1000);
      }else{
        delayMicroseconds(t_delay);
      }
      CLEARBIT(PORTB, 7);
      CLEARBIT(PORTD, 0);
      CLEARBIT(PORTD, 1);
      CLEARBIT(PRR1, PRTIM3);
      sei();
      OP_STAGE = 3;
    }else{
      Serial.println("Waiting for external trigger");
      cli();
      CLEARBIT(PRR0, PRTIM1);
      sei();
      delay(5*60*1000); //wait up to 5 minutes before setting it off on your own
      OP_STAGE = 3;
    }
  }else if(OP_STAGE == 3){ // firing
    Serial.println(F("OP_STAGE 3"));
    // do nothing, let the interrupts run it
    delay((int)(tot_duration*1000));
    OP_STAGE = 4;
  }else if(OP_STAGE == 4){ // cooldown/reset
    Serial.println(F("OP_STAGE 4"));
    cli();
    SETBIT(PORTB, 7); // LEDs off
    SETBIT(PORTD, 0);
    SETBIT(PORTD, 1);
    SETBIT(PRR1, PRTIM3); // Timer3
    SETBIT(PRR0, PRTIM1); // Timer1
    sei();
    OP_STAGE = 1;
    Serial.println(F("RETURN TO OP_STAGE 1"));
    lcd.clear(); delay(100);
    lcd.selectLine(1); lcd.print("Connect on");
    lcd.selectLine(2); lcd.print(WiFly.ip());
  }
}

void update_clock_calcs(){
  for (int i = 0; i < 3; i++)
  {
    ovf_targ[i] = 0;
    state[i] = 1;
    ovf_cnt[i] = 0;
    deadcount[i] = 0;
  }
  overflows = 0;
  delayOvfs = 0;
  targDelayOvfs = 0;
```

```
  delayCycles = 0;

  for(int i = 0; i < 3; i++){
    cycles[i][1] = F_CPU*(p[i]*1.0/1000000); // corresponds to pulse length
    cycles[i][0] = F_CPU/f[i] - cycles[i][1]; //corresponds to T-pulse length
    ovfs[i][0] = cycles[i][0] >> 16;
    ovfs[i][1] = cycles[i][1] >> 16;
    for(int j = 0; j < 2; j++){
      if( ovfs[i][j] > 0){
        cycles[i][j] &= 0xFFFF;
        if( cycles[i][j] == 0 ){
          ovfs[i][j]--;
          cycles[i][j] = 0xFFFE;
        }
      }
    }
  }

  delayCycles = F_CPU*(t_delay/1000000);
  targDelayOvfs = delayCycles >> 16;
  OCR1A = delayCycles;

  // Fix initial conditions
  for(int i = 0; i < 3; i++){
    ovf_targ[i] = ovfs[i][1];
    OCR_set(i, cycles[i][1]);
  }
}

void setTimeSettings(char* params){
  char* tok = strtok(params, "&=");
  while(tok){
    char* val = strtok(NULL, "&=");
    float v = atof(val);
    Setting s = {tok, v};
    Serial.print(s.name); Serial.print(" "); Serial.println(s.val);
    pages.changeSetting(s);
    tok = strtok(NULL, "&=");
  }
}

void setWiFiSettings(char* params){
  char* ssid;
  char* phrase;
  char* auth;
  char* tok = strtok(params, "&=");
  while(tok){
    char* val = strtok(NULL, "&=");
    if(strstr(tok, "AUTH")){
      auth = val;
    }else if(strstr(tok, "SSID")){
      ssid = val;
    }else if(strstr(tok, "PASS")){
      phrase = val;
    }
    boolean isWPA = true;
    if(strstr(auth,"0") || strstr(auth,"1")){ isWPA = false;}
    WiFly.setWiFiSettings(ssid, phrase, auth, isWPA);
    tok = strtok(NULL, "&=");
```

```
  }
}

void loadSettings(){
  p[0] = pages.getSetting("P0");
  p[1] = pages.getSetting("P1");
  p[2] = pages.getSetting("P2");
  f[0] = pages.getSetting("F0");
  f[1] = pages.getSetting("F1");
  f[2] = pages.getSetting("F2");
  tot_duration = pages.getSetting("T");
  t_delay = (int)pages.getSetting("TDELAY");
  extTrigger = (pages.getSetting("TRIG") > 0.1 ? true : false);
  Serial.print(f[0]); Serial.print(" "); Serial.print(f[1]); Serial.print(" ");Serial
      .print(f[2]); Serial.println();
  Serial.print(p[0]); Serial.print(" "); Serial.print(p[1]); Serial.print(" ");Serial
      .print(p[2]); Serial.println();
  Serial.print(t_delay); Serial.print(" "); Serial.print(tot_duration); Serial.
      println();
}

ISR(TIMER3_OVF_vect){
  overflows++;
}

void comp_func(int i){
  if(ovf_targ[i] == overflows){
    channel_toggle(i);
    state[i] = (state[i]+1) & 0x1;
    unsigned int ocr_old = OCR_get(i);
    OCR_inc(i, cycles[i][state[i]]);
    ovf_targ[i] += ovfs[i][state[i]];
    unsigned int ocr_new = OCR_get(i);
    if(ocr_new < ocr_old){
      ovf_targ[i]++;
    }
    if(ocr_new < 5){
      OCR_set(i, 5);
    }else if(ocr_new > 0xFFFA){
      OCR_set(i, 0xFFFA);
    }
  }
}
ISR(TIMER3_COMPA_vect){ comp_func(0); }
ISR(TIMER3_COMPB_vect){ comp_func(1); }
ISR(TIMER3_COMPC_vect){ comp_func(2); }

ISR(TIMER1_CAPT){
  unsigned char sreg = SREG;
  cli();
  SETBIT(TIMSK1, TOIE1);
  SETBIT(TIMSK1, OCIE1A);
  OCR1A = delayCycles;
  TCNT1 = 0x0000;
  SREG = sreg;
  sei();
}
ISR(TIMER1_OVF_vect){delayOvfs++;}
ISR(TIMER1_COMPA_vect){
```

```
  if(delayOvfs==targDelayOvfs){
    // Start the flashing
    CLEARBIT(PORTB, 7);
    CLEARBIT(PORTD, 0);
    CLEARBIT(PORTD, 1);
    CLEARBIT(PRR1, PRTIM3); // Timer3
  }
}
```

Appendix B

Appendix B: User Interface and Templating

B.1 Example Page Template

```
<!DOCTYPE html>
<html>
<head>
  <title>
    3-LED Strobe
  </title>
  <meta name='viewport' content='width=device-width, initial-scale=1'>
  <link rel='stylesheet' href='http://code.jquery.com/mobile/1.1.1/jquery.mobile
      -1.1.1.min.css' />
  <script src='http://code.jquery.com/jquery-1.7.1.min.js'></script>
  <script src='http://code.jquery.com/mobile/1.1.1/jquery.mobile-1.1.1.min.js'></
      script>
</head>

<body>

<div data-role='page' data-add-back-btn="true" data-theme="a">

  <div data-role='header'>
    <a href="/" data-icon="home" class="ui-btn-left"> </a>
    <a href="/settings" data-icon="gear" class="ui-btn-right"> </a>
    <h3>Timing Settings</h3>
  </div><!-- /header -->

  <div data-role='content'>
    <form id="time_form">
        <h4>Frequency [Hz]</h4>
        <div data-role='fieldcontain'>
          <label for='F0'>Red:</label>
          <input type='range' name='F0' id='F0' value='{{F0}}' min='0' max='{{
              MAX_F_VAL}}' step='0.1' data-mini="true" data-highlight="true" />
```

```
        </div>
        <div data-role='fieldcontain'>
          <label for='F1'>Green:</label>
          <input type='range' name='F1' id='F1' value='{{F1}}' min='0' max='{{
              MAX_F_VAL}}' step='0.1' data-mini="true" data-highlight="true" />
        </div>
        <div data-role='fieldcontain'>
          <label for='F2'>Blue:</label>
          <input type='range' name='F2' id='F2' value='{{F2}}' min='0' max='{{
              MAX_F_VAL}}' step='0.1' data-mini="true" data-highlight="true" />
        </div>
        <h4>Pulse Duration [&#956;s]</h4>
        <div data-role='fieldcontain'>
          <label for='P0'>Red:</label>
          <input type='range' name='P0' id='P0' value='{{P0}}' min='100' max='{{
              MAX_P_VAL}}' step='10' data-mini="true" data-highlight="true" />
        </div>
        <div data-role='fieldcontain'>
          <label for='P1'>Green:</label>
          <input type='range' name='P1' id='P1' value='{{P1}}' min='100' max='{{
              MAX_P_VAL}}' step='10' data-mini="true" data-highlight="true" />
        </div>
        <div data-role='fieldcontain'>
          <label for='P2'>Blue:</label>
          <input type='range' name='P2' id='P2' value='{{P2}}' min='100' max='{{
              MAX_P_VAL}}' step='10' data-mini="true" data-highlight="true" />
        </div>
        <h4>Total Duration [s]</h4>
        <div data-role='fieldcontain'>
          <label for='T'></label>
          <input type='range' name='T' id='T' value='{{T}}' min='0.05' max='{{
              MAX_T_VAL}}' step='0.01' data-mini="true" data-highlight="true" />
        </div>
        <h4>External Triggering</h4>
        <div data-role="fieldcontain">
          <label for="TRIG">Enable:</label>
          <select name="TRIG" id="TRIG" data-role="slider">
            <option value="1">ON</option>
            <option value="0">OFF</option>
          </select>
        </div>
        <div data-role='fieldcontain'>
          <label for='TDELAY'>Delay[&#956;s]:</label>
          <input type='range' name='TDELAY' id='TDELAY' value='{{TDELAY}}' min='0.1'
              max='{{MAX_TDELAY}}' step='0.1' data-mini="true" data-highlight="true"
              />
        </div>
      <input type='submit' value='Save Settings'>
    </form>
  </div><!-- /content -->

</div><!-- /page -->
<script>
trig = Math.ceil({{TRIG}});
$(function(){
  $('#time_form').submit(function(){
    $.get('/time', $('form').serialize());
    window.location.replace('/');
    return false;
```

```
  });
  $('#TRIG').val(trig).slider('refresh');
});
</script>


</body>
</html>
```

B.2 Script to convert templates to a C++ Pages class

```
import re

MATCH_STRING = r'\{\{(?P<name>[A-Z0-9_]+)\}\}'

header = """
#ifndef _PAGES_H
#define _PAGES_H

#import <Stream.h>

struct Setting{
  char* name;
  float val;
};

class Pages{
  public:
    Pages(){
      %s
    }
""" # takes setting initializers

footer = """

    void changeSetting(char* name, float val){
      for(int i = 0; i < sizeof settings; i++){
        if(strcmp(name, settings[i].name) == 0){
          settings[i].val = val;
        }
      }
    }
    void changeSetting(Setting s){
      for(int i = 0; i < sizeof settings; i++){
        if(strcmp(s.name, settings[i].name) == 0){
          settings[i] = s;
          return;
        }
      }
    }
    float getSetting(char* name){
      for(int i = 0; i < sizeof settings; i++){
        if(strcmp(name, settings[i].name) == 0){
          return settings[i].val;
        }
      }
    }
    void reloadSettings(char* params){
      Serial.println("checking out query params");
      char* tok = strtok(params, "=");
      while(tok){
        int i = atoi(tok);
        float v = atof(strtok(NULL, "&"));
        settings[i].val = v;
        Serial.print(settings[i].name); Serial.print(" : "); Serial.println(settings[
            i].val);
```

```
        tok = strtok(NULL, "=");
      }
    }
  private:
    Setting settings[%i];
};
#endif
"""

setting_init = """
      Setting a%(idx)i = {"%(name)s", 200};
      settings[%(idx)i] = a%(idx)i;
"""

template = """
    void write%s(Stream* s){
      %s
      s->println();
    }
""" # 1st: shortname, 2nd: print statements


out = open('webpages.h', 'w')
manifest = open('manifest_web.txt', 'r')
files = manifest.readlines()

params = []
pages = {}
for f in files:
  cname, fname = f.split()
  src = open(fname, 'r')
  olines = []
  for line in src.readlines():
    if len(line.strip()) > 0:
      line = "%s\\n" % line.strip().replace('"', '\\\"').replace("\n", " ")
      m = re.findall(MATCH_STRING, line)
      if m:
        for x in m:
          if x not in params:
            params.append(x)
      [olines.append(x) for x in re.split(MATCH_STRING, line)]
  comblines = [""]
  for l in olines:
    if l in params:
      comblines.append(l)
      comblines.append("")
    else:
      comblines[len(comblines)-1] += l
  pages[cname] = comblines
  src.close()
print pages

params.sort()
params.sort(key=len) # Make the params come out nicely


settingInitializers = ""
count = 0;
for p in params:
```

```
    settingInitializers += setting_init % {'idx':count, 'name':p}
    count+=1
out.write(header % settingInitializers)

for p in pages:
    printStatements = ""
    for line in pages[p]:
        printStatements += "s->print(%s);\n      " % ("settings[%i].val" % params.index(
            line) if line in params else ("F(\"%s\")" % line))
    out.write(template % (p, printStatements))



settingsTemplate = """
    void writeGenSettings(Stream* s){
      %s
      s->println();
    }
"""

template = [""]
try:
    f = open('settings_generator_template.html', 'r')
    i = 0
    for l in f.readlines():
        l = l.strip()
        if "{{SETTINGS_FORM}}" not in l:
            template[len(template)-1] += l
        else:
            for p in params:
                template[len(template)-1] += '''<div data-role="fieldcontain"> <label for="%(
                    id)i">%(name)s:</label> <input type="text" name="%(id)i" id="%(id)i"
                    value="''' % {'name':p, 'id':params.index(p)}
                template.append(p)
                template.append('''" /></div>''')
    comblines = [""]
    for l in template:
        l = l.strip().replace('"', '\\"').replace("\n", " ")
        if l in params:
            comblines.append(l)
            comblines.append("")
        else:
            comblines[len(comblines)-1] += l
    template = comblines
except Exception, e:
    raise
finally:
    f.close()

settingsStatements = ""
for t in template:
    settingsStatements += "s->print(%s);\n      " % ("settings[%i].val" % params.index(
        t) if t in params else ("\"%s\"" % t))

out.write(settingsTemplate % settingsStatements)

out.write(footer % len(params))
out.close()
```

Bibliography

[1] Arduino. Arduino mega 2560.
http://arduino.cc/en/Main/ArduinoBoardMega2560, 2012.

[2] Arduino. Arduino uno.
http://arduino.cc/en/Main/arduinoBoardUno, 2012.

[3] Dr. James W. Bales. Guidelines for 6.163 laboratory memos.
http://web.mit.edu/6.163/www/Resources/Memo_Guidelines_6.163_F11.pdf, 2011.

[4] Dr. James W. Bales. personal communication, 2012.

[5] SparkFun Electronics. 09954-01b.
via SparkFun, Creative Commons Attribution., 2012.

[6] SparkFun Electronics. Wifly shield library.
https://github.com/sparkfun/WiFly-Shield, 2012.

[7] Raspberry Pi Foundation. Raspberry pi.
http://www.raspberrypi.org/, 2012.

[8] Spence Allen Nicholas Lee and Elizabeth Smith. Motion of the tip of a snapped towel.
http://hiviz.org/hsi/galleries/eastman/hspexhibit08.pdf, 1993.

[9] Paul Stoffregen and Robin Coon. Teensy usb development board.
http://www.pjrc.com/teensy/index.html, 2012.