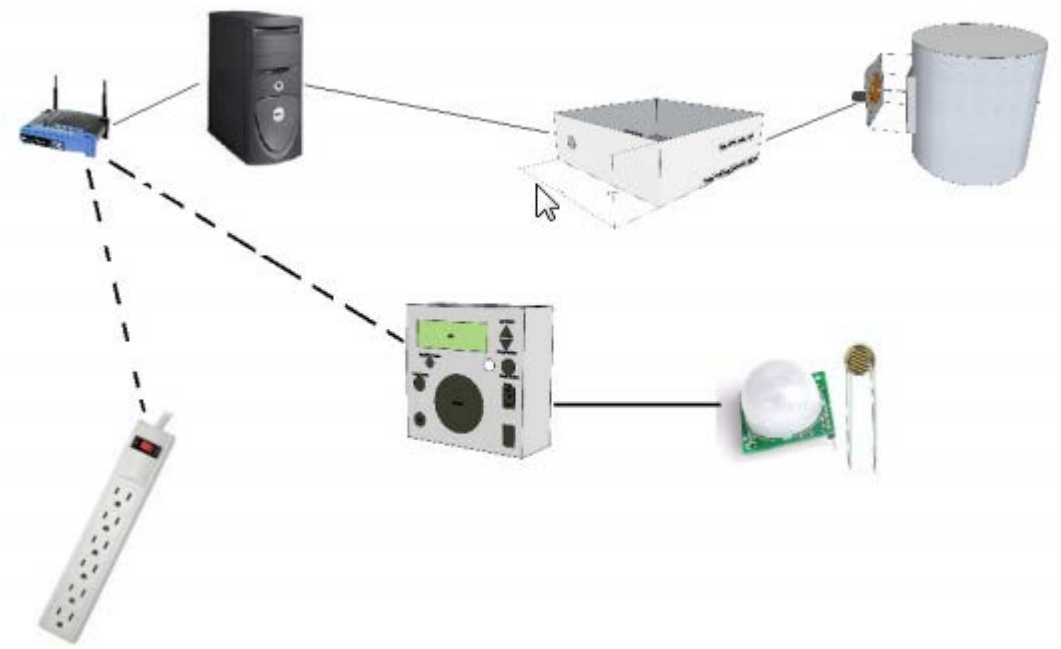
TODD ROGERS
CHRISTOPHER JOHNSON
DARIO BOSNJAK
LEVI BALLING

SMART HOME

COMPUTER ENGINEERING FINAL PROJECT

2012



T A B L E O F C O N T E N T S

INTRODUCTION

While the prices of living are going up, the main focus is to involve technology to assist us with maintaining those prices and still provide manageable living costs. With this in mind the Smart Home project will allow us to build and maintain a house that is smart enough to keep the energy levels down while providing the most automation possible. A smart home will take advantage of its environment and allow seamless control whether you are present or away. With a home that has this advantage, you can know that your home is performing at its best in energy performance.

By implementing this project we were able to explore a variety of different fields of engineering, including software programing, PCB design, Wi-Fi TCP/IP protocols, Web Server logic design, and other aspects. This project turned out to be providing great insights to both sides of engineering hardware and software vise.

OVERVIEW

OVERALL SOFTWARE FUNCTION

Implementation of multiple hardware components is necessary to provide the functionality that will be further discussed in this document. Behind the complex hardware involved in controlling the smart home project there is a fair amount of software architecture that is responsible for driving the hardware components. Each part of the project is built and designed with a different functionality in mind. Some of the software is coded in Phyton, some in Arduino code (based on C language). Functionalities of the software involved are as following:

- Control stepper motors based on values pre-determined in the code
- Collect data from input sensors (temperature, moisture, laser's current…)
- Maintain a web server running on Arduino
- Perform complex logic to control the functionality of smart home
- Manipulate relays
- Automate the smart home functionality

OVERALL HARDWARE FUNCTION

Hardware involved in the project is responsible to physically demonstrate the functionality of the smart home project. Based on some software commands received from the server specific hardware is turned on and as per designed eningeering it performs its functionality. Features of hardware are as follows:

- Stepper motors responsible for closing/opening air docks based on the temperature
- Sensors monitoring temeperature in the room's
- Sensors monitoring current occupancy of the room's
- Sensors monitoring moisture levels of soil

- Current measuring sensors that report back to the server
- Relays and Wi-Fi modules controlled by the Server

W i - F i F U N C T I O N

INTRODUCTION

In order to communicate over long distances without running wires, we came up with a more convenient way of communicating with our sensors and with controlling different I/O devices by using TCP/IP over Wi-Fi.802.11 g protocol. Data being gathered from sensors such as (heat sensors, light sensors, laser trip wire sensors) is being processed on an Arduino Micro-controller and then broad-casted with an attached WiShield v2.0 to a server over TCP/IP protocol. Arduino has an statically assigned IP address that corresponds to an individual room in the house. Each time an request is made to that IP address a HTML page is returned with implemented functionality. One of the perks of using HTML is that data can be viewed from all of the sensors in one location and also to control remotely Input/Output devices such as power strip plugs.

SOFTWARE FUNCTION

The main functionality of software is responsible to monitor the changes in attached hardware and also to to initiate controlling statements that depending on the log-in involved would trigger an invent based on that log-in.

- Monitor analog inputs go gather moisture change in ground and light intensity in order to turn on an digital output.
- Create software serial communication in order to communicate with another Arduino responsible for controlling power strip plugs
- Create a Arduino hosted web server responsible for keeping track of sensor information and current states of attached devices.
- Gather and store current sensor information and store it for clients to see.

Software is based on Arduino code that is based on C programming language. It consists of libraries that create Web Servers for Arduino MCU's and also libraries responsible of setting up software serial communication to another Arduino.

Arduino consists of code that initiates HTML page over the web server that is supported on Arduino. Once the page is established series of loops are ran on the arduino to constantly measure different sensor values that are hard wired in to the arduino over the analog inputs. Once those values are known to the code they are stored in variables that can be displayed over HTML. Once an request is made to the arduino hosting the web page the returned HTML page consists of fields populated by sensor values, and options to manually turn on power strip plugs. Software side also takes advantage of digital inputs on the arduino by utilizing them to monitor

values coming from motion detector sensor and laser trip wire sensor. If these events are triggered they automatically get stored in a variable that is translated and displayed on the web page for clients to examine. Another feature that is implemented by software side is that we are able to do logic when it comes to manipulating I/O devices. One of the features involves ability to check the moisture of soil and also check the time of day by monitoring the light intensity and if both parameters meet the logic criteria a digital enable gets sent which triggers the hardware side and starts the process designed by hardware. Web Server will be able to listen on requests that are being sent over the assigned IP address in this case http://192.168.1.102 and upon receiving will display current state of the sensors and I/O devices. In case an I/O device needs to be triggered by logic already pre determined an request of http://192.168.1.102/?LED0 … LED3 will be sent which will cause an event to be triggered and will execute a specified block of code responding to that URL page, and in most cases turning on an appliance connected to that power plug being controlled by that URL. Actual code used in the project is attached at the bottom of the document in *Appendix A.*

HARDWARE FUNCTION

So far software is responsible for doing most of the work when it comes to communication between the devices and TCP/IP protocol. In order to successfully allow the control and monitoring of different devices I am implementing Arduino Duemilanove MCU with an WiShield v2.0 Wi-Fi wirelles adapter network card that supports static IP address assignments. The power usage of the Wi-Fi Shield with Arduino is low. It requires 5-7 volts. Wi-Fi communication is done over 802.11b at 1Mbps throughput speeds with Netgear wireless N router. A 5V line is connected to the 5V pin on the arduino and the common ground is shared between the Arduino running the power strip outlets. This way noise over the software serial is dramatically limited and communication is enhanced between the two MCU controllers.

- Sleep mode: 250µA
- Transmit: 230mA
- Receive: 85mA
    - SPI
        - o Slave select (SS) : Arduino pin 10 (port B, pin 2)
        - o Clock (SCK) : Arduino pin 13 (port B, pin 5)
        - o Master in, slave out (MISO) : Arduino pin 12 (port B, pin 4)
        - o Master out, slave in (MOSI) : Arduino pin 11 (port B, pin 3)
    - Interrupt (Uses only one of the following, depending on jumper setting)
        - o INT0 : Arduino pin 2 (port D, pin 2)
        - o DIG8 : Arduino pin 8 (port B, pin 0)
    - LED : Arduino pin 9 (port B, pin 1)
        - o To regain use of this pin, remove the LED jumper cap
    - 5V power
    - GND

WiShield v2.0

Figure 1

WiShield will be placed inside a power strip box where it will be in close proximity with the MCU responsible for controlling the power strip outlets. Due to noise disturbance an Linksys Wi - Fi antenna has be soldered on the WiShield and placed outside the box to increase Wi-Fi strength signal between the router and the WiShield.

S P R I N K L E R S Y S T E M F U N C T I O N A L I T Y

INTRODUCTION

In the attempt to make the house as automated as possible a sprinkler control system has been invented that is responsible on turning on the sprinkler based on some pre set parameters. It is well known that it can be expensive to maintain green grass and the last thing you want to do is to waist water when its not necessary to water the grass. In attempt to resolve this issue, we have implemented a soil moisture sensor that measures the soil conductivity and reports it back to the Arduino MCU controller and at the same time it does that, through photo resistive it gathers the information about light intensity to determine if its light or day. This approach will limit the the sprinkler system only to be ran during the night and only if the soil moisture is less than a pre set value.

SOFTWARE FUNCTION

Coded in Arduino code based on C language, through built in hardware by Arduino we are able to monitor the analog input values of pins 0 and 1. Values that are coming in through the pin 0 are values that are gathered from the soil moisture sensor. Sensor is connected with one wire to analog input 0 on the arduino board, values coming in through this input are stored in a int variable that ranges between 0 to 700 conductivity based on the moisture in the soil. In order to come up with these values we connected the sensor to the arduino and coded it to monitor the change in the values when sensor was completely dry and then when its placed in a cup of watter. Once a desired values were achieved and observed, they would be pre coded in the function to be compared against. Analog input 1 is responsible for monitoring the change in light intensity received from the photo cell. Once the lights off the value was about 100 and with light on the value went in to 900 ohm resistance. After determining these values a loop was created

that would every 10 min check these values and based on them decide if it should turn on the sprinkler system. In case the value for light intensity of analog pin 1 was bellow 200 and moisture sensor was less than 300 a digital pin 6 would be turned on to send a High output to the relay to turn it on and to start the sprinkler system.

HARDWARE FUNCTION

The following layout describes how the Sprinkler System will be automated in order to function through the WiShield that is attached to the Arduino. *Figure 2* explains the layout of the implementation.

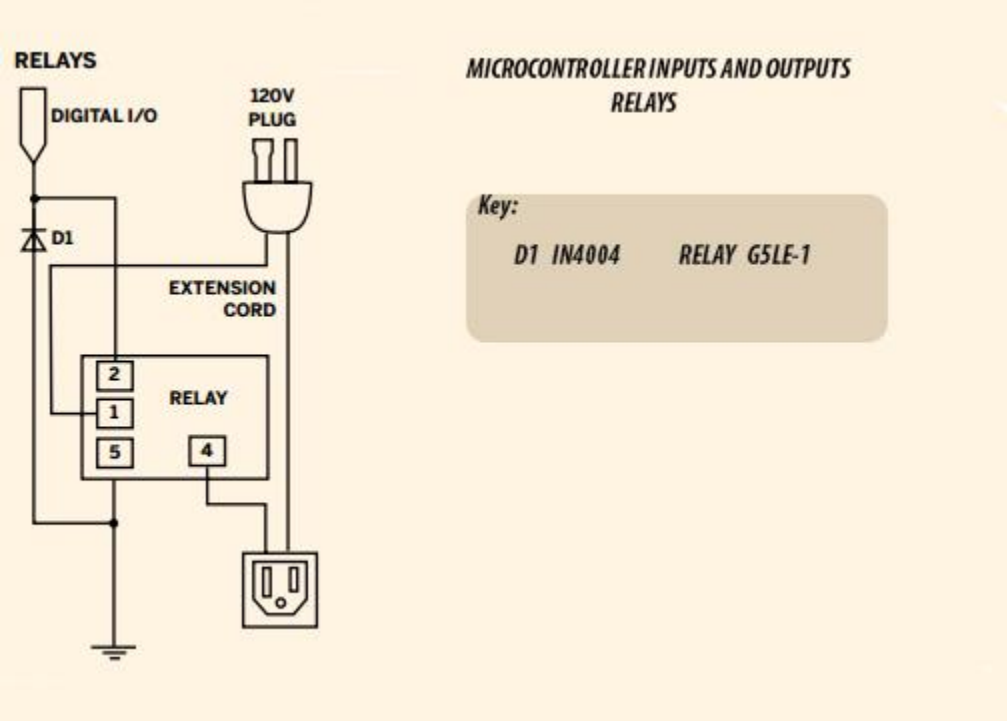


Figure 2.

Relays will be controlled with digital outputs sent from the Arduino once the webserver initiates a certain command, that command will be based upon two different factors. One of the factors is the exact based on soil moisture level in the ground and the second is based on the time of the day. If the soil moisture is under the acceptable level and its night turn on the sprinkler system. Else leave it off until one the right parameters are acquired. *Figure 3* illustrates the next two steps necessary in controlling the sprinkler system.

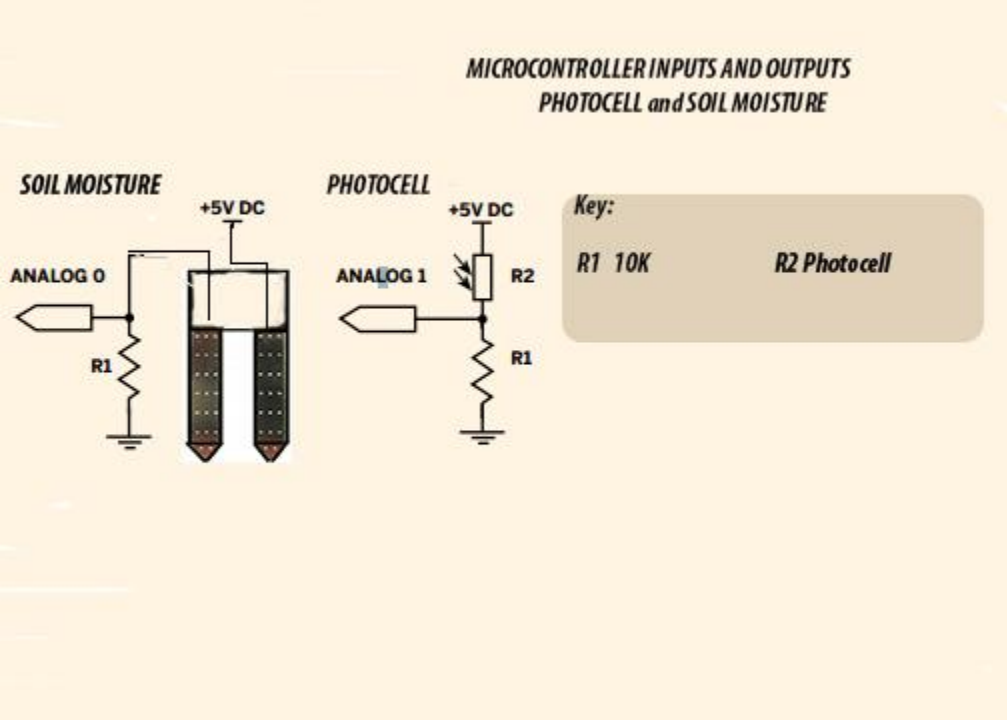


Figure 3.

OUTLET CONTROL

The outlet controller is in the form of a power strip. It is mobile and reconfigurable. This power strip is able to plug into any standard wall socket, and be wireless except for power. The sockets will be switched on and off via the server computer using the internet. Power strips will enable the home owner to control power to any device via the internet. The power strip will also monitor power for devices which are plugged in. This data will be available at the controller boxes and server computer.

SOFTWARE

The software is written in Arduino c++, it contains routines to monitor power, turn on and off switches, and communicate information through serial or wifi. The majority of processing goes into monitoring power from the current sensors. The ADCs provide values which follow the AC waveform of the power, then the software calculates the high and low peaks of the signal, from there it calculates the total current draw and transforms it into a power statistic. The switches are controlled via simple routines that provide power to the digital pins when it receives a command from serial or wifi to change their state.

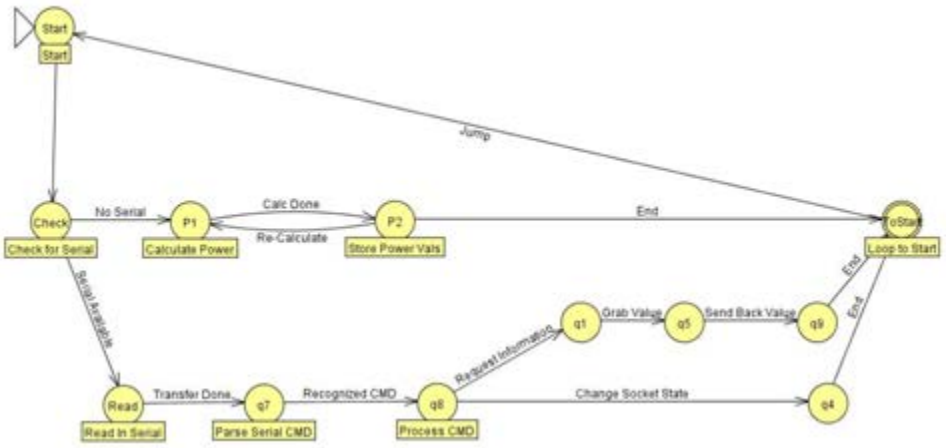


Figure 4

The communication is done through serial commands from the Wi-Fi module or the USB interface, the following is a list of user serial commands one could issue to the powerstrip, this does not include diagnostic commands. Commands are entered one line at a time, and processed one line at a time. The commands can either change a socket state, or request information about the powerstrip. When information is requested the value is sent back on the same serial connection it was requested from.

List of Commands you can enter

turn on <socketnum>	//turn on socket
turn off <socketnum>	//turn off socket
power <socketnum>	//returns power of socket
current <socketnum> high	//returns highpoint of current waveform
current <socketnum> low	//returns lowpoint of current waveform
socket <socketnum>	//returns on/off
allpower	//returns all powers on a new line each
active sockets	//returns the number of powered sockets

HARDWARE

The powerstrip box contains an Arduino ATMEGA 328 MCU, 12V power supply, WIFI Board, 4 controlled and monitored sockets, and a serial interface for peripheral expansion and

debugging. The power supply provides power to all onboard electronics. The powerstrip is protected with an over current circuit for safety. The power is heavily filtered across all circuits because of fluctuating power when an outlet is turned on or off. The four socket modules each contain a relay, a MCS714 linear IC current sensor, and an outlet, which are all controlled independently from each other. The current sensors interface with the on board ADCs on the Arduino board, providing a direct transformed AC current reading. A relay is used to switch power from the wall which is controlled via the Arduino. Each socket can source up to 5 amps with correct power readings. The following is a schematic of the basic powerstrip hardware, note that this does not include peripherals such as the WIFI board.

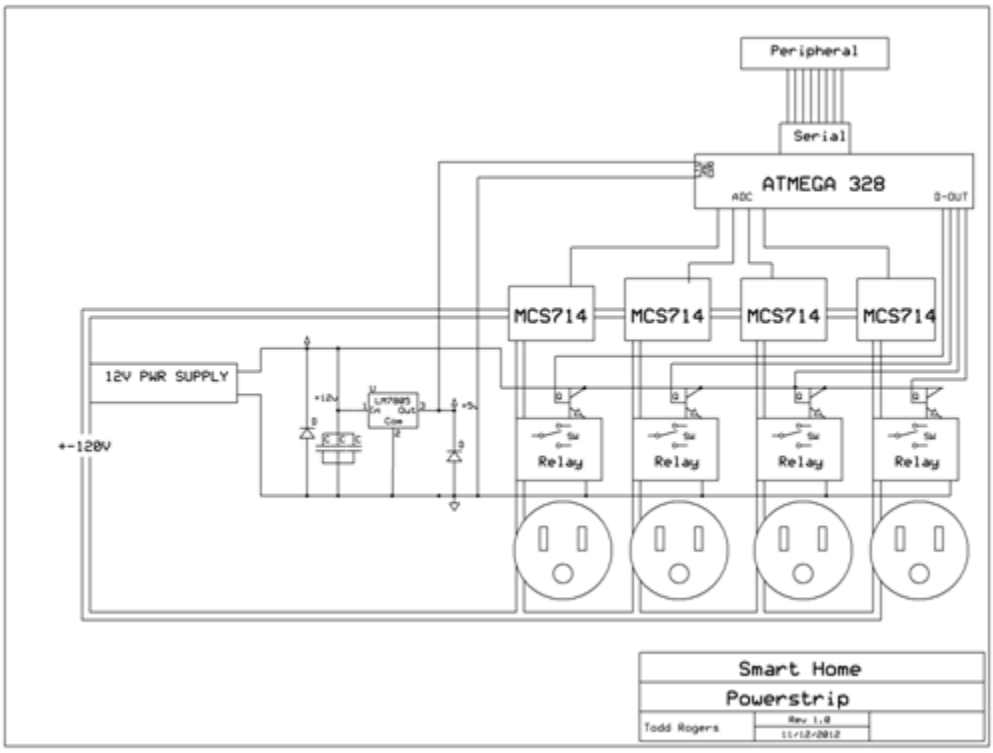


Figure 5

TEMPERATURE SENSOR

The temperature sensors are 10k ohm thermistors. The measurement of these temperature sensors is done by connecting 5V to the 10k thermistor in series with another regular 10k ohm resistor to ground. Between these two resistors, the voltage will change based on what temperature is, from the changing resistance value of the thermistor (figure x).

Reading a voltage can be done by connecting a wire from between the two resistors directly to a ADC input on a MCU. This voltage will range from 5V to 2.5V. The ADC will be a 10 bit value that will directly relate to the temperature of the thermistor.

The MCU is capable of supporting 8 analog inputs. This is a problem in representing a large home that would require dozens of temperature sensors to know a whether there is equal heat distribution throughout the house. To fix this issue we are using a 3 to 8 analog channel demux (CD4051). When using multiple demux chips and 3 control signals, we are capable of sampling 64 temperature samples, allowing a better understanding of the heat distribution in each room.

The 3 to 8 analog channel demux is controlled by a 3 digitial i/o pins. the 3 digitial i/o pins go to all 8 demux.

BLOWERS

SOFTWARE

The Air vent blowers are to assist air flow where it is needed most. Having a furnace at one end of a home, will lose air pressure as the forced air works toward the other end of the home. The typical home runs a ~1000 to 2000 CFM(cubic feet per minute). Depending on how many vents you have, and how far away from the furnace, the air flow will be really restricted. A house with a 1300 CFM blower and 14 vents will only be able to blow 100 CFM +/- 20 CFM due to air pressure. To help control the airflow to different parts of the house we will have 12V DC brushless fans. The fans are able to produce 150 CFM. When you place one of these fans in the vent it will assist the airflow throughout the house.

The fans are multipurpose, and can be used for fire blower, exercise fan, and other needs. The fans require 1.5 amps to drive, the current project only supports one fan operating at a time. There are plans to provide multiple fans operating at the same time in a future version. The fan speed is controlled by the duty cycle of a PWM signal. When the duty cycle is 100% the fan is operating at full capacity, and when its near 0% the fan stops completely. The teensy MCU will control these fans with a setting from 0 - 9. This will allow users to set the speed to their desire.

STEPPER MOTOR CONTROL

We are using a Pololu bi-polar stepper motor driver to drive the stepper motors. These simple drivers handle all of the driving and allow the user to just specify the step and direction. They are able to source up to 1.5 amps. This provides ample amount of torque to turn the stepper motor under pressure. The teensy will control the motor while waiting for a button to be pressed. The button will signal the teensy that the motor has completed its turning, and stop the stepper motor in place.

DAMPER CONTROL

In order to control multiple dampers at the same time we needed to control which path the stepper motor signals when. The teensy MCU uses 4 signals to control up to 16 dampers

SOFTWARE

HARDWARE

GARAGE CONTROL

The garage door control is a simple switch. If that switch is connected the Garage will toggle. It uses a unique voltage, and to be compatible with a variety of different garage doors, a relay is ideal to control it. The relay is controlled by a MCU GPIO pin.

The software to control the GPIO pin is simple

SOFWARE

HARDWARE

CONCLUSION

REFERENCES AND ACKNOWLEDGEMENTS

[x] E. Weddington, J Wunsch, P Fleury, T Henigan, C. O'Flynn, R Patommel, M Pfaff, S. Pool, F. Rouleau, and C. Lamas; Teensy WinAVR Makefile Base; www.pjrc.com/teensy; 2012

Appendix A - Wifi Arduino Code

```
/*
* A Web page used to control all of the information necessary for control of the whole wifi system
* The Wireless configuration parameters were borrowed from Arduino.cc in order to
* get the Wifi Board to comunicate with the router.
*/
/*
Libraries needed for the wifi implementation
*/

#include <WiServer.h>
#include <string.h>
#include <SoftwareSerial.h>

#define WIRELESS_MODE_INFRA   1
#define WIRELESS_MODE_ADHOC   2

//#define ledPin1 8 //changed this from 5 to 8
//#define ledPin2 6
//#define ledPin3 7
/*
Variable declaration for Software serial and also for the analog and
digital input values
*/
String str;
String txtLights;
boolean stringComplete;
int LaserValue = 0;
int MotionValue = 0;
int LaserPin = 6;
int TripPin = 7;

//Declaring software serial
SoftwareSerial mySerial(4, 5); //RX, TX

/*
This part of the code was borrowed from the library created by the Arduino.cc team
in order to get the WiShield v2.0 connected to the wifi router.
*/
// Wireless configuration parameters ----------------------------------------
unsigned char local_ip[] = {192,168,1,102};  // IP address of WiShield
unsigned char gateway_ip[] = {192,168,1,1};          // router or gateway IP address
unsigned char subnet_mask[] = {255,255,255,0};       // subnet mask for the local network
```

```
const prog_char ssid[] PROGMEM = {"SmartHome"};                    // max 32 bytes

unsigned char security_type = 0;     // 0 - open; 1 - WEP; 2 - WPA; 3 - WPA2

// WPA/WPA2 passphrase
const prog_char security_passphrase[] PROGMEM = {"DeadBeef"};              // max 64 characters

// WEP 128-bit keys
// sample HEX keys
prog_uchar wep_keys[] PROGMEM = {      0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b,
0x0c, 0x0d,          // Key 0
                                                                    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00,        0x00,   // Key 1
                                                                    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00,        0x00,   // Key 2
                                                                    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00,        0x00      // Key 3
                                                                 };

// setup the wireless mode
// infrastructure - connect to AP
// adhoc - connect to another WiFi device
unsigned char wireless_mode = WIRELESS_MODE_INFRA;

unsigned char ssid_len;
unsigned char security_passphrase_len;

// End of wireless configuration parameters ----------------------------------------

boolean states[4]; //Led states
char stateCounter, tmpStrCat[64], stateBuff[4], numAsCharBuff[2], ledChange; //Variable declarations

/*
Check the conversion to display on the page so we can see if the url is on or off
*/
void boolToString (boolean tests, char returnBuff[4]){
  returnBuff[0] = '\0';
  if (tests){
    strcat(returnBuff, "on");}
  else{
    strcat(returnBuff, "off");}
}
/*
State print counter
*/
void printStates(){
          for (stateCounter = 0 ; stateCounter < 4; stateCounter++){
```

```
          boolToString(states[stateCounter], stateBuff);
          int count = stateCounter;

/*
Software serial implementation in order to controll the second arduino to
perform its desired controlls
*/
          //mySerial.begin(9600);
          mySerial.print("turn ");
          mySerial.print(stateBuff);
          mySerial.print(" ");
          mySerial.print(count);
          mySerial.print('\n');
          mySerial.print("allpower\n");
         //  if(analogRead(1)<810){
          //mySerial.print("turn ");
          //mySerial.print(stateBuff); 3\n");
          //Serial.print("turn on 3\n");
        //}else{
          //mySerial.print("turn off 3\n");
          //Serial.print("turn off 3\n");
             //}
/*
This code can be ignored as it is only there for troubleshooting purposes
*/
          Serial.print("turn ");
          Serial.print(stateBuff);
          Serial.print(" ");
          Serial.println(count);
             }
             }
/*
This part of the code can also be ignored as its purpose was to see
if the LED would function based on the desired parameters.
*/
//void writeStates()
//{
             //set led states
//        digitalWrite(ledPin1, states[0]);
//        digitalWrite(ledPin2, states[1]);
//        digitalWrite(ledPin3, states[2]);
//}
/*void readSerial()
{
  while(mySerial.available()){
             char inChar = (char)mySerial.read();
             str += (String)inChar;
   if(inChar == '\n'){
```

```
    stringComplete = true;
            break;
   }

  }
  if (stringComplete==true){
  Serial.print(str);
  }
  str = "";
}*/
// Here the page gets served
/*
This part of the code serves the webpage, its a boolean funciton
that is responsible to run the code and then be served on the server*/
boolean sendPage(char* URL) {

   printStates();
  //  writeStates();



  //Using to check if the URL needs to change acting like a refresh button
  //if the url has not changed the page will not change either
  if (URL[1] == '?' && URL[2] == 'L' && URL[3] == 'E' && URL[4] == 'D') //url has a leading /
  {
            ledChange = (int)(URL[5] - 48); //get the led to change.

            for (stateCounter = 0 ; stateCounter < 4; stateCounter++)
            {
            if (ledChange == stateCounter)
            {
       states[stateCounter] = !states[stateCounter];
         //  Serial.print("Have changed ");
           Serial.println(ledChange);
            }
            }

            //after having change state, return the user to the index page.
    WiServer.print("<HTML><HEAD><meta http-equiv='REFRESH' content='0;url=/'></HEAD></HTML>");
            //Send over software serial commands in order to display the current state of the current sensor and also the
current state of the power strip
    mySerial.print("allpower\n");
            return true;
  }

  if (strcmp(URL, "/") == false) //why is this not true?
   {
      while(mySerial.available()){
```

```
        char inChar = (char)mySerial.read();
        str += (String)inChar;
     if((str.length() > 8) || (inChar == '\n')){
    stringComplete = true;
        break;
        }

}
        if (stringComplete==true){
   Serial.print(str);
}

   WiServer.print("<html>");

        // WiServer.print("<body><center>Please select the led state:<center>\n<center>");
        for (stateCounter = 0; stateCounter < 4; stateCounter++) //for each led
        {
     numAsCharBuff[0] = (char)(stateCounter + 49); //as this is displayed use 1 - 3 rather than 0 - 2
     numAsCharBuff[1] = '\0'; //strcat expects a string (array of chars) rather than a single character.
                              //This string is a character plus string terminator.

     tmpStrCat[0] = '\0'; //initialise string
     strcat(tmpStrCat, "<a href=?LED"); //start the string
     tmpStrCat[12] = (char)(stateCounter + 48); //add the led number
     tmpStrCat[13] = '\0'; //terminate the string properly for later.

     strcat(tmpStrCat, ">Led ");
     strcat(tmpStrCat, numAsCharBuff);
     strcat(tmpStrCat, ": ");

     boolToString(states[stateCounter], stateBuff);
     strcat(tmpStrCat, stateBuff);
     strcat(tmpStrCat, "</a> "); //we now have something in the range of <a href=?LED0>Led 0: Off</a>

     WiServer.print(tmpStrCat);
        }

        //Declare the variables to find the temperature
 float temp_in_celsius = 0;
 float temp_in_kelvin=0;
 float temp_in_fahrenheit=0;

//Reads the input and converts it to Kelvin degrees
 temp_in_kelvin = analogRead(0) * 0.004882812 * 100;
 //Converts Kelvin to Celsius minus 2.5 degrees error
 temp_in_celsius = temp_in_kelvin - 2.5 - 273.15;
 temp_in_fahrenheit = ((temp_in_kelvin - 2.5) * 9 / 5) - 459.67;
```

```
//Temperature
        float h = temp_in_kelvin;
        float t = temp_in_celsius;
          float f = temp_in_fahrenheit;
    //float laser = analogRead(1);
        String txt;

    //Check to see if the Laser is tripped or not
    if((analogRead(1)<810) && (analogRead(2)<300)){
        txt = "Sprinkler System is on";
    }else{
        txt = "Sprinkler System is off";
        }
    //Display if the lights are on or off
    if((digitalRead(LaserPin) == HIGH) && (digitalRead(TripPin) == HIGH)){
     txtLights = "Lights are on";
        }else{
        txtLights = "Lights are off";
        }

    LaserValue = digitalRead(LaserPin);
    MotionValue = digitalRead(TripPin);

        // check if returns are valid, if they are NaN (not a number) then something went wrong!
        if (isnan(t) || isnan(h)) {
                WiServer.print("<html>");  //Here is the code for the html page
          WiServer.print("Failed to read from Sensor Input");
                WiServer.println("           </html>");
        } else {
                WiServer.print("<html>");
                //WiServer.print("<center><H2>Room 1 Temperature Statistics: ID: 1</H2><br><br><br>");

    WiServer.print("Temperature Kelvin: ");
    WiServer.print(h);
    WiServer.print(" T\t");
    WiServer.print("Temperature Celsius ");
         WiServer.print(t);
    WiServer.println(" *C");
    WiServer.print("Temperature Fehrenhite ");
    WiServer.print(f);
    WiServer.println(" *F");
    WiServer.println("       ");
    WiServer.print("</center>");
        WiServer.print("<center>");
    WiServer.print("Sprinkler Detection: ");
    WiServer.print(txt);
    WiServer.println("       ");
        // WiServer.print(laser);
```

```
        WiServer.println("Testing Current: ");
        WiServer.println(str);
            str = "";
        WiServer.println("</center>");
        WiServer.print("<center>");
        WiServer.print("Light Status: ");
        WiServer.print(txtLights);
        WiServer.print("</center>");
        WiServer.print("<center>");
            WiServer.print("Trip Wire Status: ");
        WiServer.print(LaserValue);
        WiServer.print("</center>");
        WiServer.print("<center>");
         WiServer.print("Motion Sensor Status: ");
        WiServer.print(MotionValue);
        WiServer.print("</center>");
        WiServer.print("</html></center>");     // URL was recognized
            return true;
            }
            // URL not found
            return false;

        WiServer.print("</html> ");
            return true;
            //str = "";
   }
}

void setup() {
  // Initialize WiServer and have it use the sendMyPage function to serve pages
 // pinMode(ledPin1, OUTPUT);
  //pinMode(ledPin2, OUTPUT);
  //pinMode(ledPin3, OUTPUT);
  pinMode(3, OUTPUT);

  // Enable Serial output and ask WiServer to generate log messages (optional)
 // Serial.begin(57600);

  WiServer.enableVerboseMode(true);
  Serial.begin(9600);
            while (!Serial) {
            ; // wait for serial port to connect. Needed for Leonardo only
  }
  mySerial.begin(9600);
  WiServer.init(sendPage);
  states[0] = false;
  states[1] = false;
  states[2] = false;
```

```
}

void loop(){

 // if (mySerial.available())
   //   mySerial.print("turn on 1\n");
              //  if (mySerial.available())
              // Serial.write(mySerial.read());
      //mySerial.print("turn on 1\n");

  if((analogRead(1)<810)&&(analogRead(2)<200)){
         digitalWrite(3, HIGH);
         //mySerial.print("turn on 3");
              // mySerial.print('\n');
              }else{
         digitalWrite(3, LOW);
              // mySerial.print("turn on 3");
              // mySerial.print('\n');
              }
  // Run WiServer
  WiServer.server_task();

  delay(10);
}
```

Appendix B - Server Software

Appendix C - Teensy Controller Software

C.0 - Makefile Software Code

```
# Hey Emacs, this is a -*- makefile -*-
#----------------------------------------------------------------------------
# WinAVR Makefile Template written by Eric B. Weddington, Jörg Wunsch, et al.
#
# Released to the Public Domain
#
# Additional material for this makefile was written by:
# Peter Fleury
# Tim Henigan
# Colin O'Flynn
# Reiner Patommel
# Markus Pfaff
# Sander Pool
# Frederik Rouleau
# Carlos Lamas
#
#----------------------------------------------------------------------------
# On command line:
```

```
#
# make all = Make software.
#
# make clean = Clean out built project files.
#
# make coff = Convert ELF to AVR COFF.
#
# make extcoff = Convert ELF to AVR Extended COFF.
#
# make program = Download the hex file to the device, using avrdude.
#                Please customize the avrdude settings below first!
#
# make debug = Start either simulavr or avarice as specified for debugging,
#              with avr-gdb or avr-insight as the front end for debugging.
#
# make filename.s = Just compile filename.c into the assembler code only.
#
# make filename.i = Create a preprocessed source file for use in submitting
#                   bug reports to the GCC project.
#
# To rebuild project do "make clean" then "make all".
#----------------------------------------------------------------------------


#-------------------
# NOTES about this particular Build
# this build is good for SmartHome Project. only with the Teensy2.0++
#-------------------

# Target file name (without extension).
TARGET = mainControl



# List C source files here. (C dependencies are automatically generated.)
SRC =     $(TARGET).c src/usb_serial.c src/HVACGarage.c src/damper_Control.c src/tempSense.c src/Sprinkler.c
src/PWMTeensyTwoPlusPlus.c src/fans.c



# the microcontroller for this particular project
MCU = at90usb1286      # Teensy++ 2.0



# Processor frequency.
#   Normally the first thing your program should do is set the clock prescaler,
#   so your program will run at the correct speed.  You should also set this
#   variable to same clock speed.  The _delay_ms() macro uses this, and many
#   examples use this variable to calculate timings.  Do not add a "UL" here.
F_CPU = 16000000



# Output format. (can be srec, ihex, binary)
FORMAT = ihex
```

```
# Object files directory
#     To put object files in current directory, use a dot (.), do NOT make
#     this an empty or blank macro!
OBJDIR = objFiles

# List Assembler source files here.
#     Make them always end in a capital .S.  Files ending in a lowercase .s
#     will not be considered source files but generated files (assembler
#     output from the compiler), and will be deleted upon "make clean"!
#     Even though the DOS/Win* filesystem matches both .s and .S the same,
#     it will preserve the spelling of the filenames, and gcc itself does
#     care about how the name is spelled on its command-line.
ASRC =


# Optimization level, can be [0, 1, 2, 3, s].
#     0 = turn off optimization. s = optimize for size.
#     (Note: 3 is not always the best optimization level. See avr-libc FAQ.)
OPT = s


# Debugging format.
#     Native formats for AVR-GCC's -g are dwarf-2 [default] or stabs.
#     AVR Studio 4.10 requires dwarf-2.
#     AVR [Extended] COFF format requires stabs, plus an avr-objcopy run.
DEBUG = dwarf-2


# List any extra directories to look for include files here.
#     Each directory must be seperated by a space.
#     Use forward slashes for directory separators.
#     For a directory that has spaces, enclose it in quotes.
EXTRAINCDIRS = \include


# Compiler flag to set the C Standard level.
#     c89   = "ANSI" C
#     gnu89 = c89 plus GCC extensions
#     c99   = ISO C99 standard (not yet fully implemented)
#     gnu99 = c99 plus GCC extensions
CSTANDARD = -std=gnu99


# Place -D or -U options here for C sources
CDEFS = -DF_CPU=$(F_CPU)UL
```

```
# Place -D or -U options here for ASM sources
ADEFS = -DF_CPU=$(F_CPU)


# Place -D or -U options here for C++ sources
CPPDEFS = -DF_CPU=$(F_CPU)UL
#CPPDEFS += -D__STDC_LIMIT_MACROS
#CPPDEFS += -D__STDC_CONSTANT_MACROS



#---------------- Compiler Options C ----------------
#  -g*:         generate debugging information
#  -O*:         optimization level
#  -f...:       tuning, see GCC manual and avr-libc documentation
#  -Wall...:    warning level
#  -Wa,...:     tell GCC to pass this to the assembler.
#    -adhlns...: create assembler listing
CFLAGS = -g$(DEBUG)
CFLAGS += $(CDEFS)
CFLAGS += -O$(OPT)
CFLAGS += -funsigned-char
CFLAGS += -funsigned-bitfields
CFLAGS += -ffunction-sections
CFLAGS += -fpack-struct
CFLAGS += -fshort-enums
CFLAGS += -Wall
CFLAGS += -Wstrict-prototypes
#CFLAGS += -mshort-calls
#CFLAGS += -fno-unit-at-a-time
#CFLAGS += -Wundef
#CFLAGS += -Wunreachable-code
#CFLAGS += -Wsign-compare
CFLAGS += -Wa,-adhlns=$(<:%.c=$(OBJDIR)/%.lst)
CFLAGS += $(patsubst %,-I%,$(EXTRAINCDIRS))
CFLAGS += $(CSTANDARD)



#---------------- Compiler Options C++ ----------------
#  -g*:         generate debugging information
#  -O*:         optimization level
#  -f...:       tuning, see GCC manual and avr-libc documentation
#  -Wall...:    warning level
#  -Wa,...:     tell GCC to pass this to the assembler.
#    -adhlns...: create assembler listing
CPPFLAGS = -g$(DEBUG)
CPPFLAGS += $(CPPDEFS)
CPPFLAGS += -O$(OPT)
CPPFLAGS += -funsigned-char
CPPFLAGS += -funsigned-bitfields
CPPFLAGS += -fpack-struct
CPPFLAGS += -fshort-enums
```

```
CPPFLAGS += -fno-exceptions
CPPFLAGS += -Wall
CPPFLAGS += -Wundef
#CPPFLAGS += -mshort-calls
#CPPFLAGS += -fno-unit-at-a-time
#CPPFLAGS += -Wstrict-prototypes
#CPPFLAGS += -Wunreachable-code
#CPPFLAGS += -Wsign-compare
CPPFLAGS += -Wa,-adhlns=$(<:%.cpp=$(OBJDIR)/%.lst)
CPPFLAGS += $(patsubst %,-I%,$(EXTRAINCDIRS))
#CPPFLAGS += $(CSTANDARD)



#---------------- Assembler Options ----------------
#  -Wa,...:   tell GCC to pass this to the assembler.
#  -adhlns:   create listing
#  -gstabs:   have the assembler create line number information; note that
#             for use in COFF files, additional information about filenames
#             and function names needs to be present in the assembler source
#             files -- see avr-libc docs [FIXME: not yet described there]
#  -listing-cont-lines: Sets the maximum number of continuation lines of hex
#       dump that will be displayed for a given single line of source input.
ASFLAGS = $(ADEFS) -Wa,-adhlns=$(<:%.S=$(OBJDIR)/%.lst),-gstabs,--listing-cont-lines=100



#---------------- Library Options ----------------
# Minimalistic printf version
PRINTF_LIB_MIN = -Wl,-u,vfprintf -lprintf_min

# Floating point printf version (requires MATH_LIB = -lm below)
PRINTF_LIB_FLOAT = -Wl,-u,vfprintf -lprintf_flt

# If this is left blank, then it will use the Standard printf version.
PRINTF_LIB =
#PRINTF_LIB = $(PRINTF_LIB_MIN)
#PRINTF_LIB = $(PRINTF_LIB_FLOAT)



# Minimalistic scanf version
SCANF_LIB_MIN = -Wl,-u,vfscanf -lscanf_min

# Floating point + %[ scanf version (requires MATH_LIB = -lm below)
SCANF_LIB_FLOAT = -Wl,-u,vfscanf -lscanf_flt

# If this is left blank, then it will use the Standard scanf version.
SCANF_LIB =
#SCANF_LIB = $(SCANF_LIB_MIN)
#SCANF_LIB = $(SCANF_LIB_FLOAT)
```

```
MATH_LIB = -lm


# List any extra directories to look for libraries here.
#     Each directory must be seperated by a space.
#     Use forward slashes for directory separators.
#     For a directory that has spaces, enclose it in quotes.
EXTRALIBDIRS =



#---------------- External Memory Options ----------------

# 64 KB of external RAM, starting after internal RAM (ATmega128!),
# used for variables (.data/.bss) and heap (malloc()).
#EXTMEMOPTS = -Wl,-Tdata=0x801100,--defsym=__heap_end=0x80ffff

# 64 KB of external RAM, starting after internal RAM (ATmega128!),
# only used for heap (malloc()).
#EXTMEMOPTS = -Wl,--section-start,.data=0x801100,--defsym=__heap_end=0x80ffff

EXTMEMOPTS =



#---------------- Linker Options ----------------
#  -Wl,...:     tell GCC to pass this to linker.
#    -Map:      create map file
#    --cref:    add cross reference to  map file
LDFLAGS = -Wl,-Map=$(TARGET).map,--cref
LDFLAGS += -Wl,--relax
LDFLAGS += -Wl,--gc-sections
LDFLAGS += $(EXTMEMOPTS)
LDFLAGS += $(patsubst %,-L%,$(EXTRALIBDIRS))
LDFLAGS += $(PRINTF_LIB) $(SCANF_LIB) $(MATH_LIB)
#LDFLAGS += -T linker_script.x



#---------------- Programming Options (avrdude) ----------------

# Programming hardware
# Type: avrdude -c ?
# to get a full listing.
#
AVRDUDE_PROGRAMMER = stk500v2

# com1 = serial port. Use lpt1 to connect to parallel port.
AVRDUDE_PORT = com1    # programmer connected to serial device
```

```
AVRDUDE_WRITE_FLASH = -U flash:w:$(TARGET).hex
#AVRDUDE_WRITE_EEPROM = -U eeprom:w:$(TARGET).eep



# Uncomment the following if you want avrdude's erase cycle counter.
# Note that this counter needs to be initialized first using -Yn,
# see avrdude manual.
#AVRDUDE_ERASE_COUNTER = -y

# Uncomment the following if you do /not/ wish a verification to be
# performed after programming the device.
#AVRDUDE_NO_VERIFY = -V

# Increase verbosity level.  Please use this when submitting bug
# reports about avrdude. See <http://savannah.nongnu.org/projects/avrdude>
# to submit bug reports.
AVRDUDE_VERBOSE = -v -v

AVRDUDE_FLAGS = -p $(MCU) -P $(AVRDUDE_PORT) -c $(AVRDUDE_PROGRAMMER)
AVRDUDE_FLAGS += $(AVRDUDE_NO_VERIFY)
AVRDUDE_FLAGS += $(AVRDUDE_VERBOSE)
AVRDUDE_FLAGS += $(AVRDUDE_ERASE_COUNTER)



#---------------- Debugging Options ----------------

# For simulavr only - target MCU frequency.
DEBUG_MFREQ = $(F_CPU)

# Set the DEBUG_UI to either gdb or insight.
# DEBUG_UI = gdb
DEBUG_UI = insight

# Set the debugging back-end to either avarice, simulavr.
DEBUG_BACKEND = avarice
#DEBUG_BACKEND = simulavr

# GDB Init Filename.
GDBINIT_FILE = __avr_gdbinit

# When using avarice settings for the JTAG
JTAG_DEV = /dev/com1

# Debugging port used to communicate between GDB / avarice / simulavr.
DEBUG_PORT = 4242

# Debugging host used to communicate between GDB / avarice / simulavr, normally
```

```
#     just set to localhost unless doing some sort of crazy debugging when
#     avarice is running on a different computer.
DEBUG_HOST = localhost



#============================================================================


# Define programs and commands.
SHELL = sh
CC = avr-gcc
OBJCOPY = avr-objcopy
OBJDUMP = avr-objdump
SIZE = avr-size
AR = avr-ar rcs
NM = avr-nm
AVRDUDE = avrdude
REMOVE = rm -f
REMOVEDIR = rm -rf
COPY = cp
WINSHELL = cmd


# Define Messages
# English
MSG_ERRORS_NONE = Errors: none
MSG_BEGIN = -------- begin --------
MSG_END = --------  end  --------
MSG_SIZE_BEFORE = Size before:
MSG_SIZE_AFTER = Size after:
MSG_COFF = Converting to AVR COFF:
MSG_EXTENDED_COFF = Converting to AVR Extended COFF:
MSG_FLASH = Creating load file for Flash:
MSG_EEPROM = Creating load file for EEPROM:
MSG_EXTENDED_LISTING = Creating Extended Listing:
MSG_SYMBOL_TABLE = Creating Symbol Table:
MSG_LINKING = Linking:
MSG_COMPILING = Compiling C:
MSG_COMPILING_CPP = Compiling C++:
MSG_ASSEMBLING = Assembling:
MSG_CLEANING = Cleaning project:
MSG_CREATING_LIBRARY = Creating library:



# Define all object files.
OBJ = $(SRC:%.c=$(OBJDIR)/%.o) $(CPPSRC:%.cpp=$(OBJDIR)/%.o) $(ASRC:%.S=$(OBJDIR)/%.o)
```

```
# Define all listing files.
LST = $(SRC:%.c=$(OBJDIR)/%.lst) $(CPPSRC:%.cpp=$(OBJDIR)/%.lst) $(ASRC:%.S=$(OBJDIR)/%.lst)


# Compiler flags to generate dependency files.
GENDEPFLAGS = -MMD -MP -MF .dep/$(@F).d


# Combine all necessary flags and optional flags.
# Add target processor to flags.
ALL_CFLAGS = -mmcu=$(MCU) -I. $(CFLAGS) $(GENDEPFLAGS)
ALL_CPPFLAGS = -mmcu=$(MCU) -I. -x c++ $(CPPFLAGS) $(GENDEPFLAGS)
ALL_ASFLAGS = -mmcu=$(MCU) -I. -x assembler-with-cpp $(ASFLAGS)




# Default target.
all: begin gccversion sizebefore build sizeafter end

# Change the build target to build a HEX file or a library.
build: elf hex eep lss sym
#build: lib


elf: $(TARGET).elf
hex: $(TARGET).hex
eep: $(TARGET).eep
lss: $(TARGET).lss
sym: $(TARGET).sym
LIBNAME=lib$(TARGET).a
lib: $(LIBNAME)




# Eye candy.
# AVR Studio 3.x does not check make's exit code but relies on
# the following magic strings to be generated by the compile job.
begin:
	@echo
	@echo $(MSG_BEGIN)

end:
	@echo $(MSG_END)
	@echo


# Display size of file.
```

```
HEXSIZE = $(SIZE) --target=$(FORMAT) $(TARGET).hex
#ELFSIZE = $(SIZE) --mcu=$(MCU) --format=avr $(TARGET).elf
ELFSIZE = $(SIZE) $(TARGET).elf

sizebefore:
	@if test -f $(TARGET).elf; then echo; echo $(MSG_SIZE_BEFORE); $(ELFSIZE); \
	2>/dev/null; echo; fi

sizeafter:
	@if test -f $(TARGET).elf; then echo; echo $(MSG_SIZE_AFTER); $(ELFSIZE); \
	2>/dev/null; echo; fi



# Display compiler version information.
gccversion :
	@$(CC) --version



# Program the device.
program: $(TARGET).hex $(TARGET).eep
	$(AVRDUDE) $(AVRDUDE_FLAGS) $(AVRDUDE_WRITE_FLASH) $(AVRDUDE_WRITE_EEPROM)



# Generate avr-gdb config/init file which does the following:
#     define the reset signal, load the target file, connect to target, and set
#     a breakpoint at main().
gdb-config:
	@$(REMOVE) $(GDBINIT_FILE)
	@echo define reset >> $(GDBINIT_FILE)
	@echo SIGNAL SIGHUP >> $(GDBINIT_FILE)
	@echo end >> $(GDBINIT_FILE)
	@echo file $(TARGET).elf >> $(GDBINIT_FILE)
	@echo target remote $(DEBUG_HOST):$(DEBUG_PORT)  >> $(GDBINIT_FILE)
ifeq ($(DEBUG_BACKEND),simulavr)
	@echo load  >> $(GDBINIT_FILE)
endif
	@echo break main >> $(GDBINIT_FILE)

debug: gdb-config $(TARGET).elf
ifeq ($(DEBUG_BACKEND), avarice)
	@echo Starting AVaRICE - Press enter when "waiting to connect" message displays.
	@$(WINSHELL) /c start avarice --jtag $(JTAG_DEV) --erase --program --file \
	$(TARGET).elf $(DEBUG_HOST):$(DEBUG_PORT)
	@$(WINSHELL) /c pause

else
	@$(WINSHELL) /c start simulavr --gdbserver --device $(MCU) --clock-freq \
	$(DEBUG_MFREQ) --port $(DEBUG_PORT)
```

```
endif
	@$(WINSHELL) /c start avr-$(DEBUG_UI) --command=$(GDBINIT_FILE)



# Convert ELF to COFF for use in debugging / simulating in AVR Studio or VMLAB.
COFFCONVERT = $(OBJCOPY) --debugging
COFFCONVERT += --change-section-address .data-0x800000
COFFCONVERT += --change-section-address .bss-0x800000
COFFCONVERT += --change-section-address .noinit-0x800000
COFFCONVERT += --change-section-address .eeprom-0x810000



coff: $(TARGET).elf
	@echo
	@echo $(MSG_COFF) $(TARGET).cof
	$(COFFCONVERT) -O coff-avr $< $(TARGET).cof


extcoff: $(TARGET).elf
	@echo
	@echo $(MSG_EXTENDED_COFF) $(TARGET).cof
	$(COFFCONVERT) -O coff-ext-avr $< $(TARGET).cof



# Create final output files (.hex, .eep) from ELF output file.
%.hex: %.elf
	@echo
	@echo $(MSG_FLASH) $@
	$(OBJCOPY) -O $(FORMAT) -R .eeprom -R .fuse -R .lock -R .signature $< $@

%.eep: %.elf
	@echo
	@echo $(MSG_EEPROM) $@
	-$(OBJCOPY) -j .eeprom --set-section-flags=.eeprom="alloc,load" \
	--change-section-lma .eeprom=0 --no-change-warnings -O $(FORMAT) $< $@ || exit 0

# Create extended listing file from ELF output file.
%.lss: %.elf
	@echo
	@echo $(MSG_EXTENDED_LISTING) $@
	$(OBJDUMP) -h -S -z $< > $@

# Create a symbol table from ELF output file.
%.sym: %.elf
	@echo
```

```
	@echo $(MSG_SYMBOL_TABLE) $@
	$(NM) -n $< > $@


# Create library from object files.
.SECONDARY : $(TARGET).a
.PRECIOUS : $(OBJ)
%.a: $(OBJ)
	@echo
	@echo $(MSG_CREATING_LIBRARY) $@
	$(AR) $@ $(OBJ)


# Link: create ELF output file from object files.
.SECONDARY : $(TARGET).elf
.PRECIOUS : $(OBJ)
%.elf: $(OBJ)
	@echo
	@echo $(MSG_LINKING) $@
	$(CC) $(ALL_CFLAGS) $^ --output $@ $(LDFLAGS)


# Compile: create object files from C source files.
$(OBJDIR)/%.o : %.c
	@echo
	@echo $(MSG_COMPILING) $<
	$(CC) -c $(ALL_CFLAGS) $< -o $@


# Compile: create object files from C++ source files.
$(OBJDIR)/%.o : %.cpp
	@echo
	@echo $(MSG_COMPILING_CPP) $<
	$(CC) -c $(ALL_CPPFLAGS) $< -o $@


# Compile: create assembler files from C source files.
%.s : %.c
	$(CC) -S $(ALL_CFLAGS) $< -o $@


# Compile: create assembler files from C++ source files.
%.s : %.cpp
	$(CC) -S $(ALL_CPPFLAGS) $< -o $@


# Assemble: create object files from assembler source files.
$(OBJDIR)/%.o : %.S
```

```
	@echo
	@echo $(MSG_ASSEMBLING) $<
	$(CC) -c $(ALL_ASFLAGS) $< -o $@


# Create preprocessed source for use in sending a bug report.
%.i : %.c
	$(CC) -E -mmcu=$(MCU) -I. $(CFLAGS) $< -o $@


# Target: clean project.
clean: begin clean_list end

clean_list :
	@echo
	@echo $(MSG_CLEANING)
	$(REMOVE) $(TARGET).hex
	$(REMOVE) $(TARGET).eep
	$(REMOVE) $(TARGET).cof
	$(REMOVE) $(TARGET).elf
	$(REMOVE) $(TARGET).map
	$(REMOVE) $(TARGET).sym
	$(REMOVE) $(TARGET).lss
	$(REMOVE) $(SRC:%.c=$(OBJDIR)/%.o)
	$(REMOVE) $(SRC:%.c=$(OBJDIR)/%.lst)
	$(REMOVE) $(SRC:.c=.s)
	$(REMOVE) $(SRC:.c=.d)
	$(REMOVE) $(SRC:.c=.i)
	$(REMOVEDIR) .dep


# Create object files directory
$(shell mkdir $(OBJDIR) 2>/dev/null)


# Include the dependency files.
-include $(shell mkdir .dep 2>/dev/null) $(wildcard .dep/*)


# Listing of phony targets.
.PHONY : all begin finish end sizebefore sizeafter gccversion \
build elf hex eep lss sym coff extcoff \
clean clean_list program debug gdb-config
```

C.1 - Global Variables Code(GlobalVar.h)

```
#ifndef globalVar_h__
#define globalVar_h__

#include <avr/pgmspace.h>
```

```
/**
* this file needs to be able to complete the following tasks.
*
* 1. Open damper (specified by number 0 - 15 ( byte)
* 2. Close damper (specified by number 0 - 15 (byte)
* 3. Check damper status (false for open, and true for closed) (boolean)
*/
//define which pins are what.
//Following pins are used for Sprinkler, Damper, and Temperature.

//Pin 26(PB2)
#define DEMUX_D_CONFIG                  (DDRB |= (1<<2))
#define DEMUX_D_ON                      (PORTB |= (1<<2))
#define DEMUX_D_OFF                     (PORTB &= ~(1<<2))
//Pin 27(PB1)
#define DEMUX_C_CONFIG                  (DDRB |= (1<<1))
#define DEMUX_C_ON                      (PORTB |= (1<<1))
#define DEMUX_C_OFF                     (PORTB &= ~(1<<1))
//Pin 28(PB0)
#define DEMUX_B_CONFIG                  (DDRB |= (1<<0))
#define DEMUX_B_ON                      (PORTB |= (1<<0))
#define DEMUX_B_OFF                     (PORTB &= ~(1<<0))
//Pin 29(PE7)
#define DEMUX_A_CONFIG                  (DDRE |= (1<<7))
#define DEMUX_A_ON                      (PORTE |= (1<<7))
#define DEMUX_A_OFF                     (PORTE &= ~(1<<7))

//Damper
//Pin 30(PE6)
#define DAMPER_ENABLE_CONFIG                    (DDRE |= (1<<6))
#define DAMPER_ENABLE_OFF                       (PORTE |= (1<<6))
#define DAMPER_ENABLE_ON                        (PORTE &= ~(1<<6))
//Sprinkler this is a little different.  it is enabled high.
//Pin 25 (PB3)
#define SPRINKLER_DEMUX_EN_CONFIG       (DDRB |= (1<<3))
#define SPRINKLER_DEMUX_EN_ON                   (PORTB |= (1<<3))//SPRINKLER_DEMUX_EN_ON
#define SPRINKLER_DEMUX_EN_OFF                  (PORTB &= ~(1<<3))
// Temperature
//Pin 20(PC7)
//#define TEMPERATURE_DEMUX_EN_CONFIG           (DDRC |= (1<<7))
//#define TEMPERATURE_DEMUX_EN_OFF      (PORTC |= (1<<7))// when high it disables everything else
//#define TEMPERATURE_DEMUX_EN_ON               (PORTC &= ~(1<<7))

//Configure Fan Pins PC6, PC5, PC4, PD0, PD3
//#define XTAL 160000000L // 16Mhz

/**
* Table of values for temp sense
*       adc             A       B       C               input
*       0               0       0       0               10
*       0               0       0       0               11
```

```
*       0               0       0       0               12
*       0               0       0       0               13
*       0               0       0       0               14
*       0               0       0       0               15
*       0               0       0       0               16
*       0               0       0       0               17
*       0               0       0       0               18
*/
#endif
```

C.2 - MainController Code(mainController.c)

```
/*
 * This code was made by Levi Balling
 * portions of code were provided from:
 * LED Blink Example with USB Debug Channel for Teensy USB Development Board
 * http://www.pjrc.com/teensy/
 * Copyright (c) 2008, 2010 PJRC.COM, LLC
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */


#include <avr/io.h>
#include <avr/pgmspace.h>
#include <util/delay.h>
#include <string.h>
#include <inttypes.h>
#include <stdio.h>
#include "include/GlobalVar.h"
#include "include/usb_serial.h"//added to test the built with it.
#include "include/HVACGarage.h"// works and compiles
#include "include/damper_Control.h"
#include "include/tempSense.h"
#include "include/Sprinkler.h"
#include "include/PWMTeensyTwoPlusPlus.h"
#include <stdlib.h>


//need to add fan.h later when I figure out how to generate a pwm
```

```
// Teensy 2.0++
#define CPU_PRESCALE(n) (CLKPR = 0x80, CLKPR = (n)) // clock rate at 16Mhz
#define HEX(n) (((n) < 10) ? ((n) + '0') : ((n) + 'A' - 10))
void initializeGlobal(void);
void send_str(const char *s);
uint8_t recv_str(char *buf, uint8_t size);
void parse_and_execute_command(const char *buf, uint8_t num);// still needs updating
void convert_by_division(uint16_t value, char *temp);
uint16_t convertAsciiToInt(char*temp, uint16_t size);
//global verbose adds lots of print statements
uint8_t verbose = 1;
uint8_t invalidInput = 0;
//Simple delay command
static void delay(uint16_t us)
{
	while(us) us--;
}



int main(void)
{
	//Start HeartBeat LED PB7
	PWMNew(2);//configures PB7
	PWMStart(2);
	PWMDuty(2, 128);//

	char buf[32];
	uint8_t n;

	CPU_PRESCALE(0);
	initializeGlobal();
	initializeHVACGarage();
	InitializeDamper();
	InitializeSprinkler();
	InitializeFans();
	// still need to initialize fans

	usb_init();
	while(!usb_configured());
	_delay_ms(1000);
	//enter infinite loop
	while (1)
	{
		while (!(usb_serial_get_control() & USB_SERIAL_DTR))// caught error with ";"
		usb_serial_flush_input();

		send_str(PSTR("\r\n****************************************"));
		send_str(PSTR("\r\nWelcome to Epic Cube Serial Control Version 0.0.1\r\n"));
		send_str(PSTR("****************************************"));
		send_str(PSTR("\r\nto receive general help type \"-help\"\r\n\r\n>"));
```

```
                while (1)
                {
                        send_str(PSTR("> "));
                        n = recv_str(buf, sizeof(buf));
                        if (n == 255) break;
                        send_str(PSTR("\r\n"));
                        parse_and_execute_command(buf, n);

                }
        }
}

/**
*       Send a string to the USB serial port.  The string must be in
*       flash memory, using PSTR
**/
void send_str(const char *s)
{
        char c;
        while (1) {
                c = pgm_read_byte(s++);
                if (!c) break;
                usb_serial_putchar(c);
        }

}

/** Receive a string from the USB serial port.  The string is stored
* in the buffer and this function will not exceed the buffer size.
* A carriage return or newline completes the string, and is not
* stored into the buffer.
* The return value is the number of characters received, or 255 if
* the virtual serial connection was closed while waiting.
**/
uint8_t recv_str(char *buf, uint8_t size)
{

        int16_t r;
        uint8_t count=0;
        //send_str(PSTR("\r\nReceiving String"));
        while (count < size) {
                r = usb_serial_getchar();
                if (r != -1) {
                        if (r == '\r' || r == '\n') return count;
                        if (r >= ' ' && r <= '~') {
                                *buf++ = r;
                                usb_serial_putchar(r);
                                count++;
                        }
                } else {
                        if (!usb_configured() ||
                          !(usb_serial_get_control() & USB_SERIAL_DTR)) {
```

```
                                        // user no longer connected
                                        return 255;
                                }
                                // just a normal timeout, keep waiting
                        }
                }
                return count;

                //return 0;
}

/**
*       This function will take the input command, and parse it,
*       then execute the appropriate command.
**/
void parse_and_execute_command(const char *buf, uint8_t num)
{
        uint8_t Command, Status;// commands and status of each string
        char itemBuf[2];
        // prints the same statement back to the user.
        if(verbose)
        {
                send_str(PSTR("Received: "));
                usb_serial_write(buf, num);
                send_str(PSTR("\r\n"));
        }

        if(num < 4)
        {
                send_str(PSTR("ERROR:1 \r\nformat to small, see help for details\r\n"));
                return;
        }
        if(buf[0] == 'D')// damper section Command is Damper### ##1 for closed and ##0 for open
        {
                //disable all other Devices and enable this
                //Disable Sprinkler
                //Enable Damper
                DAMPER_ENABLE_ON;

                if(buf[6] == '?')// status request is Damper?## check to see if open or closed returns 1 for closed and 0 for
open.
                {
                        //not this will work for up to 9 dampers
                        Command = buf[8] - '0';// convert ascii value to decimal
                        Status = CheckDamper(Command);
                        send_str((char*)(Status + '0'));// returns the status
                        send_str(PSTR("\r\n"));
                        return;
                }
                else
                {
                        Command = (uint8_t)buf[7] - (uint8_t)'0';//unsigned 8 bit makes command a value from 0 - 9
                        //Command += ((uint8_t)buf[6] - (uint8_t)'0');//the higher digit either 1 or 0;
```

```
		itemBuf[0] = buf[6];
		itemBuf[1] = buf[7];
		//Command = atoi(itemBuf);

		if(((uint8_t)buf[6] - (uint8_t)'0') == 1)
		{
			Command = Command + 10; // adds the correct value 0 - 15;
		}
		if(buf[8] == '0')// open
		{
			if(verbose)
			{
				/**
				char tempBuf[2];
				//itoa(Command, tempBuf,10);

				tempBuf[0] = (char)tempCommand + '0';//
				tempBuf[1] = (char)(Command - 10) + '0';
				send_str(PSTR("Starting to open Damper"));
				usb_serial_write(tempBuf, 2);
				//usb_serial_write(itemBuf, 2);
				send_str(PSTR("\r\n"));
				**/
			}
			OpenDamper(Command);
			if(verbose)
			{
				send_str(PSTR("Damper Open Complete.\r\n"));
			}
			return;
		}
		else // Close Damper
		{
		if(verbose)
			{
				char tempBuf[2];
				//itoa(Command, tempBuf,10);

				tempBuf[0] = ((char)(Command)-10) + '0';//
				tempBuf[1] = (char)(Command) + '0';
				send_str(PSTR("Starting to close Damper"));
				usb_serial_write(tempBuf, 2);
				//usb_serial_write(itemBuf, 2);
				send_str(PSTR("\r\n"));
			}
			CloseDamper(Command);
			if(verbose)
			{
				send_str(PSTR("Damper Close Complete.\r\n"));
			}
			return;
		}
	}
```

```
}
if(buf[0] == 'H')// HVAC Section should be DONE debug
{
        if(verbose)
        send_str(PSTR("entering HVAC\r\n"));// H0V1A2V3?4
        if(buf[4] == '?')// status request
        {

                if(GFanStatus == 0)// if furnaced controlled
                {

                        //read furnace for status
                        if(ReadGFan() == 1)//if furnace is on
                        {
                                if(ReadWHeat() == 1)// and heat is on
                                {
                                        send_str(PSTR("2\r\n"));
                                        return;
                                }
                                else if(ReadYCool() == 1)// and the cooler is on
                                {
                                        send_str(PSTR("3\r\n"));
                                        return;
                                }
                                else
                                {
                                        send_str(PSTR("1\r\n"));
                                        return;
                                }
                        }
                        else //it is all off
                        {
                                send_str(PSTR("7\r\n"));
                                return;
                        }
                }
                else// Server is controlling furnace
                {
                        //so GfanStatus is already on
                        //check if hot or cold are on

                        if(WHeatStatus == 1)
                        {// 5
                                send_str(PSTR("5\r\n"));// blower and heater.
                                return;
                        }
                        if(YCoolStatus == 1)// blower and Cooler
                        {
                                send_str(PSTR("6\r\n"));
                                return;
                        }
                        if(GFanStatus == 1)
                        {
```

```
                        send_str(PSTR("4\r\n"));
                        return;
                }

                send_str(PSTR("8\r\n"));// just the blower is on
                return;
        }
}
else // command
{
        if(buf[4] == '1')// Thermo stat controlled
        {
                SetWHeat(0);
                SetYCool(0);
                SetGFan(0);
                SetHVAC(0);
                if(verbose)
                {
                        send_str(PSTR("Server controlled off\r\n"));
                }
        }
        else if(buf[4] == '2')// Blower on
        {
                SetWHeat(0);// may be overridden by Furnace
                SetYCool(0);// may be overridden by Furnace
                SetGFan(1);
                SetHVAC(1);
                if(verbose)
                {
                        send_str(PSTR("Server controlled Fan\r\n"));
                }
        }
        else if(buf[4] == '3')// heat and blower on
        {
                SetGFan(1);
                SetWHeat(1);// may be overridden by Furnace
                SetYCool(0);// may be overridden by Furnace
                SetHVAC(1);
                if(verbose)
                {
                        send_str(PSTR("Server controlled Fan and Heating\r\n"));
                }
        }
        else if(buf[4] == '4')// heat and blower on
        {
                SetGFan(1);
                SetWHeat(0);// may be overridden by Furnace
                SetYCool(0);// may be overridden by Furnace
                SetHVAC(1);
                if(verbose)
                {
                        send_str(PSTR("Server controlled Fan and Cooling\r\n"));
                }
        }
```

```
                        else if(buf[4] == '5')// Server controlled off
                        {
                                SetGFan(0);
                                SetWHeat(0);// may be overridden by Furnace
                                SetYCool(0);// may be overridden by Furnace
                                SetHVAC(1);
                                if(verbose)
                                {
                                        send_str(PSTR("Server controlled OFF\r\n"));
                                }
                                // for now Turn crapper old thermostat to off.
                        }
                        return;
                }
        }

                // Documentation change from Project description
                // instead of Attic and Basement, we are gong with just Temperature
         // sensors, and we will have to decide which temperature sensors are where
         // in the server code.
         // this is a Sts only so it is Temp?## from 0 - 63 for ##
         // example Temp?16 will read from temperature sensor 16 and return the
         // value in C

        if(buf[0] == 'T') // Temperature Section (DEBUG)
        {
                        //disable all other Devices and enable this
                //disable Damper
                DAMPER_ENABLE_OFF;

                char tempStation[2];
                tempStation[0] = buf[4];
                tempStation[1] = buf[5];
                send_str(PSTR("\n\r"));

                uint16_t ADCResult = ReadTempSensor(tempStation);// query temperature section for info. this is a number
from 0 to 1024
                char tempResultStr[4];
                //result is best left positive till C# level for easier conversion.
                convert_by_division(ADCResult, tempResultStr);
                usb_serial_write(tempResultStr, 4);
                send_str(PSTR("\r\n"));
                return;
        }
        else if(buf[0] == 'G')// Garage Section DONE
        {
                //this is only one thing. which is a pulse
                PulseGarage();
                return;
        }
        else if(buf[0] == 'S')// Sprinkler Section DONE
        {
```

```
                //disable Damper
                if(verbose)
                {
                        send_str(PSTR("Sprinkler Section\r\n"));
                }
                //now we need to create a string, and find out
                //which format is
                //Sprinkler##
                //      (#) = station
                //              (#) = Open/On(1) or Closed/Off(0)
                SprinklerCntrl(buf[9],buf[10]);
                return;
        }
        else if(buf[0] == 'F')// Fan Section
        {
                char CharStatus ='0';
                //Fan query Protocol fan

                // FAN###
//              buf[3]    (#) = Fan number(0,2,3,4)// only 4 fans.
//              buf[4]    (#) = on(1)/off(0)
//              buf[5]    (#) = Fan Speed(0-9)// represents a number from 0 - 255.
//                                              if FAN#0x <- x is a don't care
// alternative Fan Status
//                              FAN?##
//              buf[3]    (?) = status request
//              buf[4]    (#) = Fan number(0,2,3,4)// only 4 fans.
//              buf[5]          (#) = Query type 's'= speed and 'o'= on or off
                if(verbose)
                        send_str(PSTR("Fan Section\r\n"));

                if(buf[3]== '?')//Check if status request
                {
                        if(verbose)
                                send_str(PSTR("\tFan Status section\r\n"));


                        if(buf[5] == 's')// status request for speed 0 - 9
                        {
                                if(verbose)
                                        send_str(PSTR("\t\tFan Speed Status Section\r\n"));

                                CharStatus = FanSpeedStatus(buf[4]);// status of fan speed.
                        }
                        else if(buf[5] == 'o')
                        {
                                if(verbose)
                                        send_str(PSTR("\t\tFan On/Off Status Section\r\n"));
                                CharStatus = FanStatus(buf[4]);// status whether on or off.
                        }
                        else
```

```
		{
			invalidInput = 1;// triggers request for printing out instructions.
		}
		if(verbose)
			send_str(PSTR("\tFan Print Status Section>>>\r\n"));
		char charPrintStatus[2];
		charPrintStatus[1] = CharStatus;
		charPrintStatus[2] = CharStatus;

		usb_serial_write(charPrintStatus, 2);//termination char
		send_str(PSTR("\r\n"));
		return;
	}
	else if(buf[3] < ('5') && buf[3] > ('0'-1))//valid input
	{
		if(verbose)
			send_str(PSTR("\tFan Command\r\n"));

		// can still use some valid input checking.
		if(buf[4] == '0')//turn off fan.
		{
			TurnFanOff(buf[3]);
		}
		//command
		else
		{
			SetFan(buf[3], buf[5]);//this will automatically turn on the PWM
		}
	}
	else
	{
		invalidInput = 1;
	}

}
if((buf[0] == '-' && buf[1] == 'h') || buf[0] == '?' || invalidInput)// Help Section
{
	//help commands so write a bunch of things that help people.
	send_str(PSTR("\r\n****************************************\r\n"));
	send_str(PSTR("HELP I don't know what to do\r\n"));
	send_str(PSTR("List of available commands\r\n\r\n"));
	send_str(PSTR("-help:\t\tprovides this menu\r\n"));
	send_str(PSTR("Damper###:\tTurns on and off Damper\r\n"));
	send_str(PSTR("Damper?##:\tReplies whether damper is open or closed\r\n"));
	send_str(PSTR("Temp?##:\tgets the temperature value of the specific sensor\r\n"));
	send_str(PSTR("HVAC?:\t\tReturns HVAC status\r\n"));
	send_str(PSTR("HVAC#:\t\tSets the HVAC to a specific state\r\n"));
	send_str(PSTR("Garage:\t\tPulses the Garage open or closed\r\n"));
	send_str(PSTR("Sprinkler###:\tturns on/off section of sprinklers\r\n"));
	send_str(PSTR("Sprinkler?:\tGets status of sprinkler\r\n"));
	send_str(PSTR("****************************************\r\n"));
	send_str(PSTR("Examples (to-do)\r\n"));
	invalidInput = 0;
```

```
		}
			send_str(PSTR("\r\n"));
}

/**
*		initializes the variables used for Temperature Sense, Sprinkler, and Damper
**/
void initializeGlobal(void)
{
		DEMUX_A_CONFIG;
		DEMUX_B_CONFIG;
		DEMUX_C_CONFIG;
		DEMUX_D_CONFIG;
		// initialize all of the idifferent pins
		DAMPER_ENABLE_CONFIG;
}
/**
*		This function was taken from		Stackoverflow.com/questions/3694100/converting-to-ascii-in-c
*		It converts the int into a string.
**/
void convert_by_division(uint16_t value, char *temp)
{
		temp[0] = (value % 10) + '0';
		temp[1] = (value % 100) / 10 + '0';
		temp[2] = (value % 1000) / 100 + '0';
		temp[3] = (value % 1000) / 1000 + '0';
}

/**
*		This will convert the char string into a decimal. assuming correct formatting
 *if error it will return 0.
 */
uint16_t convertAsciiToInt(char*temp, uint16_t size)
{
		uint16_t result = 0;
		uint16_t i;
		uint16_t j;
		for(i = 0; i < size; i++)
   {
			uint16_t multiplier;
			uint16_t multipleOfTen;
			uint16_t tempResult;

			if(temp[i] > '9' || temp[i] < '0')
				return 0;

			multiplier = size - i - 1;
			// create a multiple of 10
			// 1,10,100,1000,10000.  for the digit
			multipleOfTen = 1;
			for(j = 0; j < multiplier; j++)
			{
				multipleOfTen *= 10;
```

```
                }

                // get uint16_t from 0 - 9;

                tempResult = temp[i] - '0';
                result += (tempResult * multipleOfTen);
        }
 return result;
}
```

C.3 - Damper Control Code

C.3.0 - Damper Control Header (damper_Control.h)

```
#ifndef damper_control_h__
#define damper_control_h__

#include <avr/pgmspace.h>

/**
* this file needs to be able to complete the following tasks.
*
* 1. Open damper (specified by number 0 - 15 ( byte)
* 2. Close damper (specified by number 0 - 15 (byte)
* 3. Check damper status (false for open, and true for closed) (boolean)
*/

//Pin 4(PD1) possible to use interrupts pd0 TO DO make interrupt
#define ISBUTTONOPEN_CONFIG    (DDRD &= ~(1<<1))
#define ISBUTTONOPENBUSHIGH                (PIND & (1<<1))
//Pin 3(PD0) set to input TOD make interrupt enabled.
#define ISBUTTONCLOSE_CONFIG  (DDRD &= ~(1<<0))
#define ISBUTTONCLOSEBUSHIGH (PIND & (1<<0))



//Pin 22(PB6)
#define MotorDirection1HIGH            (PORTB |= (1<<6))
#define MotorDirection1LOW             (PORTB &= ~(1<<6))

//Pin 23(PB5) Keep for a spare for ideal setup
#define MotorDirection2HIGH            (PORTB |= (1<<5))
#define MotorDirection2LOW             (PORTB &= ~(1<<5))
//Pin 24(PB4)
#define MotorStepHIGH       (PORTB |= (1<<4))
#define MotorStepLOW        (PORTB &= ~(1<<4))

#define MotorDirection1_CONFIG       (DDRB |= (1<<6))
#define MotorDirection2_CONFIG       (DDRB |= (1<<5))
#define MotorStep_CONFIG (DDRB |= (1<<4))
```

```
void InitializeDamper(void);
void OpenDamper(unsigned char damper);
void CloseDamper(unsigned char damper);
unsigned char CheckDamper(unsigned char damper);
void ActivateDamper(unsigned char Damper);
#endif
```

C.1.1 - Damper Control Source(damper_Control.c)

```
/**
*Author: Levi Balling
*Date: 7/17/19
*
* Purpose: this file is to control all of the different dampers it will need to store in memory,
* all the correct values.
*/

#include <avr/io.h>
#include <avr/pgmspace.h>
#include <stdlib.h>
#include <util/delay.h>



#include "../include/GlobalVar.h"
#include "../include/damper_Control.h"

uint8_t DamperVerbose = 1;
/**
*		initializes all of the dampers outputs.
*/
void InitializeDamper()
{
		MotorDirection1_CONFIG;// A4988 driver us this one
		MotorDirection2_CONFIG;
		MotorStep_CONFIG;
		ISBUTTONCLOSE_CONFIG;
		ISBUTTONOPEN_CONFIG;
}




/**
*		this activates the correct damper
*		Then while checking the Damper's status
*		runs the stepper motor.
*/
void OpenDamper( uint8_t damper)
{
		if(DamperVerbose)
		{
```

```
                send_str(PSTR("\r\nDamper: "));
                usb_serial_putchar((damper + '0'));
                send_str(PSTR("\r\nEnter OpenDamper Section \r\n"));
        }
        //set the Damper Demux correctly
        ActivateDamper(damper);
//      for(int i; i < 10000; i++);// wait for signal propagation 0.5 seconds is
        _delay_ms(100);
        //step the device open
        uint8_t DamperStatus = CheckDamper(damper);
        // if already open, return
        if(DamperStatus == 0)
        {
                return;
        }
        // if not run the motor
        MotorDirection1HIGH;
//      MotorDirection2LOW; // current setting for A4988
        // set a timeout
        unsigned int timeout = 0; // make sure the motor doesn't spin forever
        // timeout is 5 seconds is 100ms * 50 = 5 seconds
        while(DamperStatus != 0 || timeout > 50)
        {
                // step a lot// need a pause
                MotorStepHIGH;
                DamperStatus = CheckDamper(damper);
                // add ~500 milisec plenty of time for stepper motor not to crush everything
                _delay_ms(100);

                MotorStepLOW;
                timeout++;
        }
        if(timeout > 50)
        {
                // report error to main Could be the wrong board that
                // we are trying to access.
                if(DamperVerbose)
                {
                        send_str(PSTR(" Damper Open Error:TIMEOUT\r\n"));
                }
        }


}
/**
*       Closes the damper shut.
*/
void CloseDamper( uint8_t damper)
{
        if(DamperVerbose)
        {
                send_str(PSTR("\r\nDamper: "));
```

```
                usb_serial_putchar((damper + '0'));
                send_str(PSTR("\r\nEnter CloseDamper Section \r\n"));
        }

        ActivateDamper(damper);

        _delay_ms(500);

        //step the device close
        uint8_t DamperStatus = CheckDamper(damper);
        // if already close, return
        if(DamperStatus == 1)
        {
                return;
        }
        unsigned int timeout = 0; // make sure the motor doesn't spin
        MotorDirection1LOW;// go in opposite direction
        //MotorDirection2HIGH;
        // set a timeout to 5 seconds
        while(DamperStatus != 1 || timeout == 50)
        {
                // step a lot
                MotorStepHIGH;
                DamperStatus = CheckDamper(damper);
                _delay_ms(100);
                MotorStepLOW;
                timeout++;
                if(DamperVerbose)
                {
                        send_str(PSTR("ModTimer: "));
                        usb_serial_putchar((timeout / '0'));
                        send_str(PSTR("\r\n"));
                }

        }
        if(timeout == 10000)
        {
                // report error to main Could be the wrong board that
                // we are trying to access.
                if(DamperVerbose)
                {
                        send_str(PSTR(" Damper Close Error:TIMEOUT\r\n"));
                }
        }

}
/**
* So the Damper has a procedure to access the status
* first to have to activate the section
* second you have to check if it is open = 0
* third you have to check if it is close = 1
* fourth you have to check if it is in the middle = 2
```

```
* damper is a number from 0 to 15
*/
uint8_t CheckDamper(uint8_t damper)
{
	//Go through each damper and check if the input on the Bus is high or low.
	//double make sure it is good.
	ActivateDamper(damper);

	// check if it is in the middle
	for(int i; i < 10000; i++);// wait for signal propagation 0.5 seconds is
	// enough
	// it is in the middle of turning
	if(ISBUTTONOPENBUSHIGH & ISBUTTONCLOSEBUSHIGH)
	{
		if(DamperVerbose)
		{
			send_str(PSTR("Both inputs are HIGH\r\n"));
		}
		return 2;
	}
	else if(!ISBUTTONOPENBUSHIGH & ISBUTTONCLOSEBUSHIGH)
	{
		if(DamperVerbose)
		{
			send_str(PSTR("OPENBUS is LOW and CLOSE Button is HIGH\r\n"));
		}
		return 0;
	}
	else
	{

		if(DamperVerbose)
		{
			send_str(PSTR("CloseButton is Low and Open Button is HIGH\r\n"));
		}
		return 1;
	}
}
/**
*	Sets the correct damper to be active.
* Damper is high when button isn't pressed
* for simplicity it would be better to switch this to string comparisons.
*/
void ActivateDamper(uint8_t damper)
{
// converts decimal to binary digits for ABCD
// could be backwards.
// example input A hex or 1010 bin or 10 dec
	uint8_t D = damper / 8;//  D = 1bin or 1 dec
	uint8_t temp = damper % 8;// temp = 10bin and 2 dec

	uint8_t C = temp / 4;// C = (2)/4 = 0
	temp = temp % 4;// temp = (2)%4 = 2
```

```
	uint8_t B = temp / 2;// 1
	temp = temp % 2;// 0

	uint8_t A = temp;

	char tempBuf[4];
                                        //itoa(Command, tempBuf,10);

                                        tempBuf[0] = (char)A + '0';
                                        tempBuf[1] = (char)B + '0';
                                        tempBuf[2] = (char)C + '0';
                                        tempBuf[3] = (char)D + '0';
                                        send_str(PSTR("Test Outputs of ABCD:\r\n"));
                                        usb_serial_write(tempBuf, 4);
                                        //usb_serial_write(itemBuf, 2);
                                        send_str(PSTR("\r\n"));
	//set the output pins as such.
	//d to A
	if(D==1)
	{
		DEMUX_D_ON;
	}
	else
	{
		DEMUX_D_OFF;
	}
	if(C==1)
	{
		DEMUX_C_ON;
	}
	else
	{
		DEMUX_C_OFF;
	}
	if(B==1)
	{
		DEMUX_B_ON;
	}
	else
	{
		DEMUX_B_OFF;
	}
	if(A==1)
	{
		DEMUX_A_ON;
	}
	else
	{
		DEMUX_A_OFF;
	}
}
```

C.2 - Fan Control Software Code

C.2.0 - Fans Control Header (fans.h)

```
/**
*Author: Levi Balling
*Date: 7/17/19
*
* Purpose: this file is to control all of the Fans with a PWM signal
*/
#ifndef fans_h__
#define fans_h__

#include <avr/pgmspace.h>
#include <util/delay.h>

// #define DEMUX_A_CONFIG          (DDRB |= (1<<3))
//Configure Fan Pins PC6, PC5, PC4, PD0, PD3
//#define XTAL 160000000L // 16Mhz



//their are 5 fans with a
void InitializeFans(void);
uint8_t SetFan(char FanChar, char FanSpeedchar);
char FanStatus(char FanChar);
uint8_t TurnFanOff(char FanChar);
uint8_t CharToSpeed(char speed);
char FanSpeedStatus(char FanChar);

char        FanZeroStatus = '0';// need to initialize it.
char        FanOneStatus = '0';// need to initialize it.
char        FanTwoStatus = '0';// need to initialize it.
char        FanThreeStatus = '0';// need to initialize it.
char        FanFourStatus = '0';// need to initialize it.
char FanZeroSpeed = '0';//
char FanOneSpeed = '0';
char FanTwoSpeed = '0';
char FanThreeSpeed = '0';
char FanFourSpeed = '0';
// // Pin 3 (PD0)
// extern uint8_t FanZeroStatus;
// // Pin 4 (PD1)
// extern uint8_t FanOneStatus;//always on, but its for the case
// // Pin 17 (PC4)
// extern uint8_t FanTwoStatus;
// // Pin 18 (PC5)
// extern uint8_t FanThreeStatus;
// // Pin 19 (PC6)
// extern uint8_t FanFourStatus;

#endif
```

C.2.1 - Fans Control Source (fans.c)

```
/**
*Author: Levi Balling
*Date: 7/17/19
*
* Purpose: this file is to control all of the different dampers it will need to store in memory,
* all the correct values.
*
* Steps to setup of a fan PWM signal
* 1. set pin as output(OCx)
* 2. Select PWM mode of timer
* 3. Set appropriate prescaler divider
* 4. Set Compare Output Mode to Clear or Set on compare match
* 5. Write Duty cycle value to PWM(OCRx)
*/

#include <avr/io.h>
#include <avr/pgmspace.h>

#include "../include/Fans.h"
#include "../include/PWMTeensyTwoPlusPlus.h"

/*
*		This example is comprized of mainly arduino sources
// */
// uint8_t FanZeroStatus = 0;// need to initialize it.
// uint8_t FanOneStatus = 0;// need to initialize it.
// uint8_t FanTwoStatus = 0;// need to initialize it.
// uint8_t FanThreeStatus = 0;// need to initialize it.
// uint8_t FanFourStatus = 0;// need to initialize it.
/**
*		initializes the pins for the fans.
*/
void InitializeFans()
{
		//configure the fans that we want to use.
		PWMNew(3);
		PWMNew(17);
		PWMNew(18);
		PWMNew(19);
		PWMDuty(3, 0);
		PWMDuty(17, 0);
		PWMDuty(18, 0);
		PWMDuty(19, 0);
		PWMStartTimer(3,4);
		PWMStartTimer(17,4);
		PWMStartTimer(18,4);
		PWMStartTimer(19,4);


}
```

```
/**
*		Sets the speed of a fan
*  FanSelect 0-4 for the specific fan.
*		the only valid fans are 0,2,3,4
*  FanSpeed 0-255 0 for off and 255 for Max power.
*		returns 0 if successfully changed.
*		else returns the fan number(OFFSET by +1) of the other fan that is currently on.
*		this would be an Error
*/
uint8_t SetFan(char FanChar, char FanSpeedchar)
{
		uint8_t FanSelect = FanChar - '0';
		uint8_t FanSpeed = CharToSpeed(FanSpeedchar);
		//fan pins are Fan0-PD0(3),(DON'T USE Fan1-PD3(6)/Not a PWM(case fan)
		//Fan Fan2-PC4(17), Fan3-PC5(18), Fan4-PC6(19)

		//need to check status of fans 0,2,3,4 we don't want more than 1.5amps of draw.
		//can also draw off of TempSense power on power board. just wire directly to connector, and skip
		// soldering it. // then we can power 2 at the same time.

		//NOTE about circuit design.
			//bail one is already running, and we can't over draw circuit
			// can change if you supply 1.5 amp power supply to each fan.
			// to do this you would just need a L7812 voltage reg for each.
			// it would be good to have a heatsink on each of them.
		if(FanZeroStatus == '1' && FanChar != '0')//return 1 for fan 0
		{
			return 1;
		}
		else if(FanTwoStatus == '1' && FanChar != '2')//return 3 for fan 2
		{
			return 3;
		}
		else if(FanThreeStatus=='1' && FanChar != '3')//return 4 for fan 3
		{
			return 4;
		}
		else if(FanFourStatus == '1' && FanChar != '4')//return 5 for fan 4
		{
			return 5;
		}
		else if(FanSelect == 0)
		{
			FanZeroStatus = '1';
			//set duty cycle
			PWMDuty(3, FanSpeed);
			FanZeroSpeed = FanSpeedchar;
		}
		else if(FanSelect == 2)
		{
			FanTwoStatus = '1';
			//set duty cycle
```

```
                PWMDuty(17, FanSpeed);
                FanTwoSpeed = FanSpeedchar;
        }
        else if(FanSelect == 3)
        {
                FanThreeStatus = '1';
                //set duty cycle
                PWMDuty(18, FanSpeed);
                FanThreeSpeed = FanSpeedchar;
        }
        else if(FanSelect == 4)
        {
                FanThreeStatus = '1';
                //set duty cycle
                PWMDuty(19, FanSpeed);
                FanFourSpeed = FanSpeedchar;
        }
        else
        {
                return 255;//max invalid fan.
        }
}

/*
*       gets the FanSpeed of a specific Fan
* returns 0 - 100 based on the fan speed.
*/
char FanStatus(char FanChar)
{
        uint8_t FanSelect = FanChar - '0';
        if(FanSelect == 0)
        {
                return FanZeroStatus;
        }
        else if(FanSelect == 2)
        {
                return FanTwoStatus;
        }
        else if(FanSelect == 3)
        {
                return FanThreeStatus;
        }
        else
        {
                return FanFourStatus;
        }
}
/*
*       gets the FanSpeed of a specific Fan
*       returns 0,1,2..8,9
*/
char FanSpeedStatus(char FanChar)
{
        if(FanChar == '0')
```

```
		{
			return FanZeroSpeed;
		}
		else if(FanChar == '2')
		{
			return FanTwoSpeed;
		}
		else if(FanChar == '3')
		{
			return FanThreeSpeed;
		}
		else if(FanChar == '4')
		{
			return FanFourSpeed;
		}
		return 'n';// if invalid input.
}
/*
*		This will change the fan status, and change the
*		duty cycle of the fan to represent off.
*		return 0 if sucessful, and 1 for invalid input.
*/
uint8_t TurnFanOff(char FanChar)
{
		uint8_t FanSelect = FanChar - '0';
		//		Fan 0 pin3(PD0),
		if(FanSelect == 0)
		{
			PWMDuty(3, 0);
			FanZeroStatus = '0';
			FanZeroSpeed = '0';
		}
		else if(FanSelect == 2)
		{
			PWMDuty(17, 0);
			FanTwoStatus = '0';
			FanTwoSpeed = '0';
		}
		else if(FanSelect == 3)
		{
			PWMDuty(18, 0);
			FanThreeStatus = 0;
			FanThreeSpeed = '0';
		}
		else if(FanSelect == 4)
		{
			PWMDuty(19, 0);
			FanFourStatus = '0';
			FanFourSpeed = '0';

		}
		else
			return 1;
		return 0;
```

```
}
/*
*		10 speeds 0 to 9
*		returns a value from 0(0) to 255(10)
* returns 0 if invalid.
*/
uint8_t CharToSpeed(char speed)
{
		if(speed == '0')
				return 0;
		else if(speed == '1')
				return 28;
		else if(speed == '2')
				return 56;
		else if(speed == '3')
				return 85;
		else if(speed == '4')
				return 113;
		else if(speed == '5')
				return 141;
		else if(speed == '6')
				return 170;
		else if(speed == '7')
				return 198;
		else if(speed == '8')
				return 226;
		else if(speed == '9')
				return 255;
		else
				return 0;
}
```

C.4 - Fan Control Software Code

C.4.0 - Damper Control Header (fans.h)

C.4.1 - Damper Control Source(fans.c)

C.5 - HVACGarage Software Code

C.5.0 - HVACGarage Control Header (HVACGarage.h)

C.5.1 - HVACGarage Control Source(HVACGarage.c)

C.6 - PWM Control Software Code

C.6.0 - PWM Control Header (PWMTeensyTwoPlusPlus.h)

C.6.1 - PWM Control Source(PWMTeensyTwoPlusPlus.c)

C.7 - Sprinkler Control Software Code

C.7.0 - Sprinkler Control Header (Sprinkler.h)

C.7.1 - SprinklerControl Source(Sprinkler.c)

C.8 - Temperature Sensing Control Software Code

C.8.0 - Temperature Sensing Control Header (tempSense.h)

C.8.1 - Temperature Sensing Control Source(tempSense.c)

C.9- USB Serial Control Software Code

C.9.0 - USB Serial Control Header (usb_serial.h)

C.9.1 - USB Serial Control Source(usb_serial.c)