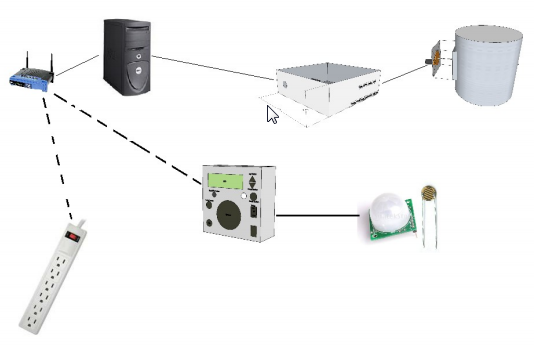
T O D D R O G E R S

C H R I S T O P H E R J O H N S O N  
D A R I O B O S N J A K  
L E V I B A L L I N G

**SMART HOME**

C O M P U T E R E N G I N E E R I N G F I N A L P R O J E C T

2 0 1 2



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

INTRODUCTION………………………………...…………………………………………………………………3

PROJECT OVERVIEW…………………………………………………………………...…………….………………………3

Software Function................................................................ .........................................................................3

Hardware Function................................................................ ........................................................................3

Wi-Fi Functionality....................................................................................................................................................4

Soil Moisture/Sprinkler Control...............................................................................................................................6

OUTLET CONTROL............................................................................................ ...................................................8

SERVER......................................................................................................................................................................10

MOTION SENSOR....................................................................................................................................................11

TEMPERATURE SENSOR......................................................................................................................................12

BLOWER CONTROL...............................................................................................................................................13

GARAGE DOOR CONTROL...................................................................................................................................17

HVAC..........................................................................................................................................................................14

STEPPER MOTOR CONTROL..............................................................................................................................15

DAMPER CONTROL...............................................................................................................................................15

SPRINKLER CONTROL..........................................................................................................................................16

CONCLUSION………………....................................................................................................................................17

ACKNOWLEDGEMENTS…....................................................................................................................................17

**Appendix A Wi-Fi/Moisture Sensor code.................................................................................................................18**

**Appendix B Server Code............................................................................................................................................25**

[**Appendix C Teensy Controller Code**](#id.ghsms1d48uaw)**........................................................................................................................35**

[**C.0 Makefile Software Code**](#id.28hiw25mlzmy)

[**C.1 Global Variables Code(GlobalVar.h**](#id.yq8eesl9xnz3)**)**

[**C.2 Main Controller Code(**mainControler.c)](#id.jmmimgrkd34z)

[**C.3 Damper Control Code**](#id.eq97w56siunw)

[**C.3.0 Damper Control Header(damper\_Control.h)......**](#id.4abjl1wae3b3)

[**C.3.1 Damper Control Source(damper\_Control.C)......**](#id.yq8eesl9xnz3)

[**C.4 Fan Control Software Code**](#id.mnmw6dj9lcej)

[**C.4.0 Damper Control Header (fans.h)**](#id.3v9y18h677q3)

[**C.4.1 Damper Control Source (fans.c)**](#id.oliyi6yhdh61)

[**C.5 HVAC and Garage Software Code**](#id.cc4hpynldizm)

[**C.5.0 HVAC and Garage Control Header (HVACGarage.h)**](#id.1ouvxkxlti82)

[**C.5.1 HVAC and Garage Control Source (HVACGarage.c)**](#id.2txakeealaub)

[**C.6 PWM Control Software Code**](#id.ytz73xna7vpp)

[**C.6.0 PWM Control Header (PWMTeensyTwoPlusPlus.h)**](#id.a4y8a5fu8na0)

[**C.6.1 PWM Control Source (PWMTeensyTwoPlusPlus.c)**](#id.yybsfyci9yk)

[**C.7 Sprinkler Control Software Code**](#id.2aas9w4zu2ba)

[**C.7.0 Sprinkler Control Header (Sprinkler.h)**](#id.bz4intxiusfw)

[**C.7.1 Sprinkler Control Source (Sprinkler.c)**](#id.c273jsrfb7sv)

[**C.8 Temperature Sensing Control Software Code**](#id.4jbi05hmdnei)

[**C.8.0 Temperature Sensing Control Header (tempSense.h)**](#id.pan735flh728)

[**C.8.1 Temperature Sensing Control Source (tempSense.c)**](#id.vvdhm472bps8)

[**C.9 USB Serial Control Software Code**](#id.li4csp480w9e)

[**C.9.0 USB Serial Control Header (usb\_serial.h)**](#id.1fe820xu6qwd)

[**C.9.1 USB Serial Control Source (usb\_serial.c)**](#id.g11pipk1e9w8)

**Appendix D Server Code...........................................................**

**I N T R O D U C T I O N**

While the cost of living is going up, there is a growing focus to involve technology to lower those prices. With this in mind the Smart Home project allows us to build and maintain a house that is smart enough to keep energy levels down while providing more automated applications. 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 engineering challenges, including software programing, PCB design, Wi-Fi, TCP/IP protocols, Web Server logic design, and other aspects. This project provided great insights to the challenges of software and hardware engineering.

**O V E R V I E W**

OVERALL SOFTWARE FUNCTION

The 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 Python, C, and Arduino (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, lasers, 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

The responsibility of the hardware in this project is to demonstrate the functionality of a smart home. Based on software commands received from the server, specific hardware is turned on and, as per design requirements, it performs its functionality. Features of hardware are as follows:

* Stepper motors are responsible for closing/opening air dampers based on the temperature
* Sensors monitoring the temperature in a room
* Sensors monitoring the current occupancy of a room
* 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**

In order to communicate over long distances without running wires, we came up with a convenient way of communicating with our sensors. The different I/O devices are controlled by using TCP/IP over the IEEE 802.11g standard protocol. Data being gathered from sensors, such as temperature sensors, light sensors, and 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 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/?LED](http://192.168.1.102/?LED1)0 … 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 wireless 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**

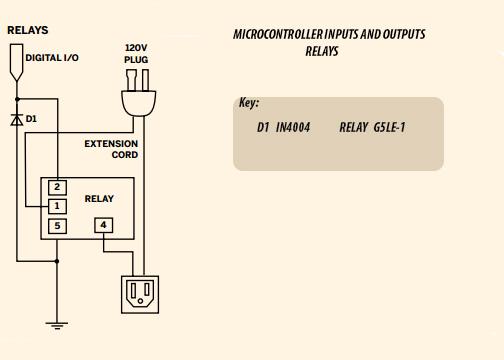
In the attempt to make the house as automated as possible a sprinkler control system has been invented that is responsible on turning on the sprinkler based on some preset parameters. It is well known that it can be expensive to maintain green grass and the last thing you want to do is to waste water when it’s 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 sprinkler system only to be run during the night and only if the soil moisture is less than a preset 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. The Sensor is connected with one wire to analog input 0 on the Arduino board. The input value is stored in an int variable that ranges from 0 to 700; it represents the moisture level of 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 it’s placed in a cup of water. Once the 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 light was off there was a ~100 ohms value and with the light on the value went in to 900 ohms 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 below 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.

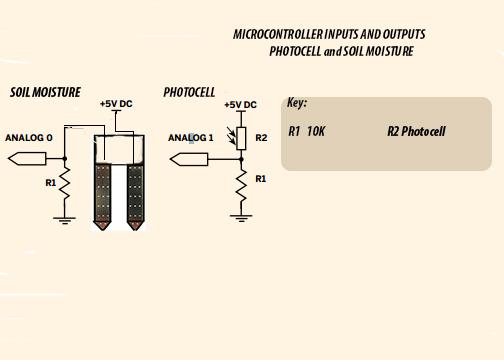
**HARDWARE FUNCTION**

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.



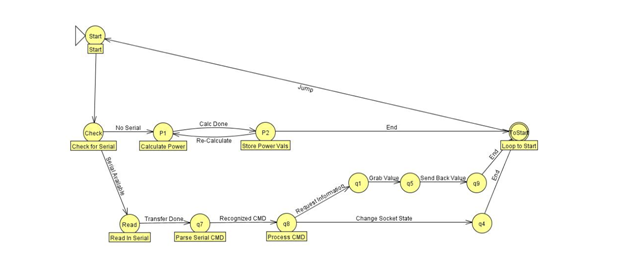
*Figure 3.*

**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 Wi-Fi. 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 Wi-Fi 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 power strip, 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 power strip. 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

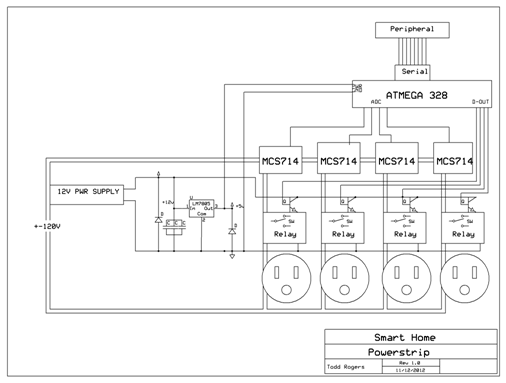
socket <socketnum> //returns on/off

allpower //returns all powers on a new line each

active sockets //returns the number of powered sockets

**HARDWARE**

The power strip 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 power strip 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 power strip hardware, note that this does not include peripherals such as the WIFI board.



*Figure 5*

**LIGHT AND APPLIANCE CONTROL**

A home’s illumination is usually the prime in the order of things to automate. People want their lights to turn on when it’s dark, when an area is occupied, and sometimes during schedules times. They want lights and certain nonessential appliances to turn off when not in use. Plug-in lamps need to integrate seamlessly with fixture lighting. It needs to be easy to add new lights and appliances to a system.

To make these things come to pass, the right software and hardware must work together. These are described next.

Hardware

Besides automating lighting according to a fixed schedule, which is often called for with exterior lighting, an innovative application of home automation is to determine room occupancy and turn the lights on or off as each room or area is populated or vacated. Typical setups to determine room occupancy, both at home and in industrial settings involve one or more motion sensors and a count-down timer.

It works like this. When motion is sensed, the lights are turned on, and a count-down timer is started. Whenever there is more motion, the timer is reset to its starting time and continues to count down. If the counter reaches 0, then the lights are turned off.

There are two related problems with this. First, when an area is vacated, the whole length of the timer must expire before the lights can be shut off. Second, sometimes occupants are present in a room but produce so little motion activity that the lights will turn off, even though a room is not vacant.

The setup can be drastically improved by adding to the motion sensors a device called a light beam interruption detector or LBID. [Diagram of LBID] An LBID detects when the boundary between rooms is crossed by sending a beam of light across a threshold to a sensor that recognizes obstruction of the light beam as a representing a person going between rooms or areas in an indeterminate direction.

When used in conjunction with one or more motion sensors, a home automation system can determine whether occupant(s) have entered a room or exited a room without having to wait for a lengthy delay. One a room’s occupancy has been positively established; there need be no turning the lights on or off or resetting count-down timers until the LBID determines that the threshold has been crossed again. Motion sensors and LBIDs used in tandem comprise a more rapid and reliable method of determining room occupancy than motion sensors alone.

**SERVER**

The server for our smart home project consists of two parts; both programmed using the Python programming language. The core of the server is a program named Mirabilis (Latin for wonderful or extraordinary). Mirabilis is responsible for maintaining the state of all entities in the home system, including lights, appliances, sprinklers, HVAC, doors, and sensors. It is intended to execute Python functions when events are triggered, both in response to changes in state of sensors and entities as well as in response to chronological events, such as timers and alarms.

The second part of the server, called Ladybug, is responsible for providing the user interface to the system. Ladybug runs on the same server computer and, in response to HTTP requests, queries the state of Mirabilis over operating system sockets. In response to activity on the web pages served, Ladybug can change the state of entities in Mirabilis or trigger other events.

**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).

Thermistor circuit

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.

Image for 8 analog paths to 8 x(3 to 8 ) analog channel demux

The 3 to 8 analog channel demux is controlled by a 3 digital I/O pins. The 3 digital I/O pins go to all 8 demux.

**BLOWERS**

The Air vent blowers are to assist air flow where it is needed most. When the furnace is at one end of a home will cause a loss in air pressure at the other end (figure x.x).



The typical home runs a blower at ~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 can 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. 

**Figure x.x PWM Datadog Systems www.datadog.com/PWM\_tutorial.pdf**

**HVAC CONTROL**

The HVAC system in a home controls the Heating, AC, and blower. The thermostat controls these setting with 24VAC communication lines. These communication lines are typically labeled G (fan), W (heat), Y (cool), R (24VAC) (figure x.x).

Picture of GWYR connection on a thermostat

The HVAC system will recognize to turn on when one or two of these wires are connected to 24VAC. When they aren’t connect to 24VAC they are floating. In order to determine whether the lines are on or off we used optocoulpers. Can be triggered with a wide voltage range, and it will behave similar to a transistor. Since the signal can be floating we have a pull up resistor on the other end of the network, to make sure we have a better signal. The signal then goes through a inverter to conform with either positive 5v signal or 0v. This input goes into the Teensy+ to determine whether it is on or off (figure x.x)

Small circuit of optocoupler and inverter.

Due to the signal being + and - 24VAC, the optocoulpers will only be active ~35% of the time. Since 60Hz is slow we can oversample the input for 100 ms, if we read high at any point during this time we will know it is on (figure x.x).

Optocoulpers saturation mode, and inverter on/off with respects to 60Hz waveform.

The control of the HVAC from the server will provide controls to allow the thermostat to remain control of the system interrupt thermostat control and control thermostat settings from the server, or the server will control all signals to be off. This will allow people to control the temperature from anywhere, and control it for any period of time. Each signal runs through a couple of relays, the first relay controls whether the server controls the signal, the second relay controls whether it is on, or connected to the thermostat (figure x.x).

Picture of 2 relays in series controlling each signal.

The relays are being controlled by the teensy+ gpio pins.

**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.

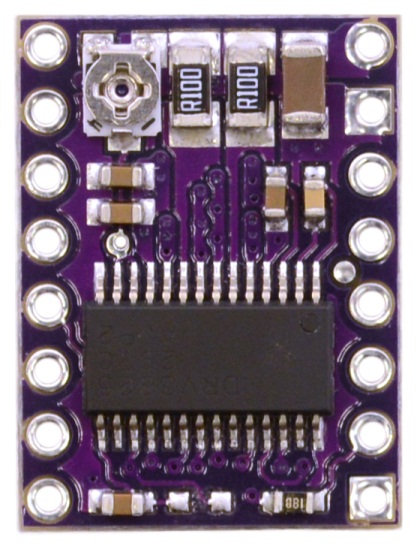


Figure x.x Pololu DRV8825 Stepper Motor Driver Carrier, High Current www.pololu.com/catalog/product/2123

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 went where. The teensy MCU uses 4 signals to control a 4 to 16 demux chip. The chip then goes through an inverter and buffer. The high signal will flip a set of 4 relays; each relay will connect 1 coil of the stepper motor driver to the motor (figure x.x).

Damper connected to relays Figure x.x

It is important to disable the stepper motor driver during the time of switching. Damage can occur if switching is done while driving a stepper motor. After the relays are switched the damper will turn open or close till it hits a button (figure x.x).

Damper open and closed states Figure x.x

This button will trigger the Teensy+ to stop. The signal line runs on a bus system, allowing only the enabled damper to respond when the button is pressed (figure x.x).

Damper Bus configuration Figure x.x

The Damper control board contains larger traces in order to handle the higher current needs. With a stronger stepper motor driver, the damper controller would be able to source 3 amps per coil (figure x.x).

Image of trace sizes and chart of current for 1 oz open traces Figure x.x

**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.

Image of simple relay circuit

Figure x.x

The software to control the GPIO pin is simple. You configure the Comport to be an output. Then you set the output to either High or Low.

**SPRINKLER CONTROL**

The sprinkler system is similar to the garage door. The sprinkler valves run on 24VAC. To turn one of them on you simply need to connect the 24VAC signal to the sprinkler wire. This will turn on the solenoid valve. Since most sprinkler systems consist of only a few valves. We made our system capable of controlling 6. Since you only need one station on at a time. We control add 6 stations we a 3 to 8 demux. Since we are only running 6 signals, out of the 8, we can tie the enable line on the 3 to 8 demux on, and only use 3 wires to control them. To disable all the stations we just set the 3 to 8 demux to 7 or 8 (figure x.x).

Sprinkler design Figure x.x

**CONCLUSION**

Are goal was to implement home automation at a lower cost than typical modern systems. To build our system cost ~$1000, and to purchase the market equivalent is $2000+. It is easy to expand this project to control addition features, voice control, lighting control, speaker control, media control, and more. I provides practical uses to help make life easier.

**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](http://www.pjrc.com/teensy); 2012

[x] Rob Paisley, Bi polar stepper motor circuits.

[x] Special thanks goes to Jennifer Balling, in assisting with grammatical corrections with papers, and her recently electronic technician skills.☺

**APPENDIXES**

**Appendix A - Wifi Arduino Code**

**Appendix B - Server Software**

**Appendix C - Teensy Controller Software**

**C.0 - Makefile Software Code**

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

**C.2 - MainController Code(mainController.c)**

**C.3 - Damper Control Code**

**C.3.0 - Damper Control Header (damper\_Control.h)**

**C.1.1 - Damper Control Source(damper\_Control.c)**

**C.2 - Fan Control Software Code**

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

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

**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)**