L E V I   B A L L I N G

D A R I O   B O S N J A K

C H R I S T O P H E R   J O H N S O N

T O D D   R O G E R S

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

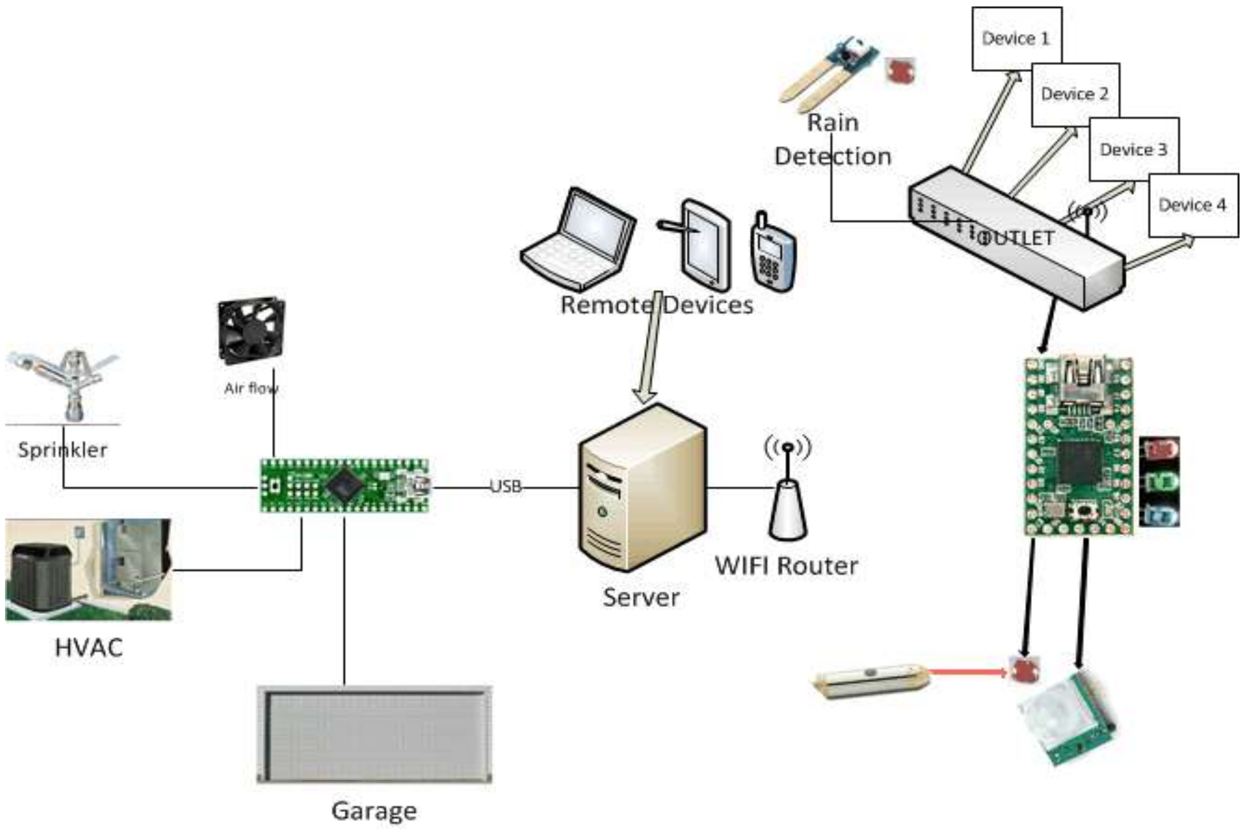

TABLE OF CONTENTS

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 the user 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 the user is 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). 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
- Manipulate relays

OVERALL HARDWARE FUNCTION

The server controls various hardware devices. 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
- Power Box Outlets controlled with an microcontroller
- Relays and Wi-Fi modules controlled by the Server

Wi-Fi FUNCTION

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 using TCP/IP over the IEEE 802.11g standard protocol. Data being gathered from sensors, such as temperature sensors, light sensors, and laser tripwire sensors, is being processed on an Arduino Micro-controller and then broadcast with an attached WiShield v2.0 to a server using the TCP/IP protocol. Arduino has a statically assigned IP address that corresponds to an individual room in the house. Each time a request is made to that IP address, an 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 input/output devices such as power strip plugs can be remotely controlled.

SOFTWARE FUNCTION

The main functionality of software is to be responsible for monitoring the changes in attached hardware and to initiate controlling statements that, depending on the data received, would trigger an event based on that condition.

- 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 other MCU controllers 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 power sensor information and store it for clients to see.

Software is based on Arduino code that is based on the C programming language. It consists of libraries that create web servers for Arduino Duemilanova MCU's Wi-Fi and also libraries responsible of setting up software serial communication to another MCU controller.

The system consists of a web server that hosts an HTML page which displays to the user different sensor information. The web server also allows the user to control output devices remotely. Sensors wired to analog inputs on the MCU controller are monitored by the server and the information is displayed on the generated HTML page. Analog inputs are monitoring temperature values from LM335A sensors, moisture values from SEN0114 sensor, and resistive photo cell responsible for monitoring the outside brightness. Values monitored from the analog inputs are assigned to variables that are passed to ADCs that convert the analog input data to digital which is then displayed to the user on the page. Once the request is made to the web server hosting the web page, the returned HTML page consists of fields populated by sensor values and options to manually turn on power strip plugs. The software takes advantage of digital inputs on the Arduino by utilizing them to monitor values coming from the motion detector sensor and the 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 the software is to integrate sensor data and make control decisions to operate the I/O devices. One of the features involves the ability to check the moisture of soil and also check the time of day by monitoring the light intensity. If both parameters meet the preset criteria, a digital enable gets sent, which triggers the digital 5V line to be sent from the MCU controller to a relay that turns on the sprinkler system.

Web Server's main communication protocol is handled by the Arduino WiShield v2.0 running Wi-Fi 802.11b protocol. Requests to the web server come through the TCP/IP protocol, depending on the received URL which consists of embedded strings that once picked by the server will execute a specific command attached to the received string. A manually assigned IP address in this case 192.168.1.102 will make an request to the server to open a listening connection and upon receiving a URL such as 192.168.1.102/?LED0 … LED3 will display the current state of sensors and I/O devices. Besides automated control of the I/O devices based on specific arguments, manual control is also available by retrieving the URL that is mentioned above. This will cause an event to be triggered and will execute a specified block of code responding to that URL page, in most cases turning on an appliance connected to that power plug being controlled by that URL. The actual code used in the project is attached at the bottom of the document in *Appendix A.*

HARDWARE FUNCTION

Software controls the communication between devices using standard TCP/IP protocol. In order to successfully allow the control and monitoring of different devices we have used an Arduino Duemilanove MCU with a WiShield v2.0 Wi-Fi 802.11b 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 using a Netgear wireless N router. A 5V line is connected to the 5V pin on the Arduino Duemilanove MCU, and the common ground is shared between the Arduino Uno running the power strip outlets. This significantly reduces noise on the serial signal lines between the two MCU controllers.

Using the Arduino WiShield v2.0 as a shield combined with the Arduino Duemilanove poses a limitation on the number of digital pins available for external use. Below are the specifications for the WiShield V2.0.

- 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 is placed inside a power strip box where it will be in close proximity with the MCU responsible for controlling the power strip outlets. The power strip box is made out of thin sheets of metal that caused the Wi-Fi shield to loose signal periodically. In order to resolve the issue a Linksys Wi-Fi antenna has been soldered on the WiShield and placed outside the box to increase Wi-Fi strength signal between the router and the WiShield.

SPRINKLER SYSTEM FUNCTIONALITY

In the attempt to make the house as automated as possible a sprinkler control system has been created that is responsible for turning on the sprinkler based on checking the moisture level of the ground and checking the light conditions. It is well known that it can be expensive to maintain green grass and watering the grass when it's not necessary is clearly undesirable. A soil moisture sensors works by measuring the conductivity in the ground and reports the level back to the Arduino MCU controller through Analog input 0. Analog input 1 measures the resistance from the photocell to determine if it's day or night. This approach will limit the sprinkler system to only run at night, and only if the soil moisture is less than the hard coded value.

SOFTWARE FUNCTIONS

Arduino code is based on C, which drives built in analog and digital I/O's that enables us to monitor the analog input values of pins 0 and 1. Values that are monitored through 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 as an integer that ranges from 0 to 700, representing the current moisture level of the soil. Values range from 0 to 300 representing soil is dry, from 301 to 500 to represent that soil is moist, and from 501 to 700 that the soil is in water. These values were collected by creating a monitor loop in C that would monitor the values as the moisture sensor was placed in different conditions, ranging from being in dry soil to being in a cup of water. Values are based on conductivity of soil once exposed to different amounts of water. When 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 photocell. Once the light was off there was an approximate 100 ohms resistance value that was observed, and with the light on the value went to 900 ohms. After determining these values, a loop was created that would check every 10 min if the values have changed and, based on the new data, decide if the sprinkler system should be turned on. 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 5V output to a 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.

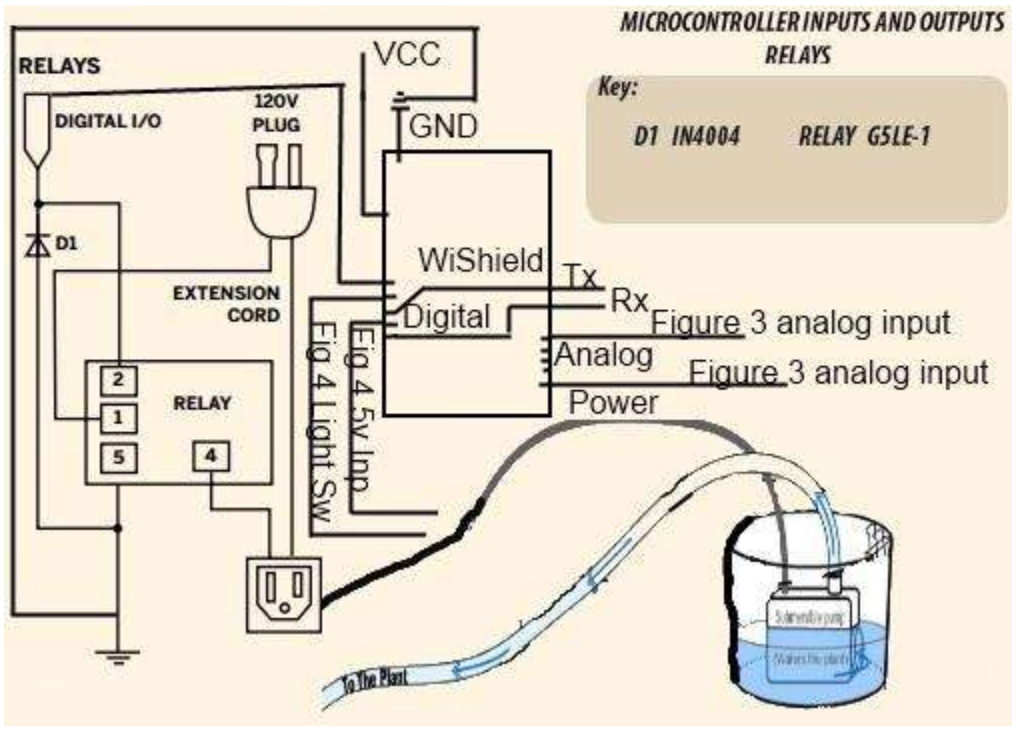

Figure 2.

Relays will be controlled with digital 5V sent from the Arduino MCU once the web server initiates a certain command, that command will be based upon two different factors. One of the factors is based on the 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 the light sensor indicates its dark then sprinkler is turned on. *Figure 3* illustrates the next two steps necessary in controlling the sprinkler system. *Figure 2's* analog inputs gather values coming from *Figure 3's* analog outputs and gathered values are used on the WiShield to perform control over sprinkler system. Above we can also see that two digital lines are assigned for software serial implementation which transcode data over software serial in order to control Arduino Uno that is responsible for driving the power box. *Figure 2* also illustrates the functionality of monitoring the light system and based on the signal received it triggers the relay controlling the lights and at the same time records the current state of lights. Digital 5V or 0V comes from the Teensy's digital out specifying if the lights should be on or off. *Figure 4* illustrates the light system, as well the Tx and Rx lines necessary for the software serial to communicate over the Digital I/O's between WiShield and Arduino Uno.

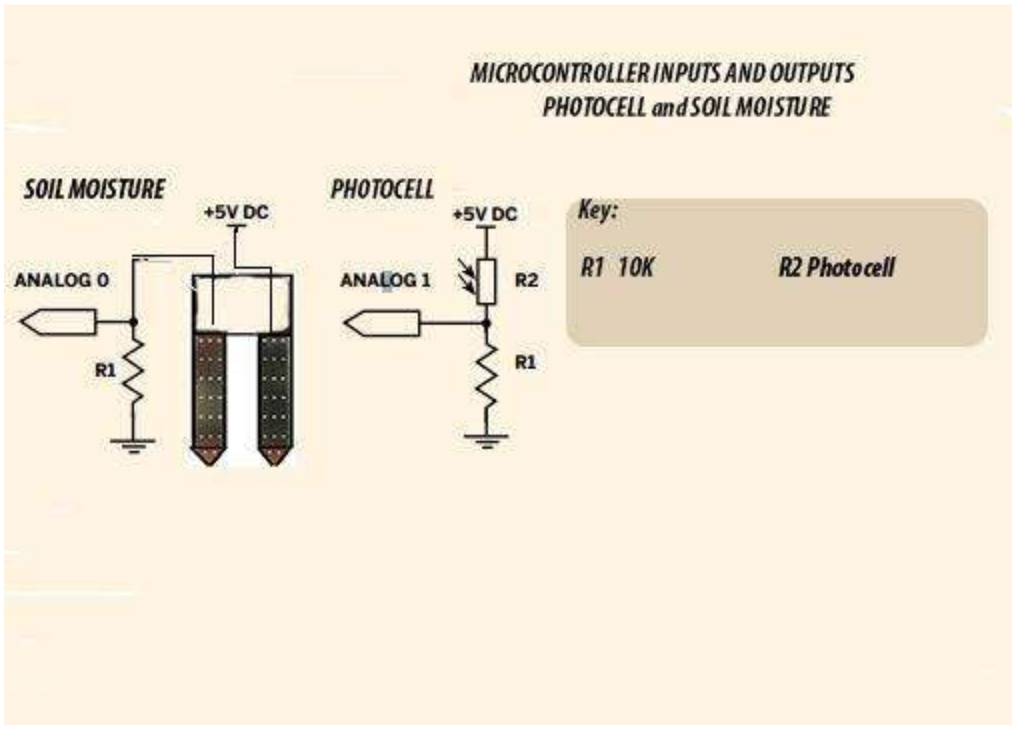

Figure 3.

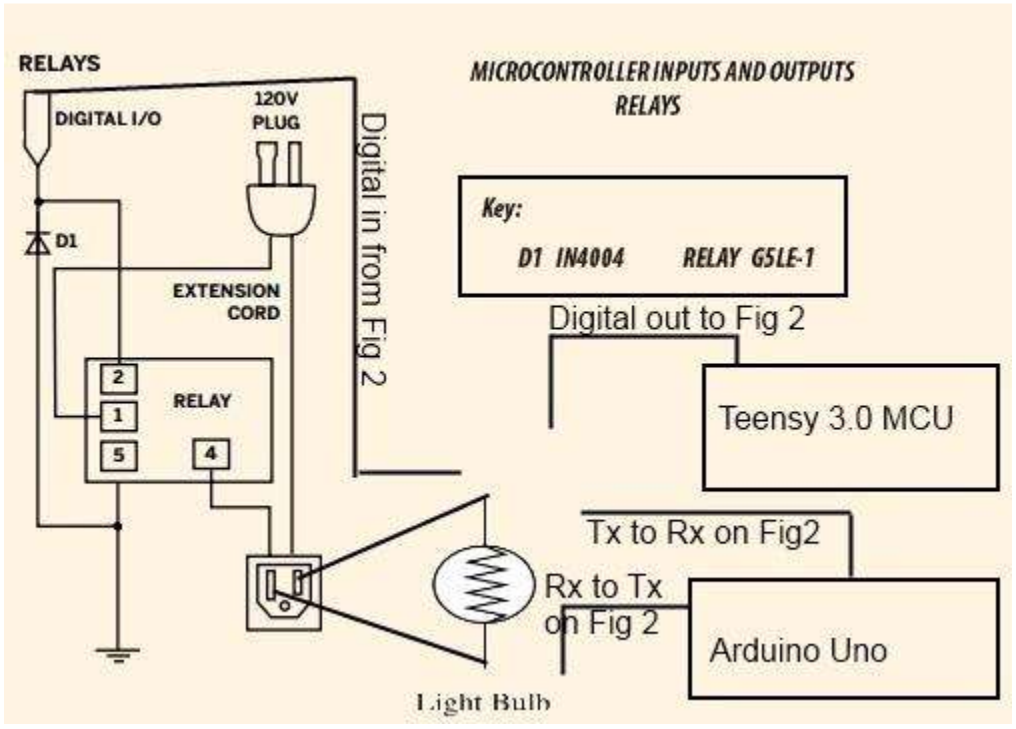

Figure 4.

OUTLET CONTROL

A major concern with homes is energy usage. Many people wonder, where does all of the electricity I use go? They often receive a higher power bills than expected and wonder why it has changed so much from last month. We wondered, how do we monitor which appliances eat energy and which are the most efficient? It occurred to us to gauge power usage of individual sockets. With real time data, we could know which kitchen appliances use the most power, which computer in my house draws the least power, and when the coffee maker or the toaster are left on, they can be shut off from a smartphone. A power strip, we decided, would be the ideal method for developing this technology. It is both 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. Power strips will enable a homeowner 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 Arduino calculates the high and low peaks of the signal. From the peak values 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.

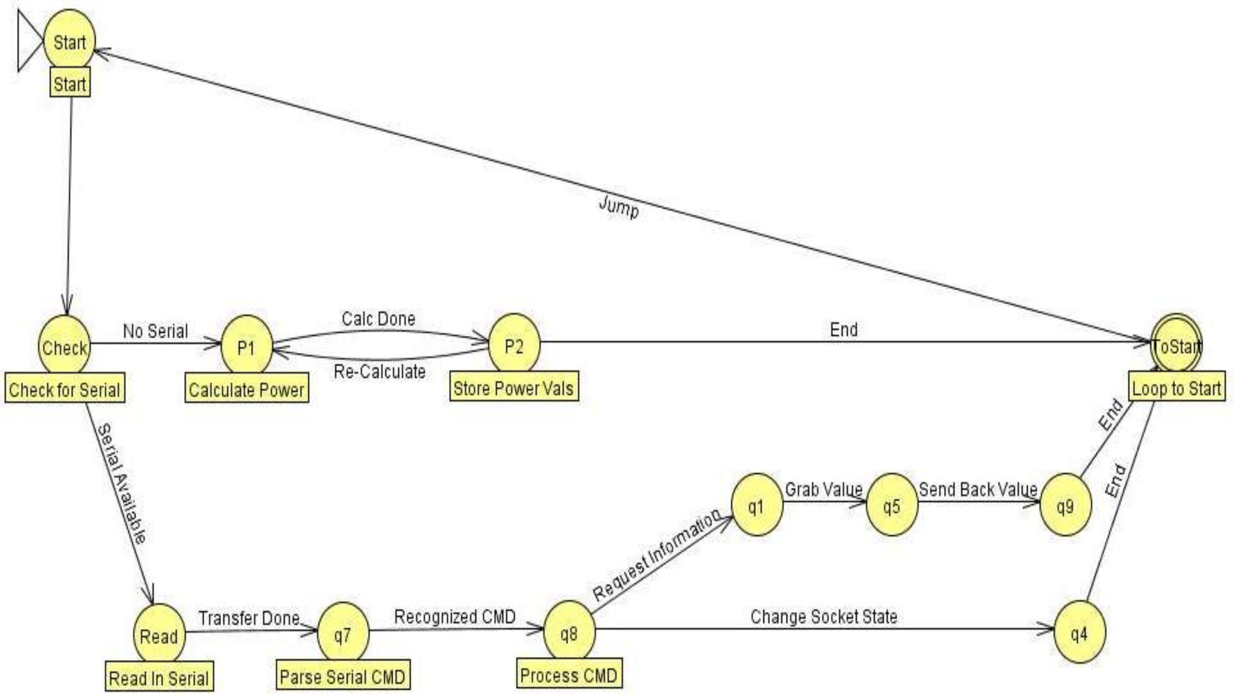

Figure 5

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, Wi-Fi 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 overcurrent 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 Wi-Fi board.

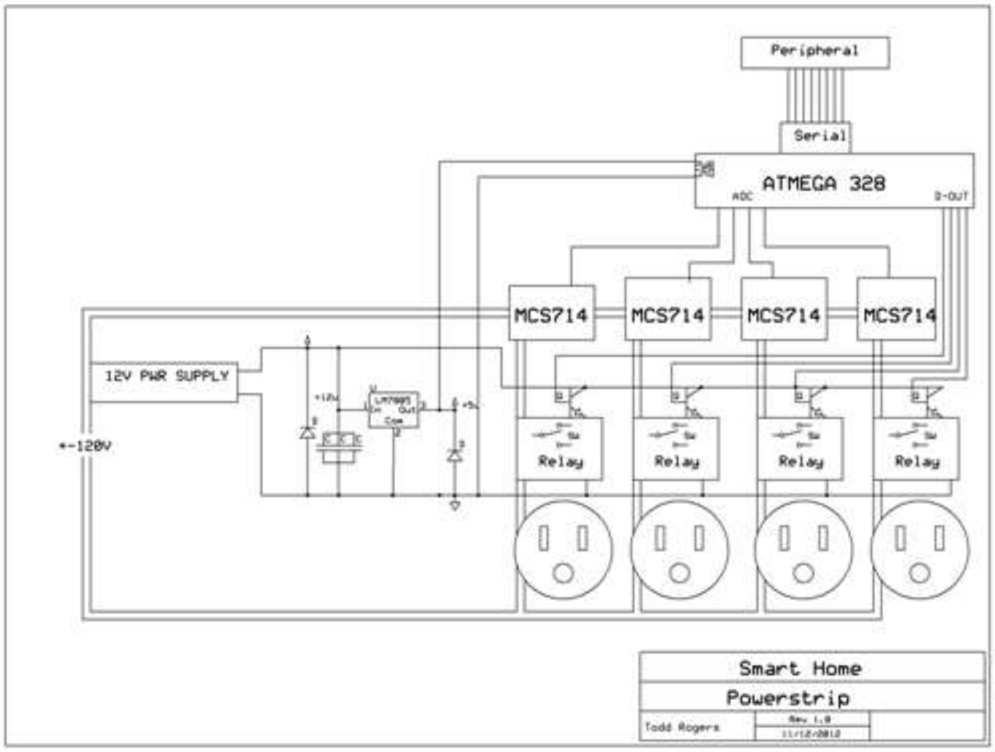

Figure 6

LIGHT AND APPLIANCE CONTROL

A home’s illumination is usually prime to automate. People want their lights to turn on when it’s dark, when an area is occupied, and sometimes during scheduled times. They want lights and certain non-essential appliances to turn off when not in use. 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 to a fixed schedule, which is often called for with exterior lighting, an innovative pursuit 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 countdown timer.

When motion is sensed, the lights are turned on, and a countdown 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 method of determining occupancy. 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. 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, although it cannot decipher the direction of the movement.

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. Once 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.

Besides the motion sensor and LBID sensor, there must be a timer and a logic implementation to combine the sensor inputs into a single output representing the occupancy state of a room or area. While the timer and logic may be implemented with discrete components, it is easy and flexible to create them using software and a processor. We originally conceived for this timer-logic combination to run on a PC as a part of our custom server software we named Mirabilis. However, the communications constraints that emerged in the Aduino WiFi component that acted as the ferrier of data between the server and the occupancy sensors necessitated a shift to using a microcontroller to implement the timer and logic.

Software

After dropping the idea of running the occupancy detection software on the PC, the software was written in C and executed on the Teensy 2.0 microcontroller. The timer was implemented as a loop of a finite number of iterations, fixed as a constant at compile time, as a clock/timer was not included as a hardware feature of the Teensy.

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.

An “entity” represents any atomic thing with a single state. A device like the power strip we built has multiple entities. The ON/OFF state of one outlet plug is a single entity. The power draw on that same plug is a separate entity. A device is defined as a component of the home automation network that has its own communication hardware.

As Mirabilis communicates with devices, its first duty is to maintain a tally of the state of all entities throughout the home. Its second and related function is to execute Python functions when events are triggered, both in response to changes in state of entities as well as in response to chronological events, such as timers and alarms. In our demonstration system, the event-handling software was omitted due to communication limitations of the Arduino WiFi as well as to concentrate on other features.

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 as Mirabilis and, in response to HTTP requests, queries the state of Mirabilis over operating system sockets to be able to display the state of the entities in the home. 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 the temperature is, from the changing resistance value of the thermistor (figure 7).

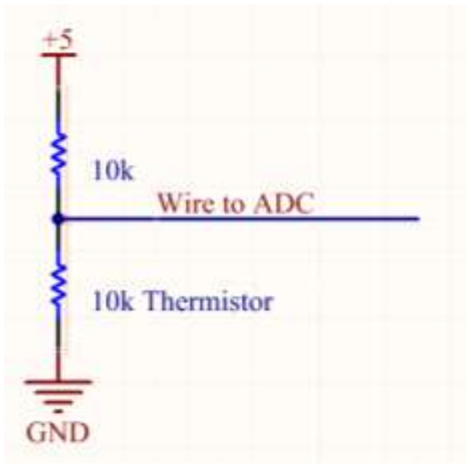

Figure 7

Reading a voltage can be done by connecting a wire between the two resistors directly to an ADC input on the MCU. This voltage will range from 5V to 2.5V. The ADC will be a 10 bit value that will indicate 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 used a 3 to 8 analog channel demux (CD4051). 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 (figure 8).

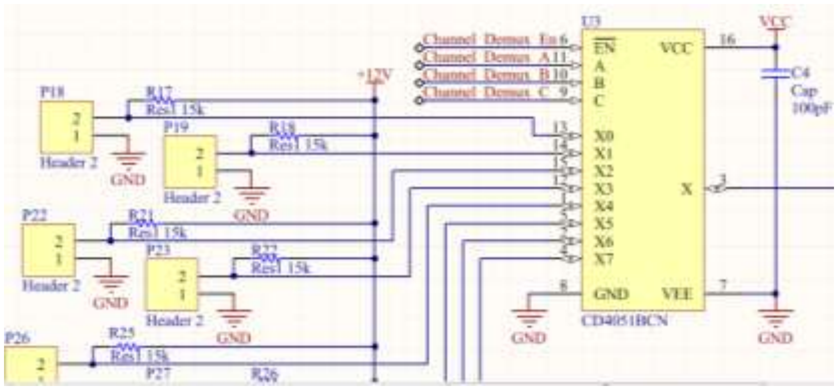

Figure 8

BLOWERS

The Air vent blowers assist airflow where it is needed most. When the furnace is at one end of a home, it will cause a loss in air pressure at the other end (figure 9).

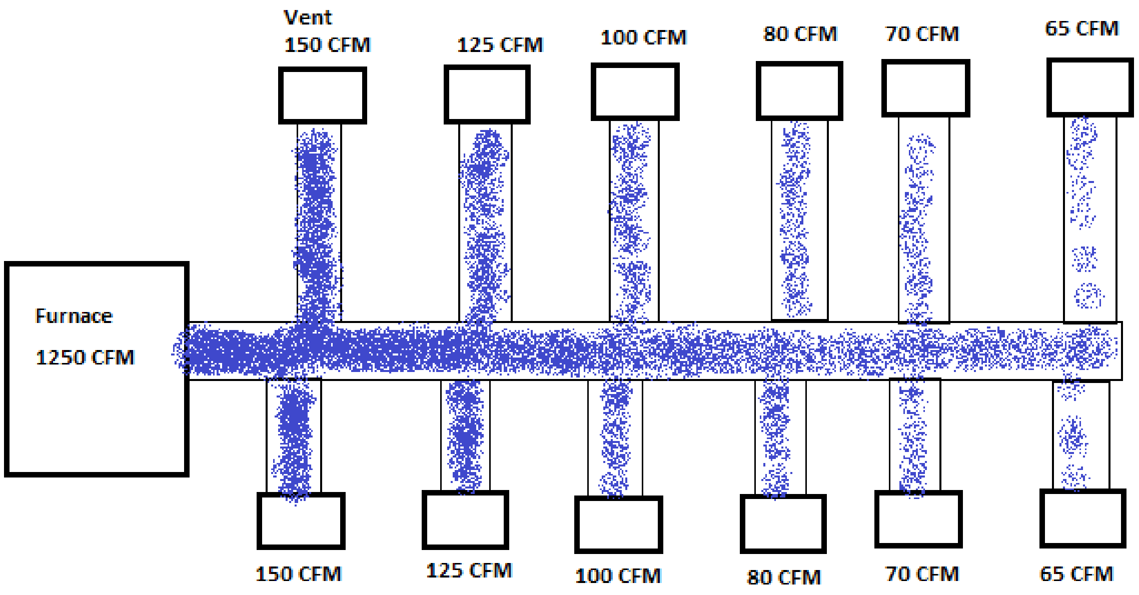

Figure 9

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 restricted. A house with a 1300 CFM furnace blower, and 14 vents, will blow 100 CFM per vent, +/- 20 CFM due to air pressure. To help control the airflow to different parts of the house, we employ 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 as a fireplace wood burning blower, exercise fan, and other needs. The fans require 1.5 amps; 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.

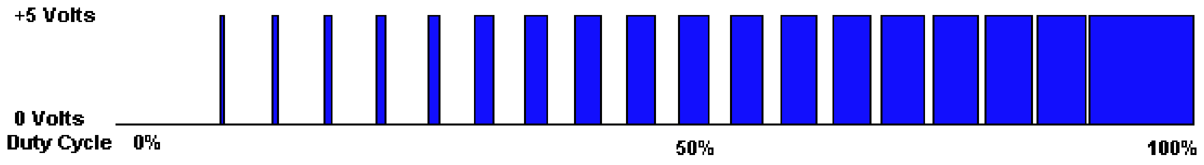

Figure 10 PWM Datadog Systems www.datadog.com/PWM_tutorial.pdf

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 color coded as: G (fan), W (heat), Y (cool), R (24VAC) (figure 11).

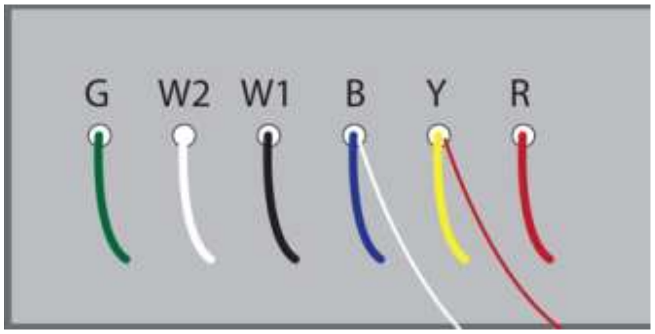

Figure 11

http://z.about.com/w/experts/Heating-Air-Conditioning-696/2010/02/wiring-diagram_4.png

The HVAC system will turn on when one or two of these wires are connected to 24VAC. When they aren't connected to 24VAC they are floating. In order to determine whether the lines are on or off we used optocoulpers. The optocoulpers allows a 24VAC signal to drive light into a phototransistor allowing a varying voltage AC signal to be used like a DC signal. 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 clear signal. The signal then goes through an inverter either being a positive 5v signal or 0v. This input goes into the Teensy++ to determine whether it is on or off (figure 12)

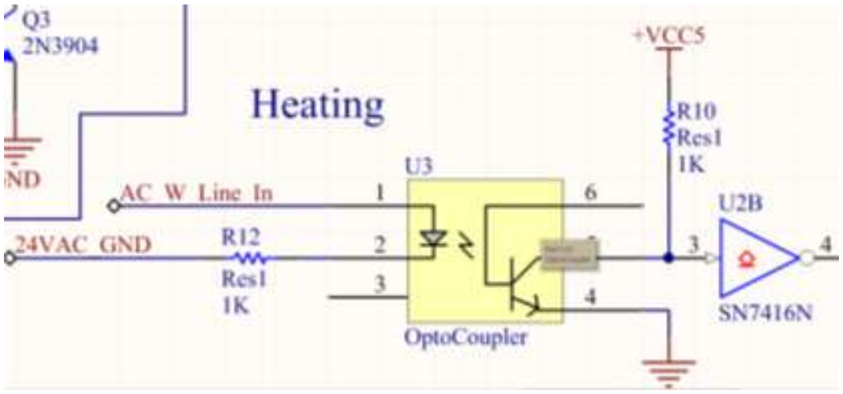

Figure 12

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 (figure 13).

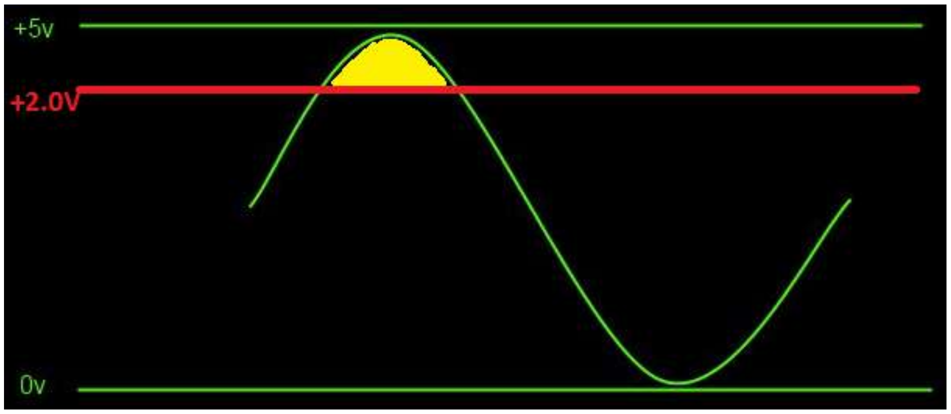

Figure 12

The control of the HVAC from the server will provide controls to allow the thermostat to 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. In order to turn off the HVAC remotely we have to interrupt the signal. To interrupt the signals you disconnect the HVAC signal path. If you want to turn on the furnace blower, you short the signal line, via a relay, to 24VAC. Similarly you can turn on the AC and Heating. We control the relays using a general purpose I/O (figure 13).

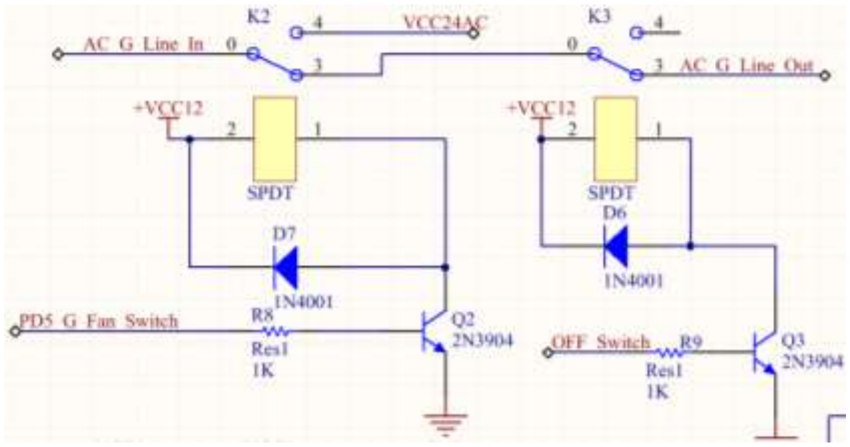

Figure 13

AIR FLOW CONTROL

We are using a Pololu bi-polar stepper motor driver to drive the stepper motors. The stepper motors are used to drive dampers through the homes vents, controlling the direction of airflow. This allows users to specify a certain room temperature. To control the motor driver you use two lines, one for motor stepping and the other for direction.

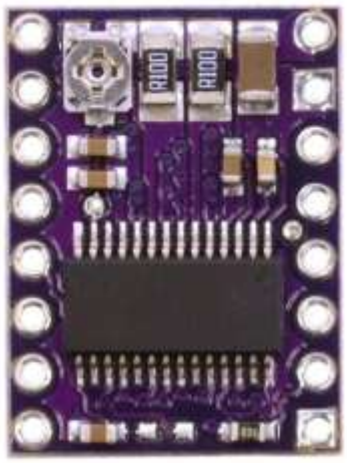

Figure 14

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

The motor drivers are able to source up to 1.5 amps. This provides ample torque to turn the stepper motor under pressure. In order to stop the damper at specific locations, we set a trigger at the open and closed points. The trigger signals to the Teensy++ to stop the motor.

In order to control multiple dampers we created a high current bus. The Teensy++ uses 4 signals to control a 4 to 16 demux chip (SN54154). 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.

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.

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.

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.

GARAGE CONTROL

The garage door control is a simple switch. Using a relay imitates the switch. The relay is controlled by a Teensy++ digital output (figure 15)

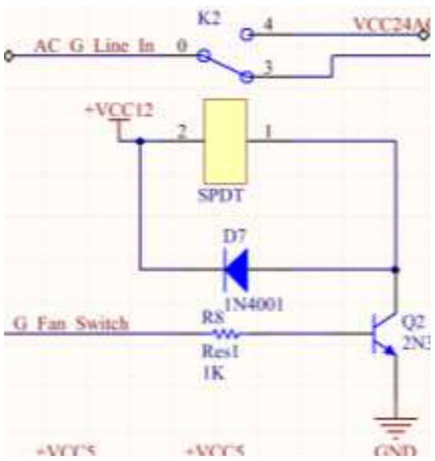

Figure 15

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 (figure x.x). Since you only need one station on at a time. We control 6 stations by using a 3 to 8 demux (74AC138).

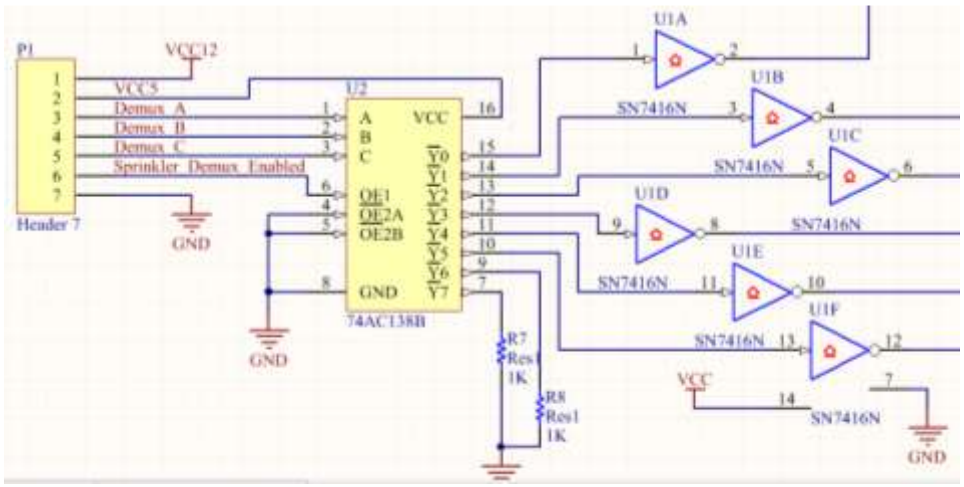

Figure 15

BILL OF MATERIALS

WIFI

Part List	**Wi-Fi & Sprinkler system**
Source Data From	Sprinkler Control.SchDoc
Project:	Wi-Fi & Sprinkler Control
Variant:	U of U Computer Engineering
Report Date:	7/13/2012 16:30
Print Date:	11/2/2012 16:31

Description	**Vendor**	**Comment**	**Value**	**Quant**	**Price**	**Total**
Arduino Duemilanove	Sparkfun Electronics	Atmega328 microcontroler	Atmega328	1	45	$45
WiShield for Arduino	Async_labs	WiShield 2.0 - Arduino Shield	WiShield 2.0	1	55	$55
Omron G5LE-1 Relay	Omron	Relay to turn on the sprinkler	G5LE-1	1	5.99	$5.99
1N4004 diode	RadioShack	Diode	Micro 1-Amp	2	1.19	2.38
Photocell	RadioShack	Photocel for time of day	N/A	1	2.99	2.99
22-gauge wire solid core	RadioShack	Wire to connect relays to power	Wire	10 ft	0.5	4.99
10k resistors	RadioShack	Resistors	N/A	5	0.5	1.5
AC extension cord	K-Mart	Extension cord for water pump	N/A	1	5.99	5.99
NetGear Wireless Router	NetGear	Wi-Fi router for connectivity	Wireless N	1	59.99	59.99
Soil Moisture Sensor	DFRobot	Sensor for measuring moisture	SEN0114	1	4.8	4.8
Total:			$189			

Outlet Box

Item	Quantity	Total Price

Polulu current sensors	4	$28
Outlets	4	$8
Arduino uno chip	1	$5
FDTI usb to uart board	1	$10
Automotive relays	4	$12

Epic Cube, or HVAC, Garage, Sprinkler, Motor driver, Damper bus

LESSONS LEARNED

Organization is important, having a set standard that everyone agrees on will allow for better processes. Triple check datasheets, several mistakes were due to not checking.

The project as a whole was relatively inexpensive. One of the goals was to try and see how much low level work can be done without outsourcing. In process a great deal of mistakes were made, from selecting the wrong chips to developing the wrong PCB. However it was extremely helpful to learn how to build a system from just chips.

Using schematics and source control were essential; we could have been pretty badly hurt if we didn't have a source control system. Schematics made it so much easier to see what was going on, and to debug any issues. Without it, the difficulty of trying to know what is going on is too great.

CONCLUSION

Are goal was to implement home automation at a lower cost than typical modern systems. To build our system cost approximately $1000. The market equivalent is $2000, or more depending on the vendor and system sophistication. It is easy to expand this project to control addition features to include, voice control, lighting control, speaker control, media control, and more.

REFERENCES

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

Steven, "How do you trigger the hvac with wiring only?"; http://diy.stackexchange.com/questions/14314/how-do-you-trigger-the-hvac-with-wiring-only; 2012

"20 Best PCB Making techniques" http://www.ingeniumblog.net/2010/06/20-best-pcb-making-techniques/; May 2012 no longer valid

Rob Paisley, "Bi Polar Stepper motor circuit"; http://home.cogeco.ca/~rpaisley4/Stepper2012Bipolar.html December 2012

"AVR Code Examples"; http://kartikmohta.com/tech/avr/tutorial/ December 2012

Kaizer Power Electronics; "Transistor base resistor calculator"; http://kaizerpowerelectronics.dk/calculators/transistor-base-resistor-calculator/

Webbot; "PWM tutorial"; http://www.societyofrobots.com/member_tutorials/node/233; December 3, 2008

QEEWiki; "PWM On The ATmega328"; https://sites.google.com/site/qeewiki/books/avr-guide/pwm-on-the-atmega328; December 2012

ACKNOWLEDGEMENT

A special thanks goes to our loved ones in supporting us through all our geeky obsessions. And to Jen Balling, in assisting with paper editing, and her recently electronic technician skills. ☺

APPENDIXES

All our code, hardware, and documentation can be pull from the following public GIT repository.
git@github.com:AshitakaLax/SmartHome.git
Schematics

Appendix A - Wifi Arduino Duemilanove Code

Code is located on GIT Repository under   repository under folder
/ArduinoCodeWifiServer/CompletedCodeFunctional/CompletedCodeFunctional.ini

Appendix B - Server Software

Appendix C - Teensy Controller Software

C.0 - Makefile Software Code
Compiles software for the Teensy++ 2.0 and all the devices it controls.

C.1 - Global Variables Code(GlobalVar.h)
Declares global variables
C.2 - MainController Code(mainController.c)
Controls serial input, and executes commands from the serial input, as well as provides a simple terminal interface.

C.3 - Damper Control Code
C.3.0 - Damper Control Header (damper_Control.h)
Declares damper variables and functions

C.1.1 - Damper Control Source(damper_Control.c)
Controls the interface to all the Dampers, motor controls, and triggers.

C.2 - Fan Control Software Code
C.2.0 - Fans Control Header (fans.h)
Declares vairiables and functions for the PWM fans.
C.2.1 - Fans Control Source (fans.c)
Generates the PWM signals to the fans, to control their speed on a scale from 0 – 9

C.5 - HVACGarage Software Code

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

Declares the variables for heating, cooling, and fan.

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

Controls the HVAC by reading in where the HVAC is already on, and changing output to control them.

C.6 - PWM Control Software Code

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

Variables for generating a PWM signal

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

Simplify code to generate a PWM signal.

C.7 - Sprinkler Control Software Code

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

Declares variables and function for sprinkler controls

C.7.1 - SprinklerControl Source(Sprinkler.c)

Controls Teensy++ 2.0 to send signals to 3 to 8 demux chip.

C.8 - Temperature Sensing Control Software Code

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

Declares variables and functions for Temp control

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

Reads in analog voltages from temperatures, while setting the analog demux.

C.9- USB Serial Control Software Code

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

Declares variables, and headers for USB serial communication

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

Allowes USB communication between Teensy ++ 2.0 and computer.