**Frank’s Corner**:

**Remote IO Links (from PooBah)**

[**SX1509 16 Output I/O Expander Breakout (I2C) $9.95**](http://www.karlssonrobotics.com/cart/sx1509-16-output-io-expander-breakout/?gclid=CjkKEQjwrLSdBRDYvIL0soO4vo0BEiQABALkqdLwJfl4CIKAif2BmkBU4dfvBVZZbV6_j3P0SjdF2Arw_wcB)

[**RS485 for remote access**](http://real2electronics.blogspot.com/2009/09/arduino-and-rs485-english-version.html)[**Part on Ebay**](http://www.ebay.com/itm/1x-MAX485-RS-485-TTL-to-RS-485-Module-converter-For-Arduino-A893-/251542343766?pt=LH_DefaultDomain_0&hash=item3a91179056)

[**Breakout Board for PCF8575 I2C Expander $12**](https://www.sparkfun.com/products/8130)

[**Digital and Analog IO Expander Shield $19.95**](http://numato.com/digital-and-analog-io-expander-shield)

[**i2c IO Expander $1.90**](http://arduino-related.livejournal.com/1591.html)

[**24Ch Analog i2C Expander Shield for Arduino by BlueberryDe.com**](http://www.techhelpblog.com/2012/06/21/24ch-analog-expander/)

**The links below demonstrate the Zumo Collision Detect program.**

The program is loaded into Arduino using the Arduino IDE.. The robot is then

detached from the desktop or laptop computer. The Zumo Robot will then execute the program loaded on the Arduino Uno. Each robot searches the computer for adversary robots. Once an adversary robot is detected, the detecting robot will accelerate and try to push

the other robot off the board. Each robot is programmed to stay in the circle. When

the white border is detected, each robot will execute a right or left turn to stay on the

board. Acceleration is detected when there is a sharp decline in acceleration. The robot

has an accelerometer feature to detect this condition.

[First Round Combat](http://www.bdpa-camp.org/zumorobot.html)

[3 robot combat](http://www.bdpa-camp.org/Zumo3Robot.html)

**Windows Phone Application Demo**

A Phone Application is simply a program or application that runs on the hardware available for windows phone. The Operating System is of course Windows 7 or 8. Applications are developed using Visual Studio Express 2010 for Windows Phone. The links below contain a zip file named

[**Visual Studio**](http://drive.google.com/open?id=0ByAPxK8Ic0qtWHV2SlNnbnhFY1U) which contain the setup files for both Visual Basic Express 2010 and Visual Studio Express for Windows Phone. This software must be installed before the actual application is created.

A second file named [**Basketball Phone App**](http://drive.google.com/open?id=0ByAPxK8Ic0qtV2VWdXBVaFlMSzQ) contains the User Interface Layout as well as the code for the application. The contents of this zip file will be used to demonstrate how a simple windows phone application is created.

[Visual Studio](http://drive.google.com/open?id=0ByAPxK8Ic0qtWHV2SlNnbnhFY1U)

[Basketball Phone App](http://drive.google.com/open?id=0ByAPxK8Ic0qtV2VWdXBVaFlMSzQ)

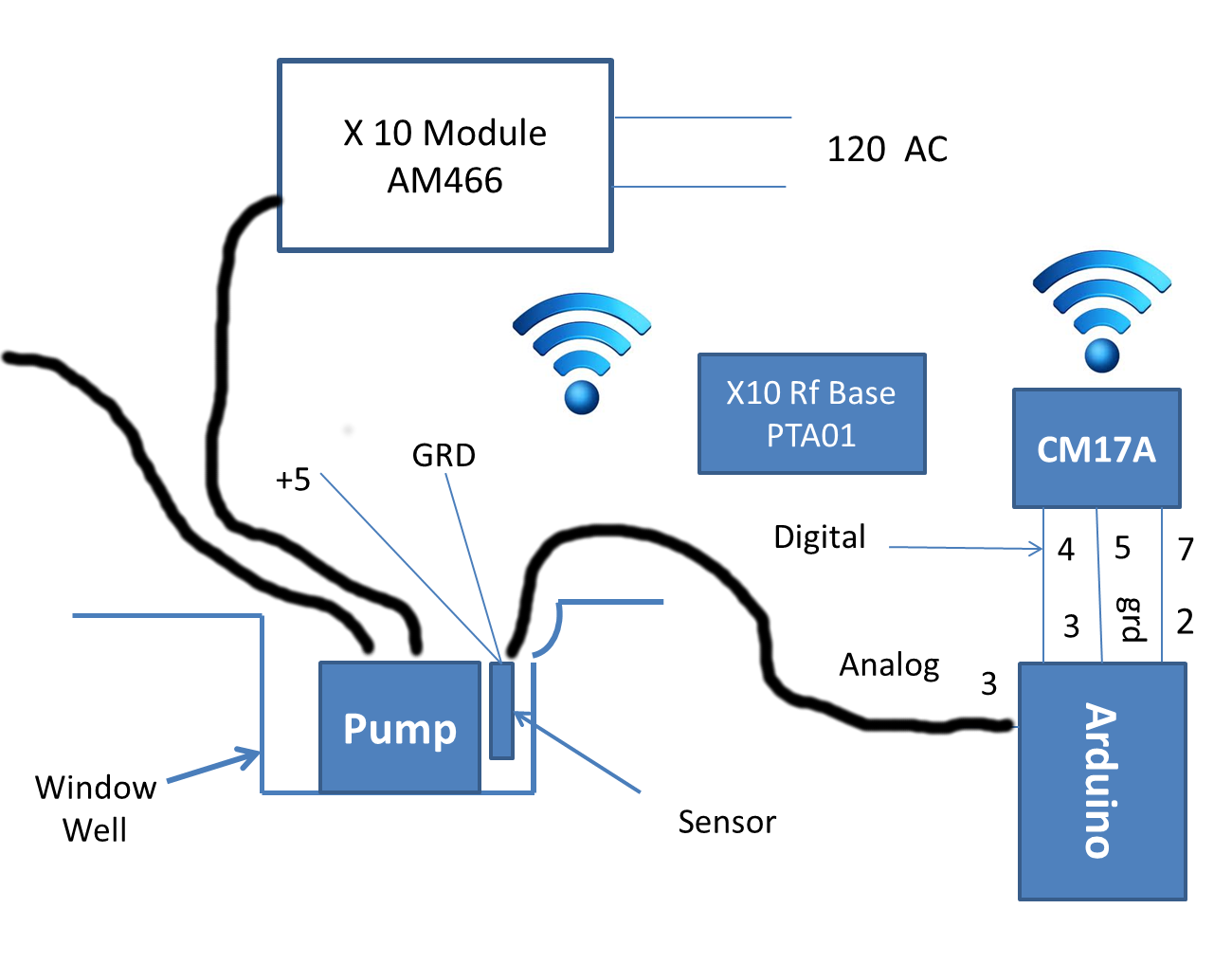
[Window Well Pump](http://drive.google.com/open?id=0ByAPxK8Ic0qtYlU4ODM0THlGY0U)

**Pump Application Design**

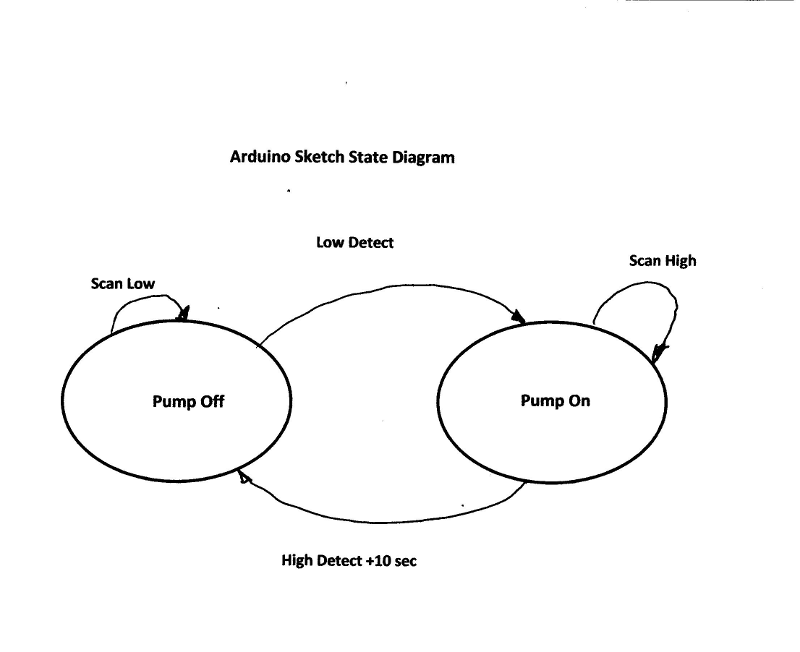
**The problem:** A pump has been installed in a window well to pump water from the window well during flood conditions. This condition occurs rarely, however, when it does occur, water gets into the rear of the basement and clean-up is costly and time consuming.

The pump will be turned on and of using X10 technology. An Arduino UNO will be used to detect the presence of water (sensor in well) and turn-on the pump using a CM17A X10 remote. The CM17A will send a signal to the X10 RF Base PAT01 and the PAT01 will then send an on/off signal to the X10 Module AM466. The AM466 can switch 500 Watts. The Arduino will scan the water sensor to determine when the water level drops below the set level. The Arduino will generate an additional 10 second time-out to make sure the window well is empty. The pump will be plugged into an X10 Appliance device.

**Functional Diagram for Automatic Well Pump Application**

****

The diagram below shows the state diagram for the software that will control the pump using the Arduino.

****

**/\***

**Overview**

**This circuit is designed to manage a submerged pump. It uses a**

**Water Sensor to detect the presence of water.**

**The analog input from the Arduino on pin 3 is connected to the sensor and analogRead()**

**is used to read the input. Pin 13 on the Arduino indicates whether the pump is on or off.**

**The circuit:**

**\* The water sensor attached to analog input 3**

**This version of the program also runs a test and prints the value of the analog input.**

**This input will vary from a low of 120 to a high of 1023. 1023 is an indicator that**

**water is not present and hasn't been present for some time. Values will vary between**

**1023 and 990 when the sensor is dry. Upon the initial transition to the wet state, the**

**sensor will return a value of 124 to 140. It may take a few minutes after water is removed**

**for the sensor to return to 1023. The sketch will use < 140 as a trigger to turn-on the pump.**

**A value above 200 > will turn it off again. The pump will run for at least 10 seconds after**

**being triggered.**

**Created by Frank Hill**

**\*/**

**#include <ArduinoX10>**

**#include <X10Firecracker.h>**

**int sensorPin = A3; // select the input pin for the water sensor**

**int ledPin = 13;**

**int sensorValue = 0; // initialize variable to store the value coming from the sensor**

**void setup() {**

**// declare the ledPin as an OUTPUT:**

**pinMode(ledPin, OUTPUT);**

**// Setup for X10 operation**

**X10.init( 2, 3, 0 );**

**Serial.begin(9600);**

**}**

**void loop() {**

**// read the value from the sensor:**

**sensorValue = analogRead(sensorPin);**

**if(sensorValue > 200) {**

**// enter X10 code for turning off the pump**

**X10.sendCmd( hcA, 1, cmdOff );**

**digitalWrite(ledPin, LOW);**

**}**

**if(sensorValue < 150) {**

**digitalWrite(ledPin, HIGH); // turn-on the pump**

**// enter X10 code for turning on the pump**

**X10.sendCmd( hcA, 1, cmdOn );**

**delay(10000); // Lead the pump running for 10 seconds to empty the well**

**}**

**Serial.print(sensorValue); // print as an ASCII-encoded decimal**

**Serial.print("\n");**

**// stop the program for for 500 milliseconds before next scan:**

**delay(500);**

**}**

**Test Application for X10 with Arduino**

**Remote Test Program**

#include <X10Firecracker.h>

void setup()

{

X10.init( 2, 3, 0 );

}

void Teston()

{

X10.sendCmd( hcA, 1, cmdOn );

}

void Testoff()

{

X10.sendCmd( hcA, 1, cmdOff );

}

void loop()

{

delay(8000);

Teston();

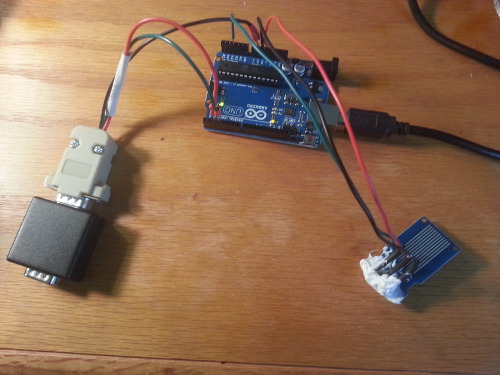
delay(8000);

Testoff();

}

**Photo of Water Detection and Remote X10 with Arduino Hardware**

**Arduino**

** Water Sensor**

**X10 Firecracker**

**Requirements and Design for Universal Home Artificial Intelligence (AI) Interface**

The goal of this system to provide an enhanced/efficient technology driven living environment. Enhancements in security, communications, entertainment and general improvements in home useability will be provided. The system will be implemented using the Computational Intelligence and Natural Language Processing AI capabilities.

The system will interact with users in the home in the same manner as the Computer in Star Trek. Wireless input sensors will be located in all parts of the house. These sensors will allow voice and some cases keypad input. **It will be important to know the location of sensors because there will be many of them and actions will depend on knowing the location**. The user should be able to speak in a normal voice to activate functions. The home AI system will execute actions and respond to the user within 2 seconds. Tentatively, the Google Voice Application will be use for decoding voice commands. There could be a problem using this mechanism due to performance characteristics of the Google API and due the fact that Google is said to be limiting access to third-party-developers.

In some cases the computer will request more information. Output devices are detailed below. The system will have the logic to receive and process multiple inputs simultaneously. This capability will be provided by a **Feature Control Function**. Feature Control will be based on Finite State Machines. This functionality will run from host processor. Commands will be sent from the host to the peripheral processor. Input from sensors will be stored on the peripheral processor.

The system will be capable of receiving and processing the following inputs:

1) Room Temperature

2) Motion

3) Moisture in basement

4) Loud Sounds

5) Light Level

6) Internet Status (connection/no connection)

7) TCP IP messages

8) Video Camera Stream Input

9) Home Entry Switches

10) Garage Door Transmitter

11) Doorbell

12) Web Client Interface

13) Microphones

14) Humidity

The system will be capable of performing the following output functions:

1) Open and Close Garage Door

2) Turn Selected Lights on and off

3) Activate and Deactivate Burglar Alarm

4) Send email

5) Send Tweet

6) Video Displays

7) Control Home Thermostat

8) Speakers

9) Manage Video Security Cameras

The system will have a master console interface to allow for software updates and configuration. Admin or Level 1 security will be required.

The system will have automatic diagnostics that can test both hardware and software interactions. Diagnostics can be set to run nightly and generate reports via email.

The system will have enough redundancy so that downtime will be on the order of 2 hours in 40 years. This is approximately the downtime of a T1 line.

The system will be able to deliver “how to instructions” for level 1, 2 and 3 security level users. Level 1 will be the highest most secure level. Level 2 will apply to members of the household and Level 3 will apply to guest.

The system will have the capability of routing video and audio inputs to various output displays as well as analyze the input for high level Feature Control functions.

The system shall be capable uploading data and programs to peripheral robots associated with the home.

The system shall have battery backup that will keep the system active for a minimum of 24 hours after a power failure.

Security Features

1) Voice

2) Fingerprint

3) Keypad digits

4) Encryption

The system will verify user access via items 1, 2 and 3 above.

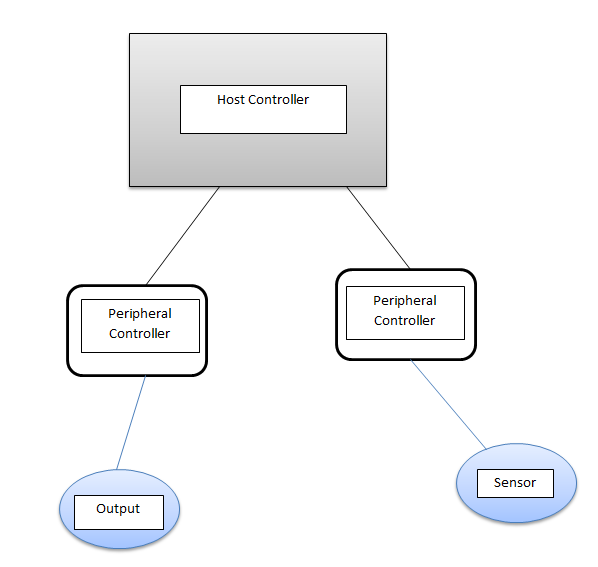
**Design Details for Home AI (HAI)**

The block diagram below shows the control structure for the HAI. The system will consist of a hierarchy of control functions. The Host Controller will act as the administrative processor and handle tasks such the Level 1 user interface and communication with the internet. These are slower functions and don’t require real-time response (<= 2 seconds). This processor will have access to 10 Gig of direct storage and remote access to 1 terabyte of data. The terabyte offline store will be used to store video and audio for entertainment purposes. The host processor will poll the peripheral processors every 500 milliseconds. The host processor will also be redundant for reliability. The processors will run mirrored and be synced within 100 ms. Both processors will have access to the peripheral processors. If one processor fails, the system will fail-over to the second processor. Wireless and Wired LAN technology will be used for processor communication. The infrastructure for lan communication and protocols exists with speeds up to 1 gigabit per second. This should be sufficient for HAI.

Peripheral Processors will be battery powered, they must therefore be very energy efficient. They must run for a single year without recharging.

The scan rate for the peripheral processors will be 50 ms. This can be lowered to save power if the slower scan rate does not impact performance. Each sensor will be scanned every 50 ms and output tasks can be processed in as fast as 50 ms. Tasks faster than 50 ms, must be performed through a hardware switching mechanism. That is, the processor will enable a connection and audio or video will flow through a physical switch. A good candidate for this function is the packet switching capability of a LAN router.

**System Architecture**



Webcams will be used to provide microphone sensors and motion detection as well as recorded video. Discrete devices will be used for temperature and moisture sensing. A system generated Ping will be used to validate network connectivity. Bluetooth will be used to relay status of garage door openers and to trigger door opening and closing. A X10 firecracker interface will be used to control light switches. Individual Wireless Speakers will be used for communication. This function will be controlled by the host processor. A wireless interface will be used to get status and activate the ADT burglar alarm system. Sending tweets and emails will be carried-out by the host processor. Scripts will be developed for this purpose.

Hardware Specifications

HAI will support both Windows Tablets and Android Tablets for the host processor. A low power version of Arduino will be used for peripheral processors. All bit manipulations will occur on the peripheral processors.

Software Specification

The operating system will handle admin functions written using scripting languages to provide the capability. The following interface functions will be required:

ScanHost(Type, Name, Number) return pointer to a struct containing results  
ParseResults(ptrStruct)  
CreateProc(FeatID)  
AddtoWorkQueue(ptrwrkOutput)

Environmental Considerations

Operating temperatures - 32f - 100f

Safety Considerations

Protections from lightning and electrical noise

Power Considerations  
  
Maintenance

Challenges

Low power requirements may affect cost

Initialization

Sanity check required on power-up. Set all subsystems to initial states

----------------------------------------------------------------------------------------------------------------

**20141204 Internet Of Things (IOT) Demo 12/4**

This demo is designed to reinforce the notion of IOT using real world objects.

It consists of a small website which resides of an Arduino UNO board. The website

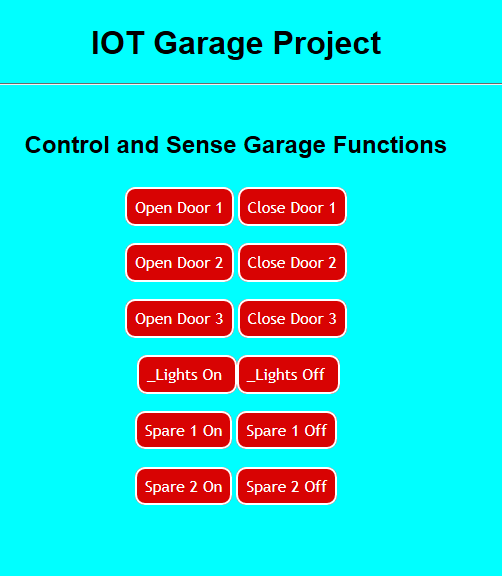
controls relays which in turn controls garage door openers and a light. The web interface is shown below.

12/18/2012 Great work Frank. The code worked!!!

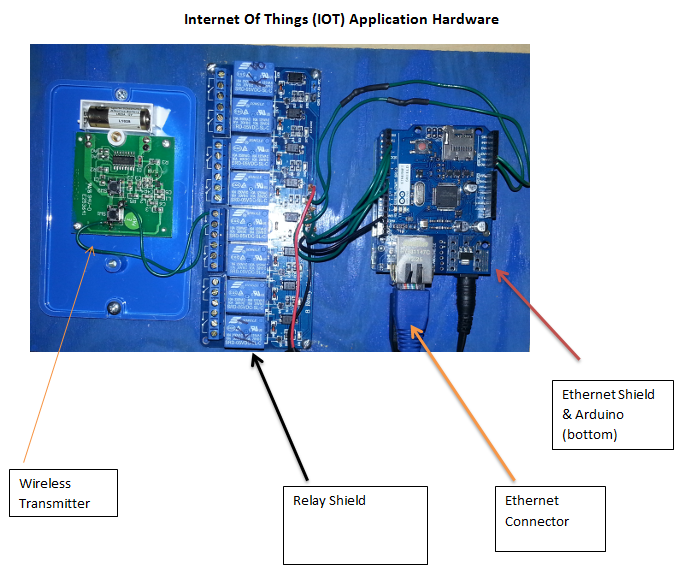
(I added two lines to get it to work on my system - updated code included below

Notes from Meeting (Grand Poo Bah):

There appears to be some sort of conflict between some 3rd party Ethernet shields and some UNOs such that the higher numbered discrete io points do not work. In this case analog IO was used instead of the higher numbered discrete IO.



The web interface controls the hardware configuration shown below.



The ethernet shield provides internet connectivity to the Arduino UNO and the UNO +responds to command from the website. Clicking the button will trigger events such as garage door up or garage door down. The Wireless transmitter controls an electrical outlet that has a string of christmas lights connected.

The software that controls the hardware is as follows:

/\*

Created by Frank Hill 11/23/2014

Inspired by Rui Santos, Leroy Miller, ...

\*/

#include <SPI.h>

#include <Ethernet.h>

int button1 = 2;

int button2 = 3;

int button3 = 4;

int button4 = 14;

int button5 = 15;

int button6 = 7;

// Enter a MAC address and IP address for your controller below.

// The IP address will be dependent on your local network:

byte mac[] = {

0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

IPAddress ip(192,168,1,177);

// Initialize the Ethernet server library

// with the IP address and port you want to use

// (port 80 is default for HTTP):

EthernetServer server(80);

String readString;

void setup() {

// Open serial communications and wait for port to open:

Serial.begin(9600);

while (!Serial) {

; // wait for serial port to connect. Needed for Leonardo only

}

pinMode(button1, OUTPUT);

pinMode(button2, OUTPUT);

pinMode(button3, OUTPUT);

pinMode(button4, OUTPUT);

pinMode(button5, OUTPUT);

pinMode(button6, OUTPUT);

//initialize, set all relays to off

digitalWrite(button1, HIGH);

digitalWrite(button2, HIGH);

digitalWrite(button3, HIGH);

digitalWrite(button4, HIGH);

digitalWrite(button5, HIGH);

digitalWrite(button6, HIGH);

// start the Ethernet connection and the server:

Ethernet.begin(mac, ip);

server.begin();

// Serial.print("server is at ");

// Serial.println(Ethernet.localIP());

}

void loop() {

// Create a client connection

EthernetClient client = server.available();

if (client) {

while (client.connected()) {

if (client.available()) {

char c = client.read();

//read char by char HTTP request

if (readString.length() < 100) {

//store characters to string

readString += c;

//Serial.print(c);

}

//if HTTP request has ended

if (c == '\n') {

// Serial.println(readString); //print to serial monitor for debugging

client.println("HTTP/1.1 200 OK"); //send new page

client.println("Content-Type: text/html");

client.println();

client.println("<HTML>");

client.println("<HEAD>");

client.println("<link rel='stylesheet' type='text/css' href='http://bdpa-camp.org/AndroidIOT.css' />");

client.println("<TITLE>IOT Garage Project</TITLE>");

client.println("</HEAD>");

client.println("<BODY>");

client.println("<H1>IOT Garage Project</H1>");

client.println("<hr />");

client.println("<br />");

client.println("<H2>Control and Sense Garage Functions</H2>");

client.println("<br />");

client.println("<a href=\"/?button1on\"\">Open Door 1</a>");

client.println("<a href=\"/?button1off\"\">Close Door 1</a><br />");

client.println("<br />");

client.println("<br />");

client.println("<a href=\"/?button2on\"\">Open Door 2</a>");

client.println("<a href=\"/?button2off\"\">Close Door 2</a><br />");

client.println("<br />");

client.println("<br />");

client.println("<a href=\"/?button3on\"\">Open Door 3</a>");

client.println("<a href=\"/?button3off\"\">Close Door 3</a><br />");

client.println("<br />");

client.println("<br />");

client.println("<a href=\"/?button4on\"\">\_Lights On </a>");

client.println("<a href=\"/?button4off\"\">\_Lights Off </a><br />");

client.println("<br />");

client.println("<br />");

client.println("<a href=\"/?button5on\"\"> Spare 1 On</a>");

client.println("<a href=\"/?button5off\"\">Spare 1 Off</a><br />");

client.println("<br />");

client.println("<br />");

client.println("<a href=\"/?button6on\"\"> Spare 2 On</a>");

client.println("<a href=\"/?button6off\"\">Spare 2 Off</a><br />");

client.println("<br />");

client.println("</BODY>");

client.println("</HTML>");

delay(1);

//stopping client

client.stop();

//controls the Arduino if you press the buttons

if (readString.indexOf("?button1on") >0){

// Pulse the relay for garage door

digitalWrite(button1, LOW);

delay(1000);

digitalWrite(button1, HIGH);

}

if (readString.indexOf("?button1off") >0){

// Pulse the relay for garage door

digitalWrite(button1, LOW);

delay(1000);

digitalWrite(button1, HIGH);

}

if (readString.indexOf("?button2on") >0){

// Pulse the relay for garage door

digitalWrite(button2, LOW);

delay(1000);

digitalWrite(button2, HIGH);

}

if (readString.indexOf("?button2off") >0){

// Pulse the relay for garage door

digitalWrite(button2, LOW);

delay(1000);

digitalWrite(button2, HIGH);

}

if (readString.indexOf("?button3on") >0){

// Pulse the relay for garage door

digitalWrite(button3, LOW);

delay(1000);

digitalWrite(button3, HIGH);

}

if (readString.indexOf("?button3off") >0){

// Pulse the relay for garage door

digitalWrite(button3, LOW);

delay(1000);

digitalWrite(button3, HIGH);

}

if (readString.indexOf("?button4on") >0){

digitalWrite(button4, LOW);

delay(1000);

digitalWrite(button4, HIGH);

}

if (readString.indexOf("?button4off") >0){

digitalWrite(button4, LOW);

delay(1000);

digitalWrite(button4, HIGH);

}

if (readString.indexOf("?button5on") >0){

digitalWrite(button5, LOW);

}

if (readString.indexOf("?button5off") >0){

digitalWrite(button5, HIGH);

}

if (readString.indexOf("?button6on") >0){

digitalWrite(button6, LOW);

}

if (readString.indexOf("?button6off") >0){

digitalWrite(button6, HIGH);

}

//clearing string for next read

readString="";

}

}

}

}

}

-----------------------------------------------------------------------------------------------------------------

Grand Poo Baa

Upate - I had to add two lines to get mine to work (lines 2 and 3)

#include <Ethernet.h>

#include <SPI.h> // added jgm

#include <w1500.h> // added jgm

int button1 = 2;

int button2 = 3;

int button3 = 4;

int button4 = 14;

int button5 = 15;

int button6 = 7;

// Enter a MAC address and IP address for your controller below.

// The IP address will be dependent on your local network:

byte mac[] = {

0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

IPAddress ip(192,168,1,177);

// Initialize the Ethernet server library

// with the IP address and port you want to use

// (port 80 is default for HTTP):

EthernetServer server(80);

String readString;

void setup() {

// Open serial communications and wait for port to open:

Serial.begin(9600);

while (!Serial) {

; // wait for serial port to connect. Needed for Leonardo only

}

pinMode(button1, OUTPUT);

pinMode(button2, OUTPUT);

pinMode(button3, OUTPUT);

pinMode(button4, OUTPUT);

pinMode(button5, OUTPUT);

pinMode(button6, OUTPUT);

//initialize, set all relays to off

digitalWrite(button1, HIGH);

digitalWrite(button2, HIGH);

digitalWrite(button3, HIGH);

digitalWrite(button4, HIGH);

digitalWrite(button5, HIGH);

digitalWrite(button6, HIGH);

// start the Ethernet connection and the server:

Ethernet.begin(mac, ip);

server.begin();

// Serial.print("server is at ");

// Serial.println(Ethernet.localIP());

}

void loop() {

// Create a client connection

EthernetClient client = server.available();

if (client) {

while (client.connected()) {

if (client.available()) {

char c = client.read();

//read char by char HTTP request

if (readString.length() < 100) {

//store characters to string

readString += c;

//Serial.print(c);

}

//if HTTP request has ended

if (c == '\n') {

// Serial.println(readString); //print to serial monitor for debugging

client.println("HTTP/1.1 200 OK"); //send new page

client.println("Content-Type: text/html");

client.println();

client.println("<HTML>");

client.println("<HEAD>");

client.println("<link rel='stylesheet' type='text/css' href='http://bdpa-camp.org/AndroidIOT.css' />");

client.println("<TITLE>IOT Garage Project</TITLE>");

client.println("</HEAD>");

client.println("<BODY>");

client.println("<H1>IOT Garage Project</H1>");

client.println("<hr />");

client.println("<br />");

client.println("<H2>Control and Sense Garage Functions</H2>");

client.println("<br />");

client.println("<a href=\"/?button1on\"\">Open Door 1</a>");

client.println("<a href=\"/?button1off\"\">Close Door 1</a><br />");

client.println("<br />");

client.println("<br />");

client.println("<a href=\"/?button2on\"\">Open Door 2</a>");

client.println("<a href=\"/?button2off\"\">Close Door 2</a><br />");

client.println("<br />");

client.println("<br />");

client.println("<a href=\"/?button3on\"\">Open Door 3</a>");

client.println("<a href=\"/?button3off\"\">Close Door 3</a><br />");

client.println("<br />");

client.println("<br />");

client.println("<a href=\"/?button4on\"\">\_Lights On </a>");

client.println("<a href=\"/?button4off\"\">\_Lights Off </a><br />");

client.println("<br />");

client.println("<br />");

client.println("<a href=\"/?button5on\"\"> Spare 1 On</a>");

client.println("<a href=\"/?button5off\"\">Spare 1 Off</a><br />");

client.println("<br />");

client.println("<br />");

client.println("<a href=\"/?button6on\"\"> Spare 2 On</a>");

client.println("<a href=\"/?button6off\"\">Spare 2 Off</a><br />");

client.println("<br />");

client.println("</BODY>");

client.println("</HTML>");

delay(1);

//stopping client

client.stop();

//controls the Arduino if you press the buttons

if (readString.indexOf("?button1on") >0){

// Pulse the relay for garage door

digitalWrite(button1, LOW);

delay(1000);

digitalWrite(button1, HIGH);

}

if (readString.indexOf("?button1off") >0){

// Pulse the relay for garage door

digitalWrite(button1, LOW);

delay(1000);

digitalWrite(button1, HIGH);

}

if (readString.indexOf("?button2on") >0){

// Pulse the relay for garage door

digitalWrite(button2, LOW);

delay(1000);

digitalWrite(button2, HIGH);

}

if (readString.indexOf("?button2off") >0){

// Pulse the relay for garage door

digitalWrite(button2, LOW);

delay(1000);

digitalWrite(button2, HIGH);

}

if (readString.indexOf("?button3on") >0){

// Pulse the relay for garage door

digitalWrite(button3, LOW);

delay(1000);

digitalWrite(button3, HIGH);

}

if (readString.indexOf("?button3off") >0){

// Pulse the relay for garage door

digitalWrite(button3, LOW);

delay(1000);

digitalWrite(button3, HIGH);

}

if (readString.indexOf("?button4on") >0){

digitalWrite(button4, LOW);

delay(1000);

digitalWrite(button4, HIGH);

}

if (readString.indexOf("?button4off") >0){

digitalWrite(button4, LOW);

delay(1000);

digitalWrite(button4, HIGH);

}

if (readString.indexOf("?button5on") >0){

digitalWrite(button5, LOW);

}

if (readString.indexOf("?button5off") >0){

digitalWrite(button5, HIGH);

}

if (readString.indexOf("?button6on") >0){

digitalWrite(button6, LOW);

}

if (readString.indexOf("?button6off") >0){

digitalWrite(button6, HIGH);

}

//clearing string for next read

readString="";

}

}

}

}

}

Frank’s multitask class using LED bar from Grand Poobah

// This file is a clone of portions of the Adafruit Tutorial 1 Mod by Frank Hill

// It demonstrates multitasking using the LED strips provided by the Grand Poo-Bah. This code template eliminates the need for the delay() function.

// The basic formula procedure for multi-tasking using the Arduino is

// 1) Create a class for each distinct object that you will be running on Arduino ie flashing lights, robot motors, buzzers ...

// 2) Create an update function for each object

// 3) Creach a Constructor so that you can instantiate (make the object run in ram) each object

// 4) Use the Arduino Loop to control and sequence the updates. The update function provides timing and allows each object time with the cpu

class Flasher

{

// Class Member Variables

// These are initialized at startup

int ledPin; // the number of the LED pin

long OnTime; // milliseconds of on-time

long OffTime; // milliseconds of off-time

// These maintain the current state

int ledState; // ledState used to set the LED

unsigned long previousMillis; // will store last time LED was updated

// Constructor - creates a Flasher

// and initializes the member variables and state

public:

Flasher(int pin, long on, long off)

{

ledPin = pin;

pinMode(ledPin, OUTPUT);

OnTime = on;

OffTime = off;

ledState = LOW;

previousMillis = 0;

}

void Update()

{

// check to see if it's time to change the state of the LED

unsigned long currentMillis = millis();

if((ledState == HIGH) && (currentMillis - previousMillis >= OnTime))

{

ledState = LOW; // Turn it off

previousMillis = currentMillis; // Remember the time

digitalWrite(ledPin, ledState); // Update the actual LED

}

else if ((ledState == LOW) && (currentMillis - previousMillis >= OffTime))

{

ledState = HIGH; // turn it on

previousMillis = currentMillis; // Remember the time

digitalWrite(ledPin, ledState); // Update the actual LED

}

}

};

Flasher led1(8, 1000, 4000);

Flasher led2(9, 3500, 3500);

Flasher led3(10, 4500, 4500);

Flasher led4(11, 5500, 5500);

Flasher led5(12, 6500, 6500);

Flasher led6(13, 8500, 8500);

void setup()

{

}

void loop()

{

led1.Update();

led2.Update();

led3.Update();

led4.Update();

led5.Update();

led6.Update();

}

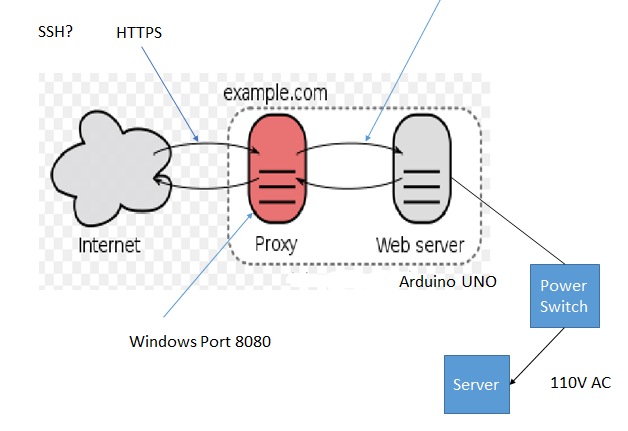
**Secure Power Cycle and Reboot Control (IOT)**

All server administrators periodically find it necessary to power cycle a server.

Increasingly operators find it necessary to complete the reboot task remotely. The Internet provides the ubiquitous medium to complete this process. This project will demonstrate a remote power switch capability with enhanced security. Security is currently a high priority problem with IOT applications. This project will display an encrypted interface using HTTPS as well as a SSH interface.

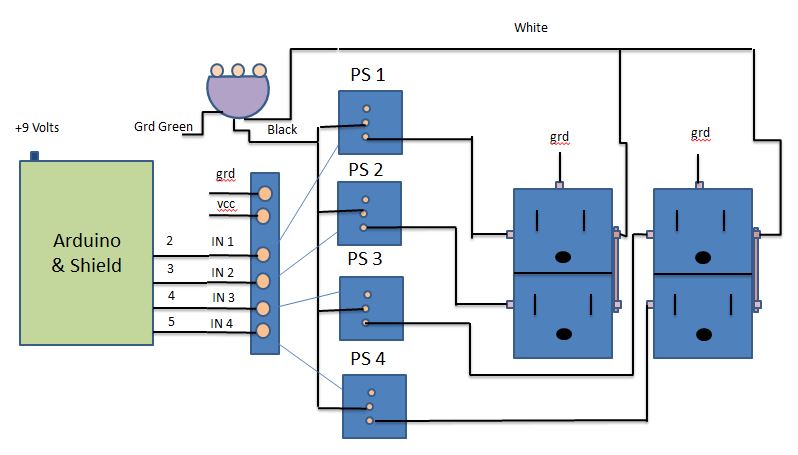
The goal is to keep the interface simple and provide tight secure coupling between the internet client and the actual power switch. As shown in the diagram below, access to the actual power switching circuit will be accomplished through a proxy server which will be connected to the internet. Only the proxy server will have access to the control software on the local network. A encrypted login and password will be used to access the switch through the proxy server. The web server will always make a keyed request for a security key within a 20 second timing window. The originating client needs to able to respond within that 20 second window. The response key will be based on the time-of-day. A combination of the login, password and response key will allow the power switch to be pulsed for 5 seconds. More than 3 failed attempts in a 2 minute interval will generate an automatic lockout for 30 minutes. The relays used for the switch will be Normally Closed (NC). A trigger from the controller will open the circuit for 5 seconds.

This project will use 4 separate power circuits.



**Circuit Diagram**

PS 1 to PS 4 are relays used to switch the hot AC connection. The relay normally closed connections are used in a manner that allows the circuit to be interrupted for a few seconds to interrupt power to a load. Regular 110 volt outlets are used. This circuit support 4 power circuits. the hot side of the outlet is split to allow for 2 separate connections on each outlet.

****

**Code for Power Cycle IOT Project - password added**

**// Added coding credits**

#include <SPI.h>

#include <Ethernet.h>

//ethernet setup

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED }; //physical mac address

IPAddress ip(192,168,1,177); // change to ip in lan

EthernetClient client; // Define a client object

EthernetServer server(80);

const int passCount = 20;

unsigned long passIDstart[passCount]; // array to record start time of user sessions

unsigned long passIDlast[passCount]; // array to record last time of user sessions

unsigned long passIDvalid = 0; // set default as passID : 0 = missing or invalid

unsigned long passExpireMil = 60000; // expiry time for automatic expiry of valid passID

String readString; // used to clear away incoming data that is not required

char myPass[] = "1234"; // Valid password (1234 for testing)

char DoAction[] = "DoAction"; // keyword for DoAction commands

char buffer[256]; // buffer for debugging

char xx1[20]; // temp char for the long converted to char

char JoinedChar[100]; // char to hold the joined strings

String LastAction;

int Button1 = 2;

int Button2 = 3;

int Button3 = 4;

int Button4 = 5;

// Action based on command click

/\*----------------------------------------------------------------------------\*/

/\* incoming data : Get a single client char \*/

/\*----------------------------------------------------------------------------\*/

char gchr(void){

while (!client.available()); /\* Await data from client \*/

return client.read(); /\* Return input character \*/

} /\* end: gchr() \*/

/\*----------------------------------------------------------------------------\*/

/\* incoming data : Get an entire line from the client \*/

/\*----------------------------------------------------------------------------\*/

char \*glin(char \*buf){

char c,\*p = buf; /\* Input char, input buffer pointer \*/

while (' ' > (c = gchr())); /\* Discard (leading) control chars \*/

do \*p++ = c; /\* Move input char to line buffer \*/

while (' ' <= (c = gchr())); /\* Until control char encountered \*/

\*p = '\0'; /\* Terminate line in buffer \*/

return buf; /\* Return pointer to input string \*/

} /\* end: glin() \*/

/\*----------------------------------------------------------------------------\*/

/\* Arduino standard setup() function \*/

/\*----------------------------------------------------------------------------\*/

void setup(void){

pinMode(4,OUTPUT); /\* pin selected to control \*/

pinMode(Button1, OUTPUT);

pinMode(Button2, OUTPUT);

pinMode(Button3, OUTPUT);

pinMode(Button4, OUTPUT);

Ethernet.begin(mac,ip); /\* Initialize ethernet device \*/

server.begin(); /\* Listen for connections \*/

Serial.begin(9600); /\* Connect to serial monitor \*/

// Initialize switches to NC state

digitalWrite(Button1, HIGH);

digitalWrite(Button2, HIGH);

digitalWrite(Button3, HIGH);

digitalWrite(Button4, HIGH);

//initialize the passIDstart array to hold zero values in all elements

for (int i=0; i < passCount; i++){

passIDstart[i] = 0;

}

} /\* end: setup() \*/

/\*----------------------------------------------------------------------------\*/

/\* Arduino standard loop() function \*/

/\*----------------------------------------------------------------------------\*/

void loop(void) {

unsigned long currentMillis = millis();

if (client = server.available()) { // Request client connection

while (client.connected()) { // Is there client data available?

glin(buffer); // Get HTTP request line by line

passIDvalid = 0; // set the passIDvalid to 0

LastAction = "";

if (strstr(buffer,"logform")) { /\* Is logform keyword present? \*/

if (strstr(buffer,myPass)) { /\* Is password present? \*/

// clear the rest of the incoming buffer

char c = client.read();

if (readString.length() < 100) { // read char by char HTTP request

readString += c; // store characters to string

}

if (c == '\n') { // if HTTP request has ended - blank line received

}

/\* Login form processing \*/

// get the first open passID storage space and create a new session ID.

for (int i=0; i < passCount; i++){

if (passIDvalid == 0 && passIDstart[i] == 0){

passIDstart[i] = currentMillis; // set the variable to use as the passID

passIDlast[i] = currentMillis; // set the last time the passID was used

passIDvalid = passIDstart[i];

LastAction = "Login Completed";

}

} // end of loop for get the first open passID storage space

} // end: if password is present

} // end: if logform in line

// Serial.println("Processing input");

// clear expired passIDs from array - housekeeping each time a server connection is made

for (int i=0; i < passCount; i++){

if (passIDstart[i] != 0){

if (currentMillis >= passExpireMil){

if (currentMillis - passIDlast[i] >= passExpireMil){

passIDstart[i] = 0;

}

}

}

}

unsigned int GotAction = 0; // shows that no DoAction command received

// look for DoAction commands in the input buffer

if (strstr(buffer,"DoAction")) { /\* If DoAction keyword is present \*/

//see if a valid passID exists in the DoAction line

for (int i=0; i < passCount; i++){

if (GotAction == 0 && passIDstart[i] != 0){

ltoa(passIDstart[i],xx1,10); // convert the long to char

//check for the passID + DoAction keyword + action code in the buffer

JoinedChar[0] = '\0'; // clear the destination array

strcat(JoinedChar, xx1); //add the passID to the array

strcat(JoinedChar, DoAction); //add the keyword to the array

strcat(JoinedChar, "on001"); //add the action code to the array

if (strstr(buffer,JoinedChar)) GotAction = 1;

JoinedChar[0] = '\0'; // clear the destination array

strcat(JoinedChar, xx1); //add the passID to the array

strcat(JoinedChar, DoAction); //add the keyword to the array

strcat(JoinedChar, "on002"); //add the action code to the array

if (strstr(buffer,JoinedChar)) GotAction = 2;

JoinedChar[0] = '\0'; // clear the destination array

strcat(JoinedChar, xx1); //add the passID to the array

strcat(JoinedChar, DoAction); //add the keyword to the array

strcat(JoinedChar, "on003"); //add the action code to the array

if (strstr(buffer,JoinedChar)) GotAction = 3;

JoinedChar[0] = '\0'; // clear the destination array

strcat(JoinedChar, xx1); //add the passID to the array

strcat(JoinedChar, DoAction); //add the keyword to the array

strcat(JoinedChar, "on004"); //add the action code to the array

if (strstr(buffer,JoinedChar)) GotAction = 4;

if(GotAction != 0){

passIDlast[i] = currentMillis; //renew the last time that the passID was used

passIDvalid = passIDstart[i];

}

}

}

if(GotAction != 0){

// clear the rest of the incoming buffer

char c = client.read();

if (readString.length() < 100) { // read char by char HTTP request

readString += c; // store characters to string

}

if (c == '\n') { // if HTTP request has ended - blank line received

// Serial.println("Header Flushed");

}

/\* DoAction processing \*/

if(GotAction == 1){

// Serial.println("Received Web Server command on001");

// do tasks in response to command 1 received

digitalWrite(Button1, LOW);

delay(2000);

digitalWrite(Button1, HIGH);

LastAction = "Command on001 processed";

}

if(GotAction == 2){

// Serial.println("Received Web Server command on002");

// do tasks in response to command 2 received

digitalWrite(Button2, LOW);

delay(2000);

digitalWrite(Button2, HIGH);

LastAction = "Command on002 processed";

}

if(GotAction == 3){

// Serial.println("Received Web Server command on003");

// do tasks in response to command 3 received

digitalWrite(Button3, LOW);

delay(2000);

digitalWrite(Button3, HIGH);

LastAction = "Command on003 processed";

}

if(GotAction == 4){

// Serial.println("Received Web Server command on003");

// do tasks in response to command 3 received

digitalWrite(Button4, LOW);

delay(2000);

digitalWrite(Button4, HIGH);

LastAction = "Command on004 processed";

}

}

}

//display the web page

client.println("HTTP/1.1 200 OK"); //send new page

client.println("Content-Type: text/html");

client.println();

client.println("<HTML>");

client.println("<HEAD>");

client.println("<link rel='stylesheet' type='text/css' href='http://bdpa-camp.org/AndroidIOT.css' />");

client.println("<TITLE>Power Control</TITLE>"); //browser tab title

client.println("</HEAD>");

client.println("<BODY>"); //start of body section

client.println("Hiltronics Power Cycle<br><br>");

// client.print("Current Millis value = ");

// client.print(currentMillis);

client.println("<br><br>");

// client.print("Last Action : ");

// client.print(LastAction);

client.println("<br><br>");

if (passIDvalid == 0){ // this is not a valid passID - ask for the password

client.print ("<form method=get>");

client.print ("<input type=password name=logform size=10>");

client.print (" <input type=submit value=Login>");

client.print ("</form>");

}

if (passIDvalid != 0){ // this IS a valid passID - display the web page

client.println("<H1>IOT Power Cycle Project</H1>");

client.print("Operate : <a href=\"./?");

client.print(passIDvalid);

client.print("DoActionon001");

client.println("\"\">[Switch 1]</a><br><br>");

client.print("Operate : <a href=\"./?");

client.print(passIDvalid);

client.print("DoActionon002");

client.println("\"\">[Switch 2]</a><br><br>");

client.print("Operate : <a href=\"./?");

client.print(passIDvalid);

client.print("DoActionon003");

client.println("\"\">[Switch 3]</a><br><br>");

client.print("Operate : <a href=\"./?");

client.print(passIDvalid);

client.print("DoActionon004");

client.println("\"\">[Switch 4]</a><br><br>");

}

client.println("</BODY>");

client.println("</HTML>");

delay(1);

client.stop(); /\* Disconnect from the server \*/

} /\* end: while client connected \*/

} /\* end: if client connection \*/

} /\* end: loop() \*/