Department of Electrical and Computer Engineering
Rutgers University – College of Engineering
New Brunswick, NJ

# Smart Switch Outlet Adapter

Ali Rahimi

12/18/2017

16:332:559

## Abstract

This paper focuses on the development of a Wi-Fi smart plug capable of remotely switching a load on or off. It monitors energy consumption and collects data from the plug sending data to a server that communicates the data in real time to a website that can be accessed on a user's smartphone. The user can send time based requests to remotely switch the load on or off.

## I. Introduction

There is an increasing interest in energy efficiency and focus on decreasing the cost of embedded network sensors to decrease the overall cost of outlet-level metering. This will allow new buildings in the future to have "smart" outlets that monitor and transmit power usage in real time at the same cost as conventional outlets. Smart outlets are energy meters that obtain information from the end user's load devices, measuring energy consumption relaying the information back to the user. Smart Plugs are electronic devices that help users communicate with their electrical applications. There are some smart plugs such as the OMG [1] that are capable of enabling/disabling an appliance via smartphone, but lack the measurement aspect of power consumption. This paper focuses on the design of a smart plug that incorporates measuring energy consumption as well as switch enabling. It is important to mention that within the consumer market, products like the DSP-W215 Wi-Fi smart plug released by DLink already includes these features as well as voice control functions.

## II. Study

### System Design

The smart plug is designed to connect to a home's Wi-Fi network. Users can interact with the smart plug with their smart phone. The plug incorporates a low power Arduino compatible development platform known as Teensy. It communicates with a wireless network using an ESP8266 Wi-Fi module. The ESP8266 is capable of transmitting data to the network as shown in fig. 1.

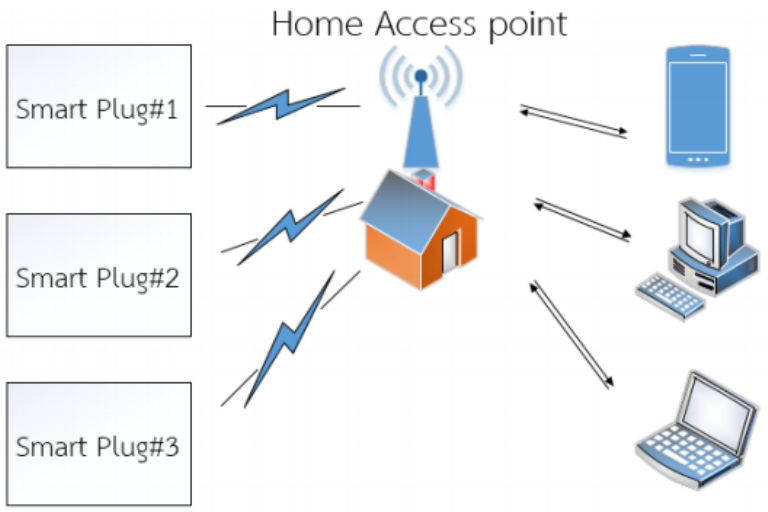


Fig. 1 Communication System Design

Fig. 2 illustrates the diagram of the smart plug. Utilizing a Kill-A-Watt (KAW), an efficient energy meter, data is received from Kill-A-Watt's LM2902 quad op-amp, at the bottom of the KAW display board. Three pins from the LM2902 are tapped into containing the voltage signal, high current range signal, and low current range signal. These inputs are received and attenuated to run at 3.3V for the Teensy. A simple voltage divider solves this issue as shown below. A 60 Hz square wave is connected to INT1, the interrupt pin on the Moteino from the collector pin from a BJT on the KAW.

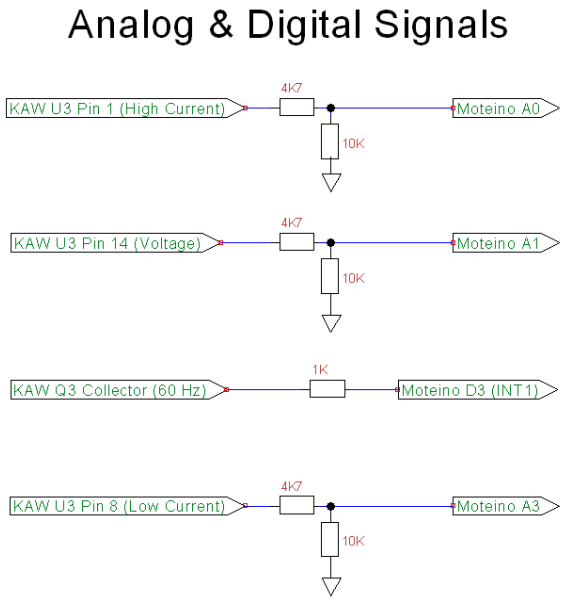

Fig. 2 Analog and Digital Signal Inputs

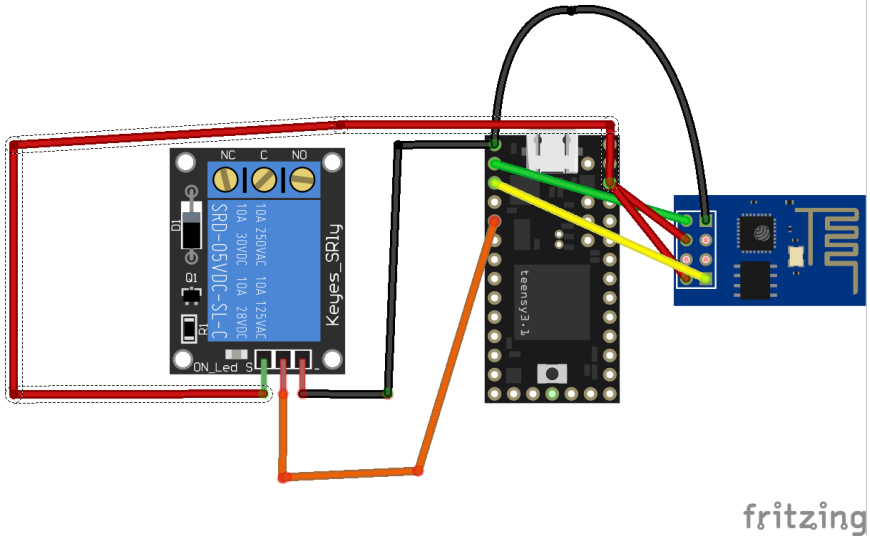


Fig. 3 Physical Setup

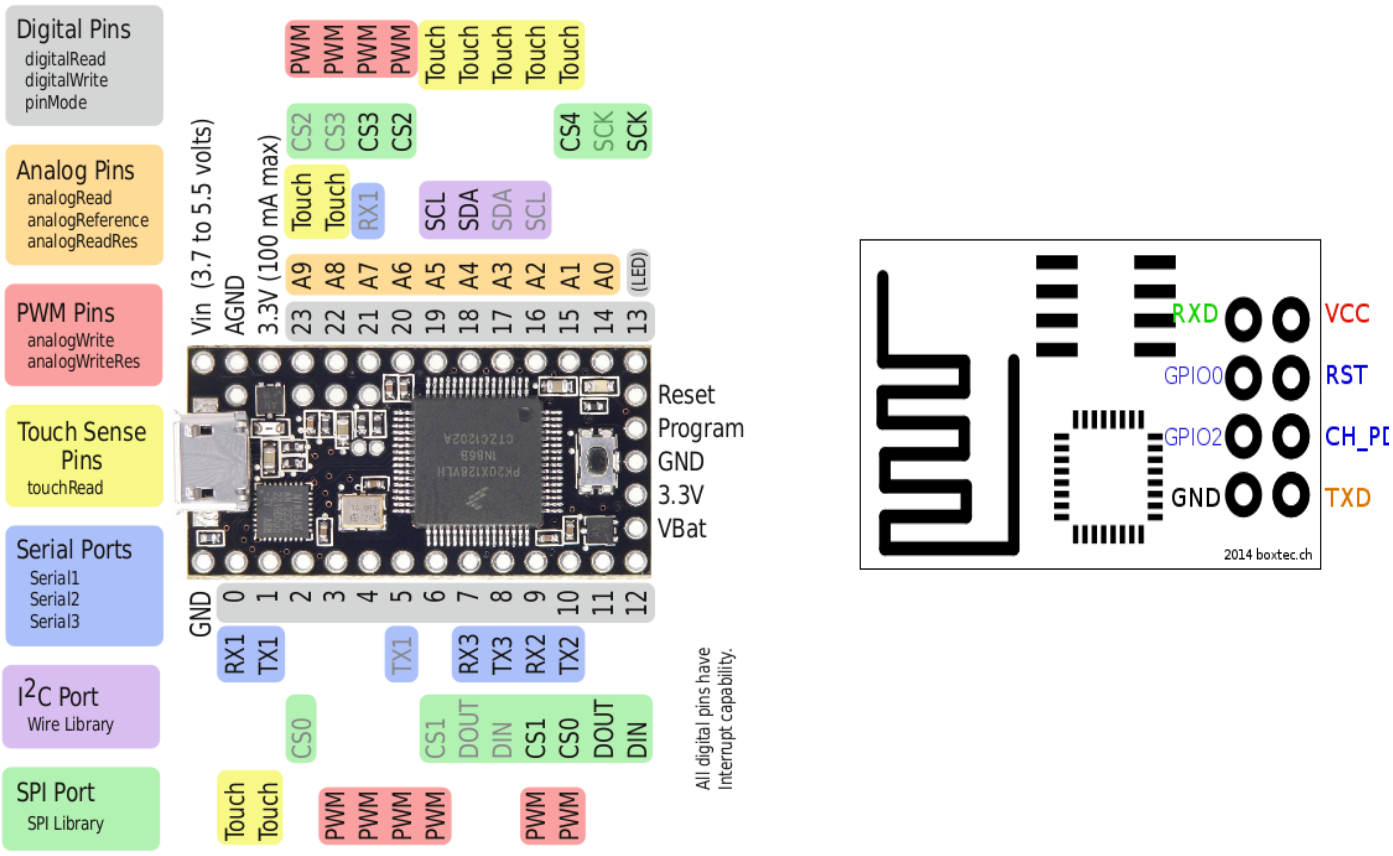


Fig. 4 Teensy and ESP8266 Pinout


Fig. 5 Kill-A-Watt

From Fig. 3 and 4, it is shown that the Teensy and ESP8266 communicate via Serial Communication. The Tx of the Teensy is connected to the Rx and the ESP8266, and vice-versa. The relay has 3 pins (VCC,GND,IN2). IN2 is connected to Pin 3 on Teensy. The Hot line of a power cable is connected to the Common and NO (normally Open) input on the relay. From Fig 5. the Kill-A-Watt has a male and female end; one side is connecting to the wall outlet and another for the electrical appliance. As the relay turns off, the entire system also turns off.

**Data Analysis**

**Energy Management System (Software)**

The data collected from the Teensy is sent to ThingSpeak via the ESP8266. ThingSpeak is an open IOT platform with MATLAB analytics. Using TCP protocol to start the connection, AT+CIPSTART command is invoked to connect to the ThingSpeak server. An HTTP Post request is sent to ThingSpeak with the power and voltage values shown on Fig 6. A HTTP GET request is made sending a string “On” or “Off” which is received and parsed through. When either String is received, Pin 3 on Teensy is HIGH or LOW. Because the connection from the hot wire is connected to Normally Open, grounding the pin actually turns the relay on.

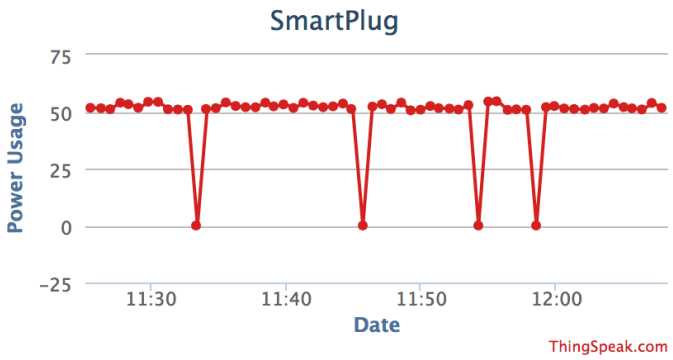


Fig 6. ThingSpeak value recording

ThingSpeak provides a function called TalkBack that Sends the “On” or “Off” string as explained above.

Commands

| Position | Command ID | Command string |
|---|---|---|
| 1 | 10147559 | On |

Fig 7. ThingSpeak Command Entry

ThingSpeak also provides TimeControl which provides the user with the ability to control at what time Talkback is triggered.

Apps / TimeControl / Time Control

Edit TimeControl

| | |
|---|---|
| Name: | Time Control |
| Frequency: | One Time |
| Time Zone: | Eastern Time (US & Canada) (edit) |
| Last Ran: | |
| Run At: | 2017-12-11 12:20 pm |
| Fuzzy Time: | ± 0 minutes |
| TalkBack: | On/off, add command: Off |
| Created: | 2017-12-07 5:21 pm |

Fig. 8 ThingSpeak Time Control

```
int n = 0;
measState = 0; // 0 is partial half cycle
toggle = !toggle;
if (toggle) {
  attachInterrupt(HzINT, AC60HZ, RISING);
} else {
  attachInterrupt(HzINT, AC60HZ, FALLING);
}
while (measState < 2); // skip partial cycle
//digitalWrite(LED,HIGH);
detachInterrupt(HzINT);
while ((measState < 3) && (n < MaxN)) { // measure 1 cycle
  //digitalWrite(LED,HIGH);
  A[n] = analogRead(ASENS); // 20.4 us
  V[n] = analogRead(VSENS); // 20.4 us
  a[n] = analogRead(aSENS); // 20.4 us
  n++;
  //digitalWrite(LED,LOW);
  // time difference between voltage and current readings is about 20.4 us
  // 20.4 us is about 0.44 degrees at 60 Hz, good enough!
  // target 50 readings over 1 cycle @ 60Hz
  // reading time: 3 * 20.4 us * 50 = 3.06 ms
  // total dead time: (1/60) - 3.06 ms = 13.61 ms
  // dead time between readings: 13.61 ms / 50 = 272 us
  delayMicroseconds(280); // fine tuned for 50 readings @ 60 Hz
}
//digitalWrite(LED,LOW);
```

Fig 9. Arduino Code Rundown P1

Teensy runs a toggle that is activated on the Rising and falling edge from Q3, the collector on the Kill-A-Watt. Every time an interrupt is ran MeasState is incremented by one. Once MeasState has the correct parameters, the values are recorded.

```
KAW.N = n; // number of readings
// compute offset levels
float VSum = 0, ASum = 0, aSum = 0, WSum = 0, wSum = 0;
for (int i = 0; i < n; i++) {
  VSum += V[i];
  ASum += A[i];
  aSum += a[i];
}
float aV, aA, aa; // averages
aV = VSum / n;
aA = ASum / n;
aa = aSum / n;
// compute std.dev. = AC RMS
VSum = ASum = aSum = 0;
for (int i = 0; i < n; i++) {
  VSum += (V[i] - aV) * (V[i] - aV);
  ASum += (A[i] - aA) * (A[i] - aA);
  aSum += (a[i] - aa) * (a[i] - aa);
  WSum += (V[i] - aV) * (A[i] - aA);
  wSum += (V[i] - aV) * (a[i] - aa);
}
KAW.V = sqrt(VSum / n);
KAW.A = sqrt(ASum / n);
KAW.a = sqrt(aSum / n);
Serial.println("RAW a");
Serial.println(KAW.a);
KAW.W = WSum / n;
KAW.w = wSum / n;
```

Fig 10. Arduino Code Rundown P2

The average is calculated for each signal. By Calculating the standard deviation, we are also calculating the RMS. At each sample point we square the difference between the sample and the mean. We sum up those squares and find the averages of that and finally square root that for V[], A[], a[], and calculate W[], w[]. W[] represents current above 2 A, while w[] represents current below 2 A.

## III. Conclusions

In conclusion this project was successful, but had some limitations. There was not enough time to create a mobile app, but the website can be accessed on your phone's browser. The Kill-A-Watt is not the best sensor to use to track power because the values received need to be mapped and calibrated to be close enough to the actual value of power consumed. This is due to not knowing what the Power Factor is for any device plugged in.

The future revision of this product will have a custom PCB board. It will have its own dedicated server that handles wireless communication. Added features will include a weekly summary of your power consumption for not just 1 smart plug but multiple.

## IV. References


Hernandez, M. (2017, July 07). Connect Your ESP8266 to Any Available Wi-Fi Network - DZone IoT. Retrieved November 10, 2017, from https://dzone.com/articles/connect-your-esp8266-to-any-available-wi-fi-networ

Tranchemontagne, M. (1970, January 01). Hooked on Arduino & Raspberry Pi. Retrieved November 10, 2017, from http://www.mikesmicromania.com/2013/04/moteino-kill-watt-hardware.html

Pora, W., & Thongkhao, Y. (2016). A low-cost Wi-Fi smart plug with on-off and Energy Metering functions. 2016 13th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 1-5.

Dhar, M. B. (2017, September 13). ESP8266 IoT Energy Monitor. Retrieved December 14, 2017, from https://www.hackster.io/whatnick/esp8266-iot-energymonitor-b199ed

Understand Your Things. (n.d.). Retrieved December 14, 2017, from https://thingspeak.com/


## V. Appendix

```
#define CALMODE false // true to xmit raw values (to establish cal factors)

#define SSID "iPhone"  //name of wireless access point to connect to

#define PASS "juicetank"  //wifi password

#define DST_IP "184.106.153.149" //my web site,

String apiKey = "ZVV7RAPNL097ED4N";
```

```
String talkbackapikey = "3SQMG3PC7BS0GBVE";
String talkbackid = "9960011";
const int checkTalkBackInterval = 15 * 1000;
boolean latch = false;
#define relay 3
#define LED 13
#define RESET 12
int loops = 0;  //a counter for testing
struct {
  int Status; // 0: OK, 1: first reading, -1: low voltage
  int      N; // # of readings (over 1 line cycle)
  float    V; // AC RMS voltage
  float    A; // AC RMS current, coarse scale
  float    W; // average watts, coarse current scale
  float    a; // AC RMS current, fine scale
  float    w; // average watts, fine current scale
} KAW;
// cal factors for map()
const float CFrawVlo = 340.5; // raw value from A2D
const float CFrawVhi = 451.8;
const float CFcalVlo = 117.7; // scaled Voltage
const float CFcalVhi = 119.2;
const float CFrawAlo = 345.9;  // A2D
```

```
const float CFrawAhi = 397.1;
const float CFcalAlo = 2.18;  // Amps (coarse scale current)
const float CFcalAhi = 7.83;
const float CFrawalo = 234.56;  // A2D
const float CFrawahi = 322.2;
const float CFcalalo = .50;  // amps (fine scale current)
const float CFcalahi = .59;
// pin assignments
const int ASENS  = 17; // AI01 = A0 (wire from pin 1, coarse current)
const int VSENS  = 20; // AI02 = A1 (wire from pin 14, voltage)
const int aSENS  = 18; // AI04 = A3 (wire from pin 8, fine current)
const int HzSENS = 12; // IRQ  = D3 (wire from Q3 collector, 60Hz)
const int HzINT  = 12; //        INT1
const int MaxN  = 65; // max # readings (good down to 48 Hz)
const int IN2 = 3;
void setup () {
  pinMode(RESET, OUTPUT);
  reset();
  pinMode(LED, OUTPUT);
  Serial1.begin(115200);    // hardware serial connects to esp8266 module
  Serial.begin(115200);
  pinMode(relay, OUTPUT);
  digitalWrite(relay, HIGH);
```

```
  delay(4000);    //wait for usb serial enumeration on 'Serial' & device startup
  boolean wifi_connected = false; //not connected yet...
  for (int i = 0; i < 5; i++) //attempt 5 times to connect to wifi - this is a good idea
  {
    if (connectWiFi()) //are we connected?
    {
      wifi_connected = true;  //yes
      break;              //get outta here!
    }
  }
  if (!wifi_connected) hang("wifi not connected");  //these seem ok - never had a problem
  delay(250);
  if (!cipmux0()) hang("cipmux0 failed");
  delay(250);
  if (!cipmode0()) hang("cipmode0 failed");
  delay(250);
  // go to sleep ASAP to give KAW a chance to boot up
  // LowPower.powerDown(SLEEP_4S, ADC_OFF, BOD_ON);
  delay(4000);
  pinMode(HzSENS, INPUT);   // D3 is IRQ1
  // speed up ADC to minimize phase measurement error
}
boolean First = true;            // the very 1st reading
```

```
int V[MaxN], A[MaxN], a[MaxN];   // waveform arrays
volatile int unsigned measState; // state machine, clocked by line frequency
boolean toggle;                  // alternate maeasurements on + or - cycle
void loop () {


 checkTalkBack();
 delay(checkTalkBackInterval);
 // acquire voltage and current samples
 int n = 0;
 measState = 0; // 0 is partial half cycle
 toggle = !toggle;
 if (toggle) {
  attachInterrupt(HzINT, AC60HZ, RISING);
 } else {
  attachInterrupt(HzINT, AC60HZ, FALLING);
 }
 while (measState < 2); // skip partial cycle
 //digitalWrite(LED,HIGH);
 detachInterrupt(HzINT);
 while ((measState < 3) && (n < MaxN)) { // measure 1 cycle
  //digitalWrite(LED,HIGH);
  A[n] = analogRead(ASENS); // 20.4 us
  V[n] = analogRead(VSENS); // 20.4 us
```

```
  a[n] = analogRead(aSENS); // 20.4 us
  n++;
  //digitalWrite(LED,LOW);
  // time difference between voltage and current readings is about 20.4 us
  // 20.4 us is about 0.44 degrees at 60 Hz, good enough!
  // target 50 readings over 1 cycle @ 60Hz
  // reading time: 3 * 20.4 us * 50 = 3.06 ms
  // total dead time: (1/60) - 3.06 ms = 13.61 ms
  // dead time between readings: 13.61 ms / 50 = 272 us
  delayMicroseconds(280); // fine tuned for 50 readings @ 60 Hz
}
//digitalWrite(LED,LOW);
KAW.N = n; // number of readings
// compute offset levels
float VSum = 0, ASum = 0, aSum = 0, WSum = 0, wSum = 0;
for (int i = 0; i < n; i++) {
  VSum += V[i];
  ASum += A[i];
  aSum += a[i];
}
float aV, aA, aa; // averages
aV = VSum / n;
aA = ASum / n;
```

```
  aa = aSum / n;
  // compute std.dev. = AC RMS
  VSum = ASum = aSum = 0;
  for (int i = 0; i < n; i++) {
    VSum += (V[i] - aV) * (V[i] - aV);
    ASum += (A[i] - aA) * (A[i] - aA);
    aSum += (a[i] - aa) * (a[i] - aa);
    WSum += (V[i] - aV) * (A[i] - aA);
    wSum += (V[i] - aV) * (a[i] - aa);
  }
  KAW.V = sqrt(VSum / n);
  KAW.A = sqrt(ASum / n);
  KAW.a = sqrt(aSum / n);
  Serial.println("RAW a");
  Serial.println(KAW.a);
  KAW.W = WSum / n;
  KAW.w = wSum / n;
#if not CALMODE  // apply cal factors
  // voltage
  KAW.V = map(KAW.V, CFrawVlo, CFrawVhi, CFcalVlo, CFcalVhi);
  // coarse current scale, above 2 A
  KAW.A = map(KAW.A, CFrawAlo, CFrawAhi, CFcalAlo, CFcalAhi);
  // coarse wattage scale, Current above 2 A
```

```
  KAW.W = map(KAW.W, CFrawVlo * CFrawAlo, CFrawVhi * CFrawAhi, CFcalVlo *
CFcalAlo, CFcalVhi * CFcalAhi);

  // fine current scale, below 2 A

  KAW.a =  map(KAW.a, CFrawalo, CFrawahi, CFcalalo, CFcalahi);


  // fine wattage scale, Current below 2 A

  KAW.w = map(KAW.w, CFrawVlo * CFrawalo, CFrawVhi * CFrawahi, CFcalVlo *
CFcalalo, CFcalVhi * CFcalahi);

  // prevent negative values (e.g. no low load current, noise)

  KAW.a = max(KAW.a, 0);

  KAW.w = max(KAW.w, 0);

  KAW.A = max(KAW.A, 0);

  KAW.W = max(KAW.W, 0);

  // use fine current scale for 2A or less load current

  if (KAW.a <= 2.0) {

    KAW.A = KAW.a;

    KAW.W = KAW.w;

  }

#endif

  if (First) {

    First = false;

    KAW.Status = 1;          // 1st reading

  } else if (KAW.V < 115 ) { // KAW analog supply sags
```

```
   KAW.Status = -1;         // accuracy warning
  } else {
   KAW.Status = 0;          // OK
  }
  delay(10);              // 10 ms visable

  if (latch == true) {
     writeToThingSpeak(KAW.w,KAW.V);
  }
 Serial.print("Amp: ");
 Serial.println(KAW.a);
  Serial.print("Watt: ");
  Serial.println(KAW.w);
  Serial.print("Voltage: ");
  Serial.println(KAW.V);
}
void AC60HZ() {
  //digitalWrite(LED,HIGH);
  measState++;
  //digitalWrite(LED,LOW);
}
// overload Arduino integer map() function to support floats, doubles
double map(double x, double in_min, double in_max, double out_min, double out_max)
```

```
{
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
void writeToThingSpeak(float power, float voltage)
{
  reset();  //only CERTAIN way I've found of keeping it going
  delay(20000);  //esp takes a while to restart
  // Serial.print("loops = ");  //check for successful connections to server
  //Serial.println(loops);
  loops++;
  String cmd = "AT+CIPSTART=\"TCP\",\"";  //make this command:
AT+CIPSTART="TCP","184.106.153.149",80
  cmd += DST_IP;
  cmd += "\",80";
  Serial1.println(cmd);  //send command to device
  delay(2000);  //wait a little while for 'Linked' response - this makes a difference
  if (Serial1.find("OK")) //message returned when connection established WEAK SPOT!!
DOESN'T ALWAYS CONNECT
  {
    Serial.print("Connected to server at ");  //debug message
    Serial.println(DST_IP);
  }
  else
```

```
  {
    Serial.println("'Linked' response not received");  //weak spot! Need to recover elegantly
  }
  String postStr = apiKey;
  postStr += "&field1=";
  postStr += String(power);
  postStr += "&field2=";
  postStr += String(voltage);
  postStr += "\r\n\r\n";
  cmd =  "POST /update HTTP/1.1\n";  //construct http GET request
  cmd += "Host: api.thingspeak.com\n";        //test file on my web
  cmd += "Connection: close\n";
  cmd += "X-THINGSPEAKAPIKEY: " + apiKey + "\n";
  cmd += "Content-Type: application/x-www-form-urlencoded\n";
  cmd += "Content-Length: ";
  cmd += postStr.length();
  cmd += "\n\n";
  cmd += postStr;
  Serial1.print("AT+CIPSEND=");               //www.cse.dmu.ac.uk/~sexton/test.txt
  Serial1.println(cmd.length());  //esp8266 needs to know message length of incoming message - .length provides this
  if (Serial1.find(">"))   //prompt offered by esp8266
  {
```

```
    Serial.println("found > prompt - issuing GET request");  //a debug message
    Serial1.println(cmd);  //this is our http GET request
  }
  else
  {
    Serial1.println("AT+CIPCLOSE");  //doesn't seem to work here?
    Serial.println("No '>' prompt received after AT+CPISEND");
  }
  Serial1.println("AT+CIPCLOSE");
  if (Serial1.find("Unlink")) //rarely seems to find Unlink? :(
  {
    Serial.println("Connection Closed Ok...");
  }
  else
  {
    //Serial.println("connection close failure");
  }
}

//--------------------------------------------------------------------------------
boolean connectWiFi()
{
  String cmd = "AT+CWJAP=\""; //form eg: AT+CWJAP="Arcane Reality Office","juicetank"
```

```
  cmd += SSID;
  cmd += "\",\"";
  cmd += PASS;
  cmd += "\"";
  Serial1.println(cmd);
  delay(5000); //give it time - my access point can be very slow sometimes
  if (Serial1.find("OK")) //healthy response
  {
    Serial.println("Connected to WiFi...");
    return true;
  }
  else
  {
    Serial.println("Not connected to WiFi.");
    return false;
  }
}
//------------------------------------------------------------------------------
//ditch this in favour of hardware reset. Done
boolean softwarereset()
{
  Serial1.println("AT+RST");
  if (Serial1.find("ready"))
```

```
  {
    return true;
  }
  else
  {
    return false;
  }
}
//-------------------------------------------------------------------------------
void reset()
{
  digitalWrite(RESET, LOW);
  digitalWrite(LED, HIGH);
  delay(100);
  digitalWrite(RESET, HIGH);
  digitalWrite(LED, LOW);
}
//-------------------------------------------------------------------------------
boolean cwmode3()
// Odd one. CWMODE=3 means configure the device as access point & station. This function can't fail?

{
```

```
  Serial1.println("AT+CWMODE=3");
  if (Serial1.find("no change"))  //only works if CWMODE was 3 previously
  {
    return true;
  }
  else
  {
    return false;
  }
}
//-----------------------------------------------------------------------------
boolean cipmux0()
{
  Serial1.println("AT+CIPMUX=0");
  if (Serial1.find("OK"))
  {
    return true;
  }
  else
  {
    return false;
  }
}
```

```
//--------------------------------------------------------------------
boolean cipmode0()
{
  Serial1.println("AT+CIPMODE=0");
  if (Serial1.find("OK"))
  {
    return true;
  }
  else
  {
    return false;
  }
}
//--------------------------------------------------------------------
void hang(String error_String)    //for debugging
{
  Serial.print("Halted...   ");
  Serial.println(error_String);
  while (1)
  {
    digitalWrite(LED, HIGH);
    delay(100);
    digitalWrite(LED, LOW);
```

```
    delay(100);
  }
}
//--------------------------------------------------------------------------
void hangreset (String error_String)    //for debugging
{
  Serial.print(error_String);
  Serial.println(" - resetting");
  reset();
}
//--------------------------------------------------------------------------
void checkTalkBack()
{
  reset();  //only CERTAIN way I've found of keeping it going
  delay(200);  //esp takes a while to restart
  // Serial.print("loops = ");  //check for successful connections to server
  //Serial.println(loops);
  loops++;
  String cmd = "AT+CIPSTART=\"TCP\",\"";  //make this command:
AT+CIPSTART="TCP","184.106.153.149",80
  cmd += DST_IP;
  cmd += "\",80";
  Serial1.println(cmd);  //send command to device
```

```
  delay(2000);  //wait a little while for 'Linked' response - this makes a difference
  if (Serial1.find("OK")) //message returned when connection established WEAK SPOT!!
DOESN'T ALWAYS CONNECT
  {
    Serial.print("Connected to server at ");  //debug message
    Serial.println(DST_IP);
  }
  else
  {
    Serial.println("'Linked' response not received");  //weak spot! Need to recover elegantly
  }
  cmd =  "GET /talkbacks/20595/commands/execute.json?api_key=3SQMG3PC7BS0GBVE\n";
//construct http GET request GET
/talkbacks/20595/commands/execute.json?api_key=3SQMG3PC7BS0GBVE Host:
184.106.153.149
  cmd += "Host: 184.106.153.149\r\n\r\n";        //test file on my web
  Serial1.print("AT+CIPSEND=");                 //www.cse.dmu.ac.uk/~sexton/test.txt
  Serial1.println(cmd.length());  //esp8266 needs to know message length of incoming message -
.length provides this
  String res;
  if (Serial1.find(">"))   //prompt offered by esp8266
  {
    Serial.println("found > prompt - issuing GET request");  //a debug message
    Serial1.println(cmd);  //this is our http GET request
```

```
  res = Serial1.readString();
  Serial.println(res);
  if (res.indexOf("On") > 0) {
    Serial.print("In On IF");
    latch = true;
    digitalWrite(relay, LOW);
    Serial1.flush();
  }
  if (res.indexOf("Off") > 0) {
    latch = false;
    Serial.print("In Off IF");
    digitalWrite(relay, HIGH);
  }
}
else
{
  Serial1.println("AT+CIPCLOSE");  //doesn't seem to work here?
  Serial.println("No '>' prompt received after AT+CPISEND");
}
Serial1.println("AT+CIPCLOSE");
if (Serial1.find("Unlink")) //rarely seems to find Unlink? :(
{
  Serial.println("Connection Closed Ok...");
```

```
  }
  else
  {
    //Serial.println("connection close failure");
  }
}
```