# MainsLogger

*Version: 2022-03-21*

## MainsLogger based on ZMPT101B and/or ZMCT103C

Starting off to make a simple 1-phase voltage linemonitor The final outcome was a more versatile PCB board that allows the following configurations:

- monitoring 1, 2 or 3 phases voltage
- monitoring 1, 2 or 3 phases current
- monitoring 1 phase voltage and current

Besides basic measuring also line frequency and cos-Φ/power factor can be measured.

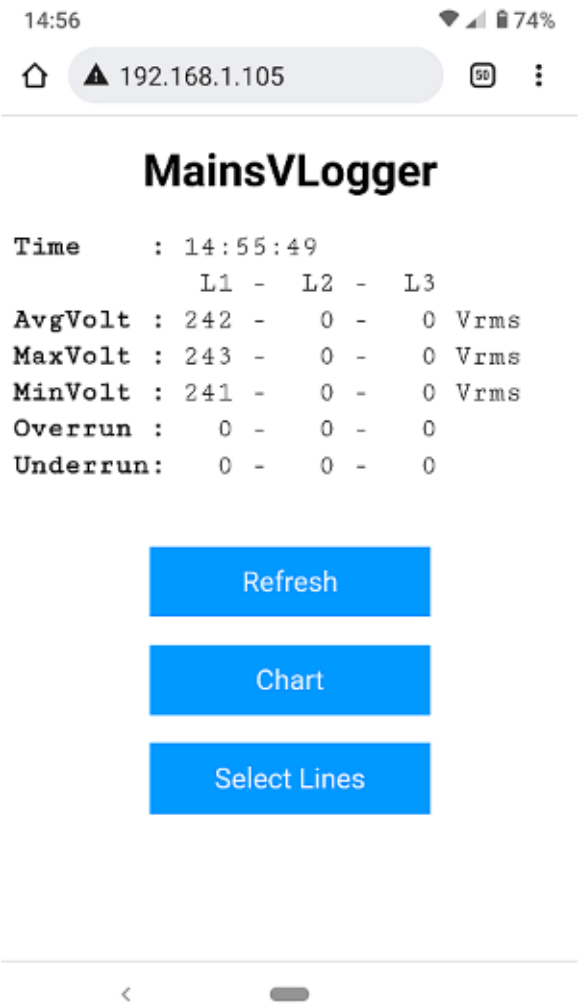
Flexible setup of WiFi-connection and MQTT server & topic using a smartphone.
A build-in webserver allows to monitor ad hoc including graphs
Data, in JSON format, can be transmitted to a MQTT server making is usable in smart home applications like Home Assistant, OpenHAB, Domoticz, ...

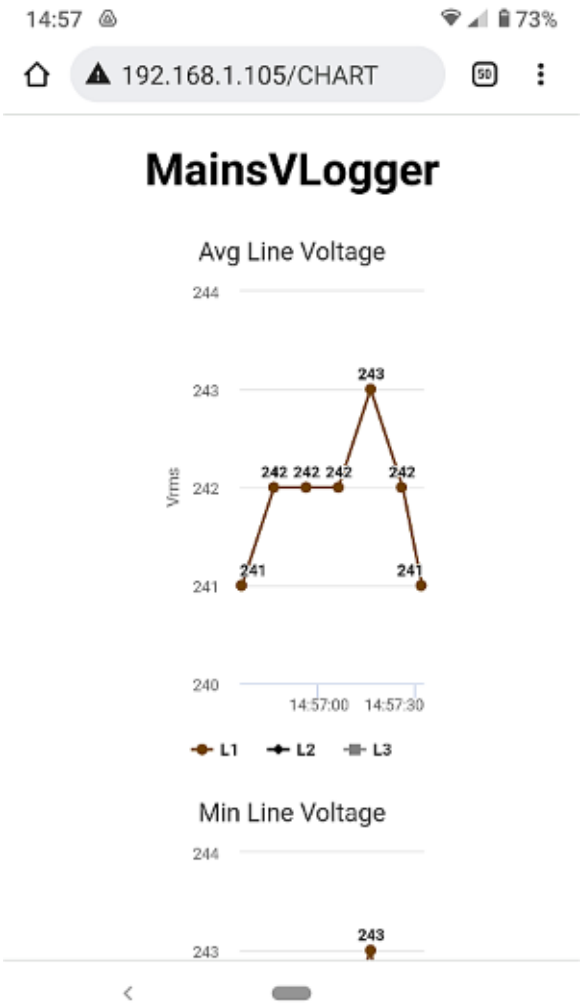
Power consumption of the device is about 100mW. The PCB can be powerd with a power supply between 7 and 20 V DC.

| | |
|---|---|
| 7 V DC | min: 140 mA |
| 10 V DC | min: 100 mA |
| 12 V DC | min: 85 mA |
| 15 V DC | min: 70 mA |
| 18 V DC | min: 60 mA |

This is a sample screenshot (Grafana)

Smartphone screenshots




The application I wrote to monitor the mains voltage/current has following functionality build-in

- up to 3 lines can be monitored (3-phase network). (can be selected)
- the WiFi portal-functionality is used to set-up WiFi connection and MQTT setting as required (MQTT can be disabled here)
- a web interface, both numerical and graphical, is available to watch the data and can be accessed from any device connected in the same network as the mainslogger
  *Note*: to use the grafical page, a connection to Internet on the device running the browser, is needed.
- a WiFi AP is running to access directly the application from the ESP32 through a webinterface. You only need to establish a WiFi-connection with the mainslogger.
- a MQTT client is running to sends data to a MQTT server (optional)
- OTA is installed to ease updating and upgrading
- NTP client is used to obtain right date/time
- mDNS is activated
- there is a button installed to allow resetting to factory defaults (push button for more than 3 seonds but less then 6 seconds = reset application, Push button 6 seconds or more = restore factory defaults)

The application is time-critical so all is done and must be done to keep looping as fast as possible.

The logic of the application is as follows:

After setting up all desired services, a timer-interupt occurs every 1 millisecond. All the interrupt routine does is incrementing a global counter(in RAM) (cfr for detailed information on volatile https://www.arduino.cc/reference/en/language/variables/variable-scope-qualifiers/volatile/)

The loop will check the counter, if it is bigger than 0, the ADC will be read and data is stored in an array.
I'll read 100 times the ADC values (= 100 ms = 5 periods of 50Hz) before calcluating the RMS (Root Mean Square see https://en.wikipedia.org/wiki/Root_mean_square) value of the values read.

RMS value of a series of data is obtained by

$$x_{\mathrm{RMS}} = \sqrt{\frac{1}{n}\left(x_1^2 + x_2^2 + \cdots + x_n^2\right)}.$$

Because the values read are offsetted by a DC values, I first calc the average AVG of all values read which is the DC-component.
To calc the RMS, I substract the AVG value from the value read, multiple it by itself, sum all together, divide by the number of values en take the squareroot of it.
This RMS values is stored in an RMS array.

When the RMS array is filled with 50 values = 5 seconds, an average is taken. This average is used a data for the outside world (webinterface and/or MQTT data).

During the calculation of average RMS value, I also check for

- The number of times the RMS value drops below the low boundary of the mains voltage (230 – 10%)
- The number of times the RMS value peaks above the high boundary of the mains voltage (230 + 10%)
- the maximun voltage that occurred
- the minimum voltage that occurred

all within the 5 second period. That data is also made available for outside world.

When all calculations are done, some cleanup is performed and interruptcounter is reset to start a fresh cycle.
I noted however that, even starting with a fresh cycle, the first 100ms reading might not be correct. Therefor you'll notice I do not take the first reading into account.

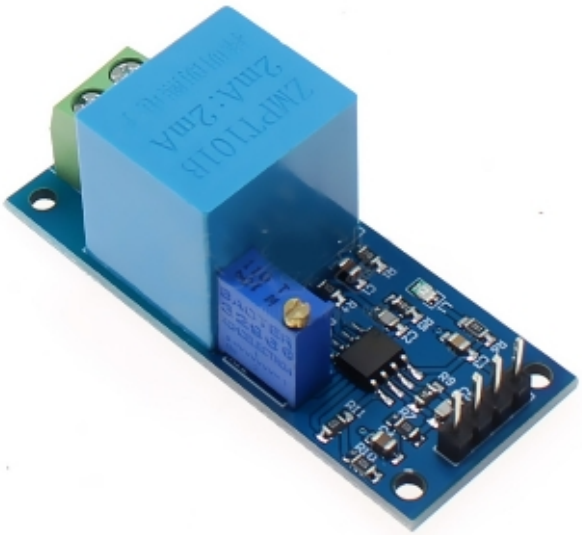
To easy calibration, the sourcecode contains also lines to accomplish this. Recompiling and uploading is needed with the #define CALIBRATE (in the mainslogger.h file) uncommented.
The calibration procedure requires also the serial monitor and serial plotter of the Arduino IDE as data is being send to the serial output of the ESP32.
Once calibration is done, recompile with the same line in comment and upload.

## MainsLogger based on ZMPT101B (Voltage up to 250V AC rms)

Starting off to make a simple 1-phase voltage linemonitor The final outcome was a moreThe ZMPT101B is a single-phase AC active output voltage mutual inductance module equipped with ZMPT101B series of high-precision voltage transformer and high-precision opamp, easy to 250Vrms within the AC power signal acquisition.

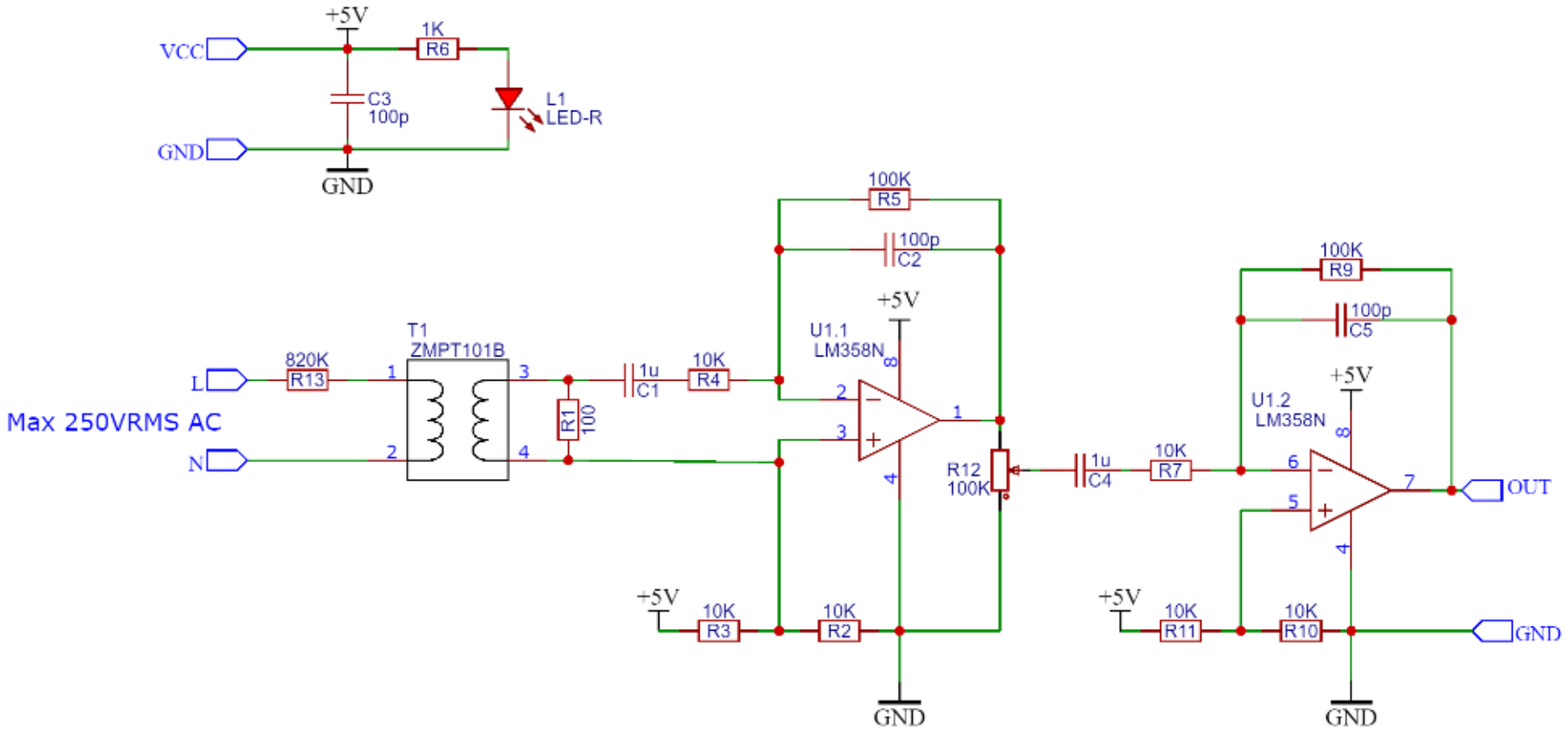
**Product Parameters**

- Onboard Precision Micro Voltage Transformer
- High-precision on-board amplifier circuit, the signal to do the exact sampling and appropriate compensation and other functions
- The module can measure AC voltage within 250Vrms, the corresponding output mode can be adjusted
- The output signal for the sine wave, the waveform of the median (DC component)
- Supply voltage is 5-30VDC

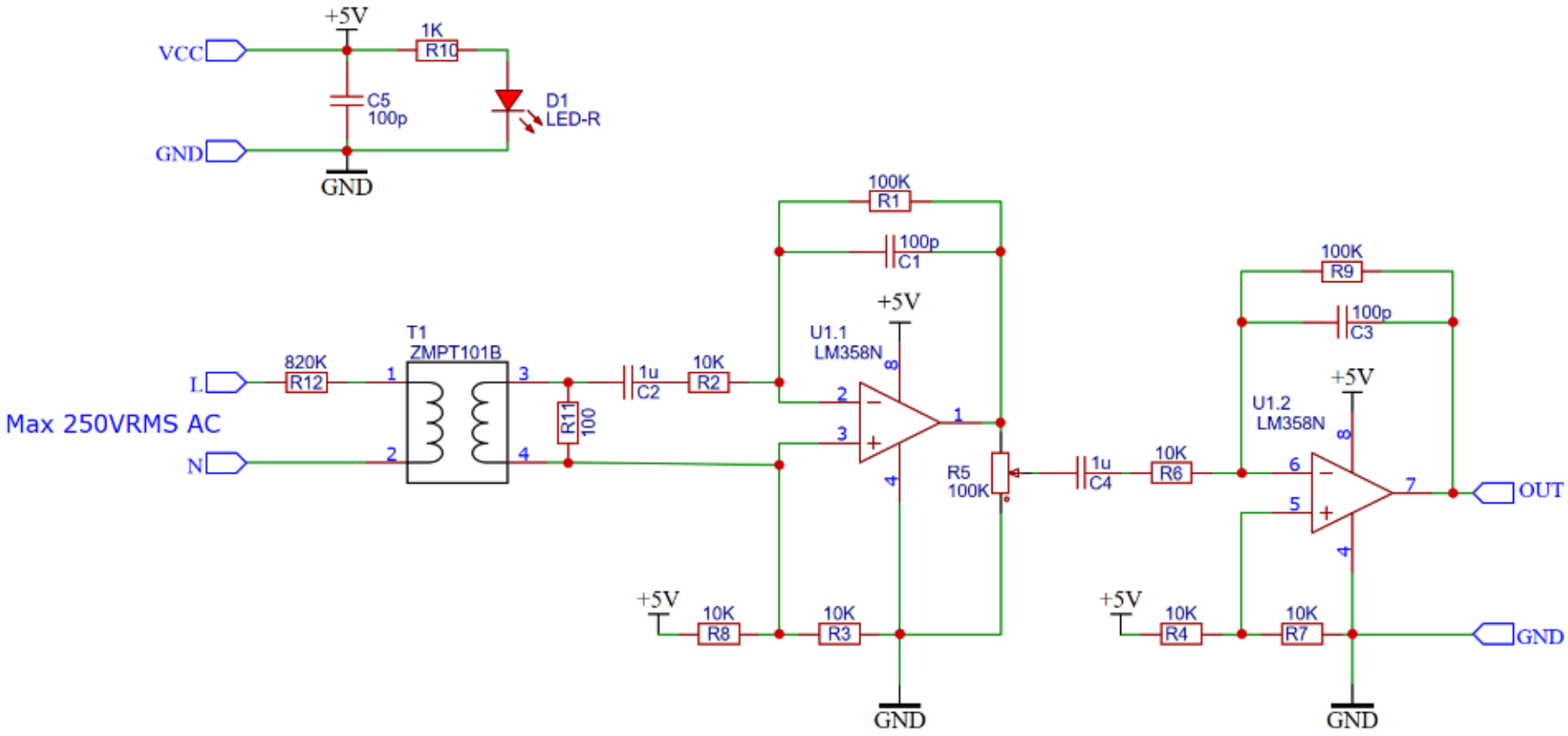




NB: I found several versions of the PCB. Below you find 2 of them.




R10 = 4k7 -> OUT: 0-3V3
R10 = 10k -> OUT: 0-5V



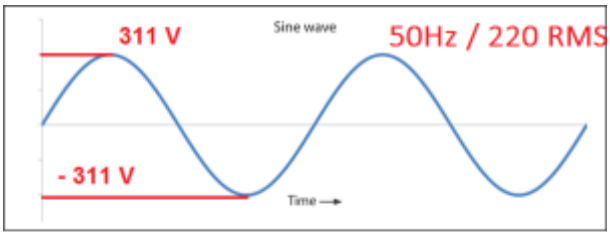

R7 = 4k7 -> OUT: 0-3V3
R7 = 10k -> OUT: 0-5V

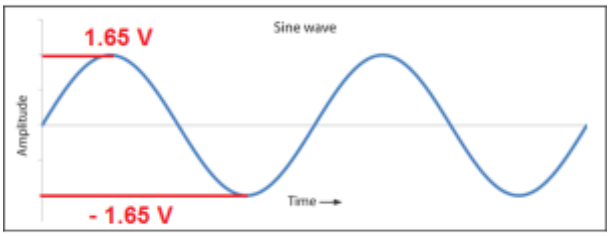
**You are working with 230V AC. So take all required safety measures to prevent electrocutation.**
**!!! Think at least twice before acting !!!**

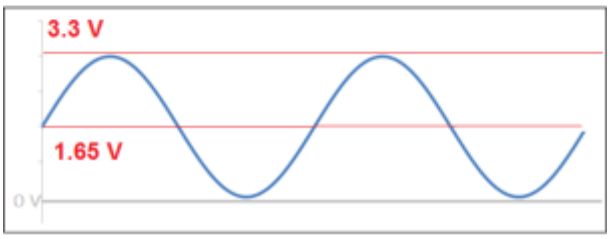
The module, called ZMPT101B, but actually this is the name of the transformer has a LM358 OpAmp onboard as you can see in the drawing. While the transformer brings down the mains voltage to less than 3.3Vpp, the circuitry adds a DC component of half of the supply voltage. So for 5VDC, thats is 2.5VDC. On top of that the AC signal is added.
Using the trimpotmeter you can bring down the whole signal to swing between 0 and 3.3VDC which almost all microcontrollers prefer nowadays as their ADC inputsignal.

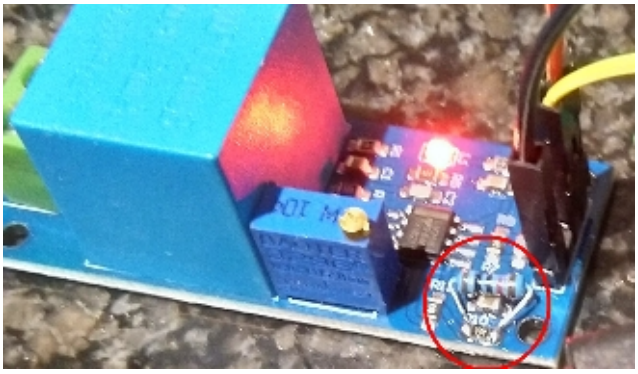
Note however that the second part of the opmap has also a DC offset of half of the supply voltage. Leaving it that way, you loose quite some swing of the AC component on top of it while keeping all down the 3.3VDC max.

Therefor, I highly recommend to solder an extra 10k resistor across R10 (or replace it with a 4.7k). Doing so, brings the DC component down from 2.5V to 1.67V when using a 5V supply and allowing to make the full swing from 0 to 3.3V.



## Calibration for Voltage

### *Adjusting the sensor through its potmeter*

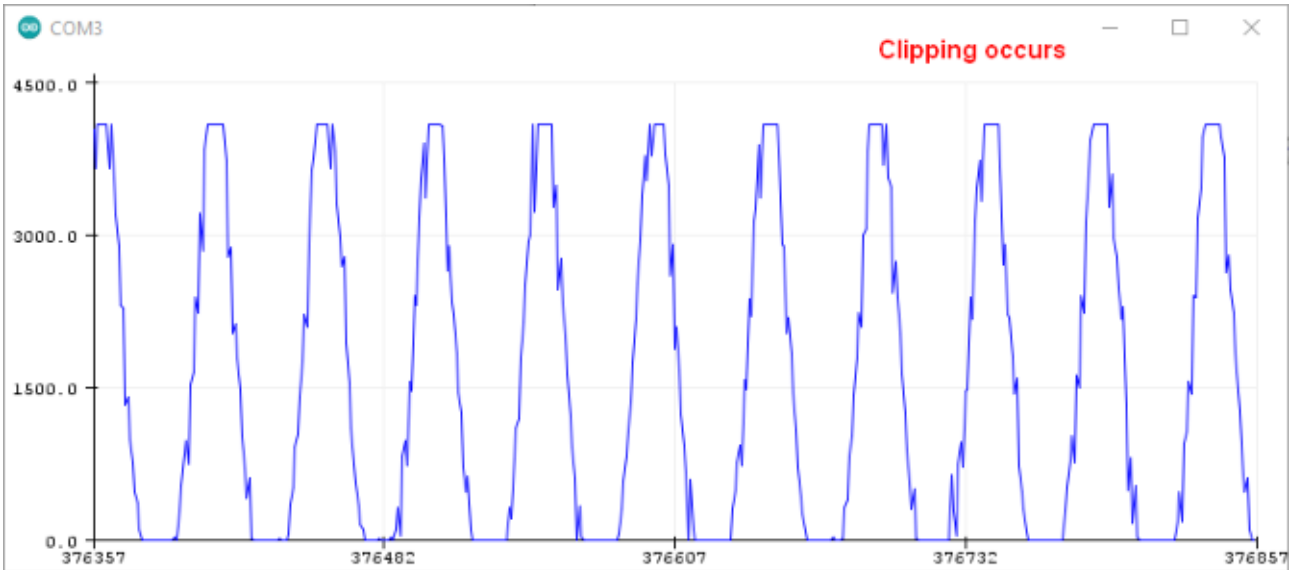
During calibration, power ESP32 only via USB and do not apply power via the on-board convertor. Only 1 power feed can be 'active'.
We need to power via USB as we need to use also serial monitor and serial plotter from the Arduino IDE.

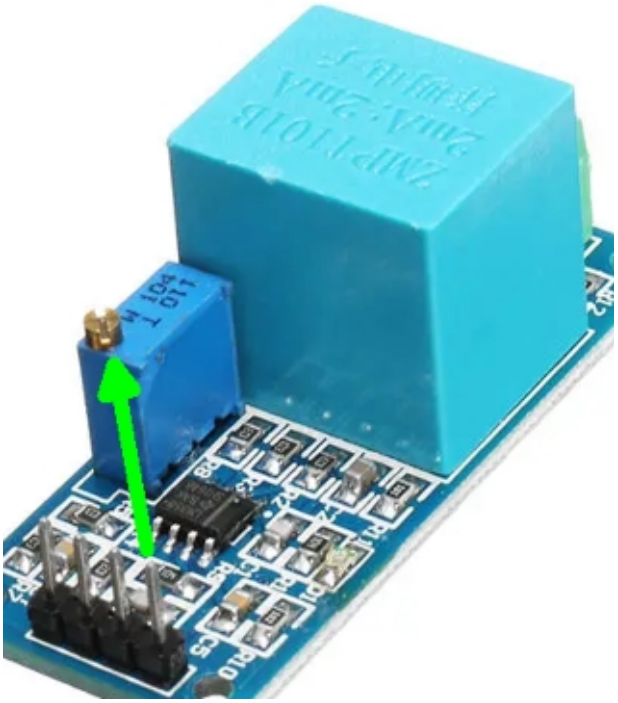
Connect the mains to the terminals of the sensor. I assume you have connected a 230V AC mains to the sensor as this is about the max the sensor can handle.
Should you use a lower voltage, adjust sensor in such a way that is can handle the max voltage you will ever measure/monitor with it

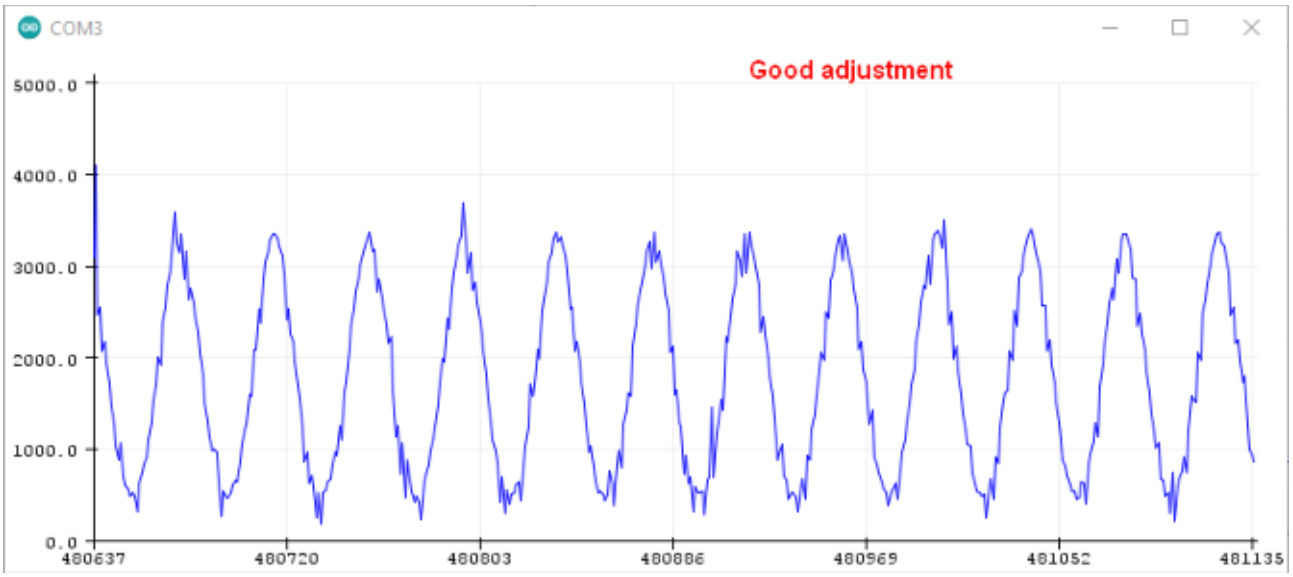
```
#define LINE  34
//#define LINE  35
//#define LINE  32

void setup()
{
  Serial.begin(115200);
  pinMode(LINE,INPUT);
}

void loop()
{
  Serial.println(analogRead(LINE));
  //delay(20);
}
```

Run this code on the ESP32 and open the Serial plotter.

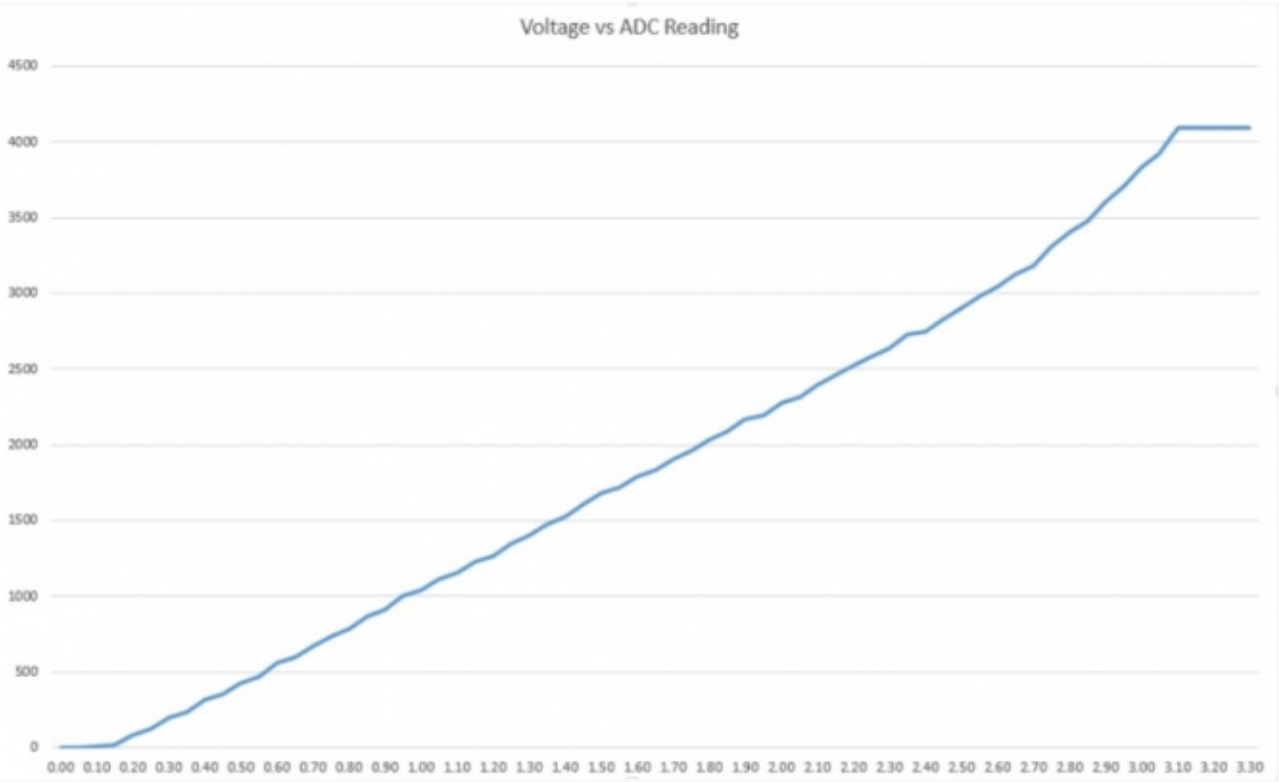
Make sure clipping does not occur and than the signal swings nicely between 100-200 and 3.300-3.500. Adjustment is done by turning the potmeter on the module






COM3
Good adjustment
5000.0
4000.0
3000.0
2000.0
1000.0
0.0
480637
480720
480803
480886
480969
481052
481135

**Note:** The ADC of an ESP32 is not fully lineair but looks like this




As you can see the lower- and upperpart flattens out. So you should stay out of these area’s to prevent misreading.

***Getting the numbers right in de code***
Once the module's output is set correctly, you have to set the right numbers in the MainsLogger.h code.

```
//LINE 1 -> index 0
#define LINE0                true
#define ADCvolt0             (float)0.2375    // volt per ADC count

//LINE 2 -> index 1
#define LINE1                false
#define ADCvolt1             (float)0.2380    // volt per ADC count

//LINE 3 -> index 2
#define LINE2                false
#define ADCvolt2             (float)0.2365    // volt per ADC count
```

To do this, recompile the MainsLogger code with the #define CALIBRATE (in the MainsLogger.h file) uncommented.
Set the line you want to calibrate to `true` and other lines to `false`. Recompile and upload.
Open Serial Monitor to watch the data being send to the output.

```
COM3

08:41:49.362 ->
08:41:54.396 ->  Zero 1 ADC value : 1821
08:41:54.396 ->  Line 1 Voltage   : 237 Vrms
08:41:54.396 ->
08:41:59.363 ->  Zero 1 ADC value : 1820
08:41:59.363 ->  Line 1 Voltage   : 236 Vrms
08:41:59.363 ->
08:42:04.398 ->  Zero 1 ADC value : 1820
08:42:04.398 ->  Line 1 Voltage   : 236 Vrms
08:42:04.398 ->
08:42:09.359 ->  Zero 1 ADC value : 1813
08:42:09.359 ->  Line 1 Voltage   : 237 Vrms
08:42:09.406 ->
```

While checking the serial monitor, measure also, with the voltmeter/mulitmeter, the voltage (V AC) of the mains.
Adjust the `ADCvoltX` value till serial output matches the voltage read on the voltmeter. You might need several recompiles and upload to get it right or close enough.
Do this for every line.
Once the calibration is done, do no longer touch the potmeter nor the values set in the code because it will affect all future calculations.
A good practice is to lock the potmeter screw with some nailpolish :-).

***Notes***:

- with the current code, a loop only reading ADC takes about 200-300 µs so that is fairly below the 1 ms of the interrupt-period. A loop with full code execution including MQTT sending but no HTTP-traffic takes about 6 ms, hence losing 6 interrupts or 1/3 of mains-cycle. That's also why in I reset the interrupt counter to 0 at exit to make sure we take full cycle samples at fixed intervals of 1 ms. Think about it ;-).

**MainsVLogger.h**

```
// System settings
// some control setting related to debugging
// uncomment to do some general debugging via Serial
//#define DEBUG
// uncomment to do debugging of WEB via Serial
//#define DEBUGWEB

//#define CALIBRATE

#define myHostname           "MainsVLogger"

#define wifiAPSSID           myHostname
#define wifiAPPassword       ""

#define ntpServer            "be.pool.ntp.org"
#define ntpUTCOffset_sec     3600
#define ntpDSTOffset_sec     3600
#define ntpUpdate_sec        60

// OTA settings
// Port default is 3232
#define otaPort              3232
#define otaPassword          ""
// Password can be set with it's md5 value as well
// Eg:  MD5 of "admin" = 21232f297a57a5a743894a0e4a801fc3
#define otaPasswordHash      ""

#define PIN_BUTTON           15
#define LONG_PRESS_TIME    6000  // 6s
#define SHRT_PRESS_TIME    3000  // 3s

#define ADC_INPUT0           34  // define the used ADC input channel
#define ADC_INPUT1           35  // define the used ADC input channel
#define ADC_INPUT2           32  // define the used ADC input channel

#define CLOCKSPEED           80  // clockspeed of ESP32 in MHz

#define nomVOLT             230  // nominal RMS voltage on the line
#define maxVOLT             253  // 230 + 10%
#define minVOLT             216  // 230 -  6%

#define nomFREQ             50.00 // nominal frequency
#define maxFREQ             50.01 // + 10mHz
#define minFREQ             49.99 // - 10mHz

//LINE 1 -> index 0
#define LINE0               true
#define ADCvolt0            (float)0.2375    // volt per ADC count

//LINE 2 -> index 1
#define LINE1               false
#define ADCvolt1            (float)0.2785    // volt per ADC count

//LINE 3 -> index 2
#define LINE2               false
#define ADCvolt2            (float)0.2475    // volt per ADC count
```

**MainsVLogger.ino**

```
//-----------------------------------------------------------------------------
// what/how to make it
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// includes
//-----------------------------------------------------------------------------
#include <Ticker.h>
#include <WiFi.h>
#include <ESPmDNS.h>
#include <WiFiUdp.h>
#include <NTPClient.h>
#include <PubSubClient.h>
#include <math.h>
#include <SPIFFS.h>
#include <WiFiSettings.h>
#include <ArduinoOTA.h>

#include "MainsVLogger.h"

//-----------------------------------------------------------------------------
// defines
//-----------------------------------------------------------------------------
#define VERSION            "1.50"

// webserver timeout in milliseconds (example: 2000ms = 2s)
#define webserverTimeout  1000

// max length of a MQTT topic
#define MQTT_TOPIC_SIZE    128

// max length of string
#define MAX_STRING_LENGTH  256

//-----------------------------------------------------------------------------
// constants
//-----------------------------------------------------------------------------
// number of samples to take before RMS is calculate
const int iADC = 2 * nomFREQ;    // one sample every 1 ms -> 100ms = 5 periods @ 50Hz
// number of Vrms to hold before average Vrms is outputted
const int iRMS =  50;            // 100 x 5 periods -> 100 * 100 ms -> 5 seconds -> every 5 seconds
MQTT message

//-----------------------------------------------------------------------------
// global variables
//-----------------------------------------------------------------------------
// time related
unsigned long time_now;
char today[11];
// to hold date & time
char *dt;
int days[] = {31,29,31,30,31,30,31,31,30,31,30,31};
char  nuDate[12];
char  nuTime[10];
char  currDST[3];
char  lastDST[3];
// Current timeing
unsigned long nowMicros  = micros();
unsigned long difMicros  = 0;
unsigned long currMillis = millis();
// Previous timing
unsigned long prevMillis = 0;

// vars related to button
unsigned long buttTime  = 0;
int           buttState;
int           buttLast  = HIGH;      // the previous state from the input pin
int           buttCurr  = HIGH;      // the current  state from the input pin

// vars related to MQTT
boolean bMQTT          = true;
String  mqttServer     = "mqtt.home";
int     mqttPort       = 1883;
String  mqttLogin      = "";
String  mqttPassword   = "";
String  mqttTopicBase  = myHostname;

char mqttClientID[30];
```

```
char mqttTopicSts[MQTT_TOPIC_SIZE];
char mqttTopicDat[MQTT_TOPIC_SIZE];

// vars to store characters
char caTemp[MAX_STRING_LENGTH];
char caTmp[MAX_STRING_LENGTH];
char caJson[MAX_STRING_LENGTH * 2];

// counters for array index
int  cntADC = 0;
int  cntRMS = 0;

// whichs lines should be used for reading
boolean bLine[3] = {LINE0, LINE1, LINE2};

// vars related to ADC and Voltages
float ADCval;

unsigned long valZER[3][iADC];
unsigned long avgZER[3][iRMS];
float         ADCzero[3] = {2000, 2000, 2000};
unsigned int  valADC[3][iADC];
unsigned int  valRMS[3][iRMS];
float         valFRQ[iRMS];
unsigned int  avgRMS[3];
float         avgFRQ;
unsigned int  uUnderV[3] = {0, 0, 0};  // drops
unsigned int  uOverV[3]  = {0, 0, 0};  // spikes
unsigned int  uMaxV[3]   = {0, 0, 0};
unsigned int  uMinV[3]   = {999, 999, 999};
unsigned int  uDiffV[3]  = {0, 0, 0};

// vars related to webpage
unsigned int  wUnderV[3] = {0, 0, 0};  // drops
unsigned int  wOverV[3]  = {0, 0, 0};  // spikes
unsigned int  wMaxV[3]   = {0, 0, 0};
unsigned int  wMinV[3]   = {999, 999, 999};
unsigned int  wDiffV[3]  = {0, 0, 0};

// Frequency measurement
unsigned int stage         = 3;          // use for switching operation
                                         // stage = 0  => voltage still positive
                                         // stage = 1  => voltage has crossed 0 and start time is
recorded
                                         // stage = 2  => looking for peak, recording time
                                         // stage = 3  => timing not started

float PREval               = 0;          // is the previous value for voltage sensor
float sumPeriod            = 0;          // is the accumulate time differences for voltage wave

unsigned long TimeStart;                 // starting time for Phase Angle and Period (in micro
seconds).
unsigned long TimePeriod;                // current  time for period of wave (in micro seconds).


// needed for timer interrupt
volatile byte cntrINT;
hw_timer_t * timer    = NULL;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

// webserver related
// Variable to store the HTTP requests
String httpHeader;

//-----------------------------------------------------------------------------
// objects
//-----------------------------------------------------------------------------
// Initiate led blinker library
Ticker       ticker;

// WiFi objects
WiFiUDP      ntpUDP;
WiFiClient   wifiClient;

// NTP object
NTPClient    ntpClient(ntpUDP, ntpServer, ntpUTCOffset_sec, ntpUpdate_sec * 1000);

// MQTT object
PubSubClient mqttClient(wifiClient);
```

```
// object for webserver
// set web server port number to 80
WiFiServer  webServer(80);
WiFiClient  webClient;

//-----------------------------------------------------------------------------
// Functions
//-----------------------------------------------------------------------------

//.............................................................................
// Interrupt functions
//.............................................................................
//** Timer interrupt
//*****************************************************************************
void IRAM_ATTR onTimer()
{
  portENTER_CRITICAL_ISR(&timerMux);
  cntrINT++;
  portEXIT_CRITICAL_ISR(&timerMux);
}
//*****************************************************************************

//.............................................................................
// Call-back functions
//.............................................................................
//** Call-back TICKER
//*****************************************************************************
void tick()
{
  // get the current state of GPIO1 pin
  int state = digitalRead(LED_BUILTIN);
  // set pin to the opposite state
  digitalWrite(LED_BUILTIN, !state);
}
//*****************************************************************************

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// +++ WIFI Functions +++
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//** Portal for settings
//*****************************************************************************
void wifiSettings()
{
  // Format SPIFF if mount fails.
  // fail might happen at first time when nothing was done before
  SPIFFS.begin(true);

  WiFiSettings.begin();
  // default to ESP32- followed by last 3 pairs of HEX numbers of MAC address, reversed
  // eg:  ac:67:b2:37:8e:60 -> esp32-608e37
  WiFiSettings.hostname = myHostname;
  //WiFiSettings.language = "en";

  WiFiSettings.heading("MQTT");
  bMQTT         = WiFiSettings.checkbox("MQTT Active", bMQTT, "MQTT Active");
  mqttServer    = WiFiSettings.string("mqtt_server", 64, mqttServer, "MQTT Server");
  mqttPort      = WiFiSettings.integer("mqtt_port", 0, 65535, mqttPort, "MQTT Port");
  mqttLogin     = WiFiSettings.string("mqtt_login", 64, mqttLogin, "MQTT Login");
  mqttPassword  = WiFiSettings.string("mqtt_password", 64, mqttPassword, "MQTT Password");
  mqttTopicBase = WiFiSettings.string("mqtt_topic", 128, mqttTopicBase, "MQTT Topic Base");

  // Set custom callback functions
  WiFiSettings.onSuccess      = [](){};  // do nothing
  WiFiSettings.onConfigSaved  = [](){};  // do nothing
  WiFiSettings.onWaitLoop     = []()
  {
    // Delay next function call by 500ms
    return 500;
  };

  WiFiSettings.onConnect = []()
  {
    // Delay next function call by 50ms
    return 50;
  };

  WiFiSettings.onFailure = []()
  {
    // Delay 2s
```

```
    delay(2000);
  };

  static unsigned int  portal_phase = 0;
  static unsigned long portal_start;
  WiFiSettings.onPortal = [](){portal_start = millis();};

  WiFiSettings.onPortalView = [](){if (portal_phase < 2){portal_phase = 2;}};

  WiFiSettings.onConfigSaved = [](){portal_phase = 3;};

  WiFiSettings.onPortalWaitLoop = []()
  {
    // anyone connected?
    if (WiFi.softAPgetStationNum() == 0)
    {
      portal_phase = 0;
    }
    else
    {
      if (!portal_phase)
      {
        portal_phase = 1;
      }
    }

    // do not wait endless
    if (portal_phase == 0 && millis() - portal_start > 5*60*1000)
    {
      // reboot if nothing happened within 5 minutes = 5 * 60 * 1000 milliseconds
      ESP.restart();
    }
  };

  // try to connect to WiFi, if not, start settings portal
  if (WiFiSettings.connect(true, 15))
  {
    #if defined(DEBUG)
      Serial.println("Connected to Wifi Network");
    #endif
  }

}
//*****************************************************************************
// +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

// +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// +++ NTP/Time Functions +++
// +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//** Set-up NTP
//*****************************************************************************
void ntpSetup()
{
  #if defined(DEBUG)
    Serial.println("NTP setup");
  #endif

  // clear some vars
  memset(lastDST, 0x00, 3);
  memset(currDST, 0x00, 3);

  // Init NTP
  ntpClient.begin();

  String nu = ntpClient.getFormattedDate();
  #if defined(DEBUG)
    Serial.print("Init    ->  Date - Time: "); Serial.print(nu); Serial.println();
  #endif

  ntpClient.forceUpdate();

  nu = ntpClient.getFormattedDate();
  #if defined(DEBUG)
    Serial.print("Updated ->  Date - Time: "); Serial.print(nu); Serial.println();
  #endif

  //adjust DST if needed
  checkDST();
```

```
  nu = ntpClient.getFormattedDate();
  #if defined(DEBUG)
    Serial.print("Final   ->  Date - Time: "); Serial.print(nu); Serial.println();
  #endif

  #if defined(DEBUG)
    Serial.println("NTP activated");
  #endif
}
//****************************************************************************

//** Check for DST
//****************************************************************************
void checkDST()
{
  #if defined(DEBUG)
    Serial.println("DST Checking ....");
  #endif

  // due to problem/bug in time /timezone, the following code
  // is required to make sure we have the right offset for DST

  // make sure we have recently updated
  ntpClient.forceUpdate();

  // get date
  String nu = ntpClient.getFormattedDate().substring(0,19);
  #if defined(DEBUG)
    Serial.println(nu);
  #endif

  int y, m, w;
  // correct for leapyear
  y = nu.substring(0,4).toInt();
  days[1] -= (y % 4) || (!(y % 100) && (y % 400));
  w = y * 365 + 97 * (y - 1) / 400 + 4;

  char buff[20];
  // find last sunday of March
  m = 2; // March
  w = (w + days[m]) % 7;
  sprintf(buff, "%04d-%02d-%02dT02:00:00", y, m + 1,days[m] - w);
  String DSTstart(buff);
  #if defined(DEBUG)
    Serial.println(DSTstart);
  #endif

  // find last sunday of Octobre
  m = 9; // Octobre
  w = (w + days[m]) % 7;
  sprintf(buff, "%04d-%02d-%02dT03:00:00", y, m + 1,days[m] - w);
  String DSTend(buff);
  #if defined(DEBUG)
    Serial.println(DSTend);
  #endif

  // if nu is between March 02:00:00 and Oct 03:00:00
  // DST is active (at least till 2024)
  if(nu >= DSTstart && nu < DSTend)
  {
    // DST = UTC + ntpUTCOffset_sec + ntpDSTOffset_sec
    ntpClient.setTimeOffset(ntpUTCOffset_sec + ntpDSTOffset_sec);
    #if defined(DEBUG)
      Serial.println("DST set");
    #endif
  }
  else
  {
    // non-DST = UTC + ntpUTCOffset_sec
    ntpClient.setTimeOffset(ntpUTCOffset_sec);
    #if defined(DEBUG)
      Serial.println("DST reset");
    #endif
  }

  // make sure we update local date/time
  ntpClient.forceUpdate();

  // save last check of DST
  sprintf(lastDST, "%02d", ntpClient.getHours());
```

```
  sprintf(currDST, "%02d", ntpClient.getHours());
  ntpClient.getFormattedTime().toCharArray(nuTime, 8);
  nuTime[9] = 0x00;

  #if defined(DEBUG)
    Serial.println("DST Checking done");
  #endif
}

//** Get Date & Time
//*****************************************************************************
char* ntpLocalDateTime()
{
  static char caDT[21];

  String nu = ntpClient.getFormattedDate().substring(0,19);
  // convert to char array
  nu.toCharArray(caDT, 20);
  // remove T
  caDT[10] = ' ';
  //terminate to string
  caDT[20] = 0x00;

  return(caDT);
}
//*****************************************************************************
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// +++ mDNS Functions +++
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//** Set-up mDNS
//*****************************************************************************
void mdnsSetup()
{
  #if defined(DEBUG)
    Serial.println(F("mDNS activating"));
  #endif

  if(!MDNS.begin(myHostname))
  {
    #if defined(DEBUG)
      Serial.println(F("mDNS activation FAILED"));
    #endif
    return;
  }

  // indicate there is a webserver on this device
  MDNS.addService("http", "tcp", 80);

  #if defined(DEBUG)
    Serial.println(F("mDNS activated"));
  #endif
}
//*****************************************************************************
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// +++ MQTT Functions +++
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//** Set-up MQTT
//*****************************************************************************
void mqttConnect()
{

  #if defined(DEBUG)
    Serial.println(F("MQTT activating"));
    Serial.print(F("MQTT attempting connection to ")); Serial.print(mqttServer);
Serial.print(F(":")); Serial.println(mqttPort);
  #endif

  // mqttClientID
  strcpy(mqttClientID, myHostname);
  #if defined(DEBUG)
    Serial.print(F("  MQTT ClientID: <")); Serial.print(mqttClientID); Serial.println(F(">"));
  #endif

  // mqttTopicSts
```

```
  strcpy(mqttTopicSts, mqttTopicBase.c_str());
  strcat(mqttTopicSts, "/status");
  #if defined(DEBUG)
    Serial.print(F("  MQTT Topic STS: <")); Serial.print(mqttTopicSts); Serial.println(F(">"));
  #endif

  // mqttTopicDat
  strcpy(mqttTopicDat, mqttTopicBase.c_str());
  strcat(mqttTopicDat, "/data");
  #if defined(DEBUG)
    Serial.print(F("  MQTT Topic DAT: <")); Serial.print(mqttTopicDat); Serial.println(F(">"));
  #endif

  mqttClient.setServer(mqttServer.c_str(), mqttPort);

  // ESP will connect to mqtt broker with clientID, last will, ...
  // bool connect(const char* id, const char* user, const char* pass, const char* willTopic, uint8_t
willQos, bool willRetain, const char* willMessage, bool cleanSession);
  if (mqttClient.connect(mqttClientID, mqttLogin.c_str(), mqttPassword.c_str(), mqttTopicSts, 1,
true, "offline", false))
  {

    #if defined(DEBUG)
      Serial.println(F("MQTT connected"));
    #endif

    // let know we are online
    mqttClient.publish(mqttTopicSts, "online", true);
    #if defined(DEBUG)
      Serial.println(F("MQTT activated"));
    #endif
  }
  else
  {
    #if defined(DEBUG)
      Serial.println(F("MQTT activation Failed"));
    #endif
  }

}
//****************************************************************************

//****************************************************************************
void mqttReconnect()
{
  uint8_t i = 0;

  while (!mqttClient.connected())
  {
    #if defined(DEBUG)
      Serial.print(F("MQTT attempting reconnection to ")); Serial.print(mqttServer);
Serial.print(F(":")); Serial.println(mqttPort);
    #endif
    if (mqttClient.connect(mqttClientID, mqttLogin.c_str(), mqttPassword.c_str(), mqttTopicSts, 1,
true, "offline", false))
    {
      #if defined(DEBUG)
        Serial.println(F("MQTT reconnected"));
      #endif
      // let know we are online
      mqttClient.publish(mqttTopicSts, "online", true);
    }
    else
    {
      #if defined(DEBUG)
        Serial.print(F("MQTT reconnection failed, rc=")); Serial.print(mqttClient.state());
Serial.println(F(" ... try again in 5 seconds"));
      #endif
      // Wait 15 seconds before retrying
      // delay non-blocking
      time_now = millis();
      while(millis() < time_now + 1500){}
      ++i;
      if (i > 5)
      {
        #if defined(DEBUG)
          Serial.println(F("Restarting..."));
        #endif
        ESP.restart();
      }
```

```
    }
  }
}
//****************************************************************************

// ** send MQTT messages
//****************************************************************************
void mqttSend(char *topic, char* data)
{
  mqttClient.publish(topic, data, true);
}
//****************************************************************************
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// +++ OTA Functions +++
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//** Set-up OTA
//****************************************************************************
void otaSetup()
{
  #if defined(DEBUG)
    Serial.println("OTA activating");
  #endif

  // Port defaults to 3232
  ArduinoOTA.setPort(otaPort);

  // Hostname defaults to esp32-[MAC]
  ArduinoOTA.setHostname(myHostname);

  // Set OTA Password. No authentication by default
  if(strlen(otaPasswordHash) != 0)
  {
    ArduinoOTA.setPasswordHash(otaPasswordHash);
  }
  else if(strlen(otaPassword) != 0)
  {
    ArduinoOTA.setPassword(otaPassword);
  }

  ArduinoOTA
    .onStart([]()
    {
      String type;
      if(ArduinoOTA.getCommand() == U_FLASH)
        type = "sketch";
      else // U_SPIFFS
        type = "filesystem";

      // NOTE: if updating SPIFFS this would be the place to unmount SPIFFS using SPIFFS.end()
      #if defined(DEBUG)
        Serial.println(" - Start updating " + type);
      #endif
    })

    .onProgress([](unsigned int progress, unsigned int total)
    {
      #if defined(DEBUG)
        Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
      #endif
    })

    .onEnd([]()
    {
      #if defined(DEBUG)
        Serial.print("\n"); Serial.println(" - End");
      #endif
    })

    .onError([](ota_error_t error)
    {
      #if defined(DEBUG)
        Serial.printf(" - Error[%u]: ", error);
        if(error == OTA_AUTH_ERROR)         Serial.println("Auth Failed");
        else if(error == OTA_BEGIN_ERROR)   Serial.println("Begin Failed");
        else if(error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
        else if(error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
        else if(error == OTA_END_ERROR)     Serial.println("End Failed");
```

```
      #endif
    }

  );

  ArduinoOTA.begin();
  #if defined(DEBUG)
    Serial.println("OTA activated");
  #endif
}
//*****************************************************************************
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// +++ Button Functions +++
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//** Button logic long press to activate Settings Portal or to restart
//*****************************************************************************
unsigned int Button()
{
  // read the state of the button:
  buttCurr = digitalRead(PIN_BUTTON);

  // Is button is pressed or released?
  if(buttLast == HIGH && buttCurr == LOW)
  {
    buttLast = buttCurr;
    #if defined(DEBUG)
      Serial.println("Portal Button: Pressed");
    #endif
    buttTime = millis();
  }
  else
  {
    if(buttLast == LOW && buttCurr == HIGH)
    {
      buttLast = buttCurr;
      #if defined(DEBUG)
        Serial.println("Portal Button: Released");
      #endif
      long pressDuration = millis() - buttTime;

      if( pressDuration > LONG_PRESS_TIME )
      {
        #if defined(DEBUG)
          Serial.println("Portal Button: A long press is detected");
        #endif
        return LONG_PRESS_TIME;
      }
      else
      {
        if ( pressDuration > SHRT_PRESS_TIME )
        {
          #if defined(DEBUG)
            Serial.println("Portal Button: A short press is detected");
          #endif
          return SHRT_PRESS_TIME;
        }
        else
        {
          #if defined(DEBUG)
            Serial.println("Portal Button: A very short press is detected");
          #endif
          return 0;
        }
      }
    }
  }
  buttLast = buttCurr;
  return false;
}
//*****************************************************************************
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// +++ Webserver Functions +++
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//** Set-up webserver
//*****************************************************************************
```

```
void wsSetup()
{
  #if defined(DEBUG)
    Serial.println("Webserver activating");
  #endif

  webServer.begin();

  #if defined(DEBUG)
    Serial.println("Webserver activated");
  #endif
}
//****************************************************************************

//** listen & talk to webclient
//****************************************************************************
void wsClientActive()
{
  char c;

  currMillis = millis();
  prevMillis = currMillis;
  // print a message out in the serial port
  #if defined(DEBUG)
    Serial.println("Client connected");
  #endif

  // make a String to hold incoming data from the webClient
  String currentLine = "";

  // loop while the webClient's connected and no timeout occured
  while (webClient.connected() && currMillis - prevMillis <= webserverTimeout)
  {
    currMillis = millis();
    // if there's bytes to read from the webClient,
    if(webClient.available())
    {
      c = webClient.read();           // read a byte, then
      #if defined(DEBUGWEB)
        Serial.write(c);              // print it out the serial monitor
      #endif
      httpHeader += c;                // add it to httpHeader var

      // if the byte is a newline character
      if(c == '\n')
      {
        #if defined(DEBUGWEB)
          Serial.println("New line");
        #endif

        // if the current line is blank, you got two newlines in a row.
        // that's the end of the webClient HTTP request
        // check what was sent, act as required and then send a response:
        if(currentLine.length() == 0)
        {
          // first send response as HTTP headers always start with a response code
          // and a content-type so the webClient knows what's coming,
          webClient.println("HTTP/1.1 200 OK");
          webClient.println("Content-type:text/html");
          webClient.println("Connection: close");
          // then a blank line:
          webClient.println("");

          // decode if action is required
          // action on home page?
          if(httpHeader.indexOf("GET /HOME") >= 0)
          {
            webHomePage();
            break;
          }
          else if(httpHeader.indexOf("GET /SELECT") >= 0)
          {
            if(httpHeader.indexOf("GET /SELECT?") >= 0)
            {
              if(httpHeader.indexOf("line0=on") >=0)
              {
                bLine[0] = true;
              }
              else
              {
```

```
                bLine[0] = false;
              }
              if(httpHeader.indexOf("line1=on") >=0)
              {
                bLine[1] = true;
              }
              else
              {
                bLine[1] = false;
              }
              if(httpHeader.indexOf("line2=on") >=0)
              {
                bLine[2] = true;
              }
              else
              {
                bLine[2] = false;
              }
              webHomePage();
              break;
            }
            webSelectPage();
            break;
          }
          else if(httpHeader.indexOf("GET /CHART") >= 0)
          {
            // send home page
            webChartPage();
            break;
          }
          else if(httpHeader.indexOf("GET /line") >= 0)
          {
            // send home page
            webLineData();
            break;
          }
          // send home page
          webHomePage();
          break;
        }
        else
        {
          // if you got a newline, then clear currentLine
          currentLine = "";
        }
      }
      else if(c != '\r')
      {
        // if you got anything else but a carriage return character,
        // add it to the end of the currentLine
        currentLine += c;
      }
      else if(c == '\r')
      {
        #if defined(DEBUGWEB)
          Serial.println("New return");
        #endif
      }
    }
  }

  #if defined(DEBUGWEB)
    Serial.print("<"); Serial.print(httpHeader); Serial.println(">");
  #endif

  // Clear the httpHeader variable
  httpHeader = "";

  // now queue is empty, stop it
  webClient.stop();
  #if defined(DEBUG)
    Serial.println("Client disconnected.");
  #endif
}
//****************************************************************************

//** send home page
//****************************************************************************
void webHomePage()
{
```

```
  // Display the HTML web page
  webClient.println("<!DOCTYPE html>");
  // Open the HTML web page
  webClient.println("<html>");
  // Open head
  webClient.println("<head><meta name=\"viewport\" content=\"width=device-width, initial-
scale=1\">");
  webClient.println("<meta http-equiv=\"refresh\" content=\"60\">");
  webClient.println("<link rel=\"icon\" href=\"data:,\">");
  // Open style for CSS settings
  webClient.println("<style>");
  // overall style setting
  webClient.println("html, body { width: 100%; height: 100%; padding: 0; margin: 0; font-
family: \"Verdana\", sans-serif;}");
  webClient.println(".centered-wrapper { position: relative; text-align: center; font-
family: \"Verdana\", sans-serif;}");
  webClient.println(".centered-wrapper:before { content: ""; position: relative; display: inline-
block; width: 0; height: 100%; vertical-align: middle; font-family: \"Verdana\", sans-serif;}");
  webClient.println(".centered-content { display: inline-block; vertical-align: middle; font-family:
\"Verdana\", sans-serif;}");
  webClient.println(".btn { background-color: #0099ff; border: none; color: white; width: 200px;
height: 50px; text-decoration: none; font-size: 20px; margin: 2px; cursor: pointer;}");
  webClient.println("</style></head>");

  // body of page
  webClient.println("<body class=\"centered-wrapper\">");
  webClient.println("<div class=\"centered-content\">");
  webClient.print("<h1>"); webClient.print(myHostname); webClient.println("</h1>");
  webClient.print("<table style='font-family: Courier, monospace; font-size: 20px'>");
  webClient.print("<tr>");
  dt = ntpLocalDateTime();
  sprintf(caTemp, "%s           ", dt +
11);
  webClient.print("<td><label><b>Time    :</b></
label><label> ");webClient.print(caTemp);webClient.print("</label></td>");
  webClient.print("</tr><tr>");
  webClient.print("<td><label>         </
label><label>  L1 -  L2 -
  L3     </label></td>");
  webClient.print("</tr><tr>");
  sprintf(caTmp, "%lu", avgRMS[0]); str_replace(caTmp, (char*)" ", (char*)" "); strcpy(caTemp,
caTmp); strcat(caTemp, " -");
  sprintf(caTmp, "%lu", avgRMS[1]); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp); strcat(caTemp, " -");
  sprintf(caTmp, "%lu", avgRMS[2]); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp);

webClient.print("<td><label><b>AvgVolt :</b></label><label>");webClient.print(caTemp);webClient
.print(" Vrms</label></td>");
  webClient.print("</tr><tr>");
  sprintf(caTmp, "%u", wMaxV[0]); str_replace(caTmp, (char*)" ", (char*)" "); strcpy(caTemp,
caTmp); strcat(caTemp, " -");
  sprintf(caTmp, "%u", wMaxV[1]); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp); strcat(caTemp, " -");
  sprintf(caTmp, "%u", wMaxV[2]); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp);

webClient.print("<td><label><b>MaxVolt :</b></label><label>");webClient.print(caTemp);webClient
.print(" Vrms</label></td>");
  webClient.print("</tr><tr>");
  sprintf(caTmp, "%u", wMinV[0]); str_replace(caTmp, (char*)" ", (char*)" "); strcpy(caTemp,
caTmp); strcat(caTemp, " -");
  sprintf(caTmp, "%u", wMinV[1]); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp); strcat(caTemp, " -");
  sprintf(caTmp, "%u", wMinV[2]); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp);

webClient.print("<td><label><b>MinVolt :</b></label><label>");webClient.print(caTemp);webClient
.print(" Vrms</label></td>");
  webClient.print("</tr><tr>");
  sprintf(caTmp, "%u", wOverV[0]); str_replace(caTmp, (char*)" ", (char*)" "); strcpy(caTemp,
caTmp); strcat(caTemp, " -");
  sprintf(caTmp, "%u", wOverV[1]); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp); strcat(caTemp, " -");
  sprintf(caTmp, "%u", wOverV[2]); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp);

webClient.print("<td><label><b>Overrun :</b></label><label>");webClient.print(caTemp);webClient
.print("     </label></td>");
  webClient.print("</tr><tr>");
```

```
  sprintf(caTmp, "%u", wUnderV[0]); str_replace(caTmp, (char*)" ", (char*)" "); strcpy(caTemp,
caTmp); strcat(caTemp, " -");
  sprintf(caTmp, "%u", wUnderV[1]); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp); strcat(caTemp, " -");
  sprintf(caTmp, "%u", wUnderV[2]); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp);

webClient.print("<td><label><b>Underrun:</b></label><label>");webClient.print(caTemp);webClient.prin
t("     </label></td>");
  webClient.print("</tr><tr>");
  webClient.print("<td> </td><td> </td><td> </td>");
  webClient.print("</tr>");
  webClient.print("</table>");
  webClient.print("<p><a href=\"/HOME\"><button class=\"btn\">Refresh</button></p>");
  webClient.print("<p><a href=\"/CHART\"><button class=\"btn\">Chart</button></p>");
  webClient.print("<p><a href=\"/SELECT\"><button class=\"btn\">Select Lines</button></p>");
  webClient.println("</div>");
  webClient.println("</body></html>");

  // The HTTP response ends with another blank line
  webClient.println("");
}
//*****************************************************************************

//** send select line page
//*****************************************************************************
void webSelectPage()
{
  // Display the HTML web page
  webClient.println("<!DOCTYPE html>");
  // Open the HTML web page
  webClient.println("<html>");
  // Open head
  webClient.println("<head><meta name=\"viewport\" content=\"width=device-width, initial-
scale=1\">");
  webClient.println("<link rel=\"icon\" href=\"data:,\">");
  // Open style for CSS settings
  webClient.println("<style>");
  // overall style setting
  webClient.println("html, body { width: 100%; height: 100%; padding: 0; margin: 0; font-
family: \"Verdana\", sans-serif;}");
  webClient.println(".centered-wrapper { position: relative; text-align: center; font-
family: \"Verdana\", sans-serif;}");
  webClient.println(".centered-wrapper:before { content: ""; position: relative; display: inline-
block; width: 0; height: 100%; vertical-align: middle; font-family: \"Verdana\", sans-serif;}");
  webClient.println(".centered-content { display: inline-block; vertical-align: middle; font-family:
\"Verdana\", sans-serif;}");
  webClient.println(".btn { background-color: #0099ff; border: none; color: white; width: 200px;
height: 50px; text-decoration: none; font-size: 20px; margin: 2px; cursor: pointer;}");
  webClient.println("</style></head>");

  //body of page
  webClient.println("<body class=\"centered-wrapper\">");
  webClient.println("<div class=\"centered-content\">");
  webClient.print("<h1>"); webClient.print(myHostname); webClient.println("</h1>");
  webClient.print("<form><table style='font-family: \"Verdana\", Courier, monospace; font-size:
20px'>");
  webClient.print("<tr><td><label>Line 1:</label> <input type=\"checkbox\" name=\"line0\" ");
webClient.printf("%s", bLine[0] ? "checked" : "unchecked"); webClient.print("></td></tr>");
  webClient.print("<tr><td><label>Line 2:</label> <input type=\"checkbox\" name=\"line1\" ");
webClient.printf("%s", bLine[1] ? "checked" : "unchecked"); webClient.print("></td></tr>");
  webClient.print("<tr><td><label>Line 3:</label> <input type=\"checkbox\" name=\"line2\" ");
webClient.printf("%s", bLine[2] ? "checked" : "unchecked"); webClient.print("></td></tr>");
  webClient.print("<tr><td> </td></tr>");
  webClient.print("<tr><td><input class=\"btn\" type=\"submit\" value=\"Save\"
formmethod=\"get\"></td></tr>");
  webClient.print("<tr><td> </td></tr>");
  webClient.print("<tr><td><a href=\"/HOME\"><button class=\"btn\">Home</button></td></tr>");
  webClient.print("</table></form>");
  webClient.print("</div>");
  webClient.println("</body></html>");

  // The HTTP response ends with another blank line
  webClient.println("");
}
//*****************************************************************************

//** send chart page
//*****************************************************************************
void webChartPage()
```

```
{
  webClient.println("<!DOCTYPE HTML><html><head>");
  webClient.println("<meta name='viewport' content='width=device-width, initial-scale=1'>");
  webClient.println("<script src='https://code.highcharts.com/highcharts.js'></script>");
  webClient.println("<style>html, body{width: 100%; height: 100%; padding: 0; margin: 0; font-
family: \"Verdana\", sans-serif;}");
  webClient.println(".centered-wrapper { position: relative; text-align: center; font-
family: \"Verdana\", sans-serif;}");
  webClient.println(".centered-wrapper:before { content: ""; position: relative; display: inline-
block; width: 0; height: 100%; vertical-align: middle; font-family: \"Verdana\", sans-serif;}");
  webClient.println(".centered-content { display: inline-block; vertical-align: middle; font-family:
\"Verdana\", sans-serif;}");
  webClient.println(".btn { background-color: #0099ff; border: none; color: white;  width: 200px;
height: 50px; text-decoration: none; font-size: 20px; margin: 2px; cursor: pointer;}");
  webClient.println("</style></head>");
  webClient.println("<body class=\"centered-wrapper\">");
  webClient.println("<div class=\"centered-content\">");
  webClient.print("<h1>"); webClient.print(myHostname); webClient.println("</h1>");
  webClient.println("<div id='chart-AVG' class='container'></div>");
  webClient.println("<div id='chart-MIN' class='container'></div>");
  webClient.println("<div id='chart-MAX' class='container'></div>");
  webClient.println("<div id='chart-UND' class='container'></div>");
  webClient.println("<div id='chart-OVR' class='container'></div>");
  webClient.println("<p><a href=\"/HOME\"><button class=\"btn\">Home</button></p>");
  webClient.println("</div></body><script>");
  webClient.println("var chartAVG= new Highcharts.Chart({chart:{renderTo : 'chart-AVG'},");
  webClient.println("title:{text: 'Avg Line Voltage'},");
  webClient.println("series:[{name: 'L1', color: '#6e3e00', showInLegend: true, data:[]},");
  webClient.println("{name: 'L2', color: '#000000', showInLegend: true, data:[]},");
  webClient.println("{name: 'L3', color: '#7f7f7f', showInLegend: true, data:[]}],");
  webClient.println("plotOptions:{line: {animation: false, dataLabels:{enabled: true}}},");
  webClient.println("xAxis:{type: 'datetime', dateTimeLabelFormats:{second: '%H:%M:%S'}},");
  webClient.println("yAxis:{title:{text: 'Vrms'}},credits:{enabled: false}});");
  webClient.println("var chartMIN= new Highcharts.Chart({chart:{renderTo : 'chart-MIN'},");
  webClient.println("title:{text: 'Min Line Voltage'},");
  webClient.println("series:[{name: 'L1', color: '#6e3e00', showInLegend: true, data:[]},");
  webClient.println("{name: 'L2', color: '#000000', showInLegend: true, data:[]},");
  webClient.println("{name: 'L3', color: '#7f7f7f', showInLegend: true, data:[]}],");
  webClient.println("plotOptions:{line: {animation: false, dataLabels:{enabled: true}}},");
  webClient.println("xAxis:{type: 'datetime', dateTimeLabelFormats:{second: '%H:%M:%S'}},");
  webClient.println("yAxis:{title:{text: 'Vrms'}},credits:{enabled: false}});");
  webClient.println("var chartMAX= new Highcharts.Chart({chart:{renderTo : 'chart-MAX'},");
  webClient.println("title:{text: 'Max Line Voltage'},");
  webClient.println("series:[{name: 'L1', color: '#6e3e00', showInLegend: true, data:[]},");
  webClient.println("{name: 'L2', color: '#000000', showInLegend: true, data:[]},");
  webClient.println("{name: 'L3', color: '#7f7f7f', showInLegend: true, data:[]}],");
  webClient.println("plotOptions:{line: {animation: false, dataLabels:{enabled: true}}},");
  webClient.println("xAxis:{type: 'datetime', dateTimeLabelFormats:{second: '%H:%M:%S'}},");
  webClient.println("yAxis:{title:{text: 'Vrms'}},credits:{enabled: false}});");
  webClient.println("var chartUND= new Highcharts.Chart({chart:{renderTo : 'chart-UND'},");
  webClient.println("title:{text: 'Underuns'},");
  webClient.println("series:[{name: 'L1', color: '#6e3e00', showInLegend: true, data:[]},");
  webClient.println("{name: 'L2', color: '#000000', showInLegend: true, data:[]},");
  webClient.println("{name: 'L3', color: '#7f7f7f', showInLegend: true, data:[]}],");
  webClient.println("plotOptions:{line: {animation: false, dataLabels:{enabled: true}}},");
  webClient.println("xAxis:{type: 'datetime', dateTimeLabelFormats:{second: '%H:%M:%S'}},");
  webClient.println("yAxis:{title:{text: '#'}},credits:{enabled: false}});");
  webClient.println("var chartOVR= new Highcharts.Chart({chart:{renderTo : 'chart-OVR'},");
  webClient.println("title:{text: 'Overruns'},");
  webClient.println("series:[{name: 'L1', color: '#6e3e00', showInLegend: true, data:[]},");
  webClient.println("{name: 'L2', color: '#000000', showInLegend: true, data:[]},");
  webClient.println("{name: 'L3', color: '#7f7f7f', showInLegend: true, data:[]}],");
  webClient.println("plotOptions:{line: {animation: false, dataLabels:{enabled: true}}},");
  webClient.println("xAxis:{type: 'datetime', dateTimeLabelFormats:{second: '%H:%M:%S'}},");
  webClient.println("yAxis:{title:{text: '#'}},credits:{enabled: false}});");
  webClient.println("setInterval(function(){");
  webClient.println("var xhttp = new XMLHttpRequest();");
  webClient.println("xhttp.onreadystatechange = function(){");
  webClient.println("if(this.readyState == 4 && this.status == 200){");
  webClient.println("var myObj = JSON.parse(this.responseText);");
  webClient.println("var x =(new Date(myObj.time)).getTime();");
  webClient.println("if(myObj.hasOwnProperty('L1')){var r = 0;");
  webClient.println("var yavg = parseFloat(myObj.L1.avg),");
  webClient.println("ymin = parseFloat(myObj.L1.min),ymax = parseFloat(myObj.L1.max),");
  webClient.println("yund = parseFloat(myObj.L1.under),yovr = parseFloat(myObj.L1.over);");
  webClient.println("if(chartAVG.series[r].data.length > 40){");
  webClient.println("chartAVG.series[r].addPoint([x, yavg], true, true, true);");
  webClient.println("} else{chartAVG.series[r].addPoint([x, yavg], true, false, true);}");
  webClient.println("if(chartMIN.series[r].data.length > 40){");
  webClient.println("chartMIN.series[r].addPoint([x, ymin], true, true, true);");
```

```
  webClient.println("} else{chartMIN.series[r].addPoint([x, ymin], true, false, true);}");
  webClient.println("if(chartMAX.series[r].data.length > 40){");
  webClient.println("chartMAX.series[r].addPoint([x, ymax], true, true, true);");
  webClient.println("} else{chartMAX.series[r].addPoint([x, ymax], true, false, true);}");
  webClient.println("if(chartUND.series[r].data.length > 40){");
  webClient.println("chartUND.series[r].addPoint([x, yund], true, true, true);");
  webClient.println("} else{chartUND.series[r].addPoint([x, yund], true, false, true);}");
  webClient.println("if(chartOVR.series[r].data.length > 40){");
  webClient.println("chartOVR.series[r].addPoint([x, yovr], true, true, true);");
  webClient.println("} else{chartOVR.series[r].addPoint([x, yovr], true, false, true);}}");
  webClient.println("if(myObj.hasOwnProperty('L2')){var r = 1;");
  webClient.println("var yavg = parseFloat(myObj.L2.avg),");
  webClient.println("ymin = parseFloat(myObj.L2.min),ymax = parseFloat(myObj.L2.max),");
  webClient.println("yund = parseFloat(myObj.L2.under),yovr = parseFloat(myObj.L2.over);");
  webClient.println("if(chartAVG.series[r].data.length > 40){");
  webClient.println("chartAVG.series[r].addPoint([x, yavg], true, true, true);");
  webClient.println("} else{chartAVG.series[r].addPoint([x, yavg], true, false, true);}");
  webClient.println("if(chartMIN.series[r].data.length > 40){");
  webClient.println("chartMIN.series[r].addPoint([x, ymin], true, true, true);");
  webClient.println("} else{chartMIN.series[r].addPoint([x, ymin], true, false, true);}");
  webClient.println("if(chartMAX.series[r].data.length > 40){");
  webClient.println("chartMAX.series[r].addPoint([x, ymax], true, true, true);");
  webClient.println("} else{chartMAX.series[r].addPoint([x, ymax], true, false, true);}");
  webClient.println("if(chartUND.series[r].data.length > 40){");
  webClient.println("chartUND.series[r].addPoint([x, yund], true, true, true);");
  webClient.println("} else{chartUND.series[r].addPoint([x, yund], true, false, true);}");
  webClient.println("if(chartOVR.series[r].data.length > 40){");
  webClient.println("chartOVR.series[r].addPoint([x, yovr], true, true, true);");
  webClient.println("} else{chartOVR.series[r].addPoint([x, yovr], true, false, true);}}");
  webClient.println("if(myObj.hasOwnProperty('L3')){var r = 2;");
  webClient.println("var yavg = parseFloat(myObj.L3.avg),");
  webClient.println("ymin = parseFloat(myObj.L3.min),ymax = parseFloat(myObj.L3.max),");
  webClient.println("yund = parseFloat(myObj.L3.under),yovr = parseFloat(myObj.L3.over);");
  webClient.println("if(chartAVG.series[r].data.length > 40){");
  webClient.println("chartAVG.series[r].addPoint([x, yavg], true, true, true);");
  webClient.println("} else{chartAVG.series[r].addPoint([x, yavg], true, false, true);}");
  webClient.println("if(chartMIN.series[r].data.length > 40){");
  webClient.println("chartMIN.series[r].addPoint([x, ymin], true, true, true);");
  webClient.println("} else{chartMIN.series[r].addPoint([x, ymin], true, false, true);}");
  webClient.println("if(chartMAX.series[r].data.length > 40){");
  webClient.println("chartMAX.series[r].addPoint([x, ymax], true, true, true);");
  webClient.println("} else{chartMAX.series[r].addPoint([x, ymax], true, false, true);}");
  webClient.println("if(chartUND.series[r].data.length > 40){");
  webClient.println("chartUND.series[r].addPoint([x, yund], true, true, true);");
  webClient.println("} else{chartUND.series[r].addPoint([x, yund], true, false, true);}");
  webClient.println("if(chartOVR.series[r].data.length > 40){");
  webClient.println("chartOVR.series[r].addPoint([x, yovr], true, true, true);");
  webClient.println("} else{chartOVR.series[r].addPoint([x, yovr], true, false, true);}}}};");
  webClient.println("xhttp.open('GET', '/line', true);xhttp.send();}, 10000 );");
  webClient.println("</script></html>");

  // The HTTP response ends with another blank line
  webClient.println("");
}
//*****************************************************************************

//** send line data
//*****************************************************************************
void webLineData()
{
  dt = ntpLocalDateTime();
  strcpy(caJson, "{");
  sprintf(caTemp, "\"time\": \"%s+00\"", dt);
  strcat(caJson, caTemp);

  for (int r=0; r<3; r++)
  {
    if(bLine[r])
    {
      sprintf(caTemp, ", \"L%d\":{" , r+1);
      strcat(caJson, caTemp);
      sprintf(caTemp, "\"avg\": %lu, " , avgRMS[r]);
      strcat(caJson, caTemp);
      sprintf(caTemp, "\"min\": %u, "  , wMinV[r]);
      strcat(caJson, caTemp);
      sprintf(caTemp, "\"max\": %u, "  , wMaxV[r]);
      strcat(caJson, caTemp);
      sprintf(caTemp, "\"under\": %u, ", wUnderV[r]);
      strcat(caJson, caTemp);
      sprintf(caTemp, "\"over\": %u"   , wOverV[r]);
```

```
      strcat(caJson, caTemp);
      strcat(caJson, "}");
    }
  }
  strcat(caJson, "}");

  #if defined(DEBUG)
    Serial.println(caJson);
  #endif

  webClient.println(caJson);
  // The HTTP response ends with another blank line
  webClient.println("");
}
//****************************************************************************

//** replace characters in string and pad with spaces
//****************************************************************************
void str_replace(char *src, char *oldchars, char *newchars)
{ // utility string function
  String tmp;

  tmp = src;
  while(tmp.length() < 4)
  {
    tmp = " " + tmp;
  }
  strcpy(src, tmp.c_str());

  char *p = strstr(src, oldchars);
  char buf[MAX_STRING_LENGTH];
  do
  {
    if (p)
    {
      memset(buf, '\0', strlen(buf));
      if (src == p)
      {
        strcpy(buf, newchars);
        strcat(buf, p + strlen(oldchars));
      }
      else
      {
        strncpy(buf, src, strlen(src) - strlen(p));
        strcat(buf, newchars);
        strcat(buf, p + strlen(oldchars));
      }
      memset(src, '\0', strlen(src));
      strcpy(src, buf);
    }
  } while (p && (p = strstr(src, oldchars)));

}
//****************************************************************************
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//============================================================================
// Setup
//============================================================================
void setup()
{
  #if defined(CALIBRATE)
    Serial.begin(115200);
    delay(1000);
    Serial.println(F(""));Serial.println(F(""));
    Serial.println(F("======================="));
    Serial.print(F("Booting (Version: ")); Serial.print(VERSION); Serial.println(F(")"));
    Serial.println(F("======================="));

    //--------------------------------------------------------------------------
    // Startup timer interrupt
    //--------------------------------------------------------------------------
    //set prescaler every microsecond eg: clockspeed 80.000.000Hz / 80 = 1.000.000Hz -> 1 µs period
    timer = timerBegin(0, CLOCKSPEED, true);
    timerAttachInterrupt(timer, &onTimer, true);
    // interrupt every 1 millisecond = 1000 µs
    timerAlarmWrite(timer, 1000, true);
    timerAlarmEnable(timer);

  #else
```

```
#if defined(DEBUG)
  Serial.begin(115200);
  delay(1000);
  Serial.println(F(""));Serial.println(F(""));
  Serial.println(F("======================="));
  Serial.print(F("Booting (Version: ")); Serial.print(VERSION); Serial.println(F(")"));
  Serial.println(F("======================="));
#endif

// Set led pin as output
pinMode(LED_BUILTIN, OUTPUT);
// Start ticker to indicate set-up mode
ticker.attach(0.5, tick);

//-----------------------------------------------------------------------------
// Set Setting Portal button Pin as input
//-----------------------------------------------------------------------------
pinMode(PIN_BUTTON, INPUT_PULLUP);

//-----------------------------------------------------------------------------
// Allow time to press button to reset to factory defaults
//-----------------------------------------------------------------------------
delay(4000);

// check if button is pressed to restore factory defaults
buttTime = millis();
// Is button pressed?
while(digitalRead(PIN_BUTTON) == LOW)
{
  delay(100);
}
long pressDuration = millis() - buttTime;
if( pressDuration > LONG_PRESS_TIME )
{
  #if defined(DEBUG)
    Serial.println("Portal Button: A long press is detected");
    Serial.println("Formatting SPIFFS");
  #endif
  SPIFFS.format();
  delay(3000);
  #if defined(DEBUG)
    Serial.println("Restarting ...");
  #endif
  ESP.restart();
}

//-----------------------------------------------------------------------------
// Startup WiFi
//-----------------------------------------------------------------------------
WiFiSettings.hostname = myHostname;
wifiSettings();

#if defined(DEBUG)
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.print("  MAC Address: "); Serial.println( WiFi.macAddress().c_str());
  Serial.print("  IP Address : "); Serial.println(WiFi.localIP());
  Serial.print("  Hostname   : "); Serial.println(WiFi.getHostname());
#endif
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Startup WiFi AP
//-----------------------------------------------------------------------------
WiFi.mode(WIFI_MODE_APSTA);
WiFi.softAP(wifiAPSSID, wifiAPPassword);

//-----------------------------------------------------------------------------
// Startup time
//-----------------------------------------------------------------------------
ntpSetup();

//-----------------------------------------------------------------------------
// Startup MDNS
//-----------------------------------------------------------------------------
mdnsSetup();

//-----------------------------------------------------------------------------
// Startup OTA
```

```
    //-----------------------------------------------------------------------
    otaSetup();

    //-----------------------------------------------------------------------
    // Setup webserver
    //-----------------------------------------------------------------------
    wsSetup();

    //-------------------------------------------------------------------------
    // Startup mqtt
    //-------------------------------------------------------------------------
    if(bMQTT)
    {
      mqttConnect();
    }

    //-------------------------------------------------------------------------
    // Startup adc reading
    //-------------------------------------------------------------------------

    //-------------------------------------------------------------------------
    // Startup timer interrupt
    //-------------------------------------------------------------------------
    //set prescaler every microsecond eg: clockspeed 80.000.000Hz / 80 = 1.000.000Hz -> 1 µs period
    timer = timerBegin(0, CLOCKSPEED, true);
    timerAttachInterrupt(timer, &onTimer, true);
    // interrupt every 1 millisecond = 1000 µs
    timerAlarmWrite(timer, 1000, true);
    timerAlarmEnable(timer);

    // Ending set-up mode, turn led off to save power
    ticker.detach();
    digitalWrite(LED_BUILTIN, LOW);
  #endif

  #if defined(CALIBRATE)
    Serial.println(F("Ready - Calibrate"));
    bLine[0] = LINE0;
    bLine[1] = LINE1;
    bLine[2] = LINE2;
  #endif

  #if defined(DEBUG)
    Serial.println(F("Ready"));
  #endif
}
//==========================================================================

//##########################################################################
// Main loop
//##########################################################################
void loop()
{

  unsigned long sum = 0;

  #if defined(CALIBRATE)

    unsigned int avgV[] = {0,0,0};
    unsigned int avgZ[] = {0,0,0};

    // OTA handling
    ArduinoOTA.handle();

    if (cntrINT > 0)
    {
      // read the ADC.
      valZER[0][cntADC] = (unsigned int)(analogRead(ADC_INPUT0));
      ADCval            = (float)(valZER[0][cntADC]);
      ADCval            = (ADCval -  ADCzero[0]) * ADCvolt0;
      valADC[0][cntADC] = (unsigned int)(ADCval);

      valZER[1][cntADC] = (unsigned int)(analogRead(ADC_INPUT1));
      ADCval            = (float)(valZER[1][cntADC]);
      ADCval            = (ADCval -  ADCzero[1]) * ADCvolt1;
      valADC[1][cntADC] = (unsigned int)(ADCval);

      valZER[2][cntADC] = (unsigned int)(analogRead(ADC_INPUT2));
      ADCval            = (float)(valZER[2][cntADC]);
      ADCval            = (ADCval -  ADCzero[2]) * ADCvolt2;
```

```
    valADC[2][cntADC] = (unsigned int)(ADCval);

    cntADC++;
    if( cntADC >= iADC)
    {
      // reset counter
      cntADC = 0;

      // find DC offset and RMS
      for (int r=0; r<3; r++)
      {
        // find DC offset
        // skip first as it might not be correct due to long loop
        sum = 0;
        for (int n=1; n<iADC; n++)
        {
          sum = sum + (unsigned long)valZER[r][n];
        }
        avgZER[r][cntRMS] = (unsigned int)(sum / (iADC - 1));
        // adjust Zero offset for current
        ADCzero[r] = avgZER[r][cntRMS];

        // find RMS
        float valV;
        unsigned long sqrV;
        // now calc RMS values and average RMS
        sum = 0;
        for (int n=1; n<iADC; n++)
        {
          valV = valADC[r][n];
          sqrV = (unsigned long)(valV * valV);
          sum = sum + sqrV;
        }
        // average rms value
        valRMS[r][cntRMS] = (unsigned int)(sqrt(sum / (iADC - 1)));
      }

      cntRMS++;
      // array filled? If so, average and send out
      // reset counters and values
      // check button
      if(cntRMS >= iRMS)
      {
        cntRMS = 0;

        for (int r=0; r<3; r++)
        {

          sum = 0;
          // find average DC offset
          for (int n=0; n<iRMS; n++)
          {
            sum   = sum + (unsigned long)avgZER[r][n];
          }
          avgZ[r] = (unsigned int)(sum / iRMS);

          // find average RMS
          sum = 0;
          for (int n=0; n<iRMS; n++)
          {
            sum = sum + valRMS[r][n];
          }
          // The average RMS value of ADC values
          avgV[r] = (unsigned int)(sum / iRMS);

          if(bLine[r])
          {
            Serial.printf(" Line %d Voltage   : %u Vrms\n", r+1, avgV[r]);
            Serial.printf(" Zero %d ADC value : %u \n\n",   r+1, avgZ[r]);
          }
        }
      }
    }
    portENTER_CRITICAL(&timerMux);
    cntrINT = 0;
    portEXIT_CRITICAL(&timerMux);
  }

#else
  unsigned int rmsVal;
```

```
float        frqVal;

if (cntrINT > 0)
{

  // read the ADC.
  if(bLine[2])
  {
    valZER[2][cntADC] = (unsigned int)(analogRead(ADC_INPUT2));
    ADCval            = (float)(valZER[2][cntADC]);
    ADCval            = (ADCval - ADCzero[2]) * ADCvolt2;
    valADC[2][cntADC] = (unsigned int)(ADCval);
  }

  if(bLine[1])
  {
    valZER[1][cntADC] = (unsigned int)(analogRead(ADC_INPUT1));
    ADCval            = (float)(valZER[1][cntADC]);
    ADCval            = (ADCval - ADCzero[1]) * ADCvolt1;
    valADC[1][cntADC] = (unsigned int)(ADCval);
  }

  if(bLine[0])
  {
    valZER[0][cntADC] = (unsigned int)(analogRead(ADC_INPUT0));
    ADCval            = (float)(valZER[0][cntADC]);
    ADCval            = (ADCval - ADCzero[0]) * ADCvolt0;
    valADC[0][cntADC] = (unsigned int)(ADCval);

    // initial begining stage of measurement when voltage wave larger than 0
    if((ADCval > 0)  && stage == 3)
    {
      // allow to change to the next stage
      stage = 0;
    }

    // when voltage wave smaller or equal than 0
    if((ADCval <= 0) && stage == 0)
    {
      // start counting time for all
      // store start time
      TimeStart = micros();
      // allow to change to the next stage
      stage     = 1;
    }

    // when voltage wave is larger than 0, prepare to find peak
    if((ADCval > 0)  && stage == 1)
    {
      // reset value
      PREval = 0;
      // allow to change to the next stage
      stage  = 2;
    }

    // if current measured value larger than previous peak value
    if((ADCval > PREval) && stage == 2)
    {
      // record current time for voltage wave
      TimePeriod = micros();
      // record current measure value replace previous peak value
      PREval   = ADCval;
    }

    // when wave voltage smaller or equal than 0
    if((ADCval <= 0) && stage == 2)
    {
      // record current time for 1 period
      TimePeriod = micros();
      // accumulate or add up time for all sample readings of period wave
      sumPeriod  = sumPeriod + (TimePeriod - TimeStart);
      // time difference between voltage peak value and current peak value
      TimeStart  = TimePeriod;
      // reset peak value
      PREval     = 0;
      // set stage mode
      stage      = 1;
    }
  }
```

```
      cntADC++;
      // every iADC milliseconds
      // average and store away
      if( cntADC >= iADC)
      {
        nowMicros = micros();

        // app handling
        cntADC = 0;

        // calc frequency - only for line 1
        frqVal = (float)(1000000 / (sumPeriod / (iADC / (1000 / nomFREQ))));
        // frequency must be between + and - 1 (by regulations) otherwise we read wrong
        // nominal variation is +/- 10mHz,  exceptionally +/- 200mHz and, when in troeuble, a short
period of +/- 800mHz
        if (frqVal >= (nomFREQ + 1) || frqVal <= (nomFREQ - 1))
        {
          if(cntRMS > 1)
          {
            // use previous frequency
            frqVal = valFRQ[cntRMS-1];
          }
          else
          {
            // use nominal frequency
            frqVal = nomFREQ;
          }
        }
        valFRQ[cntRMS] = frqVal;
        sumPeriod      = 0;

        long valV;
        for (int r=0; r<3; r++)
        {

          if(bLine[r])
          {

            // find DC offset
            // skip first as it might not be correct due to long loop
            sum = 0;
            for (int n=1; n<iADC; n++)
            {
              sum = sum + (unsigned long)valZER[r][n];
            }
            avgZER[r][cntRMS] = (unsigned int)(sum / (iADC - 1));
            // adjust Zero offset
            ADCzero[r]        = avgZER[r][cntRMS];

            sum = 0;
            // calc RMS value
            for (int n=1; n<iADC; n++)
            {
              valV = (long)valADC[r][n];
              sum = sum + (valV * valV);
            }
            // average rms value
            rmsVal = (unsigned int)(sqrt(sum / (iADC - 1)));
            valRMS[r][cntRMS] = rmsVal;

            // counts
            if(rmsVal <= minVOLT)
            {
              uUnderV[r]++;
            }
            else if(rmsVal >= maxVOLT)
            {
              uOverV[r]++;
            }

            //value
            if(rmsVal <= uMinV[r])
            {
              uMinV[r] = rmsVal;
            }
            else if(rmsVal >= uMaxV[r])
            {
              uMaxV[r] = rmsVal;
            }
```

```
      // max volt can never be lower than min volt and vice vresa
      // sometimes I see this happening in the graphs, and that should not be the case
      // still need to find the real issue why this can happen = TODO/TODEBUG
      // This is a temp "fix"
      if ( uMaxV[r] < uMinV[r])
      {
        uMaxV[r] = uMinV[r];
      }
      else if ( uMinV[r] > uMaxV[r])
      {
        uMinV[r] = uMaxV[r];
      }

      uDiffV[r] = uMaxV[r] - uMinV[r];

      // save info for webpage
      for(int r=0; r<3; r++)
      {
        wUnderV[r] = uUnderV[r];
        wOverV[r]  = uOverV[r];
        wMaxV[r]   = uMaxV[r];
        wMinV[r]   = uMinV[r];
        wDiffV[r]  = uDiffV[r];
      }
    }
    else
    {
      valRMS[r][cntRMS] = 0;
      wUnderV[r]        = 0;
      wOverV[r]         = 0;
      wMaxV[r]          = 0;
      wMinV[r]          = 0;
      wDiffV[r]         = 0;
    }
  }

  cntRMS++;
  // array filled? If so, average and send out
  // reset counters and values
  // check button
  if(cntRMS >= iRMS)
  {
    cntRMS = 0;

    if(bMQTT)
    {
      // keep mqtt connection alive
      if (!mqttClient.connected())
      {
        mqttReconnect();
      }
      mqttClient.loop();
    }

    // OTA handling
    ArduinoOTA.handle();

    float fSum = 0;
    for (int n=0; n<iRMS; n++)
    {
      fSum = fSum + valFRQ[n];
    }
    // average frequency value
    avgFRQ = (float)(fSum / iRMS);

    if(bMQTT)
    {
      strcpy(caJson, "{ ");
      dt = ntpLocalDateTime();
      sprintf(caTemp, "\"TimeStamp\": \"%s\", ", dt);
      strcat(caJson, caTemp);
      sprintf(caTemp, "\"avg\": %.2f"  , avgFRQ);
      strcat(caJson, caTemp);
      strcat(caJson, "}");
      sprintf(caTemp, "%s/Freq", mqttTopicDat);
      mqttSend(caTemp, caJson);
    }

    for (int r=0; r<3; r++)
    {
```

```
  sum = 0;
  for (int n=0; n<iRMS; n++)
  {
    sum = sum + valRMS[r][n];
  }
  // average RMS values
  avgRMS[r] = (unsigned int)(sum / iRMS);

  if(!bLine[r])
  {
    avgRMS[r]  = 0;
    uUnderV[r] = 0;
    uOverV[r]  = 0;
    uMaxV[r]   = 0;
    uMinV[r]   = 0;
    uDiffV[r]  = 0;
  }

  if(bMQTT)
  {
    strcpy(caJson, "{ ");
    dt = ntpLocalDateTime();
    sprintf(caTemp, "\"TimeStamp\": \"%s\", ", dt);
    strcat(caJson, caTemp);
    sprintf(caTemp, "\"avg\": %u, "  , avgRMS[r]);
    strcat(caJson, caTemp);
    sprintf(caTemp, "\"min\": %u, "  , uMinV[r]);
    strcat(caJson, caTemp);
    sprintf(caTemp, "\"max\": %u, "  , uMaxV[r]);
    strcat(caJson, caTemp);
    sprintf(caTemp, "\"dif\": %u, "  , uDiffV[r]);
    strcat(caJson, caTemp);
    sprintf(caTemp, "\"under\": %u, ", uUnderV[r]);
    strcat(caJson, caTemp);
    sprintf(caTemp, "\"over\": %u"   , uOverV[r]);
    strcat(caJson, caTemp);
    strcat(caJson, "}");
    sprintf(caTemp, "%s/Line%u", mqttTopicDat, r+1);
    mqttSend(caTemp, caJson);
  }

  uUnderV[r] = 0;
  uOverV[r]  = 0;
  uMaxV[r]   = 0;
  uMinV[r]   = 999;
  uDiffV[r]  = 0;
}

if(bMQTT)
{
  // let system know you are still online
  mqttSend(mqttTopicSts, (char*)"online");
}

// Listen for incoming clients
webClient = webServer.available();
// If a new webClient connects,
if(webClient)
{
  wsClientActive();
}

//-----------------------------------------------------------------------------
// checking button every ...
//-----------------------------------------------------------------------------
buttState = Button();
if (buttState == LONG_PRESS_TIME)
{
  // clearout SPIFFS and restart from scratch
  #if defined(DEBUG)
    Serial.println("Formatting SPIFFS");
  #endif
  SPIFFS.format();
  delay(3000);
  #if defined(DEBUG)
    Serial.println("Restarting ...");
  #endif
  ESP.restart();
}
else if (buttState == SHRT_PRESS_TIME)
```

```
          {
            //just restart
            #if defined(DEBUG)
              Serial.println("Restarting ...");
            #endif
            ESP.restart();
          }
        }

        #if defined(DEBUG)
          // timing measurement
          difMicros = micros() - nowMicros;
          if (difMicros > 250)
          {
            Serial.print("runtime: "); Serial.print(difMicros); Serial.println(" µs");
          }
        #endif
      }
      // to make sure we do not handle pilled up interrupts
      // and keep a pace of 1 every ms
      portENTER_CRITICAL(&timerMux);
      cntrINT = 0;
      portEXIT_CRITICAL(&timerMux);
    }
  #endif
}
```

## MainsLogger based on ZMCT103C (Current up uo 5A AC rms)

ZMCT103C AC current Sensor is the best for the purpose of the DIY project where we need to measure the accurate AC current with current transformer.
It provides

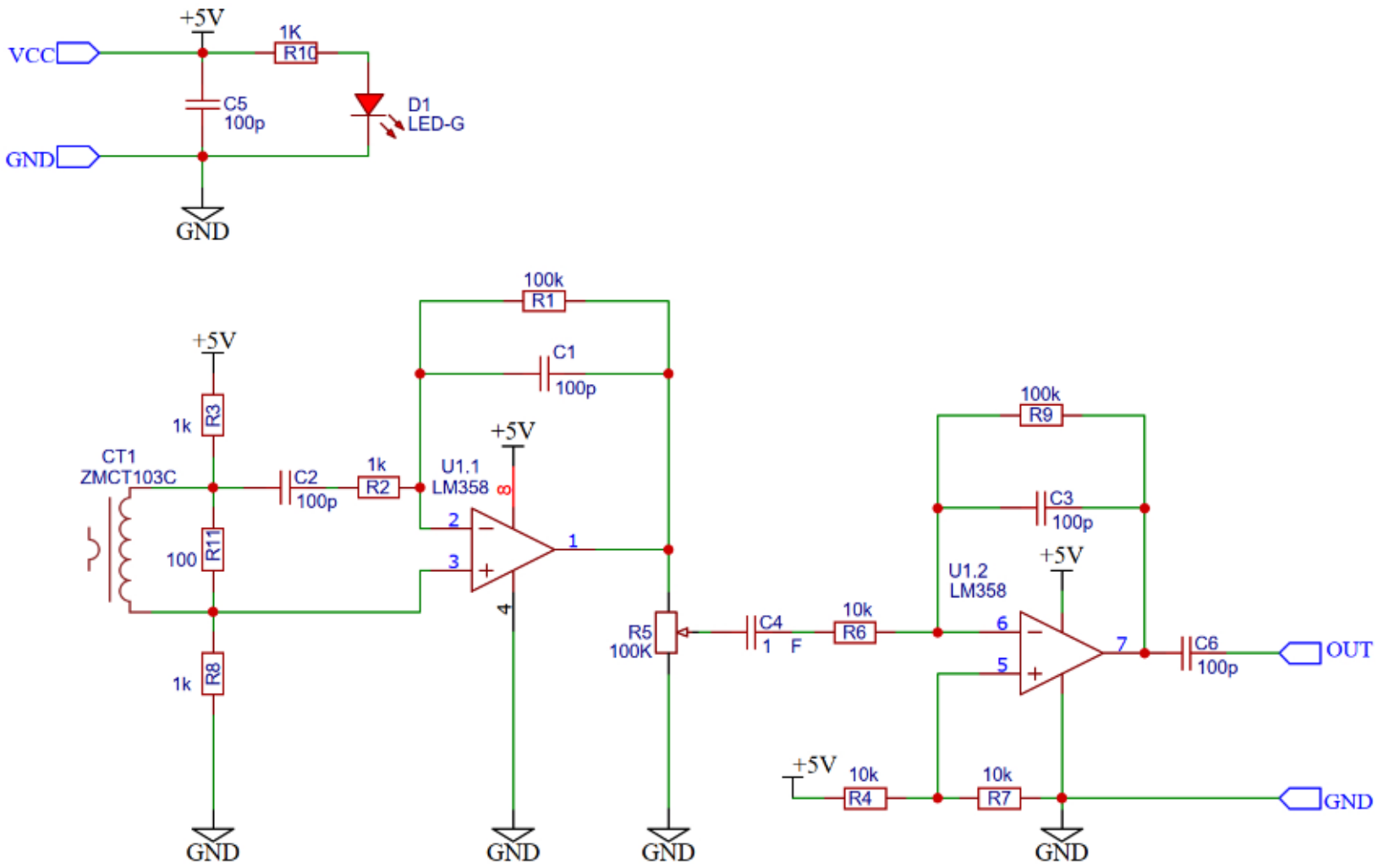
- High galvanic isolation
- High accuracy
- Good Consistency

This is a high precision micro current Transformer. This module makes it easy to monitor AC mains current up to 5 Amps . ZMCT103 is a pcb mount current transformer with 1000:1 turns ratio.

The module, called ZMCTT103C, but actually this is the name of the transformer has a LM358 OpAmp onboard as you can see in the drawing. While the transformer brings down the mains voltage to less than 3.3Vpp, the circuitry adds a DC component of half of the supply voltage. So for 5VDC, thats is 2.5VDC. On top of that the AC signal is added.
Using the trimpotmeter you can bring down the whole signal to swing between 0 and 3.3VDC which almost all microcontrollers prefer nowadays as their ADC inputsignal.

Note however that the second part of the opmap has also a DC offset of half of the supply voltage. Leaving it that way, you loose quite some swing of the AC component on top of it while keeping all down the 3.3VDC max.

Therefor, I highly recommend to solder an extra 10k resistor across R10 (or replace it with a 4.7k). Doing so, brings the DC component down from 2.5V to 1.67V when using a 5V supply and allowing to make the full swing from 0 to 3.3V.




If C6 shorted:
R7 = 4k7 -> OUT: 0-3V3
R7 = 10k -> OUT: 0-5V

## Calibration for Current

### *Adjusting the sensor through its potmeter*

During calibration, power ESP32 only via USB and do not apply power via the on-board convertor. Only 1 power feed can be 'active'.
We need to power via USB as we need to use also serial monitor and serial plotter from the Arduino IDE.

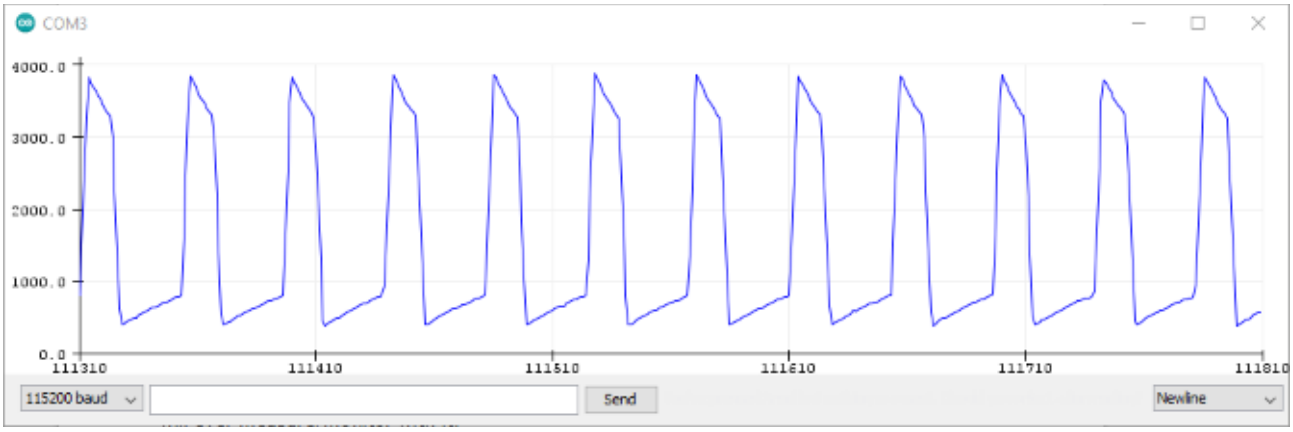
Loop the wires in such a way that 1 power supply wire of the device under test goes through th hole of the current sensor. I assume you have connected a device that draws a bit less than 5 Amps as this the max the sensor can handle.
Should you use less current, adjust sensor in such a way that is can handle the max amps you will ever measure/monitor with it.

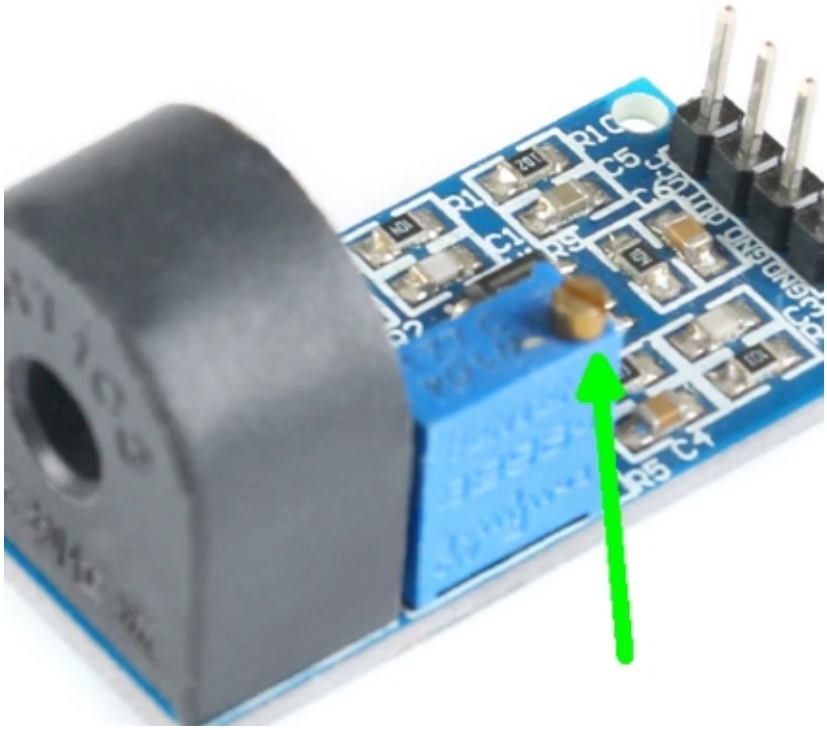
Note: a hairdryer or similar might be the right device for testing. Just make sure it does not drain more than 5 Amps.

```
#define LINE  34
//#define LINE  35
//#define LINE  32

void setup()
{
  Serial.begin(115200);
  pinMode(LINE,INPUT);
}

void loop()
{
  Serial.println(analogRead(LINE));
  //delay(20);
}
```

Run this code on the ESP32 and open the Serial plotter.

Make sure clipping does not occur and than the signal swings nicely between 100-200 and 3.300-3.500. Adjustment is done by turning the potmeter on the module.





***Getting the numbers right in de code***
Once the module's output is set correctly, you have to set the right numbers in the MainsLogger.h code.

```
//LINE 1 -> index 0
#define LINE0             true
#define ADCamp0           (float)0.0450    // amp per ADC count

//LINE 2 -> index 1
#define LINE1             false
#define ADCamp1           (float)0.0430    // amp per ADC count

//LINE 3 -> index 2
#define LINE2             false
#define ADCamp2           (float)0.0450    // amp per ADC count
```

To do this, recompile the MainsLogger code with the #define CALIBRATE (in the MainsLogger.h file) uncommented.
Set the line you want to calibrate to `true` and other lines to `false`. Recompile and upload.
Open Serial Monitor to watch the data being send to the output.

While checking the serial monitor, measure also, with the mulitmeter, the current (A AC).
Adjust the `ADCampX` value till serial output matches the current read on the multimeter. You might need several recompiles and upload to get it right or close enough.
Do this for every line.
Once the calibration is done, do no longer touch the potmeter nor the values set in the code because it will affect all future calculations.
A good practice is to lock the potmeter screw with some nailpolish :-).

***Notes***:

- with the current code, a loop only reading ADC takes about 200-300 µs so that is fairly below the 1 ms of the interrupt-period. A loop with full code execution including MQTT sending but no HTTP-traffic takes about 6 ms, hence losing 6 interrupts or 1/3 of mains-cycle. That's also why in I reset the interrupt counter to 0 at exit to make sure we take full cycle samples at fixed intervals of 1 ms. Think about it ;-).

**MainsALogger.h**

```
// System settings
// some control setting related to debugging
// uncomment to do some general debugging via Serial
//#define DEBUG
// uncomment to do debugging of WEB via Serial
//#define DEBUGWEB

//#define CALIBRATE

#define myHostname           "MainsALogger"

#define wifiAPSSID           myHostname
#define wifiAPPassword       ""

#define ntpServer            "be.pool.ntp.org"
#define ntpUTCOffset_sec     3600
#define ntpDSTOffset_sec     3600
#define ntpUpdate_sec        60

// OTA settings
// Port default is 3232
#define otaPort              3232
#define otaPassword          ""
// Password can be set with it's md5 value as well
// Eg:  MD5 of "admin" = 21232f297a57a5a743894a0e4a801fc3
#define otaPasswordHash      ""

#define PIN_BUTTON           15
#define LONG_PRESS_TIME    6000  // 6s
#define SHRT_PRESS_TIME    3000  // 3s

#define ADC_INPUT0           34  // define the used ADC input channel
#define ADC_INPUT1           35  // define the used ADC input channel
#define ADC_INPUT2           32  // define the used ADC input channel

#define CLOCKSPEED           80  // clockspeed of ESP32 in MHz

#define nomAMP               5  // max current possible
#define maxAMP               5  //
#define minAMP               0  //

#define nomFREQ             50.00 // nominal frequency
#define maxFREQ             50.01 // + 10mHz
#define minFREQ             49.99 // - 10mHz

//LINE 1 -> index 0
#define LINE0             false
#define ADCamp0           (float)0.0450    // amp per ADC count

//LINE 2 -> index 1
#define LINE1             false
#define ADCamp1           (float)0.0430    // amp per ADC count

//LINE 3 -> index 2
#define LINE2             true
#define ADCamp2           (float)0.0450    // amp per ADC count
```

**MainsALogger.ino**

```
//-----------------------------------------------------------------------------
// what/how to make it
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// includes
//-----------------------------------------------------------------------------
#include <Ticker.h>
#include <WiFi.h>
#include <ESPmDNS.h>
#include <WiFiUdp.h>
#include <NTPClient.h>
#include <PubSubClient.h>
#include <math.h>
#include <SPIFFS.h>
#include <WiFiSettings.h>
#include <ArduinoOTA.h>

#include "MainsALogger.h"

//-----------------------------------------------------------------------------
// defines
//-----------------------------------------------------------------------------
#define VERSION           "1.50"

// webserver timeout in milliseconds (example: 2000ms = 2s)
#define webserverTimeout  1000

// max length of a MQTT topic
#define MQTT_TOPIC_SIZE    128

// max length of string
#define MAX_STRING_LENGTH  256

//-----------------------------------------------------------------------------
// constants
//-----------------------------------------------------------------------------
// number of samples to take before RMS is calculate
const int iADC = 2 * nomFREQ;    // one sample every 1 ms -> 100ms = 5 periods @ 50Hz
// number of Vrms to hold before average Vrms is outputted
const int iRMS =  50;            // 100 x 5 periods -> 100 * 100 ms -> 5 seconds -> every 5 seconds
MQTT message

//-----------------------------------------------------------------------------
// global variables
//-----------------------------------------------------------------------------
// time related
unsigned long time_now;
char today[11];
// to hold date & time
char *dt;
int days[] = {31,29,31,30,31,30,31,31,30,31,30,31};
char  nuDate[12];
char  nuTime[10];
char  currDST[3];
char  lastDST[3];
// Current timeing
unsigned long nowMicros  = micros();
unsigned long difMicros  = 0;
unsigned long currMillis = millis();
// Previous timing
unsigned long prevMillis = 0;

// vars related to button
unsigned long buttTime  = 0;
int           buttState;
int           buttLast  = HIGH;      // the previous state from the input pin
int           buttCurr  = HIGH;      // the current  state from the input pin

// vars related to MQTT
boolean bMQTT          = true;
String  mqttServer     = "mqtt.home";
int     mqttPort       = 1883;
String  mqttLogin      = "";
String  mqttPassword   = "";
String  mqttTopicBase  = myHostname;

char mqttClientID[30];
```

```
char mqttTopicSts[MQTT_TOPIC_SIZE];
char mqttTopicDat[MQTT_TOPIC_SIZE];

// vars to store characters
char caTemp[MAX_STRING_LENGTH];
char caTmp[MAX_STRING_LENGTH];
char caJson[MAX_STRING_LENGTH * 2];

// counters for array index
int  cntADC = 0;
int  cntRMS = 0;

// whichs lines should be used for reading
boolean bLine[3] = {LINE0, LINE1, LINE2};

// vars related to ADC and Currents
float ADCval;

unsigned long valZER[3][iADC];
unsigned long avgZER[3][iRMS];
float         ADCzero[3] = {1000, 1000, 1000};
unsigned int  valADC[3][iADC];
unsigned int  valRMS[3][iRMS];
float         valFRQ[iRMS];
unsigned int  avgRMS[3];
float         avgFRQ;
unsigned int  uUnderA[3] = {0, 0, 0};  // drops
unsigned int  uOverA[3]  = {0, 0, 0};  // spikes
unsigned int  uMaxA[3]   = {0, 0, 0};
unsigned int  uMinA[3]   = {999, 999, 999};
unsigned int  uDiffA[3]  = {0, 0, 0};

// vars related to webpage
unsigned int  wUnderA[3] = {0, 0, 0};  // drops
unsigned int  wOverA[3]  = {0, 0, 0};  // spikes
unsigned int  wMaxA[3]   = {0, 0, 0};
unsigned int  wMinA[3]   = {999, 999, 999};
unsigned int  wDiffA[3]  = {0, 0, 0};

// Frequency measurement
unsigned int stage      = 3;          // use for switching operation
                                      // stage = 0  => current still positive
                                      // stage = 1  => current has crossed 0 and start time is
recorded
                                      // stage = 2  => looking for peak, recording time
                                      // stage = 3  => timing not started

float PREval            = 0;          // is the previous value for current sensor
float sumPeriod         = 0;          // is the accumulate time differences for current wave

unsigned long TimeStart;              // starting time for Phase Angle and Period (in micro
seconds).
unsigned long TimePeriod;             // current  time for period of wave (in micro seconds).


// needed for timer interrupt
volatile byte cntrINT;
hw_timer_t * timer    = NULL;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

// webserver related
// Variable to store the HTTP requests
String httpHeader;

//-----------------------------------------------------------------------------
// objects
//-----------------------------------------------------------------------------
// Initiate led blinker library
Ticker       ticker;

// WiFi objects
WiFiUDP      ntpUDP;
WiFiClient   wifiClient;

// NTP object
NTPClient    ntpClient(ntpUDP, ntpServer, ntpUTCOffset_sec, ntpUpdate_sec * 1000);

// MQTT object
PubSubClient mqttClient(wifiClient);
```

```
// object for webserver
// set web server port number to 80
WiFiServer  webServer(80);
WiFiClient  webClient;

//-----------------------------------------------------------------------------
// Functions
//-----------------------------------------------------------------------------

//.............................................................................
// Interrupt functions
//.............................................................................
//** Timer interrupt
//*****************************************************************************
void IRAM_ATTR onTimer()
{
  portENTER_CRITICAL_ISR(&timerMux);
  cntrINT++;
  portEXIT_CRITICAL_ISR(&timerMux);
}
//*****************************************************************************

//.............................................................................
// Call-back functions
//.............................................................................
//** Call-back TICKER
//*****************************************************************************
void tick()
{
  // get the current state of GPIO1 pin
  int state = digitalRead(LED_BUILTIN);
  // set pin to the opposite state
  digitalWrite(LED_BUILTIN, !state);
}
//*****************************************************************************

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// +++ WIFI Functions +++
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//** Portal for settings
//*****************************************************************************
void wifiSettings()
{
  // Format SPIFF if mount fails.
  // fail might happen at first time when nothing was done before
  SPIFFS.begin(true);

  WiFiSettings.begin();
  // default to ESP32- followed by last 3 pairs of HEX numbers of MAC address, reversed
  // eg:  ac:67:b2:37:8e:60 -> esp32-608e37
  WiFiSettings.hostname = myHostname;
  //WiFiSettings.language = "en";

  WiFiSettings.heading("MQTT");
  bMQTT         = WiFiSettings.checkbox("MQTT Active", bMQTT, "MQTT Active");
  mqttServer    = WiFiSettings.string("mqtt_server", 64, mqttServer, "MQTT Server");
  mqttPort      = WiFiSettings.integer("mqtt_port", 0, 65535, mqttPort, "MQTT Port");
  mqttLogin     = WiFiSettings.string("mqtt_login", 64, mqttLogin, "MQTT Login");
  mqttPassword  = WiFiSettings.string("mqtt_password", 64, mqttPassword, "MQTT Password");
  mqttTopicBase = WiFiSettings.string("mqtt_topic", 128, mqttTopicBase, "MQTT Topic Base");

  // Set custom callback functions
  WiFiSettings.onSuccess      = [](){};  // do nothing
  WiFiSettings.onConfigSaved  = [](){};  // do nothing
  WiFiSettings.onWaitLoop     = []()
  {
    // Delay next function call by 500ms
    return 500;
  };

  WiFiSettings.onConnect = []()
  {
    // Delay next function call by 50ms
    return 50;
  };

  WiFiSettings.onFailure = []()
  {
    // Delay 2s
```

```
    delay(2000);
  };

  static unsigned int  portal_phase = 0;
  static unsigned long portal_start;
  WiFiSettings.onPortal = [](){portal_start = millis();};

  WiFiSettings.onPortalView = [](){if (portal_phase < 2){portal_phase = 2;}};

  WiFiSettings.onConfigSaved = [](){portal_phase = 3;};

  WiFiSettings.onPortalWaitLoop = []()
  {
    // anyone connected?
    if (WiFi.softAPgetStationNum() == 0)
    {
      portal_phase = 0;
    }
    else
    {
      if (!portal_phase)
      {
        portal_phase = 1;
      }
    }

    // do not wait endless
    if (portal_phase == 0 && millis() - portal_start > 5*60*1000)
    {
      // reboot if nothing happened within 5 minutes = 5 * 60 * 1000 milliseconds
      ESP.restart();
    }
  };

  // try to connect to WiFi, if not, start settings portal
  if (WiFiSettings.connect(true, 15))
  {
    #if defined(DEBUG)
      Serial.println("Connected to Wifi Network");
    #endif
  }

}
//*****************************************************************************
// +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

// +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// +++ NTP/Time Functions +++
// +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//** Set-up NTP
//*****************************************************************************
void ntpSetup()
{
  #if defined(DEBUG)
    Serial.println("NTP setup");
  #endif

  // clear some vars
  memset(lastDST, 0x00, 3);
  memset(currDST, 0x00, 3);

  // Init NTP
  ntpClient.begin();

  String nu = ntpClient.getFormattedDate();
  #if defined(DEBUG)
    Serial.print("Init    ->  Date - Time: "); Serial.print(nu); Serial.println();
  #endif

  ntpClient.forceUpdate();

  nu = ntpClient.getFormattedDate();
  #if defined(DEBUG)
    Serial.print("Updated ->  Date - Time: "); Serial.print(nu); Serial.println();
  #endif

  //adjust DST if needed
  checkDST();
```

```
  nu = ntpClient.getFormattedDate();
  #if defined(DEBUG)
    Serial.print("Final   ->  Date - Time: "); Serial.print(nu); Serial.println();
  #endif

  #if defined(DEBUG)
    Serial.println("NTP activated");
  #endif
}
//****************************************************************************

//** Check for DST
//****************************************************************************
void checkDST()
{
  #if defined(DEBUG)
    Serial.println("DST Checking ....");
  #endif

  // due to problem/bug in time /timezone, the following code
  // is required to make sure we have the right offset for DST

  // make sure we have recently updated
  ntpClient.forceUpdate();

  // get date
  String nu = ntpClient.getFormattedDate().substring(0,19);
  #if defined(DEBUG)
    Serial.println(nu);
  #endif

  int y, m, w;
  // correct for leapyear
  y = nu.substring(0,4).toInt();
  days[1] -= (y % 4) || (!(y % 100) && (y % 400));
  w = y * 365 + 97 * (y - 1) / 400 + 4;

  char buff[20];
  // find last sunday of March
  m = 2; // March
  w = (w + days[m]) % 7;
  sprintf(buff, "%04d-%02d-%02dT02:00:00", y, m + 1,days[m] - w);
  String DSTstart(buff);
  #if defined(DEBUG)
    Serial.println(DSTstart);
  #endif

  // find last sunday of Octobre
  m = 9; // Octobre
  w = (w + days[m]) % 7;
  sprintf(buff, "%04d-%02d-%02dT03:00:00", y, m + 1,days[m] - w);
  String DSTend(buff);
  #if defined(DEBUG)
    Serial.println(DSTend);
  #endif

  // if nu is between March 02:00:00 and Oct 03:00:00
  // DST is active (at least till 2024)
  if(nu >= DSTstart && nu < DSTend)
  {
    // DST = UTC + ntpUTCOffset_sec + ntpDSTOffset_sec
    ntpClient.setTimeOffset(ntpUTCOffset_sec + ntpDSTOffset_sec);
    #if defined(DEBUG)
      Serial.println("DST set");
    #endif
  }
  else
  {
    // non-DST = UTC + ntpUTCOffset_sec
    ntpClient.setTimeOffset(ntpUTCOffset_sec);
    #if defined(DEBUG)
      Serial.println("DST reset");
    #endif
  }

  // make sure we update local date/time
  ntpClient.forceUpdate();

  // save last check of DST
  sprintf(lastDST, "%02d", ntpClient.getHours());
```

```
  sprintf(currDST, "%02d", ntpClient.getHours());
  ntpClient.getFormattedTime().toCharArray(nuTime, 8);
  nuTime[9] = 0x00;

  #if defined(DEBUG)
    Serial.println("DST Checking done");
  #endif
}

//** Get Date & Time
//*****************************************************************************
char* ntpLocalDateTime()
{
  static char caDT[21];

  String nu = ntpClient.getFormattedDate().substring(0,19);
  // convert to char array
  nu.toCharArray(caDT, 20);
  // remove T
  caDT[10] = ' ';
  //terminate to string
  caDT[20] = 0x00;

  return(caDT);
}
//*****************************************************************************
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// +++ mDNS Functions +++
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//** Set-up mDNS
//*****************************************************************************
void mdnsSetup()
{
  #if defined(DEBUG)
    Serial.println(F("mDNS activating"));
  #endif

  if(!MDNS.begin(myHostname))
  {
    #if defined(DEBUG)
      Serial.println(F("mDNS activation FAILED"));
    #endif
    return;
  }

  // indicate there is a webserver on this device
  MDNS.addService("http", "tcp", 80);

  #if defined(DEBUG)
    Serial.println(F("mDNS activated"));
  #endif
}
//*****************************************************************************
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// +++ MQTT Functions +++
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//** Set-up MQTT
//*****************************************************************************
void mqttConnect()
{

  #if defined(DEBUG)
    Serial.println(F("MQTT activating"));
    Serial.print(F("MQTT attempting connection to ")); Serial.print(mqttServer);
Serial.print(F(":")); Serial.println(mqttPort);
  #endif

  // mqttClientID
  strcpy(mqttClientID, myHostname);
  #if defined(DEBUG)
    Serial.print(F("  MQTT ClientID: <")); Serial.print(mqttClientID); Serial.println(F(">"));
  #endif

  // mqttTopicSts
```

```
  strcpy(mqttTopicSts, mqttTopicBase.c_str());
  strcat(mqttTopicSts, "/status");
  #if defined(DEBUG)
    Serial.print(F("  MQTT Topic STS: <")); Serial.print(mqttTopicSts); Serial.println(F(">"));
  #endif

  // mqttTopicDat
  strcpy(mqttTopicDat, mqttTopicBase.c_str());
  strcat(mqttTopicDat, "/data");
  #if defined(DEBUG)
    Serial.print(F("  MQTT Topic DAT: <")); Serial.print(mqttTopicDat); Serial.println(F(">"));
  #endif

  mqttClient.setServer(mqttServer.c_str(), mqttPort);

  // ESP will connect to mqtt broker with clientID, last will, ...
  // bool connect(const char* id, const char* user, const char* pass, const char* willTopic, uint8_t
willQos, bool willRetain, const char* willMessage, bool cleanSession);
  if (mqttClient.connect(mqttClientID, mqttLogin.c_str(), mqttPassword.c_str(), mqttTopicSts, 1,
true, "offline", false))
  {

    #if defined(DEBUG)
      Serial.println(F("MQTT connected"));
    #endif

    // let know we are online
    mqttClient.publish(mqttTopicSts, "online", true);
    #if defined(DEBUG)
      Serial.println(F("MQTT activated"));
    #endif
  }
  else
  {
    #if defined(DEBUG)
      Serial.println(F("MQTT activation Failed"));
    #endif
  }

}
//****************************************************************************

//****************************************************************************
void mqttReconnect()
{
  uint8_t i = 0;

  while (!mqttClient.connected())
  {
    #if defined(DEBUG)
      Serial.print(F("MQTT attempting reconnection to ")); Serial.print(mqttServer);
Serial.print(F(":")); Serial.println(mqttPort);
    #endif
    if (mqttClient.connect(mqttClientID, mqttLogin.c_str(), mqttPassword.c_str(), mqttTopicSts, 1,
true, "offline", false))
    {
      #if defined(DEBUG)
        Serial.println(F("MQTT reconnected"));
      #endif
      // let know we are online
      mqttClient.publish(mqttTopicSts, "online", true);
    }
    else
    {
      #if defined(DEBUG)
        Serial.print(F("MQTT reconnection failed, rc=")); Serial.print(mqttClient.state());
Serial.println(F(" ... try again in 5 seconds"));
      #endif
      // Wait 15 seconds before retrying
      // delay non-blocking
      time_now = millis();
      while(millis() < time_now + 1500){}
      ++i;
      if (i > 5)
      {
        #if defined(DEBUG)
          Serial.println(F("Restarting..."));
        #endif
        ESP.restart();
      }
```

```
    }
  }
}
//****************************************************************************

// ** send MQTT messages
//****************************************************************************
void mqttSend(char *topic, char* data)
{
  mqttClient.publish(topic, data, true);
}
//****************************************************************************
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// +++ OTA Functions +++
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//** Set-up OTA
//****************************************************************************
void otaSetup()
{
  #if defined(DEBUG)
    Serial.println("OTA activating");
  #endif

  // Port defaults to 3232
  ArduinoOTA.setPort(otaPort);

  // Hostname defaults to esp32-[MAC]
  ArduinoOTA.setHostname(myHostname);

  // Set OTA Password. No authentication by default
  if(strlen(otaPasswordHash) != 0)
  {
    ArduinoOTA.setPasswordHash(otaPasswordHash);
  }
  else if(strlen(otaPassword) != 0)
  {
    ArduinoOTA.setPassword(otaPassword);
  }

  ArduinoOTA
    .onStart([]()
    {
      String type;
      if(ArduinoOTA.getCommand() == U_FLASH)
        type = "sketch";
      else // U_SPIFFS
        type = "filesystem";

      // NOTE: if updating SPIFFS this would be the place to unmount SPIFFS using SPIFFS.end()
      #if defined(DEBUG)
        Serial.println(" - Start updating " + type);
      #endif
    })

    .onProgress([](unsigned int progress, unsigned int total)
    {
      #if defined(DEBUG)
        Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
      #endif
    })

    .onEnd([]()
    {
      #if defined(DEBUG)
        Serial.print("\n"); Serial.println(" - End");
      #endif
    })

    .onError([](ota_error_t error)
    {
      #if defined(DEBUG)
        Serial.printf(" - Error[%u]: ", error);
        if(error == OTA_AUTH_ERROR)         Serial.println("Auth Failed");
        else if(error == OTA_BEGIN_ERROR)   Serial.println("Begin Failed");
        else if(error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
        else if(error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
        else if(error == OTA_END_ERROR)     Serial.println("End Failed");
```

```
      #endif
    }

  );

  ArduinoOTA.begin();
  #if defined(DEBUG)
    Serial.println("OTA activated");
  #endif
}
//*****************************************************************************
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// +++ Button Functions +++
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//** Button logic long press to activate Settings Portal or to restart
//*****************************************************************************
unsigned int Button()
{
  // read the state of the button:
  buttCurr = digitalRead(PIN_BUTTON);

  // Is button is pressed or released?
  if(buttLast == HIGH && buttCurr == LOW)
  {
    buttLast = buttCurr;
    #if defined(DEBUG)
      Serial.println("Portal Button: Pressed");
    #endif
    buttTime = millis();
  }
  else
  {
    if(buttLast == LOW && buttCurr == HIGH)
    {
      buttLast = buttCurr;
      #if defined(DEBUG)
        Serial.println("Portal Button: Released");
      #endif
      long pressDuration = millis() - buttTime;

      if( pressDuration > LONG_PRESS_TIME )
      {
        #if defined(DEBUG)
          Serial.println("Portal Button: A long press is detected");
        #endif
        return LONG_PRESS_TIME;
      }
      else
      {
        if ( pressDuration > SHRT_PRESS_TIME )
        {
          #if defined(DEBUG)
            Serial.println("Portal Button: A short press is detected");
          #endif
          return SHRT_PRESS_TIME;
        }
        else
        {
          #if defined(DEBUG)
            Serial.println("Portal Button: A very short press is detected");
          #endif
          return 0;
        }
      }
    }
  }
  buttLast = buttCurr;
  return false;
}
//*****************************************************************************
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// +++ Webserver Functions +++
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//** Set-up webserver
//*****************************************************************************
```

```
void wsSetup()
{
  #if defined(DEBUG)
    Serial.println("Webserver activating");
  #endif

  webServer.begin();

  #if defined(DEBUG)
    Serial.println("Webserver activated");
  #endif
}
//****************************************************************************

//** listen & talk to webclient
//****************************************************************************
void wsClientActive()
{
  char c;

  currMillis = millis();
  prevMillis = currMillis;
  // print a message out in the serial port
  #if defined(DEBUG)
    Serial.println("Client connected");
  #endif

  // make a String to hold incoming data from the webClient
  String currentLine = "";

  // loop while the webClient's connected and no timeout occured
  while (webClient.connected() && currMillis - prevMillis <= webserverTimeout)
  {
    currMillis = millis();
    // if there's bytes to read from the webClient,
    if(webClient.available())
    {
      c = webClient.read();           // read a byte, then
      #if defined(DEBUGWEB)
        Serial.write(c);              // print it out the serial monitor
      #endif
      httpHeader += c;                // add it to httpHeader var

      // if the byte is a newline character
      if(c == '\n')
      {
        #if defined(DEBUGWEB)
          Serial.println("New line");
        #endif

        // if the current line is blank, you got two newlines in a row.
        // that's the end of the webClient HTTP request
        // check what was sent, act as required and then send a response:
        if(currentLine.length() == 0)
        {
          // first send response as HTTP headers always start with a response code
          // and a content-type so the webClient knows what's coming,
          webClient.println("HTTP/1.1 200 OK");
          webClient.println("Content-type:text/html");
          webClient.println("Connection: close");
          // then a blank line:
          webClient.println("");

          // decode if action is required
          // action on home page?
          if(httpHeader.indexOf("GET /HOME") >= 0)
          {
            webHomePage();
            break;
          }
          else if(httpHeader.indexOf("GET /SELECT") >= 0)
          {
            if(httpHeader.indexOf("GET /SELECT?") >= 0)
            {
              if(httpHeader.indexOf("line0=on") >=0)
              {
                bLine[0] = true;
              }
              else
              {
```

```
                bLine[0] = false;
              }
              if(httpHeader.indexOf("line1=on") >=0)
              {
                bLine[1] = true;
              }
              else
              {
                bLine[1] = false;
              }
              if(httpHeader.indexOf("line2=on") >=0)
              {
                bLine[2] = true;
              }
              else
              {
                bLine[2] = false;
              }
              webHomePage();
              break;
            }
            webSelectPage();
            break;
          }
          else if(httpHeader.indexOf("GET /CHART") >= 0)
          {
            // send home page
            webChartPage();
            break;
          }
          else if(httpHeader.indexOf("GET /line") >= 0)
          {
            // send home page
            webLineData();
            break;
          }
          // send home page
          webHomePage();
          break;
        }
        else
        {
          // if you got a newline, then clear currentLine
          currentLine = "";
        }
      }
      else if(c != '\r')
      {
        // if you got anything else but a carriage return character,
        // add it to the end of the currentLine
        currentLine += c;
      }
      else if(c == '\r')
      {
        #if defined(DEBUGWEB)
          Serial.println("New return");
        #endif
      }
    }
  }

  #if defined(DEBUGWEB)
    Serial.print("<"); Serial.print(httpHeader); Serial.println(">");
  #endif

  // Clear the httpHeader variable
  httpHeader = "";

  // now queue is empty, stop it
  webClient.stop();
  #if defined(DEBUG)
    Serial.println("Client disconnected.");
  #endif
}
//****************************************************************************

//** send home page
//****************************************************************************
void webHomePage()
{
```

```
  // Display the HTML web page
  webClient.println("<!DOCTYPE html>");
  // Open the HTML web page
  webClient.println("<html>");
  // Open head
  webClient.println("<head><meta name=\"viewport\" content=\"width=device-width, initial-
scale=1\">");
  webClient.println("<meta http-equiv=\"refresh\" content=\"60\">");
  webClient.println("<link rel=\"icon\" href=\"data:,\">");
  // Open style for CSS settings
  webClient.println("<style>");
  // overall style setting
  webClient.println("html, body { width: 100%; height: 100%; padding: 0; margin: 0; font-
family: \"Verdana\", sans-serif;}");
  webClient.println(".centered-wrapper { position: relative; text-align: center; font-
family: \"Verdana\", sans-serif;}");
  webClient.println(".centered-wrapper:before { content: ""; position: relative; display: inline-
block; width: 0; height: 100%; vertical-align: middle; font-family: \"Verdana\", sans-serif;}");
  webClient.println(".centered-content { display: inline-block; vertical-align: middle; font-family:
\"Verdana\", sans-serif;}");
  webClient.println(".btn { background-color: #0099ff; border: none; color: white; width: 200px;
height: 50px; text-decoration: none; font-size: 20px; margin: 2px; cursor: pointer;}");
  webClient.println("</style></head>");

  // body of page
  webClient.println("<body class=\"centered-wrapper\">");
  webClient.println("<div class=\"centered-content\">");
  webClient.print("<h1>"); webClient.print(myHostname); webClient.println("</h1>");
  webClient.print("<table style='font-family: Courier, monospace; font-size: 20px'>");
  webClient.print("<tr>");
  dt = ntpLocalDateTime();
  sprintf(caTemp, "%s           ", dt +
11);
  webClient.print("<td><label><b>Time    :</b></
label><label> ");webClient.print(caTemp);webClient.print("</label></td>");
  webClient.print("</tr><tr>");
  webClient.print("<td><label>         </
label><label>  L1 -  L2 -
  L3     </label></td>");
  webClient.print("</tr><tr>");
  sprintf(caTmp, "%lu", avgRMS[0]); str_replace(caTmp, (char*)" ", (char*)" "); strcpy(caTemp,
caTmp); strcat(caTemp, " -");
  sprintf(caTmp, "%lu", avgRMS[1]); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp); strcat(caTemp, " -");
  sprintf(caTmp, "%lu", avgRMS[2]); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp);

webClient.print("<td><label><b>AvgAmp :</b></label><label>");webClient.print(caTemp);webClient.
print(" Vrms</label></td>");
  webClient.print("</tr><tr>");
  sprintf(caTmp, "%u", wMaxA[0]); str_replace(caTmp, (char*)" ", (char*)" "); strcpy(caTemp,
caTmp); strcat(caTemp, " -");
  sprintf(caTmp, "%u", wMaxA[1]); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp); strcat(caTemp, " -");
  sprintf(caTmp, "%u", wMaxA[2]); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp);

webClient.print("<td><label><b>MaxAmp :</b></label><label>");webClient.print(caTemp);webClient.
print(" Vrms</label></td>");
  webClient.print("</tr><tr>");
  sprintf(caTmp, "%u", wMinA[0]); str_replace(caTmp, (char*)" ", (char*)" "); strcpy(caTemp,
caTmp); strcat(caTemp, " -");
  sprintf(caTmp, "%u", wMinA[1]); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp); strcat(caTemp, " -");
  sprintf(caTmp, "%u", wMinA[2]); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp);

webClient.print("<td><label><b>MinAmp :</b></label><label>");webClient.print(caTemp);webClient.
print(" Vrms</label></td>");
  webClient.print("</tr><tr>");
  sprintf(caTmp, "%u", wOverA[0]); str_replace(caTmp, (char*)" ", (char*)" "); strcpy(caTemp,
caTmp); strcat(caTemp, " -");
  sprintf(caTmp, "%u", wOverA[1]); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp); strcat(caTemp, " -");
  sprintf(caTmp, "%u", wOverA[2]); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp);

webClient.print("<td><label><b>Overrun :</b></label><label>");webClient.print(caTemp);webClient
.print("     </label></td>");
  webClient.print("</tr><tr>");
```

```
  sprintf(caTmp, "%u", wUnderA[0]); str_replace(caTmp, (char*)" ", (char*)" "); strcpy(caTemp,
caTmp); strcat(caTemp, " -");
  sprintf(caTmp, "%u", wUnderA[1]); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp); strcat(caTemp, " -");
  sprintf(caTmp, "%u", wUnderA[2]); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp);

webClient.print("<td><label><b>Underrun:</b></label><label>");webClient.print(caTemp);webClient.prin
t("     </label></td>");
  webClient.print("</tr><tr>");
  webClient.print("<td> </td><td> </td><td> </td>");
  webClient.print("</tr>");
  webClient.print("</table>");
  webClient.print("<p><a href=\"/HOME\"><button class=\"btn\">Refresh</button></p>");
  webClient.print("<p><a href=\"/CHART\"><button class=\"btn\">Chart</button></p>");
  webClient.print("<p><a href=\"/SELECT\"><button class=\"btn\">Select Lines</button></p>");
  webClient.println("</div>");
  webClient.println("</body></html>");

  // The HTTP response ends with another blank line
  webClient.println("");
}
//*****************************************************************************

//** send select line page
//*****************************************************************************
void webSelectPage()
{
  // Display the HTML web page
  webClient.println("<!DOCTYPE html>");
  // Open the HTML web page
  webClient.println("<html>");
  // Open head
  webClient.println("<head><meta name=\"viewport\" content=\"width=device-width, initial-
scale=1\">");
  webClient.println("<link rel=\"icon\" href=\"data:,\">");
  // Open style for CSS settings
  webClient.println("<style>");
  // overall style setting
  webClient.println("html, body { width: 100%; height: 100%; padding: 0; margin: 0; font-
family: \"Verdana\", sans-serif;}");
  webClient.println(".centered-wrapper { position: relative; text-align: center; font-
family: \"Verdana\", sans-serif;}");
  webClient.println(".centered-wrapper:before { content: ""; position: relative; display: inline-
block; width: 0; height: 100%; vertical-align: middle; font-family: \"Verdana\", sans-serif;}");
  webClient.println(".centered-content { display: inline-block; vertical-align: middle; font-family:
\"Verdana\", sans-serif;}");
  webClient.println(".btn { background-color: #0099ff; border: none; color: white; width: 200px;
height: 50px; text-decoration: none; font-size: 20px; margin: 2px; cursor: pointer;}");
  webClient.println("</style></head>");

  //body of page
  webClient.println("<body class=\"centered-wrapper\">");
  webClient.println("<div class=\"centered-content\">");
  webClient.print("<h1>"); webClient.print(myHostname); webClient.println("</h1>");
  webClient.print("<form><table style='font-family: \"Verdana\", Courier, monospace; font-size:
20px'>");
  webClient.print("<tr><td><label>Line 1:</label> <input type=\"checkbox\" name=\"line0\" ");
webClient.printf("%s", bLine[0] ? "checked" : "unchecked"); webClient.print("></td></tr>");
  webClient.print("<tr><td><label>Line 2:</label> <input type=\"checkbox\" name=\"line1\" ");
webClient.printf("%s", bLine[1] ? "checked" : "unchecked"); webClient.print("></td></tr>");
  webClient.print("<tr><td><label>Line 3:</label> <input type=\"checkbox\" name=\"line2\" ");
webClient.printf("%s", bLine[2] ? "checked" : "unchecked"); webClient.print("></td></tr>");
  webClient.print("<tr><td> </td></tr>");
  webClient.print("<tr><td><input class=\"btn\" type=\"submit\" value=\"Save\"
formmethod=\"get\"></td></tr>");
  webClient.print("<tr><td> </td></tr>");
  webClient.print("<tr><td><a href=\"/HOME\"><button class=\"btn\">Home</button></td></tr>");
  webClient.print("</table></form>");
  webClient.print("</div>");
  webClient.println("</body></html>");

  // The HTTP response ends with another blank line
  webClient.println("");
}
//*****************************************************************************

//** send chart page
//*****************************************************************************
void webChartPage()
```

```
{
  webClient.println("<!DOCTYPE HTML><html><head>");
  webClient.println("<meta name='viewport' content='width=device-width, initial-scale=1'>");
  webClient.println("<script src='https://code.highcharts.com/highcharts.js'></script>");
  webClient.println("<style>html, body{width: 100%; height: 100%; padding: 0; margin: 0; font-
family: \"Verdana\", sans-serif;}");
  webClient.println(".centered-wrapper { position: relative; text-align: center; font-
family: \"Verdana\", sans-serif;}");
  webClient.println(".centered-wrapper:before { content: ""; position: relative; display: inline-
block; width: 0; height: 100%; vertical-align: middle; font-family: \"Verdana\", sans-serif;}");
  webClient.println(".centered-content { display: inline-block; vertical-align: middle; font-family:
\"Verdana\", sans-serif;}");
  webClient.println(".btn { background-color: #0099ff; border: none; color: white;  width: 200px;
height: 50px; text-decoration: none; font-size: 20px; margin: 2px; cursor: pointer;}");
  webClient.println("</style></head>");
  webClient.println("<body class=\"centered-wrapper\">");
  webClient.println("<div class=\"centered-content\">");
  webClient.print("<h1>"); webClient.print(myHostname); webClient.println("</h1>");
  webClient.println("<div id='chart-AVG' class='container'></div>");
  webClient.println("<div id='chart-MIN' class='container'></div>");
  webClient.println("<div id='chart-MAX' class='container'></div>");
  webClient.println("<div id='chart-UND' class='container'></div>");
  webClient.println("<div id='chart-OVR' class='container'></div>");
  webClient.println("<p><a href=\"/HOME\"><button class=\"btn\">Home</button></p>");
  webClient.println("</div></body><script>");
  webClient.println("var chartAVG= new Highcharts.Chart({chart:{renderTo : 'chart-AVG'},");
  webClient.println("title:{text: 'Avg Line Current'},");
  webClient.println("series:[{name: 'L1', color: '#6e3e00', showInLegend: true, data:[]},");
  webClient.println("{name: 'L2', color: '#000000', showInLegend: true, data:[]},");
  webClient.println("{name: 'L3', color: '#7f7f7f', showInLegend: true, data:[]}],");
  webClient.println("plotOptions:{line: {animation: false, dataLabels:{enabled: true}}},");
  webClient.println("xAxis:{type: 'datetime', dateTimeLabelFormats:{second: '%H:%M:%S'}},");
  webClient.println("yAxis:{title:{text: 'Vrms'}},credits:{enabled: false}});");
  webClient.println("var chartMIN= new Highcharts.Chart({chart:{renderTo : 'chart-MIN'},");
  webClient.println("title:{text: 'Min Line Current'},");
  webClient.println("series:[{name: 'L1', color: '#6e3e00', showInLegend: true, data:[]},");
  webClient.println("{name: 'L2', color: '#000000', showInLegend: true, data:[]},");
  webClient.println("{name: 'L3', color: '#7f7f7f', showInLegend: true, data:[]}],");
  webClient.println("plotOptions:{line: {animation: false, dataLabels:{enabled: true}}},");
  webClient.println("xAxis:{type: 'datetime', dateTimeLabelFormats:{second: '%H:%M:%S'}},");
  webClient.println("yAxis:{title:{text: 'Vrms'}},credits:{enabled: false}});");
  webClient.println("var chartMAX= new Highcharts.Chart({chart:{renderTo : 'chart-MAX'},");
  webClient.println("title:{text: 'Max Line Current'},");
  webClient.println("series:[{name: 'L1', color: '#6e3e00', showInLegend: true, data:[]},");
  webClient.println("{name: 'L2', color: '#000000', showInLegend: true, data:[]},");
  webClient.println("{name: 'L3', color: '#7f7f7f', showInLegend: true, data:[]}],");
  webClient.println("plotOptions:{line: {animation: false, dataLabels:{enabled: true}}},");
  webClient.println("xAxis:{type: 'datetime', dateTimeLabelFormats:{second: '%H:%M:%S'}},");
  webClient.println("yAxis:{title:{text: 'Vrms'}},credits:{enabled: false}});");
  webClient.println("var chartUND= new Highcharts.Chart({chart:{renderTo : 'chart-UND'},");
  webClient.println("title:{text: 'Underuns'},");
  webClient.println("series:[{name: 'L1', color: '#6e3e00', showInLegend: true, data:[]},");
  webClient.println("{name: 'L2', color: '#000000', showInLegend: true, data:[]},");
  webClient.println("{name: 'L3', color: '#7f7f7f', showInLegend: true, data:[]}],");
  webClient.println("plotOptions:{line: {animation: false, dataLabels:{enabled: true}}},");
  webClient.println("xAxis:{type: 'datetime', dateTimeLabelFormats:{second: '%H:%M:%S'}},");
  webClient.println("yAxis:{title:{text: '#'}},credits:{enabled: false}});");
  webClient.println("var chartOVR= new Highcharts.Chart({chart:{renderTo : 'chart-OVR'},");
  webClient.println("title:{text: 'Overruns'},");
  webClient.println("series:[{name: 'L1', color: '#6e3e00', showInLegend: true, data:[]},");
  webClient.println("{name: 'L2', color: '#000000', showInLegend: true, data:[]},");
  webClient.println("{name: 'L3', color: '#7f7f7f', showInLegend: true, data:[]}],");
  webClient.println("plotOptions:{line: {animation: false, dataLabels:{enabled: true}}},");
  webClient.println("xAxis:{type: 'datetime', dateTimeLabelFormats:{second: '%H:%M:%S'}},");
  webClient.println("yAxis:{title:{text: '#'}},credits:{enabled: false}});");
  webClient.println("setInterval(function(){");
  webClient.println("var xhttp = new XMLHttpRequest();");
  webClient.println("xhttp.onreadystatechange = function(){");
  webClient.println("if(this.readyState == 4 && this.status == 200){");
  webClient.println("var myObj = JSON.parse(this.responseText);");
  webClient.println("var x =(new Date(myObj.time)).getTime();");
  webClient.println("if(myObj.hasOwnProperty('L1')){var r = 0;");
  webClient.println("var yavg = parseFloat(myObj.L1.avg),");
  webClient.println("ymin = parseFloat(myObj.L1.min),ymax = parseFloat(myObj.L1.max),");
  webClient.println("yund = parseFloat(myObj.L1.under),yovr = parseFloat(myObj.L1.over);");
  webClient.println("if(chartAVG.series[r].data.length > 40){");
  webClient.println("chartAVG.series[r].addPoint([x, yavg], true, true, true);");
  webClient.println("} else{chartAVG.series[r].addPoint([x, yavg], true, false, true);}");
  webClient.println("if(chartMIN.series[r].data.length > 40){");
  webClient.println("chartMIN.series[r].addPoint([x, ymin], true, true, true);");
```

```
  webClient.println("} else{chartMIN.series[r].addPoint([x, ymin], true, false, true);}");
  webClient.println("if(chartMAX.series[r].data.length > 40){");
  webClient.println("chartMAX.series[r].addPoint([x, ymax], true, true, true);");
  webClient.println("} else{chartMAX.series[r].addPoint([x, ymax], true, false, true);}");
  webClient.println("if(chartUND.series[r].data.length > 40){");
  webClient.println("chartUND.series[r].addPoint([x, yund], true, true, true);");
  webClient.println("} else{chartUND.series[r].addPoint([x, yund], true, false, true);}");
  webClient.println("if(chartOVR.series[r].data.length > 40){");
  webClient.println("chartOVR.series[r].addPoint([x, yovr], true, true, true);");
  webClient.println("} else{chartOVR.series[r].addPoint([x, yovr], true, false, true);}}");
  webClient.println("if(myObj.hasOwnProperty('L2')){var r = 1;");
  webClient.println("var yavg = parseFloat(myObj.L2.avg),");
  webClient.println("ymin = parseFloat(myObj.L2.min),ymax = parseFloat(myObj.L2.max),");
  webClient.println("yund = parseFloat(myObj.L2.under),yovr = parseFloat(myObj.L2.over);");
  webClient.println("if(chartAVG.series[r].data.length > 40){");
  webClient.println("chartAVG.series[r].addPoint([x, yavg], true, true, true);");
  webClient.println("} else{chartAVG.series[r].addPoint([x, yavg], true, false, true);}");
  webClient.println("if(chartMIN.series[r].data.length > 40){");
  webClient.println("chartMIN.series[r].addPoint([x, ymin], true, true, true);");
  webClient.println("} else{chartMIN.series[r].addPoint([x, ymin], true, false, true);}");
  webClient.println("if(chartMAX.series[r].data.length > 40){");
  webClient.println("chartMAX.series[r].addPoint([x, ymax], true, true, true);");
  webClient.println("} else{chartMAX.series[r].addPoint([x, ymax], true, false, true);}");
  webClient.println("if(chartUND.series[r].data.length > 40){");
  webClient.println("chartUND.series[r].addPoint([x, yund], true, true, true);");
  webClient.println("} else{chartUND.series[r].addPoint([x, yund], true, false, true);}");
  webClient.println("if(chartOVR.series[r].data.length > 40){");
  webClient.println("chartOVR.series[r].addPoint([x, yovr], true, true, true);");
  webClient.println("} else{chartOVR.series[r].addPoint([x, yovr], true, false, true);}}");
  webClient.println("if(myObj.hasOwnProperty('L3')){var r = 2;");
  webClient.println("var yavg = parseFloat(myObj.L3.avg),");
  webClient.println("ymin = parseFloat(myObj.L3.min),ymax = parseFloat(myObj.L3.max),");
  webClient.println("yund = parseFloat(myObj.L3.under),yovr = parseFloat(myObj.L3.over);");
  webClient.println("if(chartAVG.series[r].data.length > 40){");
  webClient.println("chartAVG.series[r].addPoint([x, yavg], true, true, true);");
  webClient.println("} else{chartAVG.series[r].addPoint([x, yavg], true, false, true);}");
  webClient.println("if(chartMIN.series[r].data.length > 40){");
  webClient.println("chartMIN.series[r].addPoint([x, ymin], true, true, true);");
  webClient.println("} else{chartMIN.series[r].addPoint([x, ymin], true, false, true);}");
  webClient.println("if(chartMAX.series[r].data.length > 40){");
  webClient.println("chartMAX.series[r].addPoint([x, ymax], true, true, true);");
  webClient.println("} else{chartMAX.series[r].addPoint([x, ymax], true, false, true);}");
  webClient.println("if(chartUND.series[r].data.length > 40){");
  webClient.println("chartUND.series[r].addPoint([x, yund], true, true, true);");
  webClient.println("} else{chartUND.series[r].addPoint([x, yund], true, false, true);}");
  webClient.println("if(chartOVR.series[r].data.length > 40){");
  webClient.println("chartOVR.series[r].addPoint([x, yovr], true, true, true);");
  webClient.println("} else{chartOVR.series[r].addPoint([x, yovr], true, false, true);}}}};");
  webClient.println("xhttp.open('GET', '/line', true);xhttp.send();}, 10000 );");
  webClient.println("</script></html>");

  // The HTTP response ends with another blank line
  webClient.println("");
}
//*****************************************************************************

//** send line data
//*****************************************************************************
void webLineData()
{
  dt = ntpLocalDateTime();
  strcpy(caJson, "{");
  sprintf(caTemp, "\"time\": \"%s+00\"", dt);
  strcat(caJson, caTemp);

  for (int r=0; r<3; r++)
  {
    if(bLine[r])
    {
      sprintf(caTemp, ", \"L%d\":{" , r+1);
      strcat(caJson, caTemp);
      sprintf(caTemp, "\"avg\": %lu, " , avgRMS[r]);
      strcat(caJson, caTemp);
      sprintf(caTemp, "\"min\": %u, "  , wMinA[r]);
      strcat(caJson, caTemp);
      sprintf(caTemp, "\"max\": %u, "  , wMaxA[r]);
      strcat(caJson, caTemp);
      sprintf(caTemp, "\"under\": %u, ", wUnderA[r]);
      strcat(caJson, caTemp);
      sprintf(caTemp, "\"over\": %u"   , wOverA[r]);
```

```
      strcat(caJson, caTemp);
      strcat(caJson, "}");
    }
  }
  strcat(caJson, "}");

  #if defined(DEBUG)
    Serial.println(caJson);
  #endif

  webClient.println(caJson);
  // The HTTP response ends with another blank line
  webClient.println("");
}
//****************************************************************************

//** replace characters in string and pad with spaces
//****************************************************************************
void str_replace(char *src, char *oldchars, char *newchars)
{ // utility string function
  String tmp;

  tmp = src;
  while(tmp.length() < 4)
  {
    tmp = " " + tmp;
  }
  strcpy(src, tmp.c_str());

  char *p = strstr(src, oldchars);
  char buf[MAX_STRING_LENGTH];
  do
  {
    if (p)
    {
      memset(buf, '\0', strlen(buf));
      if (src == p)
      {
        strcpy(buf, newchars);
        strcat(buf, p + strlen(oldchars));
      }
      else
      {
        strncpy(buf, src, strlen(src) - strlen(p));
        strcat(buf, newchars);
        strcat(buf, p + strlen(oldchars));
      }
      memset(src, '\0', strlen(src));
      strcpy(src, buf);
    }
  } while (p && (p = strstr(src, oldchars)));

}
//****************************************************************************
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//============================================================================
// Setup
//============================================================================
void setup()
{
  #if defined(CALIBRATE)
    Serial.begin(115200);
    delay(1000);
    Serial.println(F(""));Serial.println(F(""));
    Serial.println(F("======================="));
    Serial.print(F("Booting (Version: ")); Serial.print(VERSION); Serial.println(F(")"));
    Serial.println(F("======================="));

    //--------------------------------------------------------------------------
    // Startup timer interrupt
    //--------------------------------------------------------------------------
    //set prescaler every microsecond eg: clockspeed 80.000.000Hz / 80 = 1.000.000Hz -> 1 µs period
    timer = timerBegin(0, CLOCKSPEED, true);
    timerAttachInterrupt(timer, &onTimer, true);
    // interrupt every 1 millisecond = 1000 µs
    timerAlarmWrite(timer, 1000, true);
    timerAlarmEnable(timer);

  #else
```

```
#if defined(DEBUG)
  Serial.begin(115200);
  delay(1000);
  Serial.println(F(""));Serial.println(F(""));
  Serial.println(F("======================="));
  Serial.print(F("Booting (Version: ")); Serial.print(VERSION); Serial.println(F(")"));
  Serial.println(F("======================="));
#endif

// Set led pin as output
pinMode(LED_BUILTIN, OUTPUT);
// Start ticker to indicate set-up mode
ticker.attach(0.5, tick);

//-----------------------------------------------------------------------------
// Set Setting Portal button Pin as input
//-----------------------------------------------------------------------------
pinMode(PIN_BUTTON, INPUT_PULLUP);

//-----------------------------------------------------------------------------
// Allow time to press button to reset to factory defaults
//-----------------------------------------------------------------------------
delay(4000);

// check if button is pressed to restore factory defaults
buttTime = millis();
// Is button pressed?
while(digitalRead(PIN_BUTTON) == LOW)
{
  delay(100);
}
long pressDuration = millis() - buttTime;
if( pressDuration > LONG_PRESS_TIME )
{
  #if defined(DEBUG)
    Serial.println("Portal Button: A long press is detected");
    Serial.println("Formatting SPIFFS");
  #endif
  SPIFFS.format();
  delay(3000);
  #if defined(DEBUG)
    Serial.println("Restarting ...");
  #endif
  ESP.restart();
}

//-----------------------------------------------------------------------------
// Startup WiFi
//-----------------------------------------------------------------------------
WiFiSettings.hostname = myHostname;
wifiSettings();

#if defined(DEBUG)
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.print("  MAC Address: "); Serial.println( WiFi.macAddress().c_str());
  Serial.print("  IP Address : "); Serial.println(WiFi.localIP());
  Serial.print("  Hostname   : "); Serial.println(WiFi.getHostname());
#endif
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Startup WiFi AP
//-----------------------------------------------------------------------------
WiFi.mode(WIFI_MODE_APSTA);
WiFi.softAP(wifiAPSSID, wifiAPPassword);

//-----------------------------------------------------------------------------
// Startup time
//-----------------------------------------------------------------------------
ntpSetup();

//-----------------------------------------------------------------------------
// Startup MDNS
//-----------------------------------------------------------------------------
mdnsSetup();

//-----------------------------------------------------------------------------
// Startup OTA
```

```
    //-----------------------------------------------------------------------
    otaSetup();

    //-----------------------------------------------------------------------
    // Setup webserver
    //-----------------------------------------------------------------------
    wsSetup();

    //-------------------------------------------------------------------------
    // Startup mqtt
    //-------------------------------------------------------------------------
    if(bMQTT)
    {
      mqttConnect();
    }

    //-------------------------------------------------------------------------
    // Startup adc reading
    //-------------------------------------------------------------------------

    //-------------------------------------------------------------------------
    // Startup timer interrupt
    //-------------------------------------------------------------------------
    //set prescaler every microsecond eg: clockspeed 80.000.000Hz / 80 = 1.000.000Hz -> 1 µs period
    timer = timerBegin(0, CLOCKSPEED, true);
    timerAttachInterrupt(timer, &onTimer, true);
    // interrupt every 1 millisecond = 1000 µs
    timerAlarmWrite(timer, 1000, true);
    timerAlarmEnable(timer);

    // Ending set-up mode, turn led off to save power
    ticker.detach();
    digitalWrite(LED_BUILTIN, LOW);
  #endif

  #if defined(CALIBRATE)
    Serial.println(F("Ready - Calibrate"));
    bLine[0] = LINE0;
    bLine[1] = LINE1;
    bLine[2] = LINE2;
  #endif

  #if defined(DEBUG)
    Serial.println(F("Ready"));
  #endif
}
//=========================================================================

//#########################################################################
// Main loop
//#########################################################################
void loop()
{

  unsigned long sum = 0;

  #if defined(CALIBRATE)

    unsigned int avgA[] = {0,0,0};
    unsigned int avgZ[] = {0,0,0};

    // OTA handling
    ArduinoOTA.handle();

    if (cntrINT > 0)
    {
      // read the ADC.
      valZER[0][cntADC] = (unsigned int)(analogRead(ADC_INPUT0));
      ADCval            = (float)(valZER[0][cntADC]);
      ADCval            = (ADCval - ADCzero[0]) * ADCamp0;
      valADC[0][cntADC] = (unsigned int)(ADCval);

      valZER[1][cntADC] = (unsigned int)(analogRead(ADC_INPUT1));
      ADCval            = (float)(valZER[1][cntADC]);
      ADCval            = (ADCval - ADCzero[1]) * ADCamp1;
      valADC[1][cntADC] = (unsigned int)(ADCval);

      valZER[2][cntADC] = (unsigned int)(analogRead(ADC_INPUT2));
      ADCval            = (float)(valZER[2][cntADC]);
      ADCval            = (ADCval - ADCzero[2]) * ADCamp2;
```

```
    valADC[2][cntADC] = (unsigned int)(ADCval);

    cntADC++;
    if( cntADC >= iADC)
    {
      // reset counter
      cntADC = 0;

      // find DC offset and RMS
      for (int r=0; r<3; r++)
      {
        // find DC offset
        // skip first as it might not be correct due to long loop
        sum = 0;
        for (int n=1; n<iADC; n++)
        {
          sum = sum + (unsigned long)valZER[r][n];
        }
        avgZER[r][cntRMS] = (unsigned int)(sum / (iADC - 1));
        // adjust Zero offset for current
        ADCzero[r] = avgZER[r][cntRMS];

        // find RMS
        float valV;
        unsigned long sqrV;
        // now calc RMS values and average RMS
        sum = 0;
        for (int n=1; n<iADC; n++)
        {
          valV = valADC[r][n];
          sqrV = (unsigned long)(valV * valV);
          sum = sum + sqrV;
        }
        // average rms value
        valRMS[r][cntRMS] = (unsigned int)(sqrt(sum / (iADC - 1)));
      }

      cntRMS++;
      // array filled? If so, average and send out
      // reset counters and values
      // check button
      if(cntRMS >= iRMS)
      {
        cntRMS = 0;

        for (int r=0; r<3; r++)
        {

          sum = 0;
          // find average DC offset
          for (int n=0; n<iRMS; n++)
          {
            sum   = sum + (unsigned long)avgZER[r][n];
          }
          avgZ[r] = (unsigned int)(sum / iRMS);

          // find average RMS
          sum = 0;
          for (int n=0; n<iRMS; n++)
          {
            sum = sum + valRMS[r][n];
          }
          // The average RMS value of ADC values
          avgA[r] = (unsigned int)(sum / iRMS);

          if(bLine[r])
          {
            Serial.printf(" Zero %d ADC value : %u \n",       r+1, avgZ[r]);
            Serial.printf(" Line %d Current   : %u Vrms\n\n", r+1, avgA[r]);
          }
        }
      }
    }
    portENTER_CRITICAL(&timerMux);
    cntrINT = 0;
    portEXIT_CRITICAL(&timerMux);
  }

#else
  unsigned int rmsVal;
```

```
float        frqVal;

if (cntrINT > 0)
{

  // read the ADC.
  if(bLine[2])
  {
    valZER[2][cntADC] = (unsigned int)(analogRead(ADC_INPUT2));
    ADCval            = (float)(valZER[2][cntADC]);
    ADCval            = (ADCval - ADCzero[2]) * ADCamp2;
    valADC[2][cntADC] = (unsigned int)(ADCval);
  }

  if(bLine[1])
  {
    valZER[1][cntADC] = (unsigned int)(analogRead(ADC_INPUT1));
    ADCval            = (float)(valZER[1][cntADC]);
    ADCval            = (ADCval - ADCzero[1]) * ADCamp1;
    valADC[1][cntADC] = (unsigned int)(ADCval);
  }

  if(bLine[0])
  {
    valZER[0][cntADC] = (unsigned int)(analogRead(ADC_INPUT0));
    ADCval            = (float)(valZER[0][cntADC]);
    ADCval            = (ADCval - ADCzero[0]) * ADCamp0;
    valADC[0][cntADC] = (unsigned int)(ADCval);

    // initial begining stage of measurement when current wave larger than 0
    if((ADCval > 0)  && stage == 3)
    {
      // allow to change to the next stage
      stage = 0;
    }

    // when current wave smaller or equal than 0
    if((ADCval <= 0) && stage == 0)
    {
      // start counting time for all
      // store start time
      TimeStart = micros();
      // allow to change to the next stage
      stage     = 1;
    }

    // when current wave is larger than 0, prepare to find peak
    if((ADCval > 0)  && stage == 1)
    {
      // reset value
      PREval = 0;
      // allow to change to the next stage
      stage  = 2;
    }

    // if current measured value larger than previous peak value
    if((ADCval > PREval) && stage == 2)
    {
      // record current time for current wave
      TimePeriod = micros();
      // record current measure value replace previous peak value
      PREval   = ADCval;
    }

    // when wave current smaller or equal than 0
    if((ADCval <= 0) && stage == 2)
    {
      // record current time for 1 period
      TimePeriod = micros();
      // accumulate or add up time for all sample readings of period wave
      sumPeriod  = sumPeriod + (TimePeriod - TimeStart);
      // time difference between current peak value and current peak value
      TimeStart  = TimePeriod;
      // reset peak value
      PREval     = 0;
      // set stage mode
      stage      = 1;
    }
  }
```

```
cntADC++;
// every iADC milliseconds
// average and store away
if( cntADC >= iADC)
{
  nowMicros = micros();

  // app handling
  cntADC = 0;

  // calc frequency - only for line 1
  frqVal = (float)(1000000 / (sumPeriod / (iADC / (1000 / nomFREQ))));
  // frequency must be between + and - 1 (by regulations) otherwise we read wrong
  // nominal variation is +/- 10mHz,  exceptionally +/- 200mHz and, when in troeuble,
  // a short period of +/- 800mHz
  if (frqVal >= (nomFREQ + 1) || frqVal <= (nomFREQ - 1))
  {
    if(cntRMS > 1)
    {
      // use previous frequency
      frqVal = valFRQ[cntRMS-1];
    }
    else
    {
      // use nominal frequency
      frqVal = nomFREQ;
    }
  }
  valFRQ[cntRMS] = frqVal;
  sumPeriod      = 0;

  long valV;
  for (int r=0; r<3; r++)
  {

    if(bLine[r])
    {

      // find DC offset
      // skip first as it might not be correct due to long loop
      sum = 0;
      for (int n=1; n<iADC; n++)
      {
        sum = sum + (unsigned long)valZER[r][n];
      }
      avgZER[r][cntRMS] = (unsigned int)(sum / (iADC - 1));
      // adjust Zero offset
      ADCzero[r]        = avgZER[r][cntRMS];

      sum = 0;
      // calc RMS value
      for (int n=1; n<iADC; n++)
      {
        valV = (long)valADC[r][n];
        sum = sum + (valV * valV);
      }
      // average rms value
      rmsVal = (unsigned int)(sqrt(sum / (iADC - 1)));
      valRMS[r][cntRMS] = rmsVal;

      // // counts
      // if(rmsVal <= minAMP)
      // {
        // uUnderA[r]++;
      // }
      // else if(rmsVal >= maxAMP)
      // {
        // uOverA[r]++;
      // }

      //value
      if(rmsVal <= uMinA[r])
      {
        uMinA[r] = rmsVal;
      }
      else if(rmsVal >= uMaxA[r])
      {
        uMaxA[r] = rmsVal;
      }
```

```
      // max volt can never be lower than min volt and vice vresa
      // sometimes I see this happening in the graphs, and that should not be the case
      // still need to find the real issue why this can happen = TODO/TODEBUG
      // This is a temp "fix"
      // if ( uMaxA[r] < uMinA[r])
      // {
        // uMaxA[r] = uMinA[r];
      // }
      // else if ( uMinA[r] > uMaxA[r])
      // {
        // uMinA[r] = uMaxA[r];
      // }

      uDiffA[r] = uMaxA[r] - uMinA[r];

      // save info for webpage
      for(int r=0; r<3; r++)
      {
        wUnderA[r] = 0;
        wOverA[r]  = 0;
        wMaxA[r]   = uMaxA[r];
        wMinA[r]   = uMinA[r];
        wDiffA[r]  = uDiffA[r];
      }
    }
    else
    {
      valRMS[r][cntRMS] = 0;
      wUnderA[r]        = 0;
      wOverA[r]         = 0;
      wMaxA[r]          = 0;
      wMinA[r]          = 0;
      wDiffA[r]         = 0;
    }
  }

  cntRMS++;
  // array filled? If so, average and send out
  // reset counters and values
  // check button
  if(cntRMS >= iRMS)
  {

    if(bMQTT)
    {
      // keep mqtt connection alive
      if (!mqttClient.connected())
      {
        mqttReconnect();
      }
      mqttClient.loop();
    }

    // OTA handling
    ArduinoOTA.handle();

    cntRMS = 0;

    float fSum = 0;
    for (int n=0; n<iRMS; n++)
    {
      fSum = fSum + valFRQ[n];
    }
    // average frequency value
    avgFRQ = (float)(fSum / iRMS);

    if(bMQTT)
    {
      strcpy(caJson, "{ ");
      dt = ntpLocalDateTime();
      sprintf(caTemp, "\"TimeStamp\": \"%s\", ", dt);
      strcat(caJson, caTemp);
      sprintf(caTemp, "\"avg\": %.2f"  , avgFRQ);
      strcat(caJson, caTemp);
      strcat(caJson, "}");
      sprintf(caTemp, "%s/Freq", mqttTopicDat);
      mqttSend(caTemp, caJson);
    }

    for (int r=0; r<3; r++)
```

```
{
  sum = 0;
  for (int n=0; n<iRMS; n++)
  {
    sum = sum + valRMS[r][n];
  }
  // average RMS values
  avgRMS[r] = (unsigned int)(sum / iRMS);

  if(!bLine[r])
  {
    avgRMS[r]  = 0;
    uUnderA[r] = 0;
    uOverA[r]  = 0;
    uMaxA[r]   = 0;
    uMinA[r]   = 0;
    uDiffA[r]  = 0;
  }

  if(bMQTT)
  {
    strcpy(caJson, "{ ");
    dt = ntpLocalDateTime();
    sprintf(caTemp, "\"TimeStamp\": \"%s\", ", dt);
    strcat(caJson, caTemp);
    sprintf(caTemp, "\"avg\": %u, "  , avgRMS[r]);
    strcat(caJson, caTemp);
    sprintf(caTemp, "\"min\": %u, "  , uMinA[r]);
    strcat(caJson, caTemp);
    sprintf(caTemp, "\"max\": %u, "  , uMaxA[r]);
    strcat(caJson, caTemp);
    sprintf(caTemp, "\"dif\": %u, "  , uDiffA[r]);
    strcat(caJson, caTemp);
    sprintf(caTemp, "\"under\": %u, ", uUnderA[r]);
    strcat(caJson, caTemp);
    sprintf(caTemp, "\"over\": %u"   , uOverA[r]);
    strcat(caJson, caTemp);
    strcat(caJson, "}");
    sprintf(caTemp, "%s/Line%u", mqttTopicDat, r+1);
    mqttSend(caTemp, caJson);
  }

  uUnderA[r] = 0;
  uOverA[r]  = 0;
  uMaxA[r]   = 0;
  uMinA[r]   = 999;
  uDiffA[r]  = 0;
}

if(bMQTT)
{
  // let system know you are still online
  mqttSend(mqttTopicSts, (char*)"online");
}

// Listen for incoming clients
webClient = webServer.available();
// If a new webClient connects,
if(webClient)
{
  wsClientActive();
}

//-----------------------------------------------------------------------------
// checking button every ...
//-----------------------------------------------------------------------------
buttState = Button();
if (buttState == LONG_PRESS_TIME)
{
  // clearout SPIFFS and restart from scratch
  #if defined(DEBUG)
    Serial.println("Formatting SPIFFS");
  #endif
  SPIFFS.format();
  delay(3000);
  #if defined(DEBUG)
    Serial.println("Restarting ...");
  #endif
  ESP.restart();
}
```

```
          else if (buttState == SHRT_PRESS_TIME)
          {
            //just restart
            #if defined(DEBUG)
              Serial.println("Restarting ...");
            #endif
            ESP.restart();
          }
        }

        #if defined(DEBUG)
          // timing measurement
          difMicros = micros() - nowMicros;
          if (difMicros > 250)
          {
            Serial.print("runtime: "); Serial.print(difMicros); Serial.println(" µs");
          }
        #endif
      }
      // to make sure we do not handle pilled up interrupts
      // and keep a pace of 1 every ms
      portENTER_CRITICAL(&timerMux);
      cntrINT = 0;
      portEXIT_CRITICAL(&timerMux);
    }
  #endif
}
```

**MainsVALogger.h**

```
// System settings
// some control setting related to debugging
// uncomment to do some general debugging via Serial
//#define DEBUG
// uncomment to do debugging of WEB via Serial
//#define DEBUGWEB

//#define CALIBRATE

#define myHostname           "MainsVALogger"

#define wifiAPSSID           myHostname
#define wifiAPPassword       ""

#define ntpServer            "be.pool.ntp.org"
#define ntpUTCOffset_sec     3600
#define ntpDSTOffset_sec     3600
#define ntpUpdate_sec        60

// OTA settings
// Port default is 3232
#define otaPort              3232
#define otaPassword          ""
// Password can be set with it's md5 value as well
// Eg:  MD5 of "admin" = 21232f297a57a5a743894a0e4a801fc3
#define otaPasswordHash      ""

#define PIN_BUTTON           15
#define LONG_PRESS_TIME    6000  // 6s
#define SHRT_PRESS_TIME    3000  // 3s

#define ADC_INPUT0           34  // define the used ADC input channel
#define ADC_INPUT1           35  // define the used ADC input channel

#define CLOCKSPEED           80  // clockspeed of ESP32 in MHz

#define nomVOLT             230  // nominal RMS voltage on the line
#define maxVOLT             253  // 230 + 10%
#define minVOLT             216  // 230 -  6%

#define nomFREQ             50.00 // nominal frequency
#define maxFREQ             50.01 // + 10mHz
#define minFREQ             49.99 // - 10mHz

//LINE 1 -> voltage
#define ADCvolt             (float)0.2010    // volt per ADC count

//LINE 2 -> current
#define ADCamp              (float)0.0540    // ampere per ADC count (*10)
```

**MainsVALogger.ino**

```
//-----------------------------------------------------------------------------
// what/how to make it
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// includes
//-----------------------------------------------------------------------------
#include <Ticker.h>
#include <WiFi.h>
#include <ESPmDNS.h>
#include <WiFiUdp.h>
#include <NTPClient.h>
#include <PubSubClient.h>
#include <math.h>
#include <SPIFFS.h>
#include <WiFiSettings.h>
#include <ArduinoOTA.h>

#include "MainsVALogger.h"

//-----------------------------------------------------------------------------
// defines
//-----------------------------------------------------------------------------
#define VERSION           "1.50"

// webserver timeout in milliseconds (example: 2000ms = 2s)
#define webserverTimeout  1000

// max length of a MQTT topic
#define MQTT_TOPIC_SIZE    128

// max length of string
#define MAX_STRING_LENGTH  256

//-----------------------------------------------------------------------------
// constants
//-----------------------------------------------------------------------------
// number of samples to take before RMS is calculate
const int iADC = 2 * nomFREQ;    // one sample every 1 ms -> 100ms = 5 periods @ 50Hz
// number of Vrms to hold before average Vrms is outputted
const int iRMS =  50;            // 100 x 5 periods -> 100 * 100 ms -> 5 seconds -> every 5 seconds
MQTT message

//-----------------------------------------------------------------------------
// global variables
//-----------------------------------------------------------------------------
// time related
unsigned long time_now;
char today[11];
// to hold date & time
char *dt;
int days[] = {31,29,31,30,31,30,31,31,30,31,30,31};
char  nuDate[12];
char  nuTime[10];
char  currDST[3];
char  lastDST[3];
// Current timeing
unsigned long nowMicros  = micros();
unsigned long difMicros  = 0;
unsigned long currMillis = millis();
// Previous timing
unsigned long prevMillis = 0;

// vars related to button
unsigned long buttTime  = 0;
int           buttState;
int           buttLast  = HIGH;      // the previous state from the input pin
int           buttCurr  = HIGH;      // the current  state from the input pin

// vars related to MQTT
boolean bMQTT          = true;
String  mqttServer     = "mqtt.home";
int     mqttPort       = 1883;
String  mqttLogin      = "";
String  mqttPassword   = "";
String  mqttTopicBase  = myHostname;

char mqttClientID[30];
```

```
char mqttTopicSts[MQTT_TOPIC_SIZE];
char mqttTopicDat[MQTT_TOPIC_SIZE];

// vars to store characters
char caTemp[MAX_STRING_LENGTH];
char caTmp[MAX_STRING_LENGTH];
char caJson[MAX_STRING_LENGTH * 2];

// counters for array index
int  cntADC = 0;
int  cntRMS = 0;

// vars related to ADC and Voltages
float ADCval;

unsigned long valZER[2][iADC];
unsigned long avgZER[2][iRMS];
float         ADCzero[2] = {2000,1000};
unsigned int  valADC[2][iADC];
unsigned int  valRMS[2][iRMS];
float         valFRQ[iRMS];
unsigned int  avgRMS[2];

float         avgFRQ;
unsigned int  uUnderV = 0;  // drops
unsigned int  uOverV  = 0;  // spikes
unsigned int  uMaxV   = 0;
unsigned int  uMinV   = 999;
unsigned int  uDiffV  = 0;

unsigned int  uMaxA   = 0;
unsigned int  uMinA   = 999;
unsigned int  uDiffA  = 0;

// vars related to webpage
unsigned int wUnderV = 0;  // drops
unsigned int wOverV  = 0;  // spikes
unsigned int wMaxV   = 0;
unsigned int wMinV   = 999;
unsigned int wDiffV  = 0;

unsigned int wMaxA   = 0;
unsigned int wMinA   = 999;
unsigned int wDiffA  = 0;

// Frequency measurement
unsigned int stage         = 3;          // use for switching operation
                                         // stage = 0  => voltage still positive
                                         // stage = 1  => voltage has crossed 0 and start time is
recorded
                                         // stage = 2  => looking for peak, recording time
                                         // stage = 3  => timing not started

float PREval               = 0;          // is the previous value for voltage sensor
float sumPeriod            = 0;          // is the accumulate time differences for voltage wave

unsigned long TimeStart;                 // starting time for Phase Angle and Period (in micro
seconds).
unsigned long TimePeriod;                // current  time for period of wave (in micro seconds).


// needed for timer interrupt
volatile byte cntrINT;
hw_timer_t * timer    = NULL;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

// webserver related
// Variable to store the HTTP requests
String httpHeader;

//-----------------------------------------------------------------------------
// objects
//-----------------------------------------------------------------------------
// Initiate led blinker library
Ticker       ticker;

// WiFi objects
WiFiUDP      ntpUDP;
WiFiClient   wifiClient;
```

```
// NTP object
NTPClient    ntpClient(ntpUDP, ntpServer, ntpUTCOffset_sec, ntpUpdate_sec * 1000);

// MQTT object
PubSubClient mqttClient(wifiClient);

// object for webserver
// set web server port number to 80
WiFiServer  webServer(80);
WiFiClient  webClient;

//-----------------------------------------------------------------------------
// Functions
//-----------------------------------------------------------------------------

//.............................................................................
// Interrupt functions
//.............................................................................
//** Timer interrupt
//*****************************************************************************
void IRAM_ATTR onTimer()
{
  portENTER_CRITICAL_ISR(&timerMux);
  cntrINT++;
  portEXIT_CRITICAL_ISR(&timerMux);
}
//*****************************************************************************

//.............................................................................
// Call-back functions
//.............................................................................
//** Call-back TICKER
//*****************************************************************************
void tick()
{
  // get the current state of GPIO1 pin
  int state = digitalRead(LED_BUILTIN);
  // set pin to the opposite state
  digitalWrite(LED_BUILTIN, !state);
}
//*****************************************************************************

// +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// +++ WIFI Functions +++
// +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//** Portal for settings
//*****************************************************************************
void wifiSettings()
{
  // Format SPIFF if mount fails.
  // fail might happen at first time when nothing was done before
  SPIFFS.begin(true);

  WiFiSettings.begin();
  // default to ESP32- followed by last 3 pairs of HEX numbers of MAC address, reversed
  // eg:  ac:67:b2:37:8e:60 -> esp32-608e37
  WiFiSettings.hostname = myHostname;
  //WiFiSettings.language = "en";

  WiFiSettings.heading("MQTT");
  bMQTT         = WiFiSettings.checkbox("MQTT Active", bMQTT, "MQTT Active");
  mqttServer    = WiFiSettings.string("mqtt_server", 64, mqttServer, "MQTT Server");
  mqttPort      = WiFiSettings.integer("mqtt_port", 0, 65535, mqttPort, "MQTT Port");
  mqttLogin     = WiFiSettings.string("mqtt_login", 64, mqttLogin, "MQTT Login");
  mqttPassword  = WiFiSettings.string("mqtt_password", 64, mqttPassword, "MQTT Password");
  mqttTopicBase = WiFiSettings.string("mqtt_topic", 128, mqttTopicBase, "MQTT Topic Base");

  // Set custom callback functions
  WiFiSettings.onSuccess      = [](){};  // do nothing
  WiFiSettings.onConfigSaved  = [](){};  // do nothing
  WiFiSettings.onWaitLoop     = []()
  {
    // Delay next function call by 500ms
    return 500;
  };

  WiFiSettings.onConnect = []()
  {
    // Delay next function call by 50ms
```

```
    return 50;
  };

  WiFiSettings.onFailure = []()
  {
    // Delay 2s
    delay(2000);
  };

  static unsigned int  portal_phase = 0;
  static unsigned long portal_start;
  WiFiSettings.onPortal = [](){portal_start = millis();};

  WiFiSettings.onPortalView = [](){if (portal_phase < 2){portal_phase = 2;}};

  WiFiSettings.onConfigSaved = [](){portal_phase = 3;};

  WiFiSettings.onPortalWaitLoop = []()
  {
    // anyone connected?
    if (WiFi.softAPgetStationNum() == 0)
    {
      portal_phase = 0;
    }
    else
    {
      if (!portal_phase)
      {
        portal_phase = 1;
      }
    }

    // do not wait endless
    if (portal_phase == 0 && millis() - portal_start > 5*60*1000)
    {
      // reboot if nothing happened within 5 minutes = 5 * 60 * 1000 milliseconds
      ESP.restart();
    }
  };

  // try to connect to WiFi, if not, start settings portal
  if (WiFiSettings.connect(true, 15))
  {
    #if defined(DEBUG)
      Serial.println("Connected to Wifi Network");
    #endif
  }

}
//*****************************************************************************
// +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

// +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// +++ NTP/Time Functions +++
// +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//** Set-up NTP
//*****************************************************************************
void ntpSetup()
{
  #if defined(DEBUG)
    Serial.println("NTP setup");
  #endif

  // clear some vars
  memset(lastDST, 0x00, 3);
  memset(currDST, 0x00, 3);

  // Init NTP
  ntpClient.begin();

  String nu = ntpClient.getFormattedDate();
  #if defined(DEBUG)
    Serial.print("Init    ->  Date - Time: "); Serial.print(nu); Serial.println();
  #endif

  ntpClient.forceUpdate();

  nu = ntpClient.getFormattedDate();
  #if defined(DEBUG)
```

```
    Serial.print("Updated ->  Date - Time: "); Serial.print(nu); Serial.println();
  #endif

  //adjust DST if needed
  checkDST();

  nu = ntpClient.getFormattedDate();
  #if defined(DEBUG)
    Serial.print("Final   ->  Date - Time: "); Serial.print(nu); Serial.println();
  #endif

  #if defined(DEBUG)
    Serial.println("NTP activated");
  #endif
}
//****************************************************************************

//** Check for DST
//****************************************************************************
void checkDST()
{
  #if defined(DEBUG)
    Serial.println("DST Checking ....");
  #endif

  // due to problem/bug in time /timezone, the following code
  // is required to make sure we have the right offset for DST

  // make sure we have recently updated
  ntpClient.forceUpdate();

  // get date
  String nu = ntpClient.getFormattedDate().substring(0,19);
  #if defined(DEBUG)
    Serial.println(nu);
  #endif

  int y, m, w;
  // correct for leapyear
  y = nu.substring(0,4).toInt();
  days[1] -= (y % 4) || (!(y % 100) && (y % 400));
  w = y * 365 + 97 * (y - 1) / 400 + 4;

  char buff[20];
  // find last sunday of March
  m = 2; // March
  w = (w + days[m]) % 7;
  sprintf(buff, "%04d-%02d-%02dT02:00:00", y, m + 1,days[m] - w);
  String DSTstart(buff);
  #if defined(DEBUG)
    Serial.println(DSTstart);
  #endif

  // find last sunday of Octobre
  m = 9; // Octobre
  w = (w + days[m]) % 7;
  sprintf(buff, "%04d-%02d-%02dT03:00:00", y, m + 1,days[m] - w);
  String DSTend(buff);
  #if defined(DEBUG)
    Serial.println(DSTend);
  #endif

  // if nu is between March 02:00:00 and Oct 03:00:00
  // DST is active (at least till 2024)
  if(nu >= DSTstart && nu < DSTend)
  {
    // DST = UTC + ntpUTCOffset_sec + ntpDSTOffset_sec
    ntpClient.setTimeOffset(ntpUTCOffset_sec + ntpDSTOffset_sec);
    #if defined(DEBUG)
      Serial.println("DST set");
    #endif
  }
  else
  {
    // non-DST = UTC + ntpUTCOffset_sec
    ntpClient.setTimeOffset(ntpUTCOffset_sec);
    #if defined(DEBUG)
      Serial.println("DST reset");
    #endif
  }
```

```
  // make sure we update local date/time
  ntpClient.forceUpdate();

  // save last check of DST
  sprintf(lastDST, "%02d", ntpClient.getHours());
  sprintf(currDST, "%02d", ntpClient.getHours());
  ntpClient.getFormattedTime().toCharArray(nuTime, 8);
  nuTime[9] = 0x00;

  #if defined(DEBUG)
    Serial.println("DST Checking done");
  #endif
}

//** Get Date & Time
//*****************************************************************************
char* ntpLocalDateTime()
{
  static char caDT[21];

  String nu = ntpClient.getFormattedDate().substring(0,19);
  // convert to char array
  nu.toCharArray(caDT, 20);
  // remove T
  caDT[10] = ' ';
  //terminate to string
  caDT[20] = 0x00;

  return(caDT);
}
//*****************************************************************************
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// +++ mDNS Functions +++
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//** Set-up mDNS
//*****************************************************************************
void mdnsSetup()
{
  #if defined(DEBUG)
    Serial.println(F("mDNS activating"));
  #endif

  if(!MDNS.begin(myHostname))
  {
    #if defined(DEBUG)
      Serial.println(F("mDNS activation FAILED"));
    #endif
    return;
  }

  // indicate there is a webserver on this device
  MDNS.addService("http", "tcp", 80);

  #if defined(DEBUG)
    Serial.println(F("mDNS activated"));
  #endif
}
//*****************************************************************************
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// +++ MQTT Functions +++
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//** Set-up MQTT
//*****************************************************************************
void mqttConnect()
{

  #if defined(DEBUG)
    Serial.println(F("MQTT activating"));
    Serial.print(F("MQTT attempting connection to ")); Serial.print(mqttServer);
Serial.print(F(":")); Serial.println(mqttPort);
  #endif

  // mqttClientID
```

```
  strcpy(mqttClientID, myHostname);
  #if defined(DEBUG)
    Serial.print(F("  MQTT ClientID: <")); Serial.print(mqttClientID); Serial.println(F(">"));
  #endif

  // mqttTopicSts
  strcpy(mqttTopicSts, mqttTopicBase.c_str());
  strcat(mqttTopicSts, "/status");
  #if defined(DEBUG)
    Serial.print(F("  MQTT Topic STS: <")); Serial.print(mqttTopicSts); Serial.println(F(">"));
  #endif

  // mqttTopicDat
  strcpy(mqttTopicDat, mqttTopicBase.c_str());
  strcat(mqttTopicDat, "/data");
  #if defined(DEBUG)
    Serial.print(F("  MQTT Topic DAT: <")); Serial.print(mqttTopicDat); Serial.println(F(">"));
  #endif

  mqttClient.setServer(mqttServer.c_str(), mqttPort);

  // ESP will connect to mqtt broker with clientID, last will, ...
  // bool connect(const char* id, const char* user, const char* pass, const char* willTopic, uint8_t
willQos, bool willRetain, const char* willMessage, bool cleanSession);
  if (mqttClient.connect(mqttClientID, mqttLogin.c_str(), mqttPassword.c_str(), mqttTopicSts, 1,
true, "offline", false))
  {

    #if defined(DEBUG)
      Serial.println(F("MQTT connected"));
    #endif

    // let know we are online
    mqttClient.publish(mqttTopicSts, "online", true);
    #if defined(DEBUG)
      Serial.println(F("MQTT activated"));
    #endif
  }
  else
  {
    #if defined(DEBUG)
      Serial.println(F("MQTT activation Failed"));
    #endif
  }

}
//****************************************************************************

//****************************************************************************
void mqttReconnect()
{
  uint8_t i = 0;

  while (!mqttClient.connected())
  {
    #if defined(DEBUG)
      Serial.print(F("MQTT attempting reconnection to ")); Serial.print(mqttServer);
Serial.print(F(":")); Serial.println(mqttPort);
    #endif
    if (mqttClient.connect(mqttClientID, mqttLogin.c_str(), mqttPassword.c_str(), mqttTopicSts, 1,
true, "offline", false))
    {
      #if defined(DEBUG)
        Serial.println(F("MQTT reconnected"));
      #endif
      // let know we are online
      mqttClient.publish(mqttTopicSts, "online", true);
    }
    else
    {
      #if defined(DEBUG)
        Serial.print(F("MQTT reconnection failed, rc=")); Serial.print(mqttClient.state());
Serial.println(F(" ... try again in 5 seconds"));
      #endif
      // Wait 15 seconds before retrying
      // delay non-blocking
      time_now = millis();
      while(millis() < time_now + 1500){}
      ++i;
      if (i > 5)
```

```
      {
        #if defined(DEBUG)
          Serial.println(F("Restarting..."));
        #endif
        ESP.restart();
      }
    }
  }
}
//****************************************************************************

// ** send MQTT messages
//****************************************************************************
void mqttSend(char *topic, char* data)
{
  mqttClient.publish(topic, data, true);
}
//****************************************************************************
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// +++ OTA Functions +++
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//** Set-up OTA
//****************************************************************************
void otaSetup()
{
  #if defined(DEBUG)
    Serial.println("OTA activating");
  #endif

  // Port defaults to 3232
  ArduinoOTA.setPort(otaPort);

  // Hostname defaults to esp32-[MAC]
  ArduinoOTA.setHostname(myHostname);

  // Set OTA Password. No authentication by default
  if(strlen(otaPasswordHash) != 0)
  {
    ArduinoOTA.setPasswordHash(otaPasswordHash);
  }
  else if(strlen(otaPassword) != 0)
  {
    ArduinoOTA.setPassword(otaPassword);
  }

  ArduinoOTA
    .onStart([]()
    {
      String type;
      if(ArduinoOTA.getCommand() == U_FLASH)
        type = "sketch";
      else // U_SPIFFS
        type = "filesystem";

      // NOTE: if updating SPIFFS this would be the place to unmount SPIFFS using SPIFFS.end()
      #if defined(DEBUG)
        Serial.println(" - Start updating " + type);
      #endif
    })

    .onProgress([](unsigned int progress, unsigned int total)
    {
      #if defined(DEBUG)
        Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
      #endif
    })

    .onEnd([]()
    {
      #if defined(DEBUG)
        Serial.print("\n"); Serial.println(" - End");
      #endif
    })

    .onError([](ota_error_t error)
    {
      #if defined(DEBUG)
```

```
        Serial.printf(" - Error[%u]: ", error);
        if(error == OTA_AUTH_ERROR)         Serial.println("Auth Failed");
        else if(error == OTA_BEGIN_ERROR)   Serial.println("Begin Failed");
        else if(error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
        else if(error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
        else if(error == OTA_END_ERROR)     Serial.println("End Failed");
      #endif
    }

  );

  ArduinoOTA.begin();
  #if defined(DEBUG)
    Serial.println("OTA activated");
  #endif
}
//****************************************************************************
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// +++ Button Functions +++
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//** Button logic long press to activate Settings Portal or to restart
//****************************************************************************
unsigned int Button()
{
  // read the state of the button:
  buttCurr = digitalRead(PIN_BUTTON);

  // Is button is pressed or released?
  if(buttLast == HIGH && buttCurr == LOW)
  {
    buttLast = buttCurr;
    #if defined(DEBUG)
      Serial.println("Portal Button: Pressed");
    #endif
    buttTime = millis();
  }
  else
  {
    if(buttLast == LOW && buttCurr == HIGH)
    {
      buttLast = buttCurr;
      #if defined(DEBUG)
        Serial.println("Portal Button: Released");
      #endif
      long pressDuration = millis() - buttTime;

      if( pressDuration > LONG_PRESS_TIME )
      {
        #if defined(DEBUG)
          Serial.println("Portal Button: A long press is detected");
        #endif
        return LONG_PRESS_TIME;
      }
      else
      {
        if ( pressDuration > SHRT_PRESS_TIME )
        {
          #if defined(DEBUG)
            Serial.println("Portal Button: A short press is detected");
          #endif
          return SHRT_PRESS_TIME;
        }
        else
        {
          #if defined(DEBUG)
            Serial.println("Portal Button: A very short press is detected");
          #endif
          return 0;
        }
      }
    }
  }
  buttLast = buttCurr;
  return false;
}
//****************************************************************************
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// +++ Webserver Functions +++
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//** Set-up webserver
//*****************************************************************************
void wsSetup()
{
  #if defined(DEBUG)
    Serial.println("Webserver activating");
  #endif

  webServer.begin();

  #if defined(DEBUG)
    Serial.println("Webserver activated");
  #endif
}
//*****************************************************************************

//** listen & talk to webclient
//*****************************************************************************
void wsClientActive()
{
  char c;

  currMillis = millis();
  prevMillis = currMillis;
  // print a message out in the serial port
  #if defined(DEBUG)
    Serial.println("Client connected");
  #endif

  // make a String to hold incoming data from the webClient
  String currentLine = "";

  // loop while the webClient's connected and no timeout occured
  while (webClient.connected() && currMillis - prevMillis <= webserverTimeout)
  {
    currMillis = millis();
    // if there's bytes to read from the webClient,
    if(webClient.available())
    {
      c = webClient.read();           // read a byte, then
      #if defined(DEBUGWEB)
        Serial.write(c);              // print it out the serial monitor
      #endif
      httpHeader += c;                // add it to httpHeader var

      // if the byte is a newline character
      if(c == '\n')
      {
        #if defined(DEBUGWEB)
          Serial.println("New line");
        #endif

        // if the current line is blank, you got two newlines in a row.
        // that's the end of the webClient HTTP request
        // check what was sent, act as required and then send a response:
        if(currentLine.length() == 0)
        {
          // first send response as HTTP headers always start with a response code
          // and a content-type so the webClient knows what's coming,
          webClient.println("HTTP/1.1 200 OK");
          webClient.println("Content-type:text/html");
          webClient.println("Connection: close");
          // then a blank line:
          webClient.println("");

          // decode if action is required
          // action on home page?
          if(httpHeader.indexOf("GET /HOME") >= 0)
          {
            webHomePage();
            break;
          }
          else if(httpHeader.indexOf("GET /CHART") >= 0)
          {
            // send home page
            webChartPage();
```

```
            break;
          }
          else if(httpHeader.indexOf("GET /line") >= 0)
          {
            // send home page
            webLineData();
            break;
          }
          // send home page
          webHomePage();
          break;
        }
        else
        {
          // if you got a newline, then clear currentLine
          currentLine = "";
        }
      }
      else if(c != '\r')
      {
        // if you got anything else but a carriage return character,
        // add it to the end of the currentLine
        currentLine += c;
      }
      else if(c == '\r')
      {
        #if defined(DEBUGWEB)
          Serial.println("New return");
        #endif
      }
    }
  }

  #if defined(DEBUGWEB)
    Serial.print("<"); Serial.print(httpHeader); Serial.println(">");
  #endif

  // Clear the httpHeader variable
  httpHeader = "";

  // now queue is empty, stop it
  webClient.stop();
  #if defined(DEBUG)
    Serial.println("Client disconnected.");
  #endif
}
//****************************************************************************

//** send home page
//****************************************************************************
void webHomePage()
{
  // Display the HTML web page
  webClient.println("<!DOCTYPE html>");
  // Open the HTML web page
  webClient.println("<html>");
  // Open head
  webClient.println("<head><meta name=\"viewport\" content=\"width=device-width, initial-
scale=1\">");
  webClient.println("<meta http-equiv=\"refresh\" content=\"60\">");
  webClient.println("<link rel=\"icon\" href=\"data:,\">");
  // Open style for CSS settings
  webClient.println("<style>");
  // overall style setting
  webClient.println("html, body { width: 100%; height: 100%; padding: 0; margin: 0; font-
family: \"Verdana\", sans-serif;}");
  webClient.println(".centered-wrapper { position: relative; text-align: center; font-
family: \"Verdana\", sans-serif;}");
  webClient.println(".centered-wrapper:before { content: ""; position: relative; display: inline-
block; width: 0; height: 100%; vertical-align: middle; font-family: \"Verdana\", sans-serif;}");
  webClient.println(".centered-content { display: inline-block; vertical-align: middle; font-family:
\"Verdana\", sans-serif;}");
  webClient.println(".btn { background-color: #0099ff; border: none; color: white; width: 200px;
height: 50px; text-decoration: none; font-size: 20px; margin: 2px; cursor: pointer;}");
  webClient.println("</style></head>");

  // body of page
  webClient.println("<body class=\"centered-wrapper\">");
  webClient.println("<div class=\"centered-content\">");
  webClient.print("<h1>"); webClient.print(myHostname); webClient.println("</h1>");
```

```
  webClient.print("<table style='font-family: Courier, monospace; font-size: 20px'>");
  webClient.print("<tr>");
  dt = ntpLocalDateTime();
  sprintf(caTemp, "%s           ", dt +
11);
  webClient.print("<td><label><b>Time    :</b></
label><label> ");webClient.print(caTemp);webClient.print("</label></td>");
  webClient.print("</tr><tr>");
  webClient.print("<td><label>         </
label><label>  L1 -  L2     </label></td>");
  webClient.print("</tr><tr>");
  sprintf(caTmp, "%lu", avgRMS[0]); str_replace(caTmp, (char*)" ", (char*)" "); strcpy(caTemp,
caTmp); strcat(caTemp, " -");
  sprintf(caTmp, "%lu", avgRMS[1]); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp);

webClient.print("<td><label><b>AvgVolt :</b></label><label>");webClient.print(caTemp);webClient
.print(" Vrms</label></td>");
  webClient.print("</tr><tr>");
  sprintf(caTmp, "%u", wMaxV); str_replace(caTmp, (char*)" ", (char*)" "); strcpy(caTemp,
caTmp); strcat(caTemp, " -");
  sprintf(caTmp, "%u", wMaxA); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp);

webClient.print("<td><label><b>MaxVolt :</b></label><label>");webClient.print(caTemp);webClient
.print(" Vrms</label></td>");
  webClient.print("</tr><tr>");
  sprintf(caTmp, "%u", wMinV); str_replace(caTmp, (char*)" ", (char*)" "); strcpy(caTemp,
caTmp); strcat(caTemp, " -");
  sprintf(caTmp, "%u", wMinA); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp);

webClient.print("<td><label><b>MinVolt :</b></label><label>");webClient.print(caTemp);webClient
.print(" Vrms</label></td>");
  webClient.print("</tr><tr>");
  sprintf(caTmp, "%u", wOverV); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp);

webClient.print("<td><label><b>Overrun :</b></label><label>");webClient.print(caTemp);webClient
.print("     </label></td>");
  webClient.print("</tr><tr>");
  sprintf(caTmp, "%u", wUnderV); str_replace(caTmp, (char*)" ", (char*)" "); strcat(caTemp,
caTmp);

webClient.print("<td><label><b>Underrun:</b></label><label>");webClient.print(caTemp);webClient.prin
t("     </label></td>");
  webClient.print("</tr><tr>");
  webClient.print("<td> </td><td> </td><td> </td>");
  webClient.print("</tr>");
  webClient.print("</table>");
  webClient.print("<p><a href=\"/HOME\"><button class=\"btn\">Refresh</button></p>");
  webClient.print("<p><a href=\"/CHART\"><button class=\"btn\">Chart</button></p>");
  webClient.println("</div>");
  webClient.println("</body></html>");

  // The HTTP response ends with another blank line
  webClient.println("");
}
//*****************************************************************************

//** send select line page
//*****************************************************************************
//** send chart page
//*****************************************************************************
void webChartPage()
{
  webClient.println("<!DOCTYPE HTML><html><head>");
  webClient.println("<meta name='viewport' content='width=device-width, initial-scale=1'>");
  webClient.println("<script src='https://code.highcharts.com/highcharts.js'></script>");
  webClient.println("<style>html, body{width: 100%; height: 100%; padding: 0; margin: 0; font-
family: \"Verdana\", sans-serif;}");
  webClient.println(".centered-wrapper { position: relative; text-align: center; font-
family: \"Verdana\", sans-serif;}");
  webClient.println(".centered-wrapper:before { content: ""; position: relative; display: inline-
block; width: 0; height: 100%; vertical-align: middle; font-family: \"Verdana\", sans-serif;}");
  webClient.println(".centered-content { display: inline-block; vertical-align: middle; font-family:
\"Verdana\", sans-serif;}");
  webClient.println(".btn { background-color: #0099ff; border: none; color: white;  width: 200px;
height: 50px; text-decoration: none; font-size: 20px; margin: 2px; cursor: pointer;}");
  webClient.println("</style></head>");
```

```
webClient.println("<body class=\"centered-wrapper\">");
webClient.println("<div class=\"centered-content\">");
webClient.print("<h1>"); webClient.print(myHostname); webClient.println("</h1>");
webClient.println("<div id='chart-AVG' class='container'></div>");
webClient.println("<div id='chart-MIN' class='container'></div>");
webClient.println("<div id='chart-MAX' class='container'></div>");
webClient.println("<div id='chart-UND' class='container'></div>");
webClient.println("<div id='chart-OVR' class='container'></div>");
webClient.println("<p><a href=\"/HOME\"><button class=\"btn\">Home</button></p>");
webClient.println("</div></body><script>");
webClient.println("var chartAVG= new Highcharts.Chart({chart:{renderTo : 'chart-AVG'},");
webClient.println("title:{text: 'Avg Line Voltage'},");
webClient.println("series:[{name: 'L1', color: '#6e3e00', showInLegend: true, data:[]},");
webClient.println("{name: 'L2', color: '#7f7f7f', showInLegend: true, data:[]}],");
webClient.println("plotOptions:{line: {animation: false, dataLabels:{enabled: true}}},");
webClient.println("xAxis:{type: 'datetime', dateTimeLabelFormats:{second: '%H:%M:%S'}},");
webClient.println("yAxis:{title:{text: 'Vrms'}},credits:{enabled: false}});");
webClient.println("var chartMIN= new Highcharts.Chart({chart:{renderTo : 'chart-MIN'},");
webClient.println("title:{text: 'Min Line Voltage'},");
webClient.println("series:[{name: 'L1', color: '#6e3e00', showInLegend: true, data:[]},");
webClient.println("{name: 'L2', color: '#7f7f7f', showInLegend: true, data:[]}],");
webClient.println("plotOptions:{line: {animation: false, dataLabels:{enabled: true}}},");
webClient.println("xAxis:{type: 'datetime', dateTimeLabelFormats:{second: '%H:%M:%S'}},");
webClient.println("yAxis:{title:{text: 'Vrms'}},credits:{enabled: false}});");
webClient.println("var chartMAX= new Highcharts.Chart({chart:{renderTo : 'chart-MAX'},");
webClient.println("title:{text: 'Max Line Voltage'},");
webClient.println("series:[{name: 'L1', color: '#6e3e00', showInLegend: true, data:[]},");
webClient.println("{name: 'L2', color: '#7f7f7f', showInLegend: true, data:[]}],");
webClient.println("plotOptions:{line: {animation: false, dataLabels:{enabled: true}}},");
webClient.println("xAxis:{type: 'datetime', dateTimeLabelFormats:{second: '%H:%M:%S'}},");
webClient.println("yAxis:{title:{text: 'Vrms'}},credits:{enabled: false}});");
webClient.println("var chartUND= new Highcharts.Chart({chart:{renderTo : 'chart-UND'},");
webClient.println("title:{text: 'Underuns'},");
webClient.println("series:[{name: 'L1', color: '#6e3e00', showInLegend: true, data:[]},");
webClient.println("{name: 'L2', color: '#7f7f7f', showInLegend: true, data:[]}],");
webClient.println("plotOptions:{line: {animation: false, dataLabels:{enabled: true}}},");
webClient.println("xAxis:{type: 'datetime', dateTimeLabelFormats:{second: '%H:%M:%S'}},");
webClient.println("yAxis:{title:{text: '#'}},credits:{enabled: false}});");
webClient.println("var chartOVR= new Highcharts.Chart({chart:{renderTo : 'chart-OVR'},");
webClient.println("title:{text: 'Overruns'},");
webClient.println("series:[{name: 'L1', color: '#6e3e00', showInLegend: true, data:[]},");
webClient.println("{name: 'L2', color: '#7f7f7f', showInLegend: true, data:[]}],");
webClient.println("plotOptions:{line: {animation: false, dataLabels:{enabled: true}}},");
webClient.println("xAxis:{type: 'datetime', dateTimeLabelFormats:{second: '%H:%M:%S'}},");
webClient.println("yAxis:{title:{text: '#'}},credits:{enabled: false}});");
webClient.println("setInterval(function(){");
webClient.println("var xhttp = new XMLHttpRequest();");
webClient.println("xhttp.onreadystatechange = function(){");
webClient.println("if(this.readyState == 4 && this.status == 200){");
webClient.println("var myObj = JSON.parse(this.responseText);");
webClient.println("var x =(new Date(myObj.time)).getTime();");
webClient.println("if(myObj.hasOwnProperty('L1')){var r = 0;");
webClient.println("var yavg = parseFloat(myObj.L1.avg),");
webClient.println("ymin = parseFloat(myObj.L1.min),ymax = parseFloat(myObj.L1.max),");
webClient.println("yund = parseFloat(myObj.L1.under),yovr = parseFloat(myObj.L1.over);");
webClient.println("if(chartAVG.series[r].data.length > 40){");
webClient.println("chartAVG.series[r].addPoint([x, yavg], true, true, true);");
webClient.println("} else{chartAVG.series[r].addPoint([x, yavg], true, false, true);}");
webClient.println("if(chartMIN.series[r].data.length > 40){");
webClient.println("chartMIN.series[r].addPoint([x, ymin], true, true, true);");
webClient.println("} else{chartMIN.series[r].addPoint([x, ymin], true, false, true);}");
webClient.println("if(chartMAX.series[r].data.length > 40){");
webClient.println("chartMAX.series[r].addPoint([x, ymax], true, true, true);");
webClient.println("} else{chartMAX.series[r].addPoint([x, ymax], true, false, true);}");
webClient.println("if(chartUND.series[r].data.length > 40){");
webClient.println("chartUND.series[r].addPoint([x, yund], true, true, true);");
webClient.println("} else{chartUND.series[r].addPoint([x, yund], true, false, true);}");
webClient.println("if(chartOVR.series[r].data.length > 40){");
webClient.println("chartOVR.series[r].addPoint([x, yovr], true, true, true);");
webClient.println("} else{chartOVR.series[r].addPoint([x, yovr], true, false, true);}}");
webClient.println("if(myObj.hasOwnProperty('L2')){var r = 1;");
webClient.println("var yavg = parseFloat(myObj.L2.avg),");
webClient.println("ymin = parseFloat(myObj.L2.min),ymax = parseFloat(myObj.L2.max),");
webClient.println("yund = parseFloat(myObj.L2.under),yovr = parseFloat(myObj.L2.over);");
webClient.println("if(chartAVG.series[r].data.length > 40){");
webClient.println("chartAVG.series[r].addPoint([x, yavg], true, true, true);");
webClient.println("} else{chartAVG.series[r].addPoint([x, yavg], true, false, true);}");
webClient.println("if(chartMIN.series[r].data.length > 40){");
webClient.println("chartMIN.series[r].addPoint([x, ymin], true, true, true);");
webClient.println("} else{chartMIN.series[r].addPoint([x, ymin], true, false, true);}");
```

```
  webClient.println("if(chartMAX.series[r].data.length > 40){");
  webClient.println("chartMAX.series[r].addPoint([x, ymax], true, true, true);");
  webClient.println("} else{chartMAX.series[r].addPoint([x, ymax], true, false, true);}");
  webClient.println("if(chartUND.series[r].data.length > 40){");
  webClient.println("chartUND.series[r].addPoint([x, yund], true, true, true);");
  webClient.println("} else{chartUND.series[r].addPoint([x, yund], true, false, true);}");
  webClient.println("if(chartOVR.series[r].data.length > 40){");
  webClient.println("chartOVR.series[r].addPoint([x, yovr], true, true, true);");
  webClient.println("} else{chartOVR.series[r].addPoint([x, yovr], true, false, true);}}");
  webClient.println("if(myObj.hasOwnProperty('L3')){var r = 2;");
  webClient.println("var yavg = parseFloat(myObj.L3.avg),");
  webClient.println("ymin = parseFloat(myObj.L3.min),ymax = parseFloat(myObj.L3.max),");
  webClient.println("yund = parseFloat(myObj.L3.under),yovr = parseFloat(myObj.L3.over);");
  webClient.println("if(chartAVG.series[r].data.length > 40){");
  webClient.println("chartAVG.series[r].addPoint([x, yavg], true, true, true);");
  webClient.println("} else{chartAVG.series[r].addPoint([x, yavg], true, false, true);}");
  webClient.println("if(chartMIN.series[r].data.length > 40){");
  webClient.println("chartMIN.series[r].addPoint([x, ymin], true, true, true);");
  webClient.println("} else{chartMIN.series[r].addPoint([x, ymin], true, false, true);}");
  webClient.println("if(chartMAX.series[r].data.length > 40){");
  webClient.println("chartMAX.series[r].addPoint([x, ymax], true, true, true);");
  webClient.println("} else{chartMAX.series[r].addPoint([x, ymax], true, false, true);}");
  webClient.println("if(chartUND.series[r].data.length > 40){");
  webClient.println("chartUND.series[r].addPoint([x, yund], true, true, true);");
  webClient.println("} else{chartUND.series[r].addPoint([x, yund], true, false, true);}");
  webClient.println("if(chartOVR.series[r].data.length > 40){");
  webClient.println("chartOVR.series[r].addPoint([x, yovr], true, true, true);");
  webClient.println("} else{chartOVR.series[r].addPoint([x, yovr], true, false, true);}}}};");
  webClient.println("xhttp.open('GET', '/line', true);xhttp.send();}, 10000 );");
  webClient.println("</script></html>");

  // The HTTP response ends with another blank line
  webClient.println("");
}
//*****************************************************************************

//** send line data
//*****************************************************************************
void webLineData()
{
  dt = ntpLocalDateTime();
  strcpy(caJson, "{");
    sprintf(caTemp, "\"time\": \"%s+00\"", dt);
    strcat(caJson, caTemp);

    sprintf(caTemp, ", \"L0\":{");
    strcat(caJson, caTemp);
      sprintf(caTemp, "\"avg\": %lu, " , avgRMS[0]);
      strcat(caJson, caTemp);
      sprintf(caTemp, "\"min\": %u, "  , wMinV);
      strcat(caJson, caTemp);
      sprintf(caTemp, "\"max\": %u, "  , wMaxV);
      strcat(caJson, caTemp);
      sprintf(caTemp, "\"under\": %u, ", wUnderV);
      strcat(caJson, caTemp);
      sprintf(caTemp, "\"over\": %u"   , wOverV);
      strcat(caJson, caTemp);
    strcat(caJson, "}");

    sprintf(caTemp, ", \"L1\":{");
    strcat(caJson, caTemp);
      sprintf(caTemp, "\"avg\": %lu, " , avgRMS[1]);
      strcat(caJson, caTemp);
      sprintf(caTemp, "\"min\": %u, "  , wMinA);
      strcat(caJson, caTemp);
      sprintf(caTemp, "\"max\": %u, "  , wMaxA);
      strcat(caJson, caTemp);
    strcat(caJson, "}");

  strcat(caJson, "}");

  #if defined(DEBUG)
    Serial.println(caJson);
  #endif

  webClient.println(caJson);
  // The HTTP response ends with another blank line
  webClient.println("");
}
//*****************************************************************************
```

```
//** replace characters in string and pad with spaces
//*****************************************************************************
void str_replace(char *src, char *oldchars, char *newchars)
{ // utility string function
  String tmp;

  tmp = src;
  while(tmp.length() < 4)
  {
    tmp = " " + tmp;
  }
  strcpy(src, tmp.c_str());

  char *p = strstr(src, oldchars);
  char buf[MAX_STRING_LENGTH];
  do
  {
    if (p)
    {
      memset(buf, '\0', strlen(buf));
      if (src == p)
      {
        strcpy(buf, newchars);
        strcat(buf, p + strlen(oldchars));
      }
      else
      {
        strncpy(buf, src, strlen(src) - strlen(p));
        strcat(buf, newchars);
        strcat(buf, p + strlen(oldchars));
      }
      memset(src, '\0', strlen(src));
      strcpy(src, buf);
    }
  } while (p && (p = strstr(src, oldchars)));

}
//*****************************************************************************
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//=============================================================================
// Setup
//=============================================================================
void setup()
{
  #if defined(CALIBRATE)
    Serial.begin(115200);
    delay(1000);
    Serial.println(F(""));Serial.println(F(""));
    Serial.println(F("========================"));
    Serial.print(F("Booting (Version: ")); Serial.print(VERSION); Serial.println(F(")"));
    Serial.println(F("========================"));

    //-----------------------------------------------------------------------------
    // Startup timer interrupt
    //-----------------------------------------------------------------------------
    //set prescaler every microsecond eg: clockspeed 80.000.000Hz / 80 = 1.000.000Hz -> 1 µs period
    timer = timerBegin(0, CLOCKSPEED, true);
    timerAttachInterrupt(timer, &onTimer, true);
    // interrupt every 1 millisecond = 1000 µs
    timerAlarmWrite(timer, 1000, true);
    timerAlarmEnable(timer);

  #else

    #if defined(DEBUG)
      Serial.begin(115200);
      delay(1000);
      Serial.println(F(""));Serial.println(F(""));
      Serial.println(F("========================"));
      Serial.print(F("Booting (Version: ")); Serial.print(VERSION); Serial.println(F(")"));
      Serial.println(F("========================"));
    #endif

    // Set led pin as output
    pinMode(LED_BUILTIN, OUTPUT);
    // Start ticker to indicate set-up mode
    ticker.attach(0.5, tick);
```

```
//-----------------------------------------------------------------------------
// Set Setting Portal button Pin as input
//-----------------------------------------------------------------------------
pinMode(PIN_BUTTON, INPUT_PULLUP);

//-----------------------------------------------------------------------------
// Allow time to press button to reset to factory defaults
//-----------------------------------------------------------------------------
delay(4000);

// check if button is pressed to restore factory defaults
buttTime = millis();
// Is button pressed?
while(digitalRead(PIN_BUTTON) == LOW)
{
  delay(100);
}
long pressDuration = millis() - buttTime;
if( pressDuration > LONG_PRESS_TIME )
{
  #if defined(DEBUG)
    Serial.println("Portal Button: A long press is detected");
    Serial.println("Formatting SPIFFS");
  #endif
  SPIFFS.format();
  delay(3000);
  #if defined(DEBUG)
    Serial.println("Restarting ...");
  #endif
  ESP.restart();
}

//-----------------------------------------------------------------------------
// Startup WiFi
//-----------------------------------------------------------------------------
WiFiSettings.hostname = myHostname;
wifiSettings();

#if defined(DEBUG)
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.print("  MAC Address: "); Serial.println( WiFi.macAddress().c_str());
  Serial.print("  IP Address : "); Serial.println(WiFi.localIP());
  Serial.print("  Hostname   : "); Serial.println(WiFi.getHostname());
#endif
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Startup WiFi AP
//-----------------------------------------------------------------------------
WiFi.mode(WIFI_MODE_APSTA);
WiFi.softAP(wifiAPSSID, wifiAPPassword);

//-----------------------------------------------------------------------------
// Startup time
//-----------------------------------------------------------------------------
ntpSetup();

//-----------------------------------------------------------------------------
// Startup MDNS
//-----------------------------------------------------------------------------
mdnsSetup();

//-----------------------------------------------------------------------------
// Startup OTA
//-----------------------------------------------------------------------------
otaSetup();

//-----------------------------------------------------------------------------
// Setup webserver
//-----------------------------------------------------------------------------
wsSetup();

//-----------------------------------------------------------------------------
// Startup mqtt
//-----------------------------------------------------------------------------
if(bMQTT)
{
  mqttConnect();
}
```

```
    //-------------------------------------------------------------------------
    // Startup adc reading
    //-------------------------------------------------------------------------

    //-------------------------------------------------------------------------
    // Startup timer interrupt
    //-------------------------------------------------------------------------
    //set prescaler every microsecond eg: clockspeed 80.000.000Hz / 80 = 1.000.000Hz -> 1 µs period
    timer = timerBegin(0, CLOCKSPEED, true);
    timerAttachInterrupt(timer, &onTimer, true);
    // interrupt every 1 millisecond = 1000 µs
    timerAlarmWrite(timer, 1000, true);
    timerAlarmEnable(timer);

    // Ending set-up mode, turn led off to save power
    ticker.detach();
    digitalWrite(LED_BUILTIN, LOW);
  #endif

  #if defined(CALIBRATE)
    Serial.println(F("Ready - Calibrate"));
  #endif

  #if defined(DEBUG)
    Serial.println(F("Ready"));
  #endif
}
//==========================================================================

//##########################################################################
// Main loop
//##########################################################################
void loop()
{

  unsigned long sum = 0;

  //::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
  #if defined(CALIBRATE)

    unsigned int avgVA[] = {0,0};
    unsigned int avgZE[] = {0,0};

    // OTA handling
    ArduinoOTA.handle();

    if (cntrINT > 0)
    {
      // read the ADC.
      valZER[0][cntADC] = (unsigned int)(analogRead(ADC_INPUT0));
      ADCval            = (float)(valZER[0][cntADC]);
      ADCval            = (ADCval - ADCzero[0]) * ADCvolt;
      valADC[0][cntADC] = (unsigned int)(ADCval);

      valZER[1][cntADC] = (unsigned int)(analogRead(ADC_INPUT1));
      ADCval            = (float)(valZER[1][cntADC]);
      ADCval            = (ADCval - ADCzero[1]) * ADCamp;
      valADC[1][cntADC] = (unsigned int)(ADCval);

      cntADC++;
      if( cntADC >= iADC)
      {
        // reset counter
        cntADC = 0;

        // find DC offset and RMS
        for (int r=0; r<2; r++)
        {
          // find DC offset
          // skip first as it might not be correct due to long loop
          sum = 0;
          for (int n=1; n<iADC; n++)
          {
            sum = sum + (unsigned long)valZER[r][n];
          }
          avgZER[r][cntRMS] = (unsigned int)(sum / (iADC - 1));
          // adjust Zero offset for current
          ADCzero[r] = avgZER[r][cntRMS];
```

```
        // find RMS
        long valV;
        unsigned long sqrV;
        // now calc RMS values and average RMS
        sum = 0;
        for (int n=1; n<iADC; n++)
        {
          valV = valADC[r][n];
          sqrV = (unsigned long)(valV * valV);
          sum = sum + sqrV;
        }
        // average rms value
        valRMS[r][cntRMS] = (unsigned int)(sqrt(sum / (iADC - 1)));
      }

      cntRMS++;
      // array filled? If so, average and send out
      // reset counters and values
      // check button
      if(cntRMS >= iRMS)
      {
        cntRMS = 0;

        for (int r=0; r<2; r++)
        {
          sum = 0;
          // find average DC offset
          for (int n=0; n<iRMS; n++)
          {
            sum   = sum + (unsigned long)avgZER[r][n];
          }
          avgZE[r] = (unsigned int)(sum / iRMS);

          // find average RMS
          sum = 0;
          for (int n=0; n<iRMS; n++)
          {
            sum = sum + valRMS[r][n];
          }
          // The average RMS value of ADC values
          avgVA[r] = (unsigned int)(sum / iRMS);
        }

        Serial.printf(" Line Voltage   : %u Vrms\n",    avgVA[0]);
        Serial.printf(" Zero ADC value : %u \n\n",      avgZE[0]);
        Serial.printf(" Line Current   : %.1f Arms\n",  (float)avgVA[1] / 10);
        Serial.printf(" Zero ADC value : %u \n",        avgZE[1]);
        Serial.printf("-----------------------------------\n");

      }
    }
    portENTER_CRITICAL(&timerMux);
    cntrINT = 0;
    portEXIT_CRITICAL(&timerMux);
  }

//::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
#else
//::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
  unsigned int rmsVal;
  float        frqVal;

  if (cntrINT > 0)
  {

    //-------------------------------------------------------------------------
    // read the ADC.
    //-------------------------------------------------------------------------
    // current
    valZER[1][cntADC] = (unsigned int)(analogRead(ADC_INPUT1));
    ADCval            = (float)(valZER[1][cntADC]);
    ADCval            = (ADCval - ADCzero[1]) * ADCamp;
    valADC[1][cntADC] = (unsigned int)(ADCval);

    //-------------------------------------------------------------------------
    // voltage
    valZER[0][cntADC] = (unsigned int)(analogRead(ADC_INPUT0));
    ADCval            = (float)(valZER[0][cntADC]);
    ADCval            = (ADCval - ADCzero[0]) * ADCvolt;
    valADC[0][cntADC] = (unsigned int)(ADCval);
```

```
//-----------------------------------------------------------------------------
// initial begining stage of measurement when voltage wave larger than 0
if((ADCval > 0)  && stage == 3)
{
  // allow to change to the next stage
  stage = 0;
}

// when voltage wave smaller or equal than 0
if((ADCval <= 0) && stage == 0)
{
  // start counting time for all
  // store start time
  TimeStart = micros();
  // allow to change to the next stage
  stage     = 1;
}

// when voltage wave is larger than 0, prepare to find peak
if((ADCval > 0)  && stage == 1)
{
  // reset value
  PREval = 0;
  // allow to change to the next stage
  stage  = 2;
}

// if current measured value larger than previous peak value
if((ADCval > PREval) && stage == 2)
{
  // record current time for voltage wave
  TimePeriod = micros();
  // record current measure value replace previous peak value
  PREval   = ADCval;
}

// when wave voltage smaller or equal than 0
if((ADCval <= 0) && stage == 2)
{
  // record current time for 1 period
  TimePeriod = micros();
  // accumulate or add up time for all sample readings of period wave
  sumPeriod  = sumPeriod + (TimePeriod - TimeStart);
  // time difference between voltage peak value and current peak value
  TimeStart  = TimePeriod;
  // reset peak value
  PREval     = 0;
  // set stage mode
  stage      = 1;
}

//-----------------------------------------------------------------------------
// every iADC millisecond
// average and store away
//-----------------------------------------------------------------------------
cntADC++;
if( cntADC >= iADC)
{
  nowMicros = micros();

  // app handling
  cntADC = 0;

  //-------------------------------------------------------------------------
  // calc frequency - only for line 1
  frqVal = (float)(1000000 / (sumPeriod / (iADC / (1000 / nomFREQ))));
  // frequency must be between + and - 1 (by regulations) otherwise we read wrong
  // nominal variation is +/- 10mHz,  exceptionally +/- 200mHz and,
  // when in trouble, a short period of +/- 800mHz
  if (frqVal >= (nomFREQ + 1) || frqVal <= (nomFREQ - 1))
  {
    if(cntRMS > 1)
    {
      // use previous frequency
      frqVal = valFRQ[cntRMS-1];
    }
    else
    {
      // use nominal frequency
```

```
    frqVal = nomFREQ;
  }
}
valFRQ[cntRMS] = frqVal;
sumPeriod      = 0;

//-------------------------------------------------------------------------
// voltage = line0

// find DC offset
// skip first as it might not be correct due to long loop
sum = 0;
for (int n=1; n<iADC; n++)
{
  sum = sum + (unsigned long)valZER[0][n];
}
avgZER[0][cntRMS] = (unsigned int)(sum / (iADC - 1));
// adjust Zero offset
ADCzero[0]        = avgZER[0][cntRMS];

long valV;
sum = 0;
// calc RMS value
for (int n=1; n<iADC; n++)
{
  valV = (long)valADC[0][n];
  sum = sum + (valV * valV);
}
// average rms value
rmsVal = (unsigned int)(sqrt(sum / (iADC - 1)));
valRMS[0][cntRMS] = rmsVal;

// counts
if(rmsVal <= minVOLT)
{
  uUnderV++;
}
else if(rmsVal >= maxVOLT)
{
  uOverV++;
}

//value
if(rmsVal <= uMinV)
{
  uMinV = rmsVal;
}
else if(rmsVal >= uMaxV)
{
  uMaxV = rmsVal;
}

// max volt can never be lower than min volt and vice vresa
// sometimes I see this happening in the graphs, and that should not be the case
// still need to find the real issue why this can happen = TODO/TODEBUG
// This is a temp "fix"
if ( uMaxV < uMinV)
{
  uMaxV = uMinV;
}
else if ( uMinV > uMaxV)
{
  uMinV = uMaxV;
}

uDiffV = uMaxV - uMinV;

// save info for webpage
wUnderV = uUnderV;
wOverV  = uOverV;
wMaxV   = uMaxV;
wMinV   = uMinV;
wDiffV  = uDiffV;

//-------------------------------------------------------------------------
// current = line1

// find DC offset
// skip first as it might not be correct due to long loop
sum = 0;
```

```
for (int n=1; n<iADC; n++)
{
  sum = sum + (unsigned long)valZER[1][n];
}
avgZER[1][cntRMS] = (unsigned int)(sum / (iADC - 1));
// adjust Zero offset
ADCzero[1]        = avgZER[1][cntRMS];

// find RMS
long valA;
unsigned long sqrA;
// now calc RMS values and average RMS
sum = 0;
for (int n=1; n<iADC; n++)
{
  valA = valADC[1][n];
  sqrA = (unsigned long)(valA * valA);
  sum = sum + sqrA;
}
// average rms value
valRMS[1][cntRMS] = (unsigned int)(sqrt(sum / (iADC - 1)));

//-------------------------------------------------------------------------
// array filled? If so, average and send out
// reset counters and values
// check button
//-------------------------------------------------------------------------
cntRMS++;
if(cntRMS >= iRMS)
{
  cntRMS = 0;

  //-----------------------------------------------------------------------
  // some stuff to keep things going
  //-----------------------------------------------------------------------
  if(bMQTT)
  {
    // keep mqtt connection alive
    if (!mqttClient.connected())
    {
      mqttReconnect();
    }
    mqttClient.loop();
  }

  // OTA handling
  ArduinoOTA.handle();

  //-----------------------------------------------------------------------
  // frequentie
  float fSum = 0;
  for (int n=0; n<iRMS; n++)
  {
    fSum = fSum + valFRQ[n];
  }
  // average frequency value
  avgFRQ = (float)(fSum / iRMS);

  if(bMQTT)
  {
    strcpy(caJson, "{ ");
    dt = ntpLocalDateTime();
    sprintf(caTemp, "\"TimeStamp\": \"%s\", ", dt);
    strcat(caJson, caTemp);
    sprintf(caTemp, "\"avg\": %.2f"  , avgFRQ);
    strcat(caJson, caTemp);
    strcat(caJson, "}");
    sprintf(caTemp, "%s/Freq", mqttTopicDat);
    mqttSend(caTemp, caJson);
  }

  //-----------------------------------------------------------------------
  // voltage = line0
  sum = 0;
  for (int n=0; n<iRMS; n++)
  {
    sum = sum + valRMS[0][n];
  }
  // average RMS values
  avgRMS[0] = (unsigned int)(sum / iRMS);
```

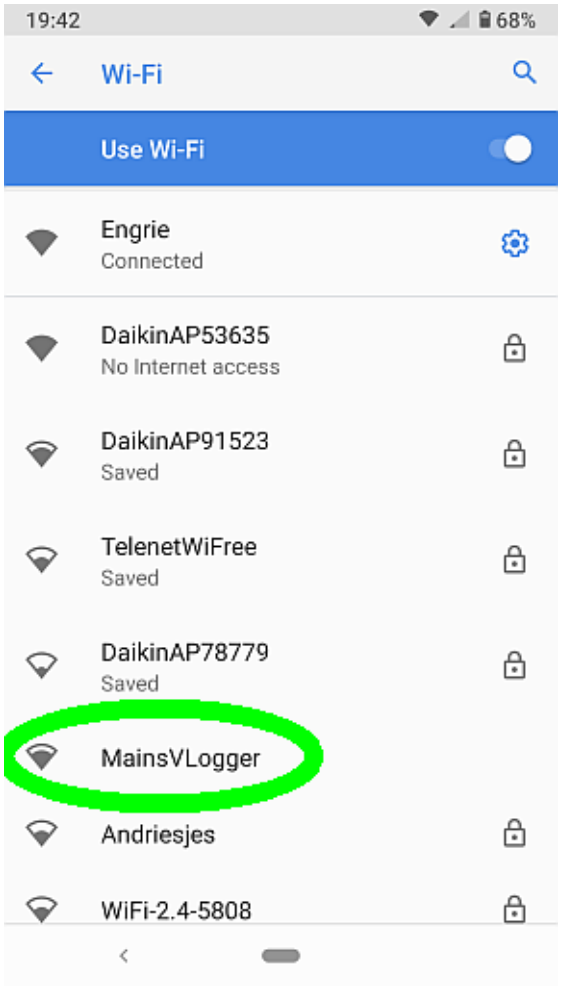
```
if(bMQTT)
{
  strcpy(caJson, "{ ");
  dt = ntpLocalDateTime();
  sprintf(caTemp, "\"TimeStamp\": \"%s\", ", dt);
  strcat(caJson, caTemp);
  sprintf(caTemp, "\"avg\": %u, "  , avgRMS[0]);
  strcat(caJson, caTemp);
  sprintf(caTemp, "\"min\": %u, "  , uMinV);
  strcat(caJson, caTemp);
  sprintf(caTemp, "\"max\": %u, "  , uMaxV);
  strcat(caJson, caTemp);
  sprintf(caTemp, "\"dif\": %u, "  , uDiffV);
  strcat(caJson, caTemp);
  sprintf(caTemp, "\"under\": %u, ", uUnderV);
  strcat(caJson, caTemp);
  sprintf(caTemp, "\"over\": %u"   , uOverV);
  strcat(caJson, caTemp);
  strcat(caJson, "}");
  sprintf(caTemp, "%s/Volt", mqttTopicDat);
  mqttSend(caTemp, caJson);
}

uUnderV = 0;
uOverV  = 0;
uMaxV   = 0;
uMinV   = 999;
uDiffV  = 0;

//-----------------------------------------------------------------------------
// current = line1
sum = 0;
for (int n=0; n<iRMS; n++)
{
  sum = sum + valRMS[1][n];
}
// average RMS values
avgRMS[1] = (unsigned int)(sum / iRMS);

if(bMQTT)
{
  strcpy(caJson, "{ ");
  dt = ntpLocalDateTime();
  sprintf(caTemp, "\"TimeStamp\": \"%s\", ", dt);
  strcat(caJson, caTemp);
  sprintf(caTemp, "\"avg\": %.1f", (float)avgRMS[1] / 10);
  strcat(caJson, caTemp);
  strcat(caJson, "}");
  sprintf(caTemp, "%s/Curr", mqttTopicDat);
  mqttSend(caTemp, caJson);
}

if(bMQTT)
{
  // let system know you are still online
  mqttSend(mqttTopicSts, (char*)"online");
}

uMaxA   = 0;
uMinA   = 999;
uDiffA  = 0;

//-----------------------------------------------------------------------------
// Listen for incoming clients
//-----------------------------------------------------------------------------
webClient = webServer.available();
// If a new webClient connects,
if(webClient)
{
  wsClientActive();
}

//-----------------------------------------------------------------------------
// checking button every ...
//-----------------------------------------------------------------------------
buttState = Button();
if (buttState == LONG_PRESS_TIME)
{
  // clearout SPIFFS and restart from scratch
```

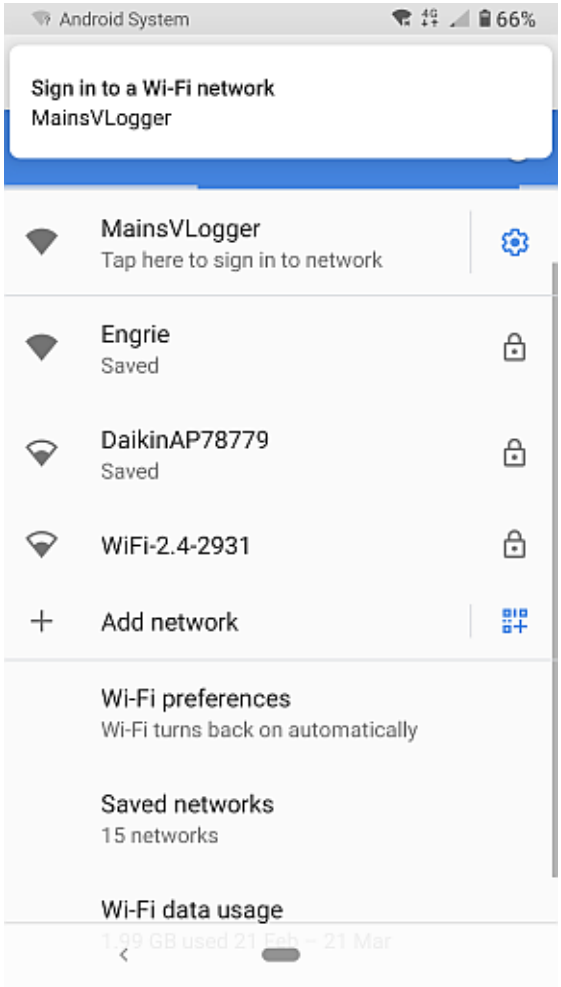
```
            #if defined(DEBUG)
              Serial.println("Formatting SPIFFS");
            #endif
            SPIFFS.format();
            delay(3000);
            #if defined(DEBUG)
              Serial.println("Restarting ...");
            #endif
            ESP.restart();
          }
          else if (buttState == SHRT_PRESS_TIME)
          {
            //just restart
            #if defined(DEBUG)
              Serial.println("Restarting ...");
            #endif
            ESP.restart();
          }
        }

        #if defined(DEBUG)
          // timing measurement
          difMicros = micros() - nowMicros;
          if (difMicros > 250)
          {
            Serial.print("runtime: "); Serial.print(difMicros); Serial.println(" µs");
          }
        #endif
      }
      // to make sure we do not handle pilled up interrupts
      // and keep a pace of 1 every ms
      portENTER_CRITICAL(&timerMux);
      cntrINT = 0;
      portEXIT_CRITICAL(&timerMux);
    }
  #endif
}
```

## Configuration of MainsLogger

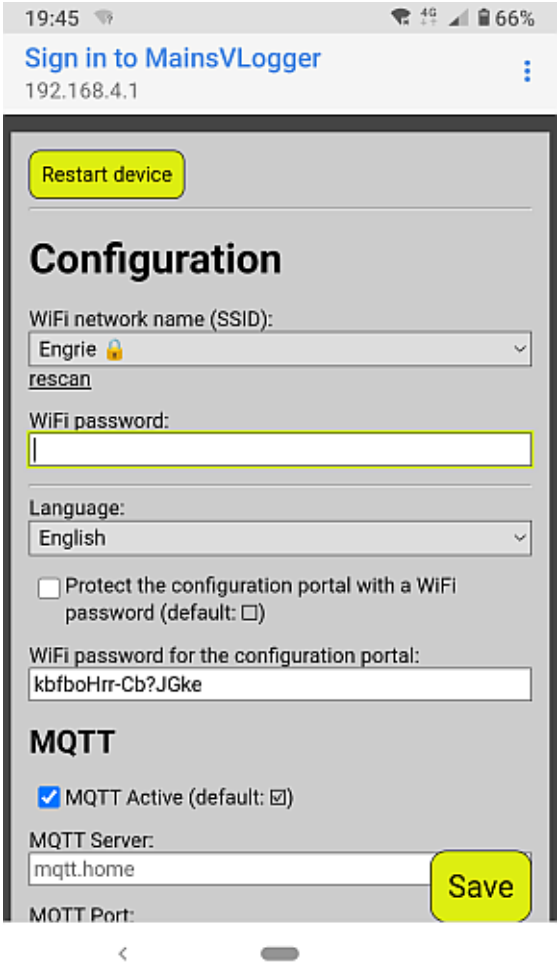
Using this method eases the deployment and relocation of MainsLogger but requires some extra steps at first startup.

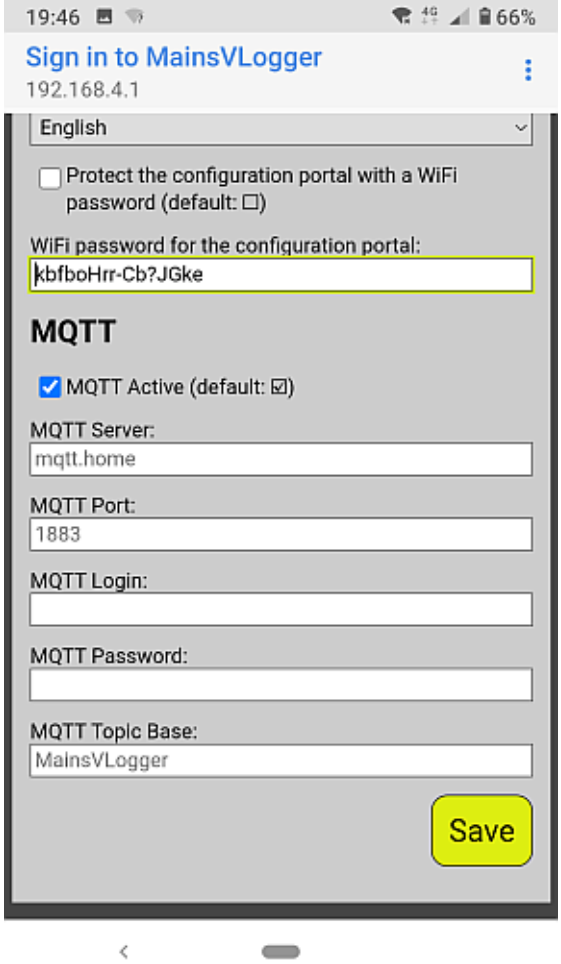
After flashing the ESP32, the WiFi-Portal will launch its own Wi-Fi network `MainsLogger` (or `MainsVLogger` or `MainsALogger` or `MainsVALogger` or ) to which you can connect using e.g. a smartphone. Connect to it by selecting and tapping.



It will start the web portal, scanning the air for Wi-Fi networks and presenting you with a page where you and make choices and enter the settings required for your environment




When done entering all settings, hit the '`Save`' button and next, hit the '`Restart device`' button.

Now, your ESP32 will restart, connect to the Wi-Fi network you specified and start running the code with the settings defined.
All settings are stored on the ESP32 in flash memory and will be re-used at reboot/reset.

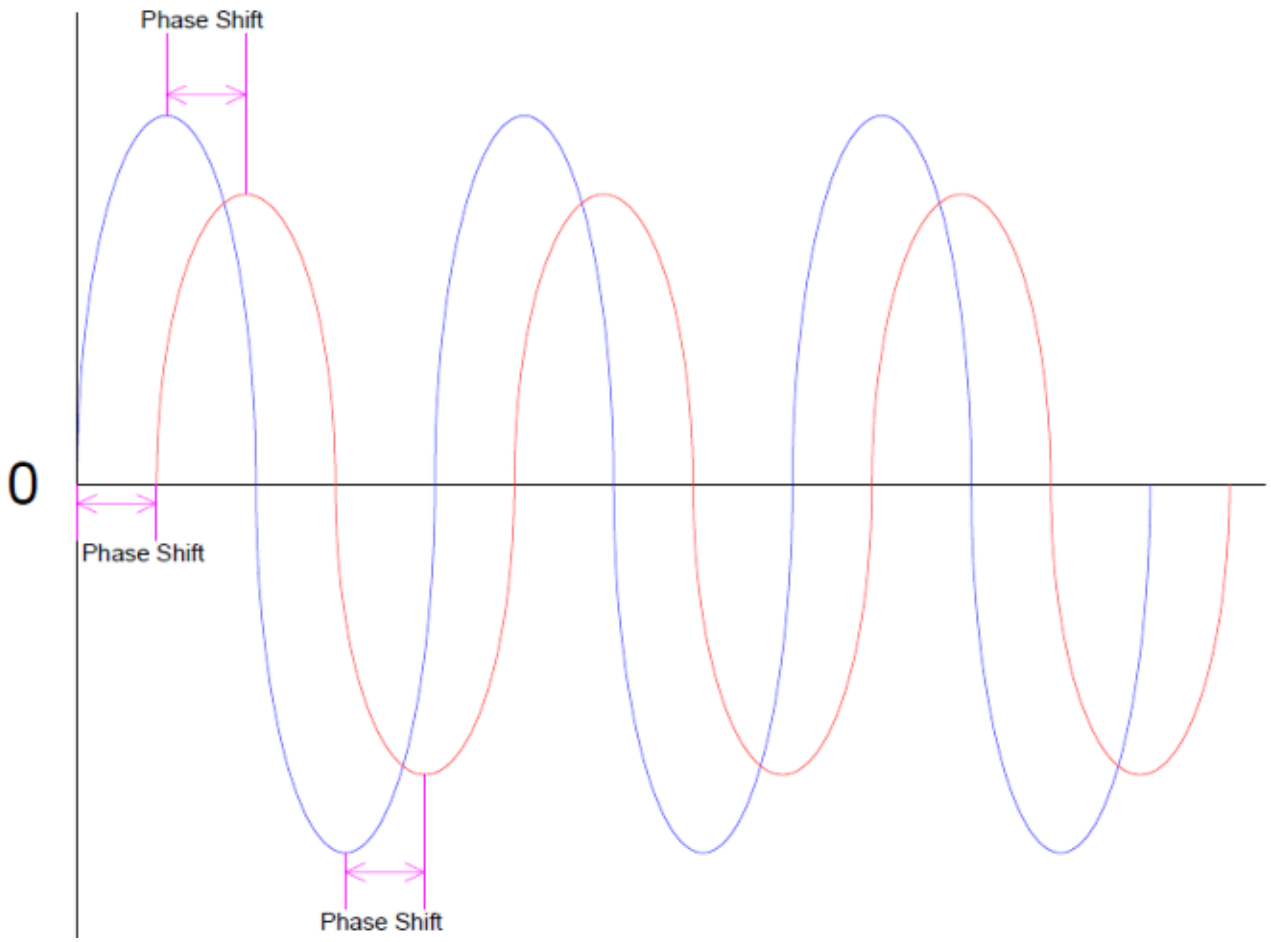
Pushing the small button on the PCB board for more than 3 seconds but less than 6 seconds, will reboot the ESP32. Pushing that button for more than 6 seconds, the settings will be removed from the flash memory and so, all is back as if the ESP32 was flashed for the first time. Hence, you need to re-enter all settings using the portal.

# !!!!!!  WORK IN PROGRESS !!!!!!

# All info hereafter is still W.I.P.

**How to measure Phase Angle?**

***Introduction***
In Alternating Current (AC) electrical system, wave patterns are involved where values oscillate with time. Phase shift is the difference of oscillation time between two waves pattern. It is a delay between two waves that have same period or frequency. Phase shift is expressed in positive or negative angle which is within -180 to 180 degree and called Phase Angle.




Phase shift can be applied between Voltage and Current wave of a particular electrical system. It is often applied by metering systems to obtain the quality of the loads / generator. By knowing the phase angle between the Voltage and Current waves, terms such as Power Factor, Real Power, Apparent Power and Reactive Power can be defined.

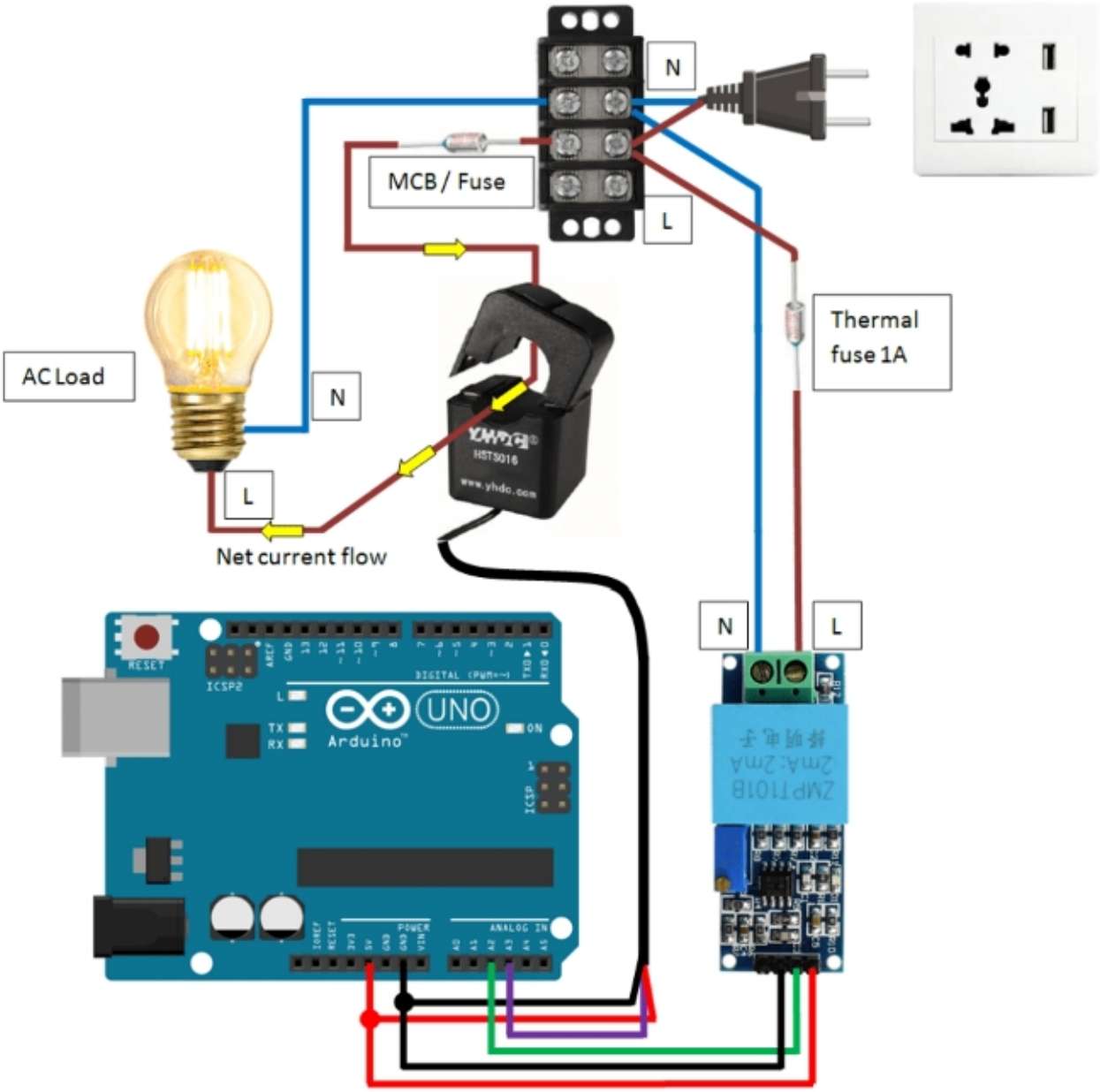
Phase shift can also be used between Voltage and Voltage wave in a 3-Phase system. It is an important electrical parameter to determine the phase sequence for 3 phase system. Each of the voltage phase is co-related with adjacent phases by 120 degree phase angle according to right rotation angle. By messing up the phase sequence, there might be issues with your 3-phase loads especially with loads using motor or compressor operation.

For the phase angle comparison, the first wave is taken as a reference to the other. The wave that is always taken as reference is the voltage wave while current wave is compared to the voltage wave.

***Hardware Connection***

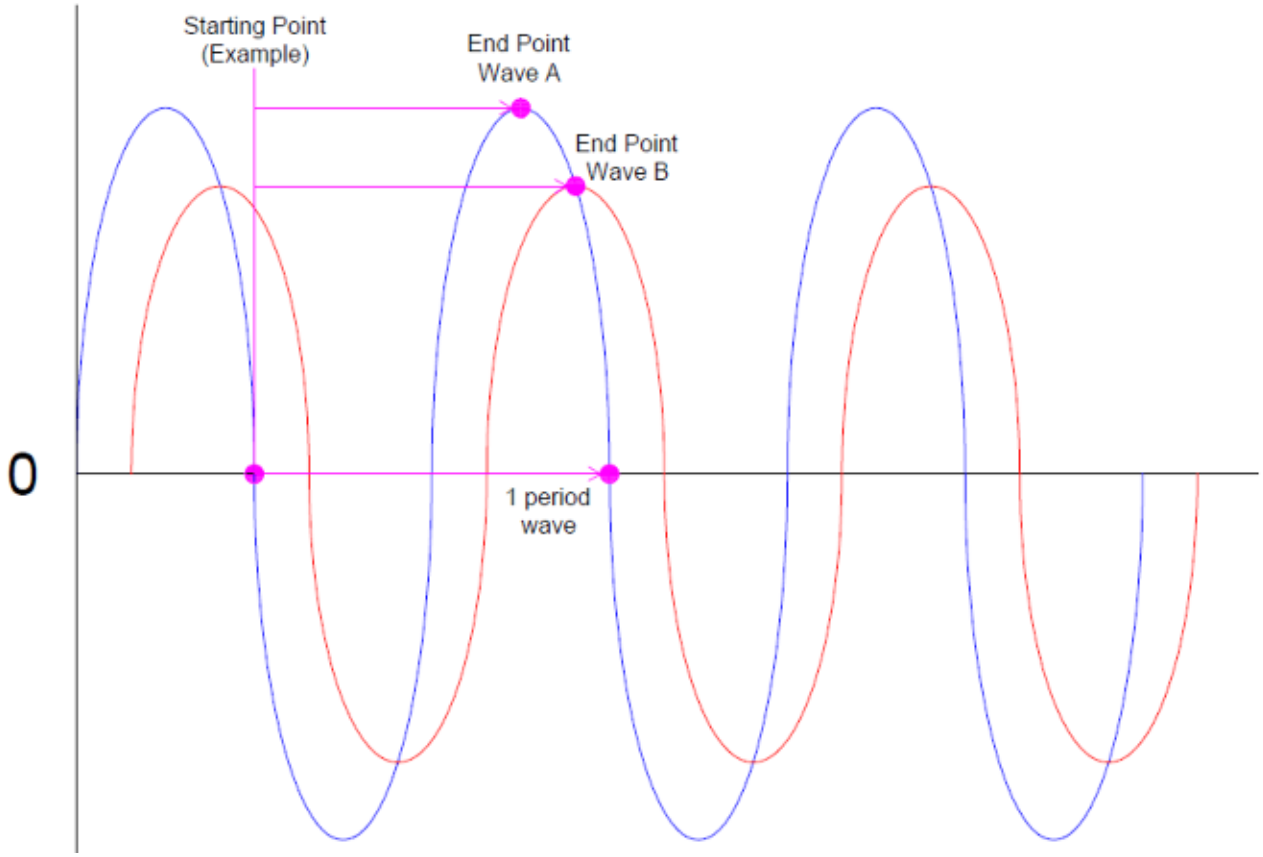
Once everything ready, make sure the split core is closed tightly.

Make sure underneath Voltage sensor does not have exposed conductor or metal plates which may cause you accidental short circuit at the terminal of the module. You need to get the low current fast blown fuse for voltage sensor cable while MCB or fuse of sufficient current rating based on your load to protect against any potential short circuit and fire risk.



***The method***

Phase shift can be obtained by measuring the time difference to reach peak value between both waves while using the same starting point. The 2 reference points are the starting point and peak value point. It is suitable for clean waves as well as distorted or high disturbance waves. However this method also requires to measure the period of wave in order to determine the angle value.

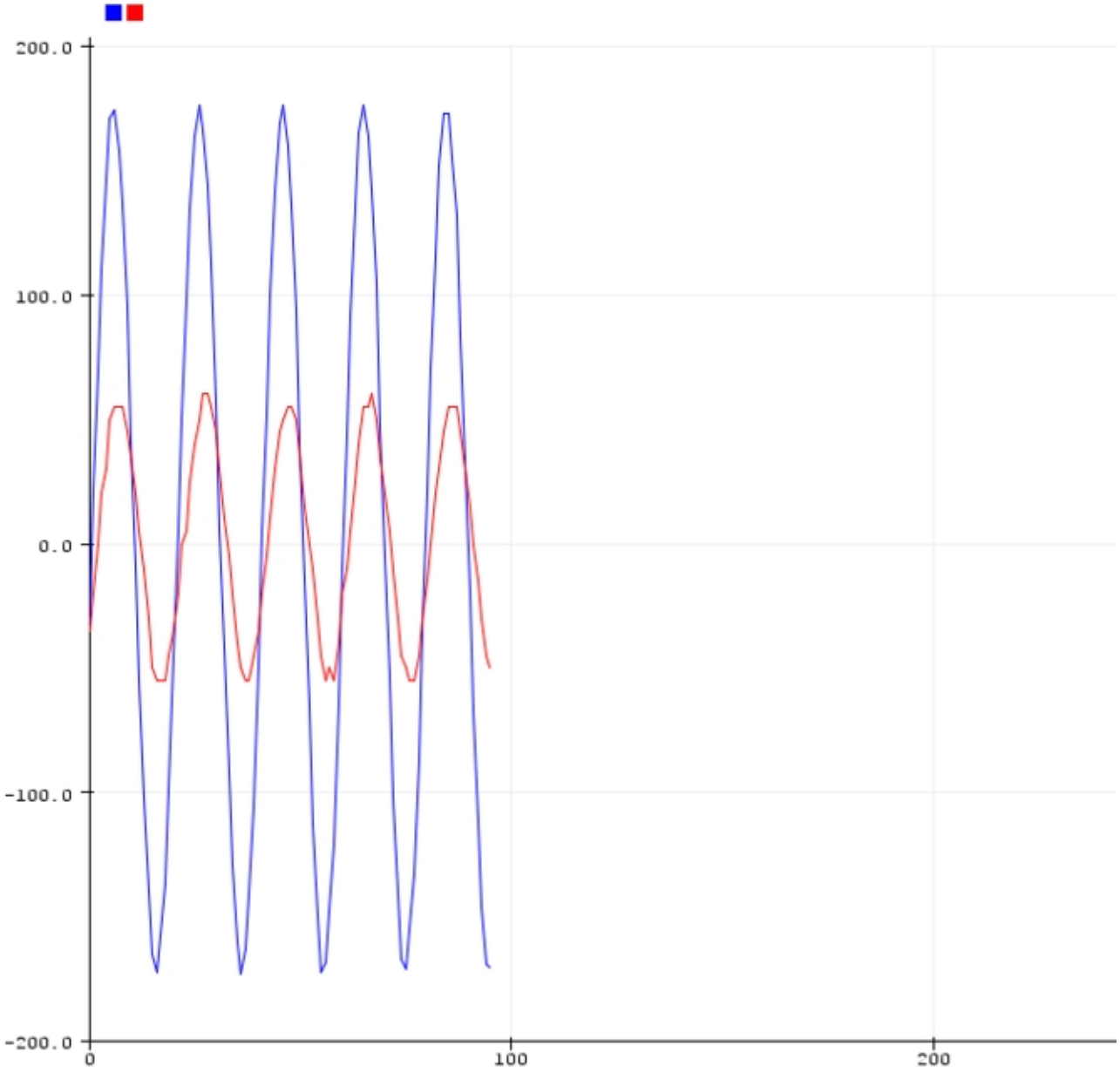


***Simple testing***

Signals detected are in analog values. Below are the signals being detected by the voltage (blue) and current sensor (red). You can copy the below code to try your own. You can see the waveform in Serial Plotter.

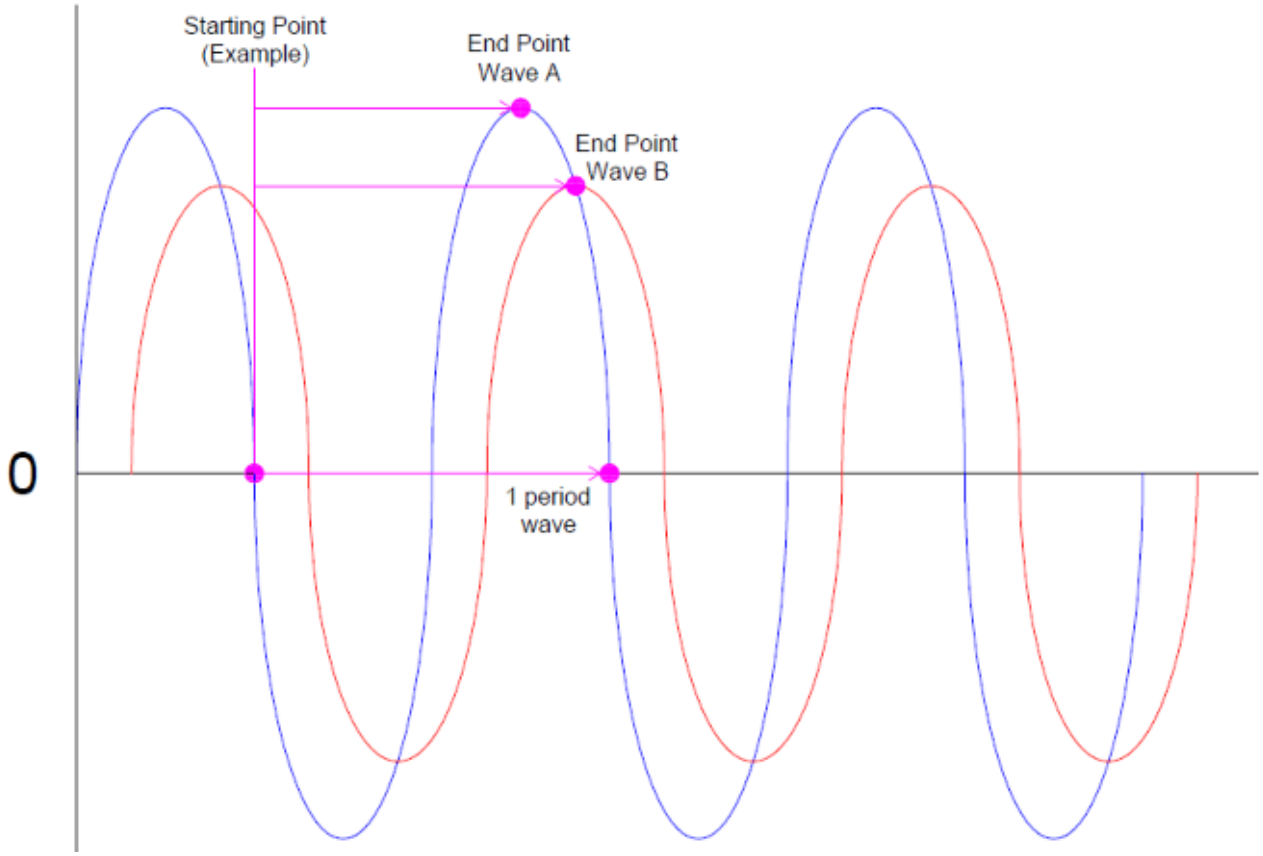
The phase shift or phase difference between voltage and current also subject to applied load. The magnitude of voltage and current are not a concerned here as we only need to measure the wave difference with respect to time.

```
#define ADC_INPUTV  34
#define ADC_INPUTI  35

unsigned int ADCv;
unsigned int ADCi;

void setup()
{
  Serial.begin(115200);
}

void loop()
{
  // read the ADC
  ADCv = (unsigned int)(analogRead(ADC_INPUTV));
  ADCi = (unsigned int)(analogRead(ADC_INPUTI));

  Serial.print(ADCv); Serial.print(" "); Serial.println(ADCi);

  //delay(300);
}
```


200.0
100.0
0.0
-100.0
-200.0
0
100
200

***The Real Code***

The code has few stages to make sure to go through the whole the wave pattern to prevent miscalculation. It first determine the entry point to start recording time. The point is where the voltage wave cross over or below 0. The starting time is the same for all 3 measurement parameters : wave Voltage = A, wave Current = B and frequency of wave Voltage).




When wave Voltage is more than 0, it keeps checking till it goes through 0.
When wave Voltage goes through 0 the current time is stored in start time.
The end time recording starts and continues while the current voltage values are larger than previous measured value.
If at the peak, where current value is no longer larger than previous value, recording the end time stops and it will be the last recording time.
The same applies to wave Current with a separate time recording and end time at peak value.
The time difference taken between wave Voltage and wave Current to reach peak is actually the phase shift time from which phase angle can be calculated.
When the wave Voltage crosses 0 again, the time recording for period is stopped.
Period is the time difference between start time and end time for the period.
At the same time, the current time is set as starting time for the next cycle and the process is repeated.
When we have enough cycles, averages are taken, calculations are doen and resluts are available.

```
// Summary
// This Code monitors phase difference between 2 waves and also its frequency value.
// The value shown in Serial Monitor is refreshed every second, can be used for 50Hz and 60Hz.
// The frequency is measured by counting time and average it for every 50 samples taken
//   = for 50 hz. 1 sample is 1 cycle.
//////////////////////////////////////////////////////////////////////////////////////////////

#define ADC_INPUTV  34
#define ADC_INPUTI  35

#define ADCzeroV          (float)1825    // ADC count for '0'-level
#define ADCvolt           (float)0.2375  // volt per ADC count

#define ADCzeroI          (float)1700    // ADC count for '0'-level
#define ADCamp            (float)0.0450  // amp per ADC count

// Phase Angle, Frequency and Power Factor measurement
unsigned int stage          = 3;         // use for switching operation
                                         // stage = 0  => voltage still positive
                                         // stage = 1  => voltage has crossed 0 and start time is recorded
                                         // stage = 2  => looking for peak, recording time
                                         // stage = 3  => timing not started

int iPeriods                = 50;        // frequency of main grid (50 or 60 Hz)
int cntPeriods              = 0;         // to count how many set of samples

// Measurements
unsigned int ADCv;
unsigned int ADCi;

float ADCvalVolt            = 0;         // is the analog value for voltage sensor
float PREVvalVolt           = 0;         // is the previous value for voltage sensor

float ADCvalAmp             = 0;         // is the analog value for current sensor
float PREVvalAmp            = 0;         // is the previous value for current sensor

float TimePhase             = 0;         // is the time difference between the 2 sensor (in micro seconds)
float prevTimePhase         = 0;         // previous sample reading to replace false value less than 100 micro
seconds
float sumTimePhase          = 0;         // is the accumulate time differences for thr phase timing
float avgTimePhase          = 0;         // is the average time difference

float periodSample          = 0;         // is the time difference for a period of voltage wave (in micro
seconds)
float sumPeriod             = 0;         // is the accumulate time differences for voltage wave
float avgPeriod             = 0;         // is the average time for a period of voltage wave

float PhaseAngle            = 0;         // the phase angle in degree
float frequency             = 0;         // the frequency of the voltage sensor
float powerFactor           = 0;         // the power factor

unsigned long TimeStart;                 // starting time for Phase Angle and Period (in micro seconds).
unsigned long TimeVolt;                  // current  time for voltage (in micro seconds).
unsigned long TimeAmp;                   // current  time for current (in micro seconds).
unsigned long TimePeriod;                // current  time for period of wave (in micro seconds).


void setup()
{
  // General
  Serial.begin(9600);
}

void loop()
{

  // read ADC asap
  ADCv = (float)(analogRead(ADC_INPUTV));
  ADCi = (float)(analogRead(ADC_INPUTI));

  // do some adjustments to obtain real values
  ADCvalVolt = (ADCval - ADCzeroV) * ADCvolt;
  ADCvalAmp  = (ADCval - ADCzeroI) * ADCamp;


  // initial begining stage of measurement when voltage wave larger than 0
  if((ADCvalVolt > 0)  && stage == 3)
  {
    // allow to change to the next stage
    stage = 0;
  }

  // when voltage wave smaller or equal than 0
  if((ADCvalVolt <= 0) && stage == 0)
  {
    // start counting time for all
    // store start time
    TimeStart = micros();
    // allow to change to the next stage
    stage = 1;
  }
```

```
  // when voltage wave is larger than 0, prepare to find peak
  if((ADCvalVolt > 0)  && stage == 1)
  {
    // reset value
    PREVvalVolt = 0;
    PREVvalAmp  = 0;
    // allow to change to the next stage
    stage = 2;
  }

  // if current measured value larger than previous peak value
  if((ADCvalVolt > PREVvalVolt) && stage == 2)
  {
    // record current time for voltage wave
    TimeVolt    = micros();
    // record current measure value replace previous peak value
    PREVvalVolt = ADCvalVolt;
  }

  // if current measured value larger than previous peak value
  if((ADCvalAmp > PREVvalAmp)   && stage == 2)
  {
    // record current time for current wave
    TimeAmp  = micros();
    // record current measure value replace previous peak value
    PREVvalAmp = ADCvalAmp;
  }

  // when wave voltage smaller or equal than 0
  if((ADCvalVolt <= 0) && stage == 2)
  {
    // record current time for 1 period
    TimePeriod = micros();
    // accumulate or add up time for all sample readings of period wave
    sumPeriod  = sumPeriod + (TimePeriod - TimeStart);
    // time difference between voltage peak value and current peak value
    TimePhase  = TimeAmp - TimeVolt;
    // prevent/overcome some possible false readings
    // if time difference more than 100 micro seconds
    // replace previous value using new current value
    if(TimePhase >= 100)
    {
      prevTimePhase = TimePhase;
    }
    // if time difference less than 100 micro seconds (might be noise or fake values)
    // take previous value instead using low value
    if(TimePhase < 100)
    {
      TimePhase = prevTimePhase;
    }
    // accumulate or add up time for all sample readings of time difference
    sumTimePhase = sumTimePhase + TimePhase;
    cntPeriods   = cntPeriods + 1;
    // reset beginning time
    TimeStart    = TimePeriod;
    // reset peak value
    PREVvalVolt  = 0;
    PREVvalAmp   = 0;
    // set stage mode
    stage = 1;
  }

  // if number of total sample recorded equals what we want
  if(cntPeriods == iPeriods)
  {
    // average time for a period of wave from all the sample readings
    avgPeriod    = sumPeriod / iPeriods;
    // the calculated frequency value
    frequency    = 1000000 / avgPeriod;
    // average time difference between 2 sensor peak values from all the sample readings
    avgTimePhase = sumTimePhase / iPeriods;
    // the calculated phase angle in degree (out of 360)
    PhaseAngle   = ((avgTimePhase * 360) / avgPeriod);
    // power factor. Cos is in radian, the formula on the left has converted the degree to rad.
    powerFactor  = cos(PhaseAngle * 0.017453292);

    Serial.print("Phase Angle  :"); Serial.print(PhaseAngle, 2);    Serial.print("°   ");
    Serial.print("Frequency    :"); Serial.print(frequency, 2);     Serial.print("Hz  ");
    Serial.print("Power Factor :"); Serial.println(powerFactor, 2); Serial.print("    ");

    // reset counters
    cntPeriods   = 0;
    sumPeriod    = 0;
    sumTimePhase = 0;
  }
}
```

***The Alternate Method To Measure Real And Apparent Power***
Basically there are 3 terms for AC Power: the Apparent Power, Active Power and Reactive Power.

Apparent Power is referring to total power in an AC circuit which is including dissipated and absorbed power. This term is important for power generator and cable design as to make sure all cabling and power can support all kind of loads. Apparent Power can be calculated by multiply RMS voltage value and RMS current value.

The Active or Real Power is the actual power consumed by the load. Active Power can be calculated by averaging the product of instantaneous Voltage and Current waveform values.

The Reactive Power is referring to power that is wasted and not being used by the load as useful work due to phase difference between voltage and current waveforms. When Reactive Power is added up with Active Power, it will give you the Apparent Power. Or Reactive Power = Apparent Power – Real Power

The Power Factor is obtained by dividing Real Power by Apparent Power

**The Apparent Power Calculation**
The Apparent Power is the product of RMS AC Voltage and RMS AC Current values. The RMS AC Voltage and RMS AC Current are measured and calculate separately.
Our code is designed to calculate a value which is derived from averaging x samples per set. Each single sample analog value is being squared initially and the values are accumulated. The average value, being the accumulated values divided by x, is then being square-rooted in order to come out the RMS analog value for 1 sensor.
Do the similar calculation separately for the other sensor and multiply RMS values to obtain the Apparent Power.

***The Active or Real Power Calculation***
Active or Real Power is the average value of all the multiplications of instantaneous voltage and instantaneous current value. Both voltage and current values are first converted into measured voltage and current values. The voltage is then multiplied by its instantaneous current value and becomes a sample reading.

Similar to Apparent power method, the Active Power is also derived from averaging all samples per set. Each sample is recorded every millisecond, in other words the total complete set is equivalent to 50 waves with each wave is divided into 20 sections or readings (for 50Hz).

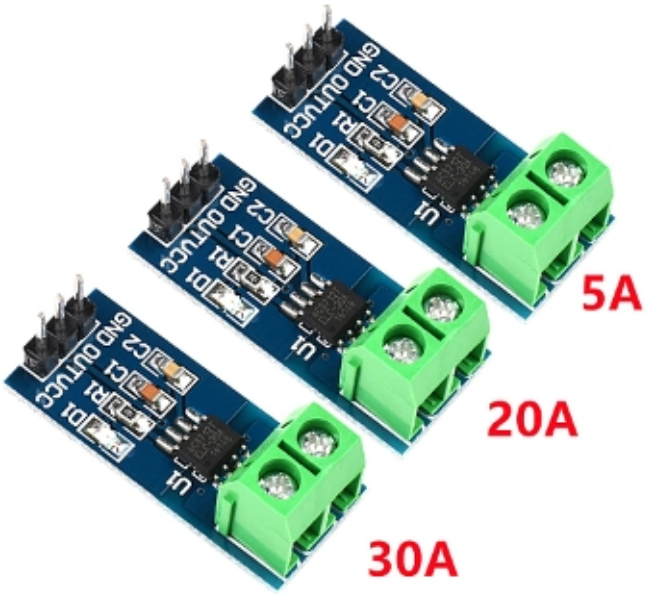
```
// Summary
// This Code monitors RMS Voltage, RMS current, RMS AC Power (Apparent Power)
//   and Instantaneous AC Power (Active Power)
// The value shown in Serial Monitor is refreshed every second,
//   can be used for 50Hz and 60Hz.
// The frequency is measured by counting time and average it for every 50 samples taken
//   = 1 sample is 1 cycle for 50Hz by default.
////////////////////////////////////////////////////////////////////////////////////////

#define ADC_INPUTV  34
#define ADC_INPUTI  35

//
#define ADCzeroV          (float)1825       // ADC count for '0'-level
#define ADCvolt           (float)0.2375     // volt per ADC count

#define ADCzeroI          (float)1700       // ADC count for '0'-level
#define ADCamp            (float)0.0450     // amp per ADC count


// General
float lstTime             = 0;              // to count time for each sample. 1 milli second
float cntr                = 0;              // to count number of sample.
float iSamples            = 1000;           // number of samples before calculating

// Measurements
unsigned int ADCv;
unsigned int ADCi;

float ADCvalVolt          = 0;              // to read the value of a sample
float sumVolt             = 0;              // accumulation of sample readings
float rmsVolt             = 0;              // square root of average Volt

float ADCvalAmp           = 0;              // to read the value of a sample
float sumAmp              = 0;              // accumulation of sample readings
float rmsAmp              = 0;              // square root of average Current

float ADCvalPwr           = 0;              // to read the current X voltage value of a sample
float sumPwr              = 0;              // accumulation of sample readings

float PwrAppar            = 0;              // the apparent power
float PwrReal             = 0;              // the real power
float PwrReact            = 0;              // the reactive power
float PwrFactor           = 0;              // the power factor


void setup()
{
  Serial.begin(115200);
}

void loop()
{

  // Measurements
  // every 1 milli second taking 1 reading
  if(millis() >= lstTime + 1)
  {

    // read ADC asap
    ADCv = (float)(analogRead(ADC_INPUTV));
    ADCi = (float)(analogRead(ADC_INPUTI));
    // to reset the time again so that next cycle can start again
    lstTime    = millis();

    // do some adjustments to obtain real values
    ADCvalVolt = (ADCv - ADCzeroV) * ADCvolt;
    ADCvalAmp  = (ADCi - ADCzeroI) * ADCamp;

    // accumulate value with older sample readings
    sumVolt    = sumVolt + sq(ADCvalVolt);
    sumAmp     = sumAmp  + sq(ADCvalAmp);

    // real power sample value
    ADCvalPwr  = ADCvalVolt * ADCvalAmp;
    // accumulate value with older sample readings
    sumPwr     = sumPwr + ADCvalPwr;

    cntr       = cntr + 1;
  }
```

```
  // Calculations
  // after iSamples count or iSamples milli seconds = 1 second, do the calculation
  if(cntr == iSamples)
  {
    // square root of the average value
    rmsVolt     = sqrt(sumVolt / iSamples);
    rmsAmp      = sqrt(sumAmp  / iSamples);

    // calculate average values of Power
    PwrAppar    = rmsAmp   * rmsVolt;
    PwrReal     = sumPwr   / iSamples;
    PwrReact    = PwrAppar - PwrReal;
    PwrFactor   = PwrReal  / PwrAppar;
    if(PwrFactor > 1 || PwrFactor < 0)
    {
      PwrFactor = 0;
    }

    Serial.print("Voltage RMS        : "); Serial.print(rmsVolt,   0); Serial.println(" V  ");
    Serial.print("Current RMS        : "); Serial.print(rmsAmp,    0); Serial.println(" A  ");
    Serial.print("Apparent Power (VA): "); Serial.print(PwrAppar,  0); Serial.println(" VA ");
    Serial.print("Real Power (W)     : "); Serial.print(PwrReal,   0); Serial.println(" W  ");
    Serial.print("Reactive Power (VA): "); Serial.print(PwrReact,  0); Serial.println(" VA ");
    Serial.print("Power Factor       : "); Serial.print(PwrFactor, 0); Serial.println("    ");

    // to reset values for the next cycle
    sumVolt  = 0;
    sumAmp   = 0;
    sumPwr   = 0;
    cntr     = 0;
  }
}
```

**Alternatives for ZMC103C**

***ACS712 Current Sensor Module***
The standard ACS712 Current Sensor Module rated at 5A, 20A and 30A which are suitable to most applications. The 5A module has the resolution of 185mV/ampere, 20A module has 100mV/ampere while 30A module has the resolution of 66mV/ampere.

These modules require direct contact which I think is a major drawback. It has to be connected in series to the measured value.The wiring of the existing system need to be altered in order to fit the module into the existing system.




***Hall-Effect Split-Core Sensor HSTS016L***
There is also a hall-effect sensor type with split core transformer type (as picture on left). It is the Hall-Effect Split-Core Sensor HSTS016L module. The model ranges from 10A up to 200A. With split core current sensor type, not alteration on the existing system required. The output voltage of this sensor is 2.5V +/- 0.625V with decent accuracy.