Project
-
BoilerController

Versie: 2022-11-26 – Update 1

Doel van project

Met dit project is het mijn bedoeling om een elektrische warmwaterboiler (80l of meer – enkelfasig verwarmingselement) te kunnen gecontroleerd 'op te laden' hetzij om geen extra impact te creëren op het afgenomen vermogen van het elektriciteitsnet (capaciteitstarief) hetzij om de boiler te laten opladen wanneer er een overschot is aan de locale opgewekte zonne-energie.

Het ontwerp heeft enkel tot doel te meten en deze meetwaarden door te sturen naar een 'home automation' systeem en om aangestuurd te worden door hetzelfde 'home automation' systeem. De intelligentie zit dus in het 'home automation' systeem en de beslissingen om al dan niet de boiler aan te schakelen worden dus ook door het 'home automation' systeem genomen.

Verder is het project modulair en hoeven niet alle componenten aanwezig te zijn of te worden gebruikt.

Belangrijke Opmerkingen: dit project is NIET geschikt voor warm water boilers met een warmtepomp of hybrides (warmtepomp en weerstand)!!! Dus alléén voor boilers met een weerstand (droog of nat - dat maakt niet uit)

In dit ontwerp wordt gewerkt met netspanning van 230V AC in combinatie met water. Gelieve alle nodige maatregelen te nemen om ervoor te zorgen dat jij noch andere personen geëlektrocuteerd kunnen worden zowel tijdens de opbouw als tijdens regulier gebruik.

De bedoeling is om dit tussen de netaansluiting en de voeding van de boiler aan te sluiten zodat alles beveiligingen van de boier intact en actief blijven. Dus niet tussen de netaansluiting en de weerstand van de boilier. Je mag nooit de bestaande veiligheidsmaatregelen van de boiler overbruggen of verwijderen.

Het ontwerp - Hardware

Het ontwerp is gebaseerd op een ESP32 D2 Mini en wordt voorzien van 3 meetmodules: een module om de spanning te meten, een module om de stroom te meten, en 3 temperatuur-sensors. Voor sturing is er een relais en een DAC die een range van 0 tot 10V DC aankan. Het geheel kan gevoed worden door een voedingsadapter van 7V tot 20V.

De ESP32 D2 Mini werd gekozen om zijn compact vorm, WiFi build-in.
https://nl.aliexpress.com/item/1005001621844145.html

De (optionele) spanningsmodule, ZMPT101B, meet de AC spanning (in RMS) die op de boiler-weerstand komt te staan. Max 250V AC rms
https://nl.aliexpress.com/item/1005002042876220.html

De (optionele) stroommodule, ACS712, meet de AC stroom (in RMS) die door de boiler-weerstand loopt. Max 20A AC rms
https://nl.aliexpress.com/item/32314318168.html

Noot: Opgelet!! Er zijn ACS712 modules waarbij Vcc en Gnd zijn omgewisseld. Dus goed nakijken bij bestellen EN monteren.

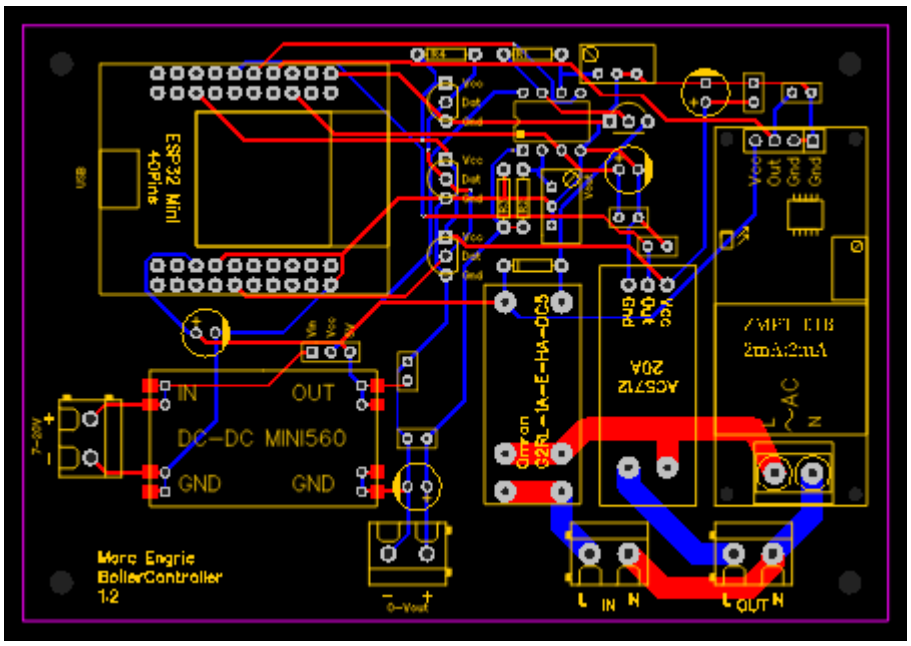
De 3 (optionele) temperatuurssensors, DS18B20, kunnen naar eigen goeddunken worden gebruikt om temperatuurinformatie door te sturen. Vb: 1 sensor om de warmte in de behuizing te meten, 1 sensor om de temperatuur van de koelvin te meten (zie later), 1 sensor om de temperatuur van de omgeving waarin de boiler staat te meten (vb anti-vorst). Maar andere opties zijn mogelijk.
https://nl.aliexpress.com/item/32467815969.htm

Met de build-in DAC van de ESP32 waar ik een buffer/versterker op basis van de LM358 heb voorgezet, kan een systeem gestuurd worden dat het vermogen regelt van vb een regelbare dimmer.
Noot: gebruik steeds fase-aansnijding die zowel de positieve als de negatieve cyclus aansnijdt en dus niet een die vb enkel de positieve cyclus of enkel de negatieve cyclus. Ondervinding leert mij dat een MPTT/DCAC-converter van de zonnepanelen dit laatste weinig apprecieert.

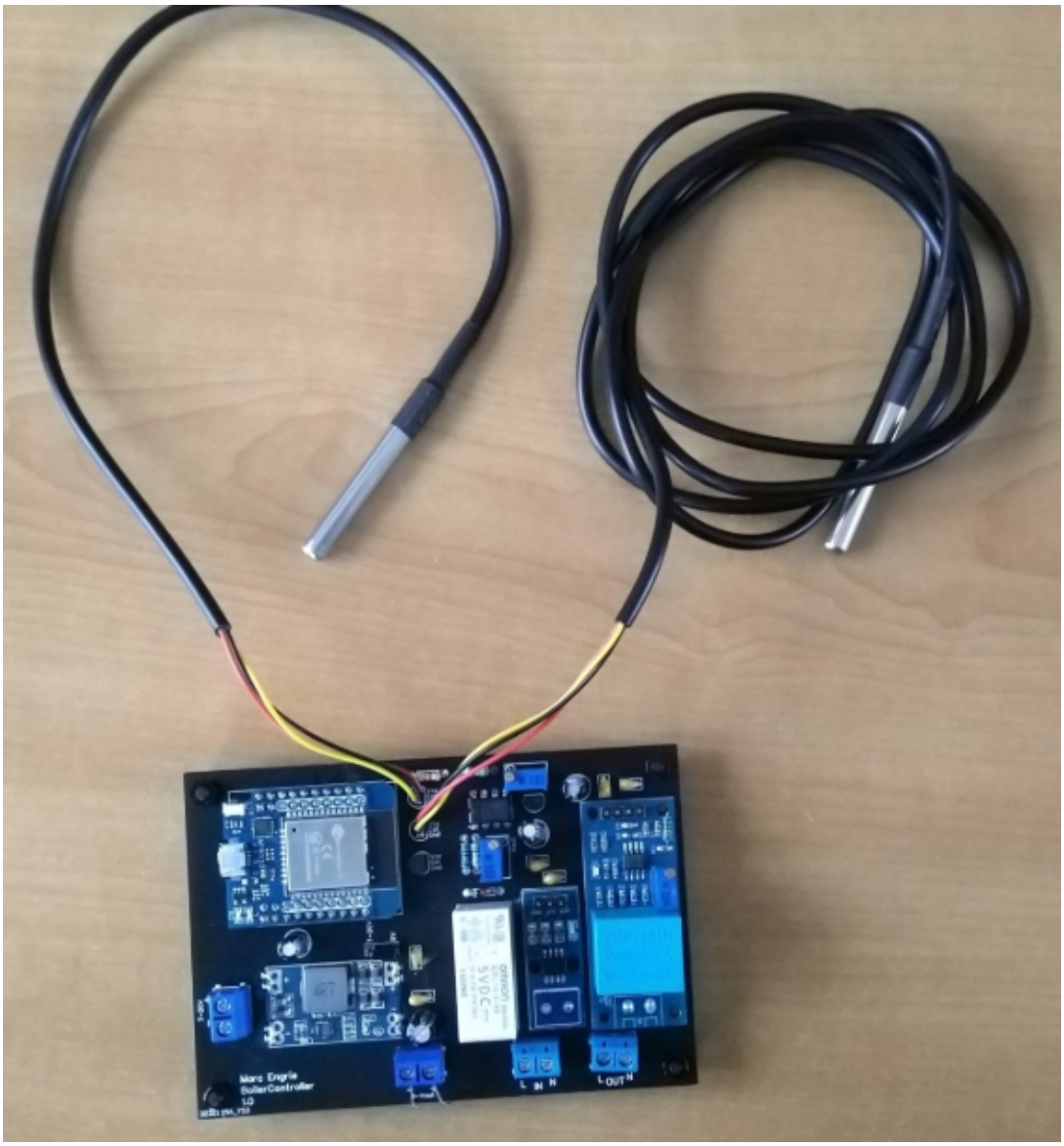
De relais, Omron G2RL-1A-E-HA-DC5, laat toe om de boiler aan of af te schakelen.
Noot: het gaat hier om een specifiek relais dat in staat is om 16A te schakelen bij 440V AC met een spoelspanning van slechts 5V. En zeer schappelijk in prijs is.
https://www.conrad.be/nl/p/omron-g2rl-1a-e-ha-dc5-printrelais-5-v-dc-16-a-1x-no-1-stuk-s-bag-2593562.html

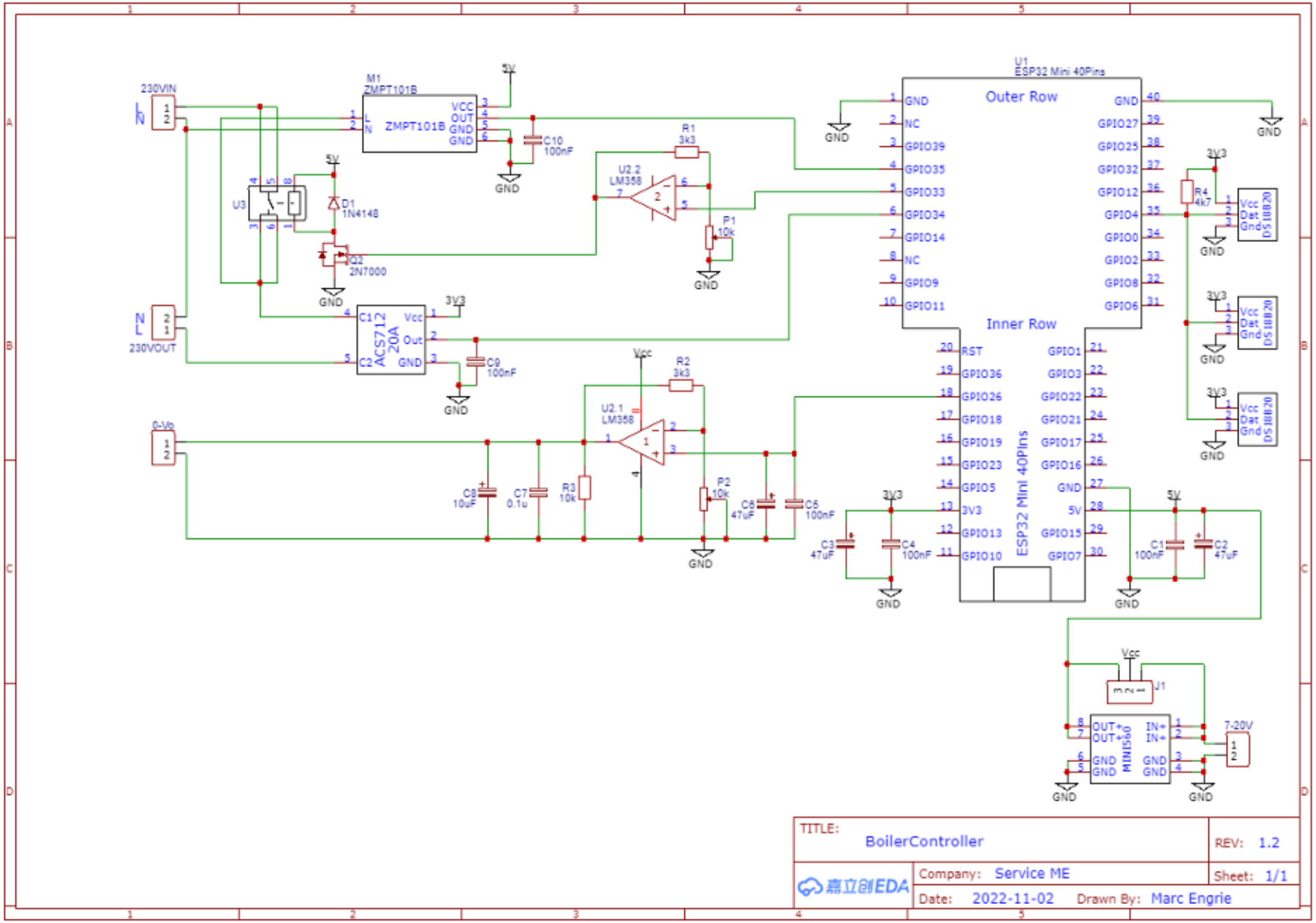
De powerconvertor/step-down convertor is een MINI560 5V.
https://nl.aliexpress.com/item/1005001629723875.html

Noot: Op de foto's zie je versie 1.0 van het ontwerp en de PCB. Tijdens de opbouw en testing kwamen wat 'onhebbelijkheden' aan het licht en werden deze opgelost in versie 1.2. Dit is dan ook de versie die in de tekeningen wordt getoond.



Een volledig bestukte printplaat kan er zo uitzien. Maar, zoals reeds geschreven, niet alle modules hoeven erop gezet worden.




230VIN
230VOUT
0-Vo
M1
ZMPT101B
U3
D1
1N4148
Q2
2N7000
U2.2
LM358
U2.1
LM358
R1
3k3
R2
3k3
R3
10k
P1
10k
P2
10k
R4
4k7
C1
100nF
C2
47uF
C3
47uF
C4
100nF
C5
100nF
C6
47uF
C7
0.1u
C8
10uF
C9
100nF
C10
100nF
U1
ESP32 Mini 40Pins
Outer Row
Inner Row
ESP32 Mini 40Pins
GND
NC
GPIO39
GPIO35
GPIO33
GPIO34
GPIO14
GPIO9
GPIO11
GPIO27
GPIO25
GPIO32
GPIO12
GPIO4
GPIO0
GPIO2
GPIO8
GPIO6
RST
GPIO36
GPIO26
GPIO18
GPIO19
GPIO23
GPIO5
3V3
GPIO13
GPIO10
GPIO1
GPIO3
GPIO22
GPIO21
GPIO17
GPIO16
5V
GPIO15
GPIO7
DS18B20
Vcc
Dat
Gnd
J1
MINI560
7-20V
OUT+
OUT-
IN+
IN-
TITLE:
BoilerController
REV: 1.2
Company: Service ME
Sheet: 1/1
Date: 2022-11-02
Drawn By: Marc Engrie

Indien de ACS712 module of de relais niet wordt gemonteerd, hoeft er enkel een (stevige) draadbrug gelegd te worden tussen de meetpunten van de ACS712-module en/of de contactenpunten van de relais.

Er zijn 2 trimmers op de PCB die moeten afgesteld worden. Eentje, aangeduid door 'Vout' dient om de output van de DAC af te regelen op max. voltage output die nodig voor de sturing. Vb: 5V of 10V.
Noot: 5V output kan enkel bereikt worden indien er minstens een 7V voeding wordt aangesloten en 10V output enkel indien er een 12V voeding wordt aangesloten.

De 2de trimmer dient om ervoor te zorgen dat de spanning op de gate van de 2N7000 beperkt blijft tot 5V.

Externe componenten
Om het afgegeven vermogen van de boilerweerstand te bepalen (tussen 0 en 100%), koos ik voor een module Kemo M028N. Deze is volledig waterdicht en stuurbaar hetzij met een potentiometer (wordt meegeleverd) hetzij met de extra module Kemo M150.

https://www.conrad.be/nl/p/kemo-m028n-vermogensregelaar-module-110-v-ac-230-v-ac-190516.html

https://www.conrad.be/nl/p/kemo-electronic-gmbh-spanningsomvormer-met-pulsomzetter-module-uitgangsspanning-bereik-110-230-v-ac-1402639.html

Noot 1: ik heb voorlopig een koelvin (uit mijn "als ik ooit eens nodig heb"-voorraad) voorzien op de Kemo M028N omdat ik nog geen idee heb hoe warm dit ding wordt als we gaan sturen tussen 5% en 95%. Daarom ook dat ik één van de temperatuursensors op de koelvin heb gemonteerd. Mocht blijken dat dit niet nodig is, gaat ze eraf natuurlijk.
Indien je dit niet 'op voorraad' hebt: Vb: https://nl.aliexpress.com/item/32830696828.html

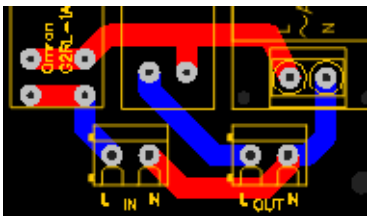
Noot 2: alternatief voor Kemo M028N + Kemo M150 is de Kemo M240 die beide combineert maar beperkt is tot 2300W. Kemo M028N kan tot 4000W aan.

Verder voorzie ik in mijn opstelling ook een powerlinefilterr om storingen op het net, door de module M028N veroorzaakt, tot een minimum te herleiden. Mijn buurman zal wel op een andere manier te weten komen dat ik mijn boiler aanstuur.

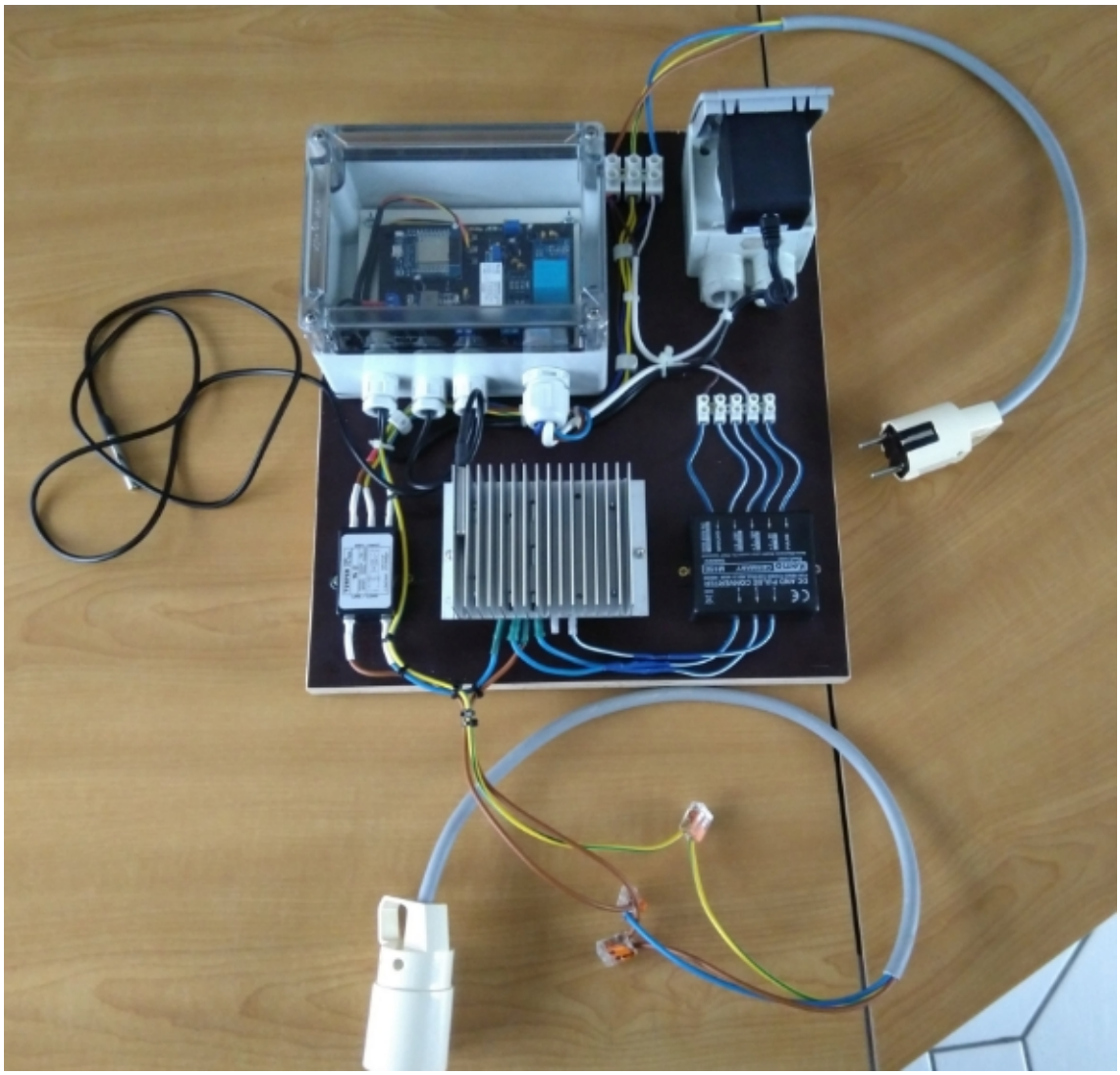
https://www.conrad.be/nl/p/yunpen-yg10t5-ontstoringsfilter-250-v-ac-10-a-0-3-mh-l-x-b-x-h-68-x-55-x-25-mm-1-stuk-s-521349.html

Noot: mijn boiler, 200l, heeft een weerstand van 1800W, enkelfasig, en dus de stroom door het circuit is max. 8A. Vandaar de filter van 10A. Is jouw boiler hoger in vermogen, moet je even uitrekenen wat de max. stroom is en het gepaste filter op Conrad of AliExpress zoeken.

Opmerking: om zeker te zijn dat de grote stromen die door een deel van de print vloeien, niet voor al te grote opwarming zorgen, soldeer ik over deze printbanen een extra 1,5mm² koperdraad (van punt naar punt) en leg er een laag vernis op (in meerdere keren aanbrengen of spuiten).

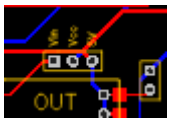


Mijn volledig test-opzet ziet er zo uit



Het ontwerp - Software

Belangrijk: Tijdens het testen van de code kan je de ESP32 voeden met een USB-kabel vanuit je computer en zo via de seriële monitor, de run van de code volgen om dan, eventueel, bij te sturen waar nodig. **Let er wel op dat je dan geen power voorziet via de Mini560**. Dus deze moet je loskoppelen.
Om de DAC-versterker te voeden kan je tijdelijk de jumper naar 5V verplaasten.



Om het testen van het bord te vergemakkelijken, is voor iedere module een test-applicatie geschreven die tegelijkertijd dan ook laat zien hoe je de module moet aansturen en hoe je kan kalibreren.

Relais

```
// set pin numbers
#define RELAY_GPIO 33


void setup()
{
  Serial.begin(115200);
  // initialize GPIO as an output
  pinMode(RELAY_GPIO, OUTPUT);
}

void loop()
{
  // adjust trimmer as such that only 5V max is on the gate of MOSFet 2N7000
  // set output high
  digitalWrite(RELAY_GPIO, HIGH);
  Serial.println("Output HIGH");
  delay(5000);
  // set output low
  digitalWrite(RELAY_GPIO, LOW);
  Serial.println("Output LOW");
  delay(5000);
}
```

DAC

```
//DAC output port
#define DAC_GPIO  26

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

void loop()
{
  for(int i = 0; i < 256; i+=16)
  {
    dacWrite(DAC_GPIO, i);
    Serial.printf(" %d \n", i);
    delay(1000);
  }
  // adjust trimmer Vout to the output required when DAC is at max = 255.
  dacWrite(DAC_GPIO, 255);
  Serial.println("Max (255) reached");
  delay(5000);

  for(int i = 254; i > 0; i-=16)
  {
    dacWrite(DAC_GPIO, i);
    Serial.printf(" %d \n", i);
    delay(1000);
  }
  dacWrite(DAC_GPIO, 0);
  Serial.println("Min (0) reached");
  delay(5000);
}
```

DS18B20

```
#include <OneWire.h>
#include <DallasTemperature.h>

// uncomment to list all DS18B20 addresses
//#define LISTDS

//OneWire data line connected to GPIO
#define ONE_WIRE_BUS  4

// create objects required
OneWire           oneWire(ONE_WIRE_BUS);
#if not defined(LISTDS)
  DallasTemperature sensors(&oneWire);

  // these can be obtained using the LISTDS code
  DeviceAddress ds18b20_1 = { 0x28, 0x2F, 0x9C, 0x56, 0xB5, 0x01, 0x3C, 0x9B };
  DeviceAddress ds18b20_2 = { 0x28, 0xFF, 0x64, 0x02, 0xC8, 0x7D, 0x29, 0xB7 };
  DeviceAddress ds18b20_3 = { 0x28, 0xFF, 0x64, 0x1E, 0x0C, 0x04, 0xC5, 0x1E };
#endif

// setup
void setup(void)
{
  Serial.begin(115200);
  #if not defined(LISTDS)
    sensors.begin();
  #endif
}

// loop
void loop(void)
{

  #if defined(LISTDS)

    byte addr[8];
    static int cnt = 0;

    if (!oneWire.search(addr))
    {
      Serial.println();
      oneWire.reset_search();
      cnt = 0;
      delay(3000);
      return;
    }
    else
    {
      cnt++;
    }

    Serial.printf("DeviceAddress ds18b20_%d = {", cnt);
    for (byte i = 0; i < 8; i++)
    {
      if(addr[i] < 16)
      {
        Serial.write(" 0x0"); Serial.print(addr[i], HEX);
      }
      else
      {
        Serial.write(" 0x"); Serial.print(addr[i], HEX);
      }
      if(i < 7)
      {
        Serial.write(",");
      }
    }
    Serial.println(" };");

  #else

    Serial.println("");
    Serial.println("Requesting temperatures...");
    // Send the command to get temperatures
    sensors.requestTemperatures();

    // display temps
    Serial.print("Sensor 1 :"); Serial.printf(" %3.1f C\n", sensors.getTempC(ds18b20_1));
    Serial.print("Sensor 2 :"); Serial.printf(" %3.1f C\n", sensors.getTempC(ds18b20_2));
    Serial.print("Sensor 3 :"); Serial.printf(" %3.1f C\n", sensors.getTempC(ds18b20_3));

    delay(2000);

  #endif
}
```

ZMPT101B

```
//-------------------------------------------------------------------------
// defines
//-------------------------------------------------------------------------
//#define CALIBRATE1                        // calibrate first using pot-trimmer for max swing
//#define CALIBRATE2                        // calibrate ADCvolt in code

#define ADCvolt           (float)0.210     // volt per ADC count

#define ZMPT101B_GPIO     35               // define the used ADC input channel
#define CLOCKSPEED        80               // clockspeed of ESP32 in MHz
#define nomFREQ           50.00            // nominal frequency

//-------------------------------------------------------------------------
// 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

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

// vars related to ADC and Currents
float ADCval;
float ADCzero   = 1700;

unsigned long valZER[iADC];
unsigned long avgZER[iRMS];
float         valADC[iADC];
float         valRMS[iRMS];

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

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

//=========================================================================
// Setup
//=========================================================================
void setup()
{

  Serial.begin(115200);
  delay(1000);

  #if defined(CALIBRATE1)

    pinMode(ZMPT101B_GPIO, INPUT);

  #else

    //-----------------------------------------------------------------------
    // 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);

  #endif

}
//=========================================================================
```

```
//#############################################################################
// Main loop
//#############################################################################
void loop() {
  #if defined(CALIBRATE1)
    // use Serial Plotter to view waveform
    Serial.println(analogRead(ZMPT101B_GPIO));
  #else
    float        sum    = 0;
    float        rmsVal = 0;
    unsigned int avgZer = 0;

    // interrupty occured if cntrINT > 1
    if (cntrINT > 0) {
      // read the ADC and save value
      valZER[cntADC] = (unsigned int)(analogRead(ZMPT101B_GPIO));
      ADCval         = (float)(valZER[cntADC]);
      ADCval         = (ADCval - ADCzero) * ADCvolt;
      valADC[cntADC] = (float)(ADCval);

      cntADC++;
      // if array is full, calc rms
      if( cntADC >= iADC)
      {
        // reset counter
        cntADC = 0;

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

        // find RMS
        // RMS = the square root of the mean square (the arithmetic mean of the squares) of the set
        float valV;
        sum = 0;
        // skip first as it might not be correct due to long loop
        for (int n=1; n<iADC; n++)
        {
          valV = (float)valADC[n];
          sum = sum + (float)(valV * valV);
        }
        valRMS[cntRMS] = (float)(sqrt(sum / (iADC - 1)));

        cntRMS++;
        // array filled? If so, average
        if(cntRMS >= iRMS)
        {
          // reset counter
          cntRMS = 0;

          #if defined(CALIBRATE2)
            sum = 0;
            // find average DC offset
            for (int n=0; n<iRMS; n++)
            {
              sum = sum + (float)avgZER[n];
            }
            avgZer = (unsigned int)(sum / iRMS);
          #endif

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

          Serial.printf(" Line Voltage   : %4.0f Vrms\n", rmsVal);
          #if defined(CALIBRATE2)
            Serial.printf(" Zero ADC value : %u \n\n", avgZer);
          #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
}
//#############################################################################
```

ACS712

```
//--------------------------------------------------------------------------
// defines
//--------------------------------------------------------------------------
//#define CALIBRATE1                        // calibrate first using pot-trimmer for max swing
#define CALIBRATE2                        // calibrate ADCamp in code

#define ADCamp            (float)0.0195    // amp per ADC count

#define ACS712_GPIO       34               // define the used ADC input channel
#define CLOCKSPEED        80               // clockspeed of ESP32 in MHz
#define nomFREQ           50.00            // nominal frequency

//--------------------------------------------------------------------------
// 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

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

// vars related to ADC and Currents
float ADCval;
float ADCzero   = 1900;

unsigned long valZER[iADC];
unsigned long avgZER[iRMS];
float         valADC[iADC];
float         valRMS[iRMS];

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

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

//=========================================================================
// Setup
//=========================================================================
void setup()
{

  Serial.begin(115200);
  delay(1000);

  #if defined(CALIBRATE1)

    pinMode(ACS712_GPIO, INPUT);

  #else

    //-----------------------------------------------------------------------
    // 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);

  #endif

}
//=========================================================================
```

```
//#############################################################################
// Main loop
//#############################################################################
void loop() {
  #if defined(CALIBRATE1)
    // use Serial Plotter to view waveform
    Serial.println(analogRead(ACS712_GPIO));
 #else
    float         sum    = 0;
    float         rmsVal = 0;
    unsigned int  avgZer = 0;

    // interrupty occured if cntrINT > 1
    if (cntrINT > 0) {
      // read the ADC and save value
      valZER[cntADC] = (unsigned int)(analogRead(ACS712_GPIO));
      ADCval         = (float)(valZER[cntADC]);
      ADCval         = (ADCval - ADCzero) * ADCamp;
      valADC[cntADC] = (float)ADCval;

      cntADC++;
      // if array is full, calc rms
      if( cntADC >= iADC) {
        // reset counter
        cntADC = 0;

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

        // find RMS
        // RMS = the square root of the mean square (the arithmetic mean of the squares) of the set
        float valV;
        sum = 0;
        // skip first as it might not be correct due to long loop
        for (int n=1; n<iADC; n++)
        {
          valV = (float)valADC[n];
          sum = sum + (float)(valV * valV);
        }
        valRMS[cntRMS] = (float)(sqrt(sum / (iADC - 1)));

        cntRMS++;
        // array filled? If so, average
        if(cntRMS >= iRMS)
        {
          // reset counter
          cntRMS = 0;

          #if defined(CALIBRATE2)
            sum = 0;
            // find average DC offset
            for (int n=0; n<iRMS; n++)
            {
              sum = sum + (float)avgZER[n];
            }
            avgZer = (unsigned int)(sum / iRMS);
          #endif

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

          Serial.printf(" Line Voltage   : %4.2f Arms\n", rmsVal);
          #if defined(CALIBRATE2)
            Serial.printf(" Zero ADC value : %u \n\n", avgZer);
          #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
}
//#############################################################################
```

Om de communicatie met het “home automation” systeem te maken, wordt het MQTT protocol gebruikt dat door zowat alle home automation systemen wordt ondersteund.

De settings voor MQTT worden in BoilerController.h aangepast. (zie appendix)

```
// MQTT settings
#define mqttServer             "your mqtt server”
#define mqttPort               1883
#define mqttLogin              ""
#define mqttPassword           ""
#define mqttTopicBase          "your base topic"
// send MQTT message every x seconds
#define mqttInterval           60
```

Hier kan je je MQTT-server en zijn poort aanpassen. De “mqttTopicBase” is de basis om informatie te publishen en commands door te sturen. Stel even dat de topicbase “engrie111/bc” is dan krijg je volgende structuur in je MQTT server

Dus onder de topicbase komt een extra tak met “bc-” gevolgd door het MAC-adres van de ESP32. Zodoende kunnen meerdere BC’s gebruikt worden. Daaronder vallen dan

- online : geeft aan of BC online of offline ie
- data
  - gen : meer algemene data
  - dat : de data die wordt uitgelezen
- command : hier kan je volgende command posten
  Relay=ON of Relay=OFF
  Out=xxx met xxx een getal van 0 to 100, zijnde 0 tot 100% van de DAC output

Noot: Om zeer gemakkelijk met MQTT servers te kunnen werken, beveel ik “MQTT Explorer” aan. http://mqtt-explorer.com/ . De huidige versie is 0.4.0-beta1 en werkt uitstekend.

OTA wordt ook in de code gestoken zodat updating van de ESP32 gemakkelijker wordt.
Ook de settings voor OTA zijn in de BoilerController.h aanpasbaar.

Om de juiste tijd te hebben, wordt NTP gebruikt. Ook die setting in BoilerController.h

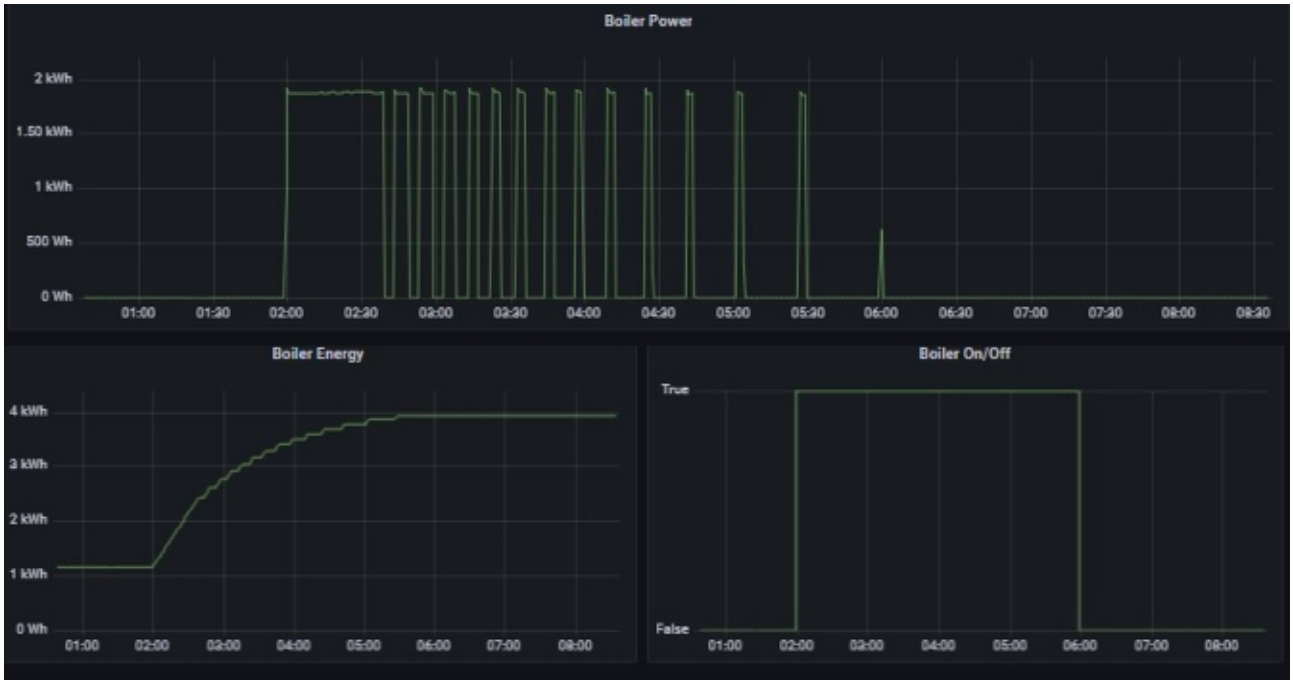
En als laatste kan je ook je WiFi settings in Boilerconroller.h aanpassen.

Andere aanpassingen moeten in de code gebeuren. De volledige, operationele, code vind je in de appendix.

Weetjes

Mijn boiler laadt niet op zoals ik dacht: spanning erop, opwarmen, temperatuur bereikt en slaat af. Kous af.
Hieronder zie je hoe een opwarmingscyclus verloopt. Warmt op gedurende een fikse periode, schakelt af, en dan een reeks van uitdeindende puls-opwarmingen die ook steeds korte van duur worden.

Mijn verklaring hiervan is: water rond weerstand warmt op tot gewenste temperatuur, thermostaat schakelt uit. Warm water stijgt langzaam naar boven en koud water zakt langzaam. Dus thermostaat schakelt aan, koudere water warmt op, thermostaat schakelt uit, en zo verder.



Een uitgeprobeerd idee maar niet OK:
als ik nu eens minder snel zou opwarmen? enerzijds om warm water de tijd de gegeven te stijgen/koud water te dalen en anderzijds zou dit in een capaciteitstarief ook nuttig zijn (kleinere piek). Dus een diode er tussen zodat we maar op halve 'kracht' opwarmen, weinig verliezen (***Noot***: Diode 15A 600V verstookt toch nog 12W = 8A met spanningsval van +/- 1,5V) . En ja hoor, halve kracht. Maar dat was blijkbaar niet naar de zin van mijn omvormer van mijn zonnepanelen. Dat liet hij zeer duidelijk horen met vervaarlijk gebrom. (logisch want geen zuiver sinus meer maar slechts een halve). Dus dat idee heb ik maar laten varen vooraleer mijn omvormer het helemaal gehad heeft met mij.

Appendix

BoilerController.h

```
//WiFi settings
#define wifiSSID                "yourSSID"
#define wifiPassword            "yuorPassword"
#define location                "Home"

// NTP settings
#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         ""

// MQTT settings
#define mqttServer              "your mqtt server”
#define mqttPort                1883
#define mqttLogin               ""
#define mqttPassword            ""
#define mqttTopicBase           "your base topic"
// send MQTT message every x seconds
#define mqttInterval            60
```

BoilerController.ino

```
//------------------------------------------------------------------------------
// Versioning
//------------------------------------------------------------------------------
#define VERSION    "1.20"

//------------------------------------------------------------------------------
// Debugging flag
//------------------------------------------------------------------------------
//#define DEBUG

//------------------------------------------------------------------------------
// includes
//------------------------------------------------------------------------------
#include <Ticker.h>
#include <WiFi.h>
#include <ESPmDNS.h>
#include <NTPClient.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
#include <PubSubClient.h>
#include <OneWire.h>
#include <DallasTemperature.h>

// application settings
#include "BoilerController.h"

//------------------------------------------------------------------------------
// defines
//------------------------------------------------------------------------------

// GPIO for controlling relay
#define RELAY_GPIO         33
#define RELAYOFF           false
#define RELAYON            true

//DAC output port
#define DAC_GPIO           26

// OneWire data line connected to GPIO
#define ONE_WIRE_BUS        4

// ADC input channel for ZMPT101B
#define ZMPT101B_GPIO      35

// ADC input channel for ACS712
#define ACS712_GPIO        34

#define ADCvolt            (float)0.210       // volt per ADC count
#define ADCamps            (float)0.0195      // amp per ADC count

#define CLOCKSPEED         80                 // clockspeed of ESP32 in MHz
#define nomFREQ            50.00              // nominal frequency

// max MQTT topic size
#define MQTT_TOPIC_SIZE    150

// easy wrapper for counting seconds instead of millisceonds
#define seconds()          (millis()/1000)
// easy wrapper for executing something every t seconds
#define every(t)           for (static unsigned long _lasttime; (unsigned long)((unsigned long)millis() -
_lasttime) >= (t); _lasttime = millis())
// easy wrapper for getting number of element in array
#define ARRAY_SIZE(array)  ((sizeof(array))/(sizeof(array[0])))

//------------------------------------------------------------------------------
// 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

//------------------------------------------------------------------------------
// global variables
//------------------------------------------------------------------------------
// to hold MAC address
String wifiMACAddress = WiFi.macAddress();
char   caMAC[20];
char   hostname[64];

bool useLED = true;

// time related
unsigned long time_now;
char today[11];
// to hold date & time
char  nuDate[12];
```

```
char  nuTime[10];
char  currDST[3];
char  lastDST[3];
char *dt;
int   days[] = {31,29,31,30,31,30,31,31,30,31,30,31};

char  mqttClientID[30];
char  mqttTopicCmd[MQTT_TOPIC_SIZE];
char  mqttTopicSts[MQTT_TOPIC_SIZE];
char  mqttTopicDat1[MQTT_TOPIC_SIZE];
char  mqttTopicDat2[MQTT_TOPIC_SIZE];
char  payload[MQTT_MAX_PACKET_SIZE - MQTT_TOPIC_SIZE];

// a temporary buffer to store characters mostly to print/mail stuff
char caTemp[256];
char caJson[256];

// DS18B20 addresses
DeviceAddress ds18b20_1 = { 0x28, 0x2F, 0x9C, 0x56, 0xB5, 0x01, 0x3C, 0x9B };
DeviceAddress ds18b20_2 = { 0x28, 0xFF, 0x64, 0x02, 0xC8, 0x7D, 0x29, 0xB7 };
DeviceAddress ds18b20_3 = { 0x28, 0xFF, 0x64, 0x1E, 0x0C, 0x04, 0xC5, 0x1E };

float temp_1;
float temp_2;
float temp_3;

// DAC
unsigned int dacn = 0;
unsigned int upct = 0;

// Relay
unsigned int relay_state;

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

// vars related to ADC and Currents
float ADCval;

// voltage
float ADCzeroV   = 1700;

unsigned long valZERv[iADC];
unsigned long avgZERv[iRMS];
float         valADCv[iADC];
float         valRMSv[iRMS];

float         rmsValV = 0;

// amperes
float ADCzeroA   = 1900;

unsigned long valZERa[iADC];
unsigned long avgZERa[iRMS];
float         valADCa[iADC];
float         valRMSa[iRMS];

float         rmsValA = 0;

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


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

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

// MQTT / PubSub needings
WiFiClient       wifiClient;
PubSubClient     mqttClient(wifiClient);

// 1-Wire object
OneWire           oneWire(ONE_WIRE_BUS);

// DS18B20 object
DallasTemperature sensors(&oneWire);
```

```
//------------------------------------------------------------------------------
// 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 +++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//** Set-up WIFI
//*********************************************************************
void wifiConnect()
{

  // reset WiFi configuration
  WiFi.disconnect();
  // Next line was needed before Core 1.0.5 to set the hostname correctly
  // since Core 1.0.5 rc6 no longer. see https://github.com/espressif/arduino-esp32/issues/4732
  // leaving the line will result in getting 255.255.255.255 as IP-address.
  // WiFi.config(INADDR_NONE, INADDR_NONE, INADDR_NONE, INADDR_NONE);
  // removing line above curres getting ip 255.255.255.255 but
  // brings back the issue of hostname not being set in DHCP -  DAMNED!!!!
  WiFi.mode(WIFI_STA);
  WiFi.setHostname(hostname);
  WiFi.begin(wifiSSID, wifiPassword);

  #if defined(DEBUG)
    Serial.print("WiFi attempting connection to "); Serial.println(wifiSSID);
  #endif

  uint8_t i = 0;
  while (WiFi.status() != WL_CONNECTED)
  {
    #if defined(DEBUGWIFI)
      Serial.print('.');
    #endif

    // delay non-blocking
    unsigned long currTime = millis();
    while(millis() < currTime + 500){}
    if((++i % 16) == 0)
    {
      #if defined(DEBUGWIFI)
        Serial.println("");
        Serial.println("still trying to connect");
      #endif
    }

    if(i > 32)
    {
      #if defined(DEBUGWIFI)
        Serial.println("Restarting... (WIFI)");
      #endif
      ESP.restart();
    }
  }
}
//*********************************************************************
```

```
//** WiFi call-back
//************************************************************************
void wifiEvent(WiFiEvent_t event)
{
  Serial.printf("  [WiFi-event : %d] = ", event);

  switch (event)
  {
    case SYSTEM_EVENT_WIFI_READY:
      Serial.println("Interface ready");
      break;

    case SYSTEM_EVENT_SCAN_DONE:
      Serial.println("Completed scan for APs");
      break;

    case SYSTEM_EVENT_STA_START:
      Serial.println("STA Client started");
      break;

    case SYSTEM_EVENT_STA_STOP:
      Serial.println("STA Client stopped");
      break;

    case SYSTEM_EVENT_STA_CONNECTED:
      Serial.println("Connected to AP");
      break;

    case SYSTEM_EVENT_STA_DISCONNECTED:
      Serial.println("Disconnected from AP");
      break;

    case SYSTEM_EVENT_STA_AUTHMODE_CHANGE:
      Serial.println("Auth changed");
      break;

    case SYSTEM_EVENT_STA_GOT_IP:
      Serial.print("Obtained IP: ");
      Serial.println(WiFi.localIP());
      break;

    case SYSTEM_EVENT_STA_LOST_IP:
      Serial.println("Lost IP");
      break;

    case SYSTEM_EVENT_STA_WPS_ER_SUCCESS:
      Serial.println("WPS succeeded");
      break;

    case SYSTEM_EVENT_STA_WPS_ER_FAILED:
      Serial.println("WPS failed");
      break;

    case SYSTEM_EVENT_STA_WPS_ER_TIMEOUT:
      Serial.println("WPS timeout");
      break;

    case SYSTEM_EVENT_STA_WPS_ER_PIN:
      Serial.println("WPS pin code");
      break;

    case SYSTEM_EVENT_AP_START:
      Serial.println("AP started");
      break;

    case SYSTEM_EVENT_AP_STOP:
      Serial.println("AP stopped");
      break;

    case SYSTEM_EVENT_AP_STACONNECTED:
      Serial.println("AP Client connected");
      break;

    case SYSTEM_EVENT_AP_STADISCONNECTED:
      Serial.println("AP Client disconnected");
      break;

    case SYSTEM_EVENT_AP_STAIPASSIGNED:
      Serial.println("Assigned IP to client");
      break;

    case SYSTEM_EVENT_AP_PROBEREQRECVED:
      Serial.println("Received probe request");
      break;

    case SYSTEM_EVENT_GOT_IP6:
      Serial.println("IPv6 preferred");
      break;

    case SYSTEM_EVENT_ETH_START:
      Serial.println("Ethernet started");
```

```
      break;

    case SYSTEM_EVENT_ETH_STOP:
      Serial.println("Ethernet stopped");
      break;

    case SYSTEM_EVENT_ETH_CONNECTED:
      Serial.println("Ethernet connected");
      break;

    case SYSTEM_EVENT_ETH_DISCONNECTED:
      Serial.println("Ethernet disconnected");
      break;

    case SYSTEM_EVENT_ETH_GOT_IP:
      Serial.println("Obtained IP");
      break;

    default:
        break;
  }
}
//**************************************************************************

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

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

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

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

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

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

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

  // clear some vars
  memset(lastDST, NULL, 3);
  memset(currDST, NULL, 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] = NULL;

  #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] = NULL;

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

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// +++ 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(hostname);

  // Set OTA Password. No authentication by default
  if(otaPasswordHash != "")
  {
    ArduinoOTA.setPasswordHash(otaPasswordHash);
  }
  else if(otaPassword != "")
  {
    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
}
//**************************************************************************
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// +++ 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

  // setup and connect to MQTT server
  // mqttClientID
  wifiMACAddress.replace(":","");
  wifiMACAddress.toCharArray(caMAC, 20);
  strcpy(mqttClientID,"bc_");
  strcat(mqttClientID, caMAC);
  #if defined(DEBUG)
    Serial.print(F("  MQTT ClientID: <")); Serial.print(mqttClientID); Serial.println(F(">"));
  #endif

  // mqttTopicCmd
  strcpy(mqttTopicCmd, mqttTopicBase);
  strcat(mqttTopicCmd, "/bc_");
  strcat(mqttTopicCmd, caMAC);
  strcat(mqttTopicCmd, "/command");
  #if defined(DEBUG)
    Serial.print(F("  MQTT Topic CMD: <")); Serial.print(mqttTopicCmd); Serial.println(F(">"));
  #endif

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

  // mqttTopicDat
  strcpy(mqttTopicDat1, mqttTopicBase);
  strcat(mqttTopicDat1, "/bc_");
  strcat(mqttTopicDat1, caMAC);
  strcat(mqttTopicDat1, "/data/gen");
  #if defined(DEBUG)
    Serial.print(F("  MQTT Topic DAT1: <")); Serial.print(mqttTopicDat1); Serial.println(F(">"));
  #endif

  strcpy(mqttTopicDat2, mqttTopicBase);
  strcat(mqttTopicDat2, "/bc_");
  strcat(mqttTopicDat2, caMAC);
  strcat(mqttTopicDat2, "/data/dat");
  #if defined(DEBUG)
    Serial.print(F("  MQTT Topic DAT2: <")); Serial.print(mqttTopicDat2); Serial.println(F(">"));
  #endif

  mqttClient.setServer(mqttServer, mqttPort);
  mqttClient.setCallback(mqttOnMessage);

  // 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, mqttPassword, mqttTopicSts, 1, true, "offline", false))
  {

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

    // subscribe to command topic
    mqttClient.subscribe(mqttTopicCmd, 1);
    // 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
  }

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

```
// ** reconnect MQTT
//*****************************************************************************
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, mqttPassword, mqttTopicSts, 1, true, "offline", false))
    {
      #if defined(DEBUG)
        Serial.println(F("MQTT reconnected"));
      #endif
      // subscribe to command topic
      mqttClient.subscribe(mqttTopicCmd, 1);
      // 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);
}
//*****************************************************************************

//** receive MQTT messages
//*****************************************************************************
void mqttOnMessage(char* topic, byte* payload, unsigned int payloadlength)
{
  //callback includes topic and payload
  #if defined(DEBUG)
    Serial.print(F("Message arrived on topic [")); Serial.print(topic); Serial.print(F("] : "));
  #endif

  char caTemp[payloadlength + 1];
  strncpy(caTemp, (char*)payload, payloadlength);
  caTemp[payloadlength] = NULL;

  #if defined(DEBUG)
    Serial.print(F("  <")); Serial.print(caTemp); Serial.println(F(">"));
  #endif

  // do what is needed based on the payload
  if(strncasecmp(caTemp, "Out=", 4) == 0)
  {
    #if defined(DEBUG)
      Serial.print("MQTT Message Received: "); Serial.println(caTemp);
    #endif

    char *pct;
    // part before =
    pct = strtok(caTemp, "=");
    // part after =
    pct = strtok(NULL, "=");
    //#if defined(DEBUG)
    //   Serial.print("%: "); Serial.println(pct);
    //#endif
    if (strlen(pct) > 0)
    {
      setDAC(pct);
    }
  }
```

```
  else if(strcasecmp(caTemp, "Relay=ON") == 0)
  {
    #if defined(DEBUG)
      Serial.println("MQTT Message Received: Relay=ON");
    #endif
    setRelay(RELAYON);
  }
  else if(strcasecmp(caTemp, "Relay=OFF") == 0)
  {
    #if defined(DEBUG)
      Serial.println("MQTT Message Received: Relay=OFF");
    #endif
    setRelay(RELAYOFF);
  }
  else if(strcasecmp(caTemp, "ON") == 0)
  {
    #if defined(DEBUG)
      Serial.println("MQTT Message Received: ON");
    #endif
    setRelay(RELAYON);
  }
  else if(strcasecmp(caTemp, "OFF") == 0)
  {
    #if defined(DEBUG)
      Serial.println("MQTT Message Received: OFF");
    #endif
    setRelay(RELAYOFF);
  }
}
//*****************************************************************************

//** send MQTT data
//*****************************************************************************
void mqttSendData()
{
  //---------------------------------------------------------------------------
  // Read DS18B20 sensors
  //---------------------------------------------------------------------------
  #if defined(DEBUG)
    Serial.println("");
    Serial.println("Requesting temperatures...");
  #endif
  // Send the command to get temperatures
  sensors.requestTemperatures();

  temp_1 = sensors.getTempC(ds18b20_1);
  temp_2 = sensors.getTempC(ds18b20_2);
  temp_3 = sensors.getTempC(ds18b20_3);

  #if defined(DEBUG)
    // display temps
    Serial.print("Sensor 1 :"); Serial.printf(" %3.1f C\n", temp_1);
    Serial.print("Sensor 2 :"); Serial.printf(" %3.1f C\n", temp_2);
    Serial.print("Sensor 3 :"); Serial.printf(" %3.1f C\n", temp_3);
  #endif

  if(useLED)
  {
    digitalWrite(LED_BUILTIN, HIGH);
  }

  dt = ntpLocalDateTime();

  #if defined(DEBUG)
    Serial.println(F("Sending data via MQTT"));
  #endif
  strcpy(caJson, "{ ");
  sprintf(caTemp, "\"IP\": \"%s\", ", WiFi.localIP().toString().c_str());
  strcat(caJson, caTemp);
  sprintf(caTemp, "\"Hostname\": \"%s\", ", hostname);
  strcat(caJson, caTemp);
  sprintf(caTemp, "\"Location\": \"%s\", ", location);
  strcat(caJson, caTemp);
  dt = ntpLocalDateTime();
  sprintf(caTemp, "\"TimeStamp\": \"%s\" ", dt);
  strcat(caJson, caTemp);
  strcat(caJson, "}");
  mqttSend(mqttTopicDat1, caJson);
  #if defined(DEBUG)
    Serial.printf("  --> %s\n", mqttTopicDat1);
    Serial.printf("      %s\n", caJson);
  #endif

  strcpy(caJson, "{ ");
  sprintf(caTemp, "\"Volt\": %.0f, ", rmsValV);
  strcat(caJson, caTemp);
  sprintf(caTemp, "\"Amps\": %.1f, ", rmsValA);
  strcat(caJson, caTemp);
  sprintf(caTemp, "\"Power\": %.0f, ", rmsValV * rmsValA);
  strcat(caJson, caTemp);
  sprintf(caTemp, "\"Temp1\": %.1f, ", temp_1);
```

```
  strcat(caJson, caTemp);
  sprintf(caTemp, "\"Temp2\": %.1f, ", temp_2);
  strcat(caJson, caTemp);
  sprintf(caTemp, "\"Temp3\": %.1f, ", temp_3);
  strcat(caJson, caTemp);
  sprintf(caTemp, "\"Out\": %u, ", upct);
  strcat(caJson, caTemp);
  sprintf(caTemp, "\"Relay\": %u", relay_state);
  strcat(caJson, caTemp);
  strcat(caJson, "}");
  mqttSend(mqttTopicDat2, caJson);
  #if defined(DEBUG)
    Serial.printf("  --> %s\n", mqttTopicDat2);
    Serial.printf("      %s\n", caJson);
  #endif
  mqttSend(mqttTopicSts, "online");
  if(useLED)
  {
    digitalWrite(LED_BUILTIN, LOW);
  }
}
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//** set Relay
//*****************************************************************************
void setRelay(bool OnOff)
{

  if(OnOff == RELAYON)
  {
    // set output high
    digitalWrite(RELAY_GPIO, RELAYON);
    #if defined(DEBUG)
      Serial.println("Relay ON");
    #endif
  }
  else
  {
    // set output low
    digitalWrite(RELAY_GPIO, RELAYOFF);
    #if defined(DEBUG)
      Serial.println("Relay OFF");
    #endif
  }

  if (digitalRead(RELAY_GPIO) == RELAYON)
  {
    relay_state = 1;
  }
  else
  {
    relay_state = 0;
  }
  mqttSendData();
}
//*****************************************************************************

//** set DAC
//*****************************************************************************
void setDAC(char *percent)
{

  // convert string to float and then to a DAC number
  upct = (unsigned int)(abs(atoi(percent)));

  // keep between boundaries
  if(upct < 0)
  {
    upct = 0;
  }
  else if (upct > 100)
  {
    upct = 100;
  }

  dacn = (unsigned int)((255 * upct) / 100);
  #if defined(DEBUG)
    Serial.printf("Setting DAC: PCT: %u - DACN: %u\n", upct, dacn);
  #endif
  dacWrite(DAC_GPIO, dacn);
  mqttSendData();

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

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// +++ ADC VA Functions +++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//** reading voltage
//*************************************************************************
void readVolt(bool ADC, bool RMS)
{

  float sum;

  // read the ADC and save value
  valZERv[cntADC] = (unsigned int)(analogRead(ZMPT101B_GPIO));
  ADCval          = (float)(valZERv[cntADC]);
  ADCval          = (ADCval - ADCzeroV) * ADCvolt;
  valADCv[cntADC] = (float)(ADCval);

  if( ADC )
  {
    // 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 + (float)valZERv[n];
    }
    avgZERv[cntRMS] = (unsigned int)(sum / (iADC - 1));
    // adjust Zero offset for current
    ADCzeroV = (float)avgZERv[cntRMS];

    // find RMS
    // RMS = the square root of the mean square (the arithmetic mean of the squares) of the set
    float valV;
    sum = 0;
    // skip first as it might not be correct due to long loop
    for (int n=1; n<iADC; n++)
    {
      valV = (float)valADCv[n];
      sum  = sum + (float)(valV * valV);
    }
    valRMSv[cntRMS] = (float)(sqrt(sum / (iADC - 1)));
  }

  if( RMS )
  {
    // find average RMS
    sum = 0;
    for (int n=0; n<iRMS; n++)
    {
      sum = sum +(float)valRMSv[n];
    }
    // The average RMS value of ADC values
    rmsValV = (float)(sum / iRMS);

    #if defined(DEBUG)
      Serial.printf(" Line Voltage   : %4.0f Vrms\n", rmsValV);
    #endif
  }
}
//*************************************************************************


//** reading amperes
//*************************************************************************
void readAmps(bool ADC, bool RMS)
{

  float sum;

  // read the ADC and save value
  valZERa[cntADC] = (unsigned int)(analogRead(ACS712_GPIO));
  ADCval          = (float)(valZERa[cntADC]);
  ADCval          = (ADCval - ADCzeroA) * ADCamps;
  valADCa[cntADC]  = (float)ADCval;

  if( ADC )
  {
    // 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 + (float)valZERa[n];
    }
    avgZERa[cntRMS] = (unsigned int)(sum / (iADC - 1));
    // adjust Zero offset for current
    ADCzeroA = (float)avgZERa[cntRMS];

    // find RMS
    // RMS = the square root of the mean square (the arithmetic mean of the squares) of the set
    float valV;
```

```
    sum = 0;
    // skip first as it might not be correct due to long loop
    for (int n=1; n<iADC; n++)
    {
      valV = (float)valADCa[n];
      sum  = sum + (float)(valV * valV);
    }
    valRMSa[cntRMS] = (float)(sqrt(sum / (iADC - 1)));
  }

  if( RMS )
  {
    // find average RMS
    sum = 0;
    for (int n=0; n<iRMS; n++)
    {
      sum = sum +(float)valRMSa[n];
    }
    // The average RMS value of ADC values
    rmsValA = (float)(sum / iRMS);

    #if defined(DEBUG)
      Serial.printf(" Line Amperes   : %4.2f Arms\n", rmsValA);
    #endif
  }
}
//*************************************************************************
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//=========================================================================
void setup()
{
  #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);

  //-------------------------------------------------------------------------
  // Startup WiFi
  //-------------------------------------------------------------------------
  String sTmp = wifiMACAddress;
  #if defined(DEBUG)
    WiFi.onEvent(wifiEvent);
  #endif
  // setup hostname
  sTmp.replace(":", "");
  sTmp.toUpperCase();
  sTmp.toCharArray(caMAC, 20);
  strcpy(hostname, "ESP32-BC-");
  strcat(hostname, caMAC);
  wifiConnect();

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

  //-------------------------------------------------------------------------
  // Startup time
  //-------------------------------------------------------------------------
  ntpSetup();
  // get time
  dt = ntpLocalDateTime();
  strncpy(today, dt, 10);
  today[10] = NULL;
  #if defined(DEBUG)
    Serial.print(dt); Serial.print(F(" - Today : [")); Serial.print(today); Serial.println(F("]"));
  #endif

  //-------------------------------------------------------------------------

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

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

  //------------------------------------------------------------------------
  // Stop Bluetooth
  //------------------------------------------------------------------------
  btStop();
  //------------------------------------------------------------------------

  //------------------------------------------------------------------------
  // Startup mqtt
  //------------------------------------------------------------------------
  mqttConnect();
  //------------------------------------------------------------------------

  //------------------------------------------------------------------------
  // Startup DS18B20 sensors
  //------------------------------------------------------------------------
  sensors.begin();
  //------------------------------------------------------------------------

  //------------------------------------------------------------------------
  // Set-up Relay port
  //------------------------------------------------------------------------
  pinMode(RELAY_GPIO, OUTPUT);
  //------------------------------------------------------------------------

  //------------------------------------------------------------------------
  // 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);

  //------------------------------------------------------------------------
  // init Realy and DAC
  //------------------------------------------------------------------------
  setRelay(RELAYOFF);
  setDAC(itoa(upct,caTemp,10));
  //------------------------------------------------------------------------

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

```
//=============================================================================
void loop() {
  //-----------------------------------------------------------------------------
  // OTA handling
  //-----------------------------------------------------------------------------
  ArduinoOTA.handle();

  //-----------------------------------------------------------------------------
  // Read ADC
  //-----------------------------------------------------------------------------

  // interrupt occured if cntrINT > 1
  if (cntrINT > 0) {
    cntADC++;
    // if array is full, calc rms
    if(cntADC >= (iADC - 1))
    {
      cntRMS++;

      // if array is full, calc average rms
      if(cntRMS >= (iRMS - 1))
      {
        readVolt(true, true);
        readAmps(true, true);
        // reset counter
        cntADC = -1;
        cntRMS = -1;
      }
      else
      {
        readVolt(true, false);
        readAmps(true, false);
        // reset counter
        cntADC = -1;
      }
    }
    else
    {
      readVolt(false, false);
      readAmps(false, false);
    }

    // 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);
  }

  //-----------------------------------------------------------------------------
  // MQTT handling
  //-----------------------------------------------------------------------------
  if (!mqttClient.connected()) {
    mqttReconnect();
  }
  // start mqtt listener
  mqttClient.loop();

  every(mqttInterval * 1000)
  {
    mqttSendData();
  }

  //-----------------------------------------------------------------------------
  // reboot every day around midnight
  //-----------------------------------------------------------------------------
  // check only so often = 5 minutes, not every loop
  every(300 * 1000)
  {
    char  nu[11];
    dt = ntpLocalDateTime();
    // only compare date , not time
    strncpy(nu, dt, 10);
    nu[10] = NULL;

    #if defined(DEBUG)
      Serial.print(F(" - Today : [")); Serial.print(today); Serial.println(F("]"));
      Serial.print(F(" - Now   : [")); Serial.print(nu);    Serial.println(F("]"));
    #endif

    // if (strcmp(today, nu) < 0) {
      // #if defined(DEBUG)
        // Serial.println(F("Restarting..."));
      // #endif
      // ESP.restart();
    // }
    strcpy(today, nu);
  }
}
```