Relazione tecnica

Interfaccia Web
per il controllo dei consumi elettrici
da remoto

Introduzione:

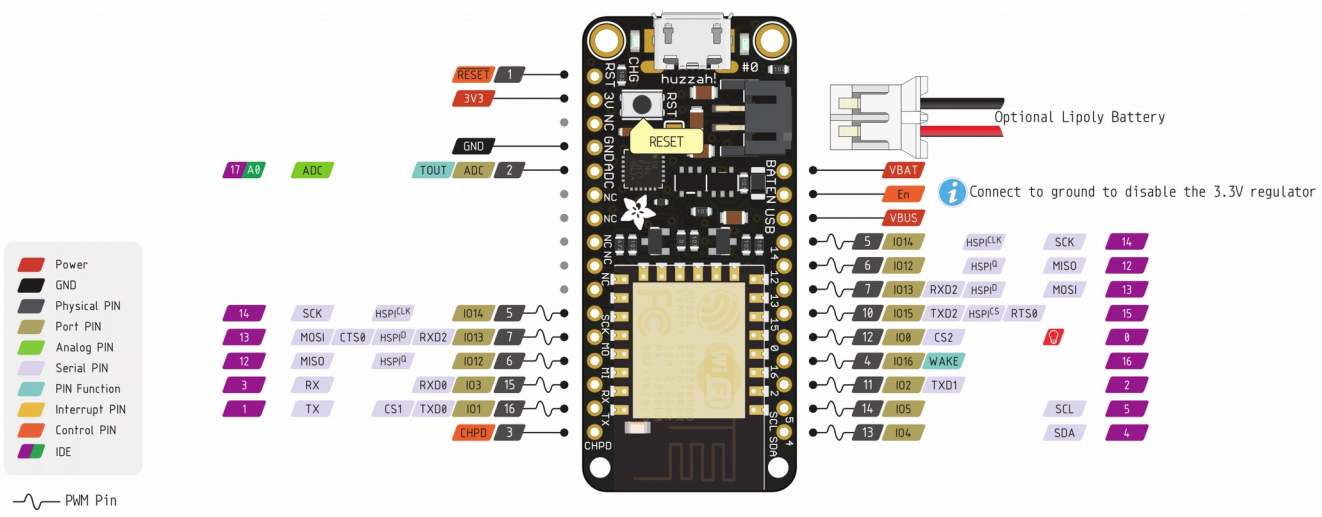
Immaginando un sistema di domotica futuristico, abbiamo pensato di realizzare un sistema per controllare i consumi di corrente (kWatt), in modo da poter risparmiare sulla bolletta di casa, o anche solo per avere una visione completa della potenza assorbita dagli elettrodomestici.
In campo industriale lo stesso sistema può essere applicato in scala maggiore per monitorare i consumi dei macchinari, inoltre, con uno studio più approfondito è possibile osservare la risposta temporale della corrente per un dato macchinario, determinando curve e picchi di potenza caratteristici.

Hardware utilizzato per il circuito:

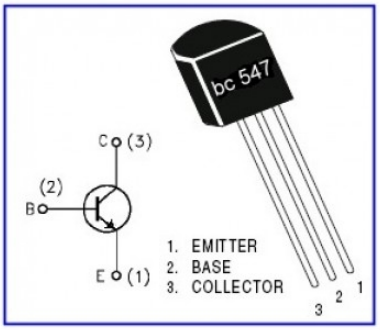
Nella realizzazione del progetto siamo stati in grado di leggere, mediante una sonda amperometrica, i valori di corrente e potenza istantanei. La sonda amperometrica utilizzata (YHDC_SCT013000) permette di leggere correnti fino a 100 A al primario del trasformatore interno, il quale, con un rapporto di spire 2000:1, trasforma la corrente in valori molto più piccoli, che possono raggiungere al massimo 50 mA.
Nel circuito da noi realizzato, tenuto conto dei valori di corrente molto più bassi in un comune impianto elettrico domestico, è stato preso come valore massimo una corrente di 30 A. Il circuito di fatto è progettato per reggere carichi di corrente non superiori a questa soglia.

Il circuito, converte un segnale in corrente, in un segnale in tensione, grazie alla resistenza di burden e lo invia al microcontrollore, che elabora il dato, restituendo valori in ampere e kilowatt del primario. Il microcontrollore, interfacciato al Wi-Fi, comanda un relè (il nostro attuatore) accendendo in contrapposizione fra loro un LED rosso e uno verde. L'utilizzo dei LED è a titolo di esempio, possono essere sostituiti da un qualsiasi circuito da accendere o spegnere.

Il microcontrollore impiegato è un Adafruit Feather Huzzah ESP8266 a 3,3 V. Il pin analogico ADC può acquisire tensioni da 0 a 1 V, convertiti in valori discreti da 0 a 1023 (10 bit) e correnti non superiori a 6 mA.



Il relè utilizzato è un **SRD-05VDC-S L-C** a 5 V, per alimentarlo è necessario l'utilizzo di un **transistor NPN**, che riceva alla base (B) un segnale da parte dell'ESP8266, segnale di attivazione di una corrente a una tensione maggiore (5 V) fra collettore (C) ed emettitore (E). Quest'ultima è capace di far attivare il relè, accendendo a sua volta il LED verde e spegnendo quello rosso.

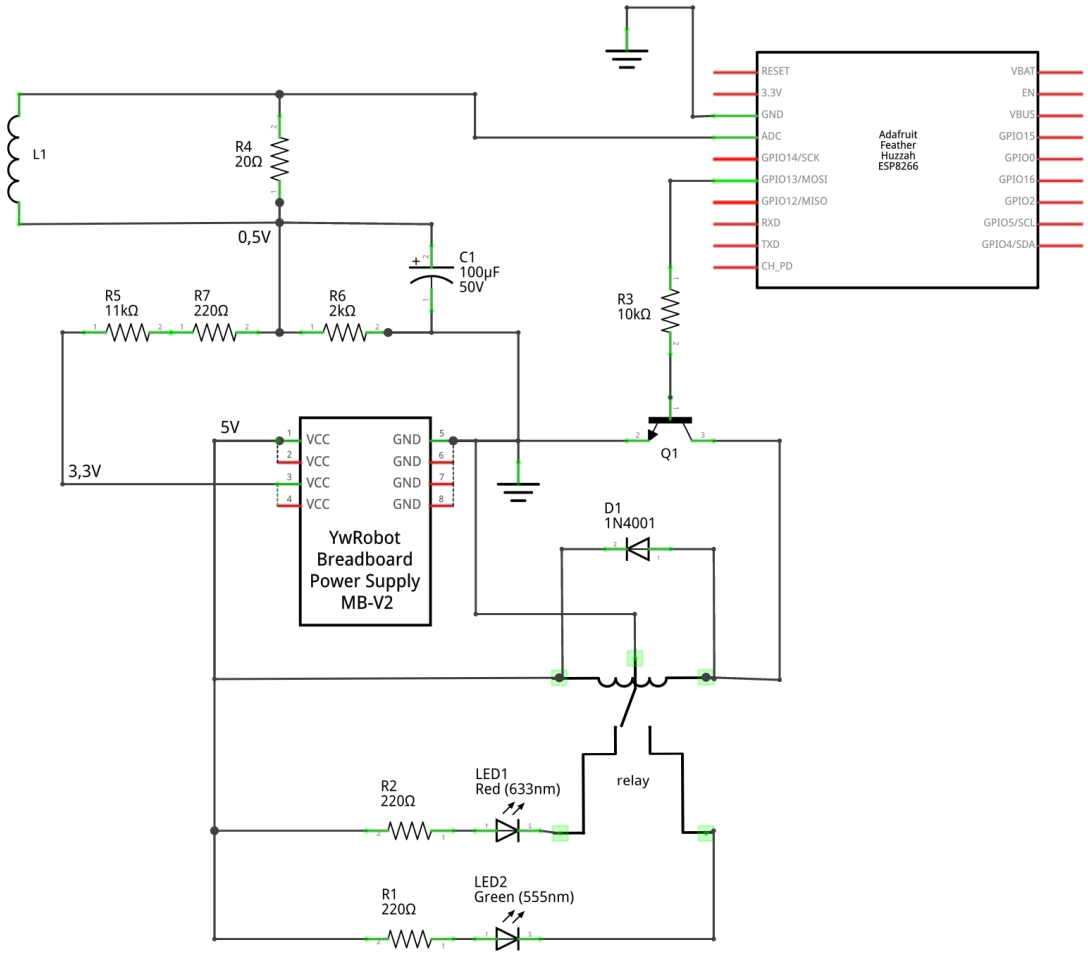


Il circuito è stato progettato seguendo due logiche differenti:

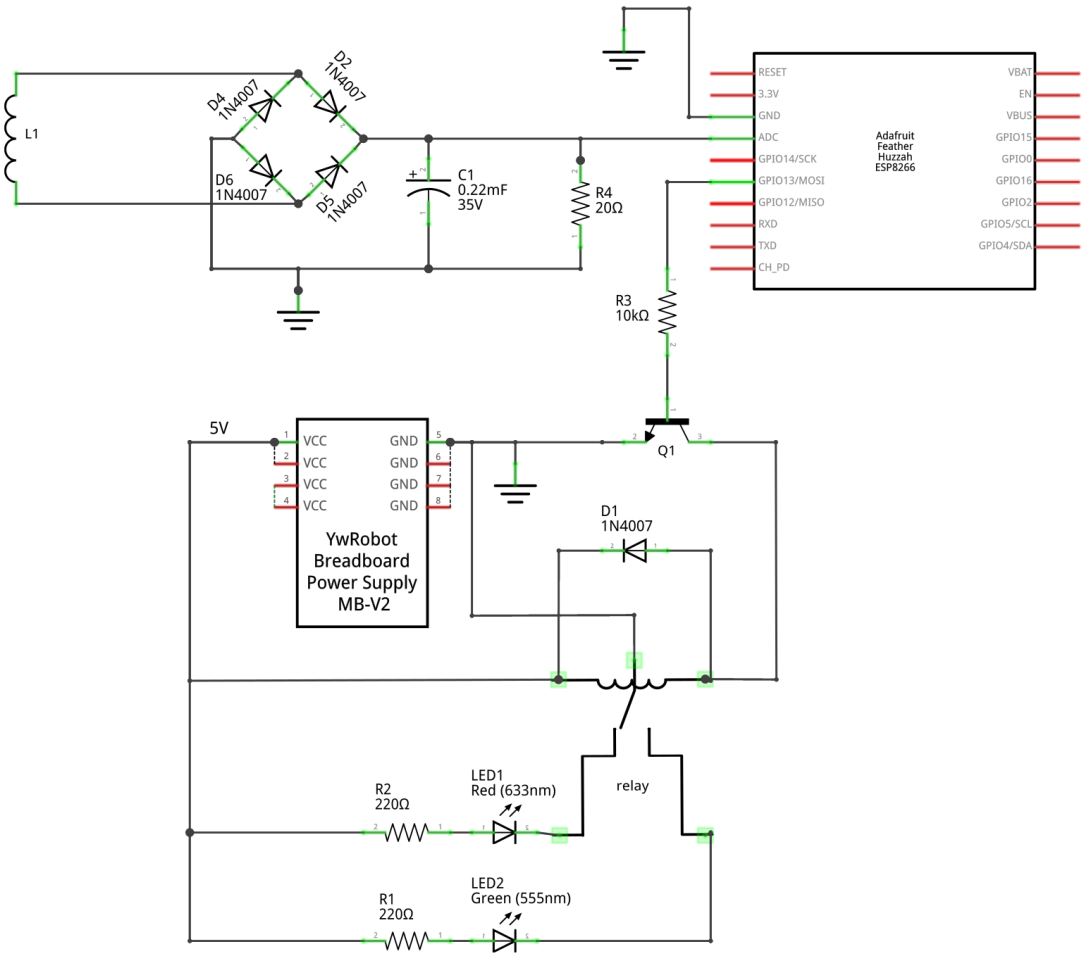
1. Lettura del valore più alto di segnali in corrente alternata (**AC**) sul pin ADC.
2. Acquisizione del segnale in corrente continua (**DC**) sul pin ADC del microcontrollore;

Con la prima logica abbiamo letture molto più lente, con problemi di lag fra una lettura e la successiva, dovuti a calcoli e intercettazione del valore massimo in un array di letture, anche se il circuito è notevolmente più semplice, con valori che restano lineari con le correnti in gioco sul primario.
Riguardo alla seconda logica, invece, la lettura è istantanea, evitando di conseguenza eventuali lag, nonostante sia di più complicata realizzazione, dato l'impiego di un ponte di Graetz (diodi), che causa cadute di tensione nel circuito. La resistenza di burden in questo caso è a valle del ponte di diodi, in questo modo il valore di tensione letto può essere riconvertito nel valore di corrente al secondario della sonda amperometrica.

Schema circuito AC:

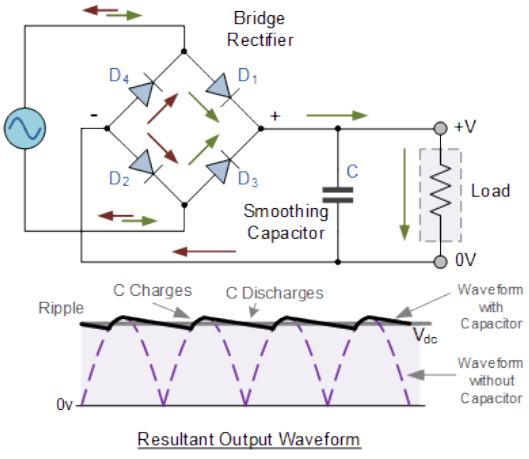


Schema circuito DC:

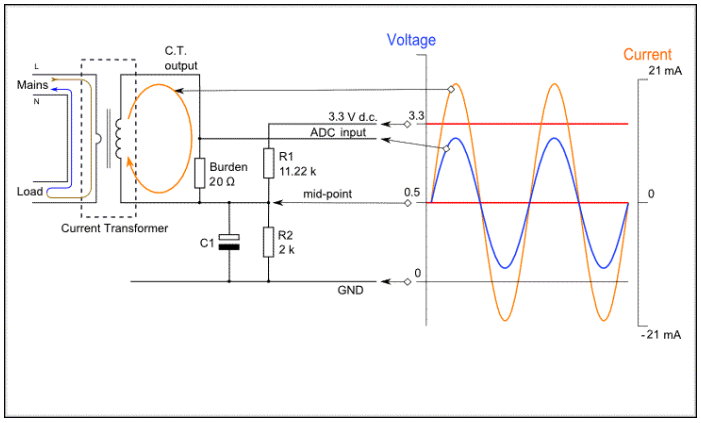


Considerazioni sul circuito:

Nel circuito DC è utilizzato un raddrizzatore di corrente a doppia semionda, formato da quattro diodi. La forma d'onda della tensione risultante è costituita dalle semionde positive, più le semionde negative ribaltate di segno (come nell'immagine). La tensione pulsante fornita dal raddrizzatore viene quasi sempre livellata mediante un filtro capacitivo. In questo modo il condensatore a valle del ponte di diodi riduce il ripple (fattore di ondulazione) della tensione. Si carica all'aumentare della tensione rettificata, quando invece la tensione raddrizzata che entra nel condensatore inizia il suo declino, il condensatore scarica molto lentamente l'energia immagazzinata, fornendo energia al carico.



Resultant Output Waveform



Per quanto riguarda il circuito AC, viene considerata una corrente massima di 30 A al primario, che corrisponde a una corrente di picco al secondario di 21 mA.
Con una resistenza di burden si limita la tensione entro certi valori massimi, altrimenti si raggiungerebbero tensioni altissime. Dalla legge di Ohm si ottiene il valore della resistenza di burden da usare, calcolando il rapporto fra la metà della massima tensione leggibile sull'input ADC (1 V / 2 = 0,5 V) e la corrente di picco al secondario.

Rb = 0,5 V / 21 mA ≈ 20 Ω

In questo modo il picco (+) di tensione sarà di 0,5 V, mentre il picco (-) sarà di -0,5 V. Con un partitore di tensione e un condensatore si porta a 0,5 V uno dei due capi del circuito, costringendo la tensione a oscillare sopra e sotto questo valore di riferimento, toccando un picco di 1 V massimo.

Protocollo di comunicazione:

Il microcontrollore dialoga via Wi-Fi con il server realizzato con **Node-Red**, che permette di gestire mediante interfaccia grafica l'attivazione e lo spegnimento del relè, visualizza lo status delle letture, e avvisa tramite notifiche e allarmi se le soglie di corrente impostate sono state superate.

Per la comunicazione fra server Node-Red e l'ESP8266 si è optato per il protocollo **MQTT** (Message Queue Telemetry Transport), un protocollo specificatamente studiato per le situazioni in cui è richiesto un basso consumo e dove la banda è limitata, lo standard di riferimento per la comunicazione per l'Internet of Things.

Il protocollo MQTT adotta un meccanismo di pubblicazione e sottoscrizione per scambiare messaggi tramite un apposito "**message broker**". Invece di inviare messaggi a un determinato set di destinatari, i mittenti pubblicano i messaggi su un certo argomento (detto **topic**) sul message broker, Ogni destinatario si iscrive al topic e, ogni volta che un nuovo messaggio viene pubblicato su quel determinato topic, il broker lo distribuisce a tutti i destinatari. In questo modo è molto semplice configurare una messaggistica uno-a-molti.

Il broker MQTT utilizzato: *nash.abinsula.com*
I payload inviati sono Json con tutti i parametri e comandi gestiti schematicamente.

I topic utilizzati sono:

/ABI_TIR/EVENTS per le noifiche di allarme del superamento delle soglie di corrente impostate;
/ABI_TIR/REQ per le richieste di settaggio o di lettura da parte del server Node-Red verso l'ESP8266;
/ABI_TIR/RES per le risposte con i valori da parte dell'ESP8266 verso il server;

Le richieste:

- per l'accensione o spegnimento del relè
  "{'REQUEST': {'request_id':111, 'action_type':'SET', 'action_detail':'relay', 'action_value':'ON'}}"
  "{'REQUEST': {'request_id':222, 'action_type':'SET', 'action_detail':'relay', 'action_value':'OFF'}}"
- dello stato attuale del relè
  "{'REQUEST': {'request_id':333, 'action_type':'GET', 'action_detail':'relay', 'action_value':'STATUS'}}"
- di lettura dei valori di corrente e potenza
  "{'REQUEST': {'request_id':444, 'action_type':'GET', 'action_detail':'current', 'action_value':'STATUS'}}"
- di settaggio del valore soglia massimo di corrente, oltre il quale viene spento il relè
  "{'REQUEST': {'request_id':555, 'action_type':'SET', 'action_detail':'shutdown', 'action_value':10}}"
- di settaggio del valore di corrente, oltre il quale viene inviata una notifica
  "{'REQUEST': {'request_id':666, 'action_type':'SET', 'action_detail':'high_current', 'action_value':8}}"
- del valore soglia di corrente massima in uso
  "{'REQUEST': {'request_id':777, 'action_type':'GET', 'action_detail':'shutdown', 'action_value':'STATUS'}}"
- del valore soglia di corrente per la notifica in uso
  "{'REQUEST': {'request_id':888, 'action_type':'GET', 'action_detail':'high_current', 'action_value':'STATUS'}}"

Le risposte:

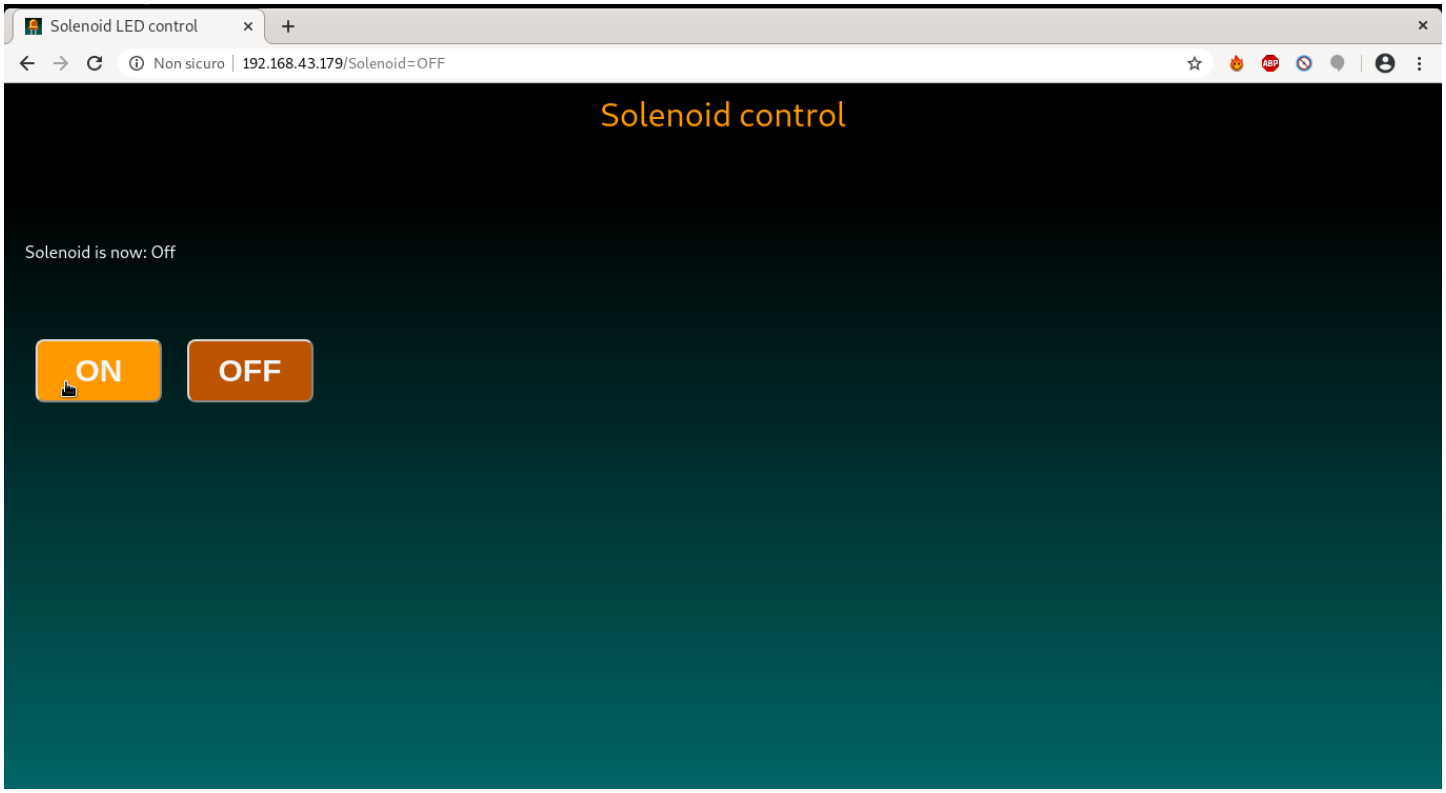
- con i valori di corrente e potenza istantanei
  "{'RESPONSE':{'request_id':444,'current_value':1.035,'power_value':0.238,'return_status':'OK'}}"
- con lo stato attuale del relè
  "{'RESPONSE':{'request_id':333,'return_value':'OFF','return_status':'OK'}}"

Gli eventi di notifica:

- dell'allarme di superamento soglia massima di corrente
  "{'EVENTS': {'event_type':'ALARM', 'notify_value':123, 'alarm_value':'max_current'}}"
- del superamento soglia di corrente alta
  "{'EVENTS': {'event_type':'NOTIFY', 'notify_value':123, 'alarm_value':'high_current'}}"

Sito web nella rete locale:

Nel codice dell'ESP8266, oltre al controllo delle letture di corrente e potenza e al sistema di comunicazione MQTT col server Node-Red, avviene anche la creazione di una pagina web nella rete locale, da cui è possibile, mediante l'interfaccia grafica, controllare l'accensione e lo spegnimento del relè attraverso due semplici pulsanti ON e OFF. Inserendo l'indirizzo IP del microcontrollore si accede alla pagina.



Sketch ESP8266:

```
#include <Arduino.h>
#include <ArduinoJson.h>
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

WiFiClient espClient;
PubSubClient client(espClient);

const char* ssid = "My ASUS";                 // ssid and password of wifi AP
const char* password = "17772hdjd";

const char* mqtt_server = "nash.abinsula.com";  // MQTT-broker ip address (indirizzo del tuo pc se usi mosquitto) 192.168.43.148
const char* clientName = "ESP8266_Ampere";      // name of this client, it will be visible from the broker

const int pinRelay = 13;                    // GPIO13---D7 of NodeMCU

String msgIn;                             // MQTT message input, from byte payload

WiFiServer server(80);                      // server on port 80
int value = LOW;                          // site flag-value for the relay last state (on or off)

float ValADC = 0;                         // variables for electric current reading:
const float diodsTensFall = 1.4;
const float Nspire = 2000;
```

```
float Vrms;
float Irms;
float kWatt;

float maxCurrent = 30;                  // settings for Current Alarm
float highCurrent = 20;

unsigned long timer;



//////////////////////////////- SETUP -////////////////////////////////

void setup()
{
  pinMode(0, OUTPUT);
  pinMode(2, OUTPUT);
  pinMode(pinRelay, OUTPUT);
  digitalWrite(pinRelay, LOW);
  digitalWrite(0, HIGH);
  digitalWrite(2, HIGH);

  Serial.begin(115200);

  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
}


 ////_____ WIFI setup _____////

 void setup_wifi(){
   delay(10);

   Serial.println();                          // Connect to WiFi network
   Serial.println();
   Serial.print("Connecting to ");
   Serial.println(ssid);

   WiFi.begin(ssid, password);                    // Attempting to connect to the AP

   while (WiFi.status() != WL_CONNECTED) {
     delay(500);
     Serial.print(".");
   }
   Serial.println("");
   Serial.println("WiFi connected");                // Connected

   server.begin();                             // Start the server
   Serial.println("Server started");

   Serial.print("Use this URL to connect: ");         // Print the IP address on serial monitor
   Serial.print("http://");                       // URL IP to be typed in mobile/desktop browser
   Serial.print(WiFi.localIP());
   Serial.println("/");

   digitalWrite(0, LOW);                          // red Led if connected and server working
 }
```

```
void callback(char* topic, byte* payload, unsigned int length) {
  msgIn = "";
  digitalWrite(2, LOW);
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    msgIn = msgIn + ((char)payload[i]);
  }
  Serial.println(msgIn);
  delay(100);
  ControlsByMessage((String)topic, msgIn);
}


////_____ MQTT message Request control _____////

void ControlsByMessage(String tpc, String msg){

  Serial.println(tpc);
  Serial.println(msg);

  if (tpc == "/ABI_TIR/REQ"){

    StaticJsonDocument<200> doc;
    DeserializationError err = deserializeJson(doc, msg);
    if (err) {
       Serial.print(F("deserializeJson() failed with code "));
       Serial.println(err.c_str());
    }

    long int reqId = doc["REQUEST"]["request_id"];
    String reqType = doc["REQUEST"]["action_type"];
    String reqDetail = doc["REQUEST"]["action_detail"];
    String reqValue = doc["REQUEST"]["action_value"];

    //#### Set relay status ####//
     if (reqType == "SET"){
       if (reqDetail == "relay" && reqValue == "ON"){            // set relay ON
             digitalWrite(pinRelay, HIGH);
             value = HIGH;
             Serial.println("Set relay ON");
             String respStatus = "OK";
             genResponse(reqId, reqValue, respStatus);

       }else if (reqDetail == "relay" && reqValue == "OFF"){      // set relay OFF
             digitalWrite(pinRelay, LOW);
             value = LOW;
             Serial.println("Set relay OFF");
             String respStatus = "OK";
             genResponse(reqId, reqValue, respStatus);


       }else if (reqDetail == "shutdown"){                    // set max current value
             maxCurrent = reqValue.toFloat();                 // is necessary a cast to convert a String to a float
             Serial.print("Set max_current to ");
             Serial.println(maxCurrent);
             String respStatus = "OK";
            sendMaxHighCurrent(reqId, maxCurrent, respStatus);
```

```
      }else if (reqDetail == "high_current"){                  // set high current value
            highCurrent = reqValue.toFloat();                  // is necessary a cast to convert a String to a float
            Serial.print("Set high_current to ");
            Serial.println(highCurrent);
            String respStatus = "OK";
            sendMaxHighCurrent(reqId, highCurrent, respStatus);

      }else{
            String errorValue = "ERROR";
            String respStatus = "NOT_OK";
            genResponse(reqId, errorValue, respStatus);
      }
   }

   //#### Info request ####//
   else if (reqType == "GET"){
      if (reqDetail == "relay" && reqValue == "STATUS"){                     // relay status
            String statusRelay = (digitalRead(pinRelay)==1)?"ON":"OFF";
            Serial.println("The Relay is "+statusRelay);
            String respStatus = "OK";
            genResponse(reqId, statusRelay, respStatus);

      }else if (reqDetail == "current" && reqValue == "STATUS"){             // current and power consumption status
            getCurrent();
            sendCurrent(reqId);

      }else if (reqDetail == "shutdown" && reqValue == "STATUS"){            // max current status
            Serial.print("Max_current is set to ");
            Serial.println(maxCurrent);
            String respStatus = "OK";
            sendMaxHighCurrent(reqId, maxCurrent, respStatus);

      }else if (reqDetail == "high_current" && reqValue == "STATUS"){        // high current status
            Serial.print("High_current is set to ");
            Serial.println(highCurrent);
            String respStatus = "OK";
            sendMaxHighCurrent(reqId, highCurrent, respStatus);

      }else{
            String errorValue = "ERROR";
            String respStatus = "NOT_OK";
            genResponse(reqId, errorValue, respStatus);
      }
   }
  }else{}
}



////_____ Generate a Response MQTT _____////

void genResponse(long int reqId, String returnValue, String respStatus){
    StaticJsonDocument<200> doc;
    JsonObject RESPONSE = doc.createNestedObject("RESPONSE");   // create a nested object
    RESPONSE["request_id"] = reqId;
    RESPONSE["return_value"] = returnValue;
    RESPONSE["return_status"] = respStatus;
    char resp_buff[200];                                // char buffer for upload the response Json to Mqtt pub
    serializeJson(doc, resp_buff);
    client.publish("/ABI_TIR/RES",resp_buff);
}
```

```
////_____ Send HIGH or MAX current values by MQTT _____////

void sendMaxHighCurrent(long int reqId, int returnValue, String respStatus){
    StaticJsonDocument<200> doc;
    JsonObject RESPONSE = doc.createNestedObject("RESPONSE");   // create a nested object
    RESPONSE["request_id"] = reqId;
    RESPONSE["return_value"] = returnValue;
    RESPONSE["return_status"] = respStatus;
    char resp_buff[200];                              // char buffer for upload the response Json to Mqtt pub
    serializeJson(doc, resp_buff);
    client.publish("/ABI_TIR/RES",resp_buff);
}


////_____ Send Current value by MQTT _____////

void sendCurrent(long int reqId){
    StaticJsonDocument<200> doc;
    JsonObject RESPONSE = doc.createNestedObject("RESPONSE");    // create a nested object
    RESPONSE["request_id"] = reqId;
    RESPONSE["current_value"] = Irms;
    RESPONSE["power_value"] = kWatt;
    RESPONSE["return_status"] = "OK";

    char curr_buff[200];                              // char buffer for upload the response Json to Mqtt pub
    serializeJson(doc, curr_buff);
    client.publish("/ABI_TIR/RES",curr_buff);
}


////________ Loop read and control Current value and call EVENT MQTT ________////

void readLoopCurrent(){
    getCurrent();
    if (Irms >= highCurrent && Irms < maxCurrent){
        Serial.println("NOTIFY");
        String type = "NOTIFY";
        String eventValue = "high_current";
        genAlarm(type, eventValue);
    }else if(Irms >= maxCurrent){
        Serial.println("ALARM");
        String type = "ALARM";
        String eventValue = "max_current";
        genAlarm(type, eventValue);
    }else{
    }
}


////_____ Reading Current value from A0 _____////

void getCurrent(){

    Irms = 0;
    kWatt = 0;
    ValADC = analogRead(A0);

    Vrms = (ValADC/1024)*diodsTensFall;
    Irms = (Vrms/20)*2000;
    kWatt = Irms*230/1000;

    Irms = round(Irms*1000)/1000;                     // Current round to the third decimal digit
    kWatt = round(kWatt*1000)/1000;                   // Power round to the third decimal digit
```

```
    Serial.println("");
    Serial.print(ValADC);
    Serial.println(" on A0 pin;");
    Serial.print(Vrms);
    Serial.print(" V;  ");
    Serial.print(Irms, 3);
    Serial.print(" A;  ");
    Serial.print(kWatt, 3);                    // Serial.print rounded float to 3 decimal digits
    Serial.println(" kW;");
    Serial.println("");
}


////_____ Generate an Alarm MQTT _____////

void genAlarm(String type, String eventValue){
    StaticJsonDocument<200> doc;
    JsonObject EVENTS = doc.createNestedObject("EVENTS");   // create a nested object
    EVENTS["event_type"] = type;
    EVENTS["notify_value"] = Irms;
    EVENTS["alarm_value"] = eventValue;
    char alarm_buff[200];                      // char buffer for upload the response Json to Mqtt pub
    serializeJson(doc, alarm_buff);
    client.publish("/ABI_TIR/EVENTS",alarm_buff);            // in a different topic: /EVENTS
}


////_______ reconnect to MQTT broker _____////

void reconnect() {
 while (!client.connected()) {                 // If MQTT connection lost
  digitalWrite(0, HIGH);                       // Loop until we're reconnected
  digitalWrite(2, HIGH);
  Serial.print("Attempting MQTT connection...");

  if (client.connect(clientName)) {            // Attempt to connect
   digitalWrite(0, LOW);                       // red Led if connected and server working
   Serial.println("connected");
   client.subscribe("/ABI_TIR/REQ");           // subscription to Topic
  } else {
   Serial.print("failed, rc=");
   Serial.print(client.state());
   Serial.println(" try again in 5 seconds");
   delay(5000);                                // Wait 5 seconds before retrying
  }
 }
}


////________ Server HTTP ________////

void runServer(){
    WiFiClient client = server.available();          // Check if a client has connected
    if (!client) {
     return;
    }

    Serial.println("new client");                // Wait until the client sends some data
    digitalWrite(0, HIGH);
    digitalWrite(2, LOW);
```

```
    String request = client.readStringUntil('\r');    // Read the first line of the request
    Serial.println(request);
    client.flush();
                                                                    // Match the request:
    if (request.indexOf("/Solenoid=ON") != -1)  {
      digitalWrite(pinRelay, HIGH);
      value = HIGH;
    }
    if (request.indexOf("/Solenoid=OFF") != -1)  {
      digitalWrite(pinRelay, LOW);
      value = LOW;
    }

                                                                     // Return the response
    client.println("HTTP/1.1 200 OK");
    client.println("Content-Type: text/html");
    client.println("");                                              //  do not forget this one
    client.println("<!DOCTYPE HTML>");
    client.println("<html>");

    client.println("<head>");
    client.println("<link href='
U7sALDAAABwfAAAoKwAAFxoAAE7i/
ACSkpIAP0UAAE9XAAAqLgAAOkAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAiIiGaIZoiIiIiIZohmiIiIiIhmiGaIiIiIiGaIZoiIiIiIZohmiIiIiAAAAA
AAiId3AAAAAAB3d3d1VVVVV3d6qqBQAAAKqqqqoFAAAAqqoREQUAAAAREZmZBQAAAJmZMzMFUAAAMzMzMzBVUAMzMyIiIgAA
IiIiREREREREREQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAA+B8AAP//AAA=' rel='icon' type='image/x-icon' />");
    client.println("<link href='
U7sALDAAABwfAAAoKwAAFxoAAE7i/
ACSkpIAP0UAAE9XAAAqLgAAOkAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAiIiGaIZoiIiIiIZohmiIiIiIhmiGaIiIiIiGaIZoiIiIiIZohmiIiIiAAAAA
AAiId3AAAAAAB3d3d1VVVVV3d6qqBQAAAKqqqqoFAAAAqqoREQUAAAAREZmZBQAAAJmZMzMFUAAAMzMzMzBVUAMzMyIiIgAA
IiIiREREREREREQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAA+B8AAP//AAA=' rel='shortcut icon' type='image/x-icon' />");
    client.println("<meta charset='utf-8'>");
    client.println("<meta name='viewport' content='width=device-width, initial-scale=1.0'>");
    client.println("<meta name='apple-mobile-web-app-capable' content='yes'>");
    client.println("<meta name='mobile-web-app-capable' content='yes'>");
    client.println("<title>Solenoid LED control</title>");
    client.println("</head>");

    client.println("<body  leftmargin='0' topmargin='0' style='background-color:black; height: -webkit-fill-available; font-family: -
apple-system, BlinkMacSystemFont, Segoe UI, Roboto, Oxygen-Sans, Ubuntu, Cantarell, Helvetica Neue, sans-serif;'>");
    client.println("<h1 style='color: #ff9900; size: 60px; margin-top:10px; margin-left: 10px; margin-right: 10px; margin-bottom:
10px; font-weight: 500; position: relative; text-align: center;'>Solenoid control</h1>");
    client.println("<div style='background: linear-gradient(to bottom, #000000 0%, #006666 100%); position: fixed; position: fixed;
top: 100px; right: 0px; left: 0px; padding: 20px; bottom: 0px;'>");
    client.print("<p style='color: white; size:6px; position: relative; margin-top: 30px; margin-right: 30px;'>Solenoid is
now: ");

    if(value == HIGH) {
      client.print("On");
    } else {
      client.print("Off");
    }

    client.println("</p>");
    client.println("<br><br>");
    client.println("<button onclick='powerOn()'>On </button>");
    client.println("<button onclick='powerOff()'>Off </button><br />");
    client.println("</div>");
    client.println("</body>");
```

```
    client.println("<script>");
    client.println("function powerOn(){ window.location.href = '/Solenoid=ON'; }");
    client.println("function powerOff(){ window.location.href = '/Solenoid=OFF'; }");
    client.println("</script>");

    client.println("<style>");
    client.println("button:hover, button:active { background-color: #ff9900; color: #f3f5f6;}");
    client.println("button { text-decoration: none; cursor: pointer; display: inline-block; background-color: #bb5300; color: #f3f5f6;
font-size:30px; position: relative; min-height: 60px; width: 120px; text-align: center; padding: 6px 12px; margin-top: 20px; margin-
right:10px; margin-left:10px; border-radius: 8px; font-weight: bold; text-transform: uppercase;}");
    client.println("</style>");
    client.println("</html>");

    //delay(10);
    Serial.println("Client disonnected");
    Serial.println("");
}

////////////////////////////////- LOOP -////////////////////////////////

void loop() {

  if (!client.connected()) {
      reconnect();
  }
  runServer();
  client.loop();

  if (millis() >= (timer + 10000)){
      readLoopCurrent();
      timer = millis();
  }

  if (timer > millis()){                    // if millis() reset to 0 milliseconds, this will reset timer too
      timer = millis();
  }

}
```