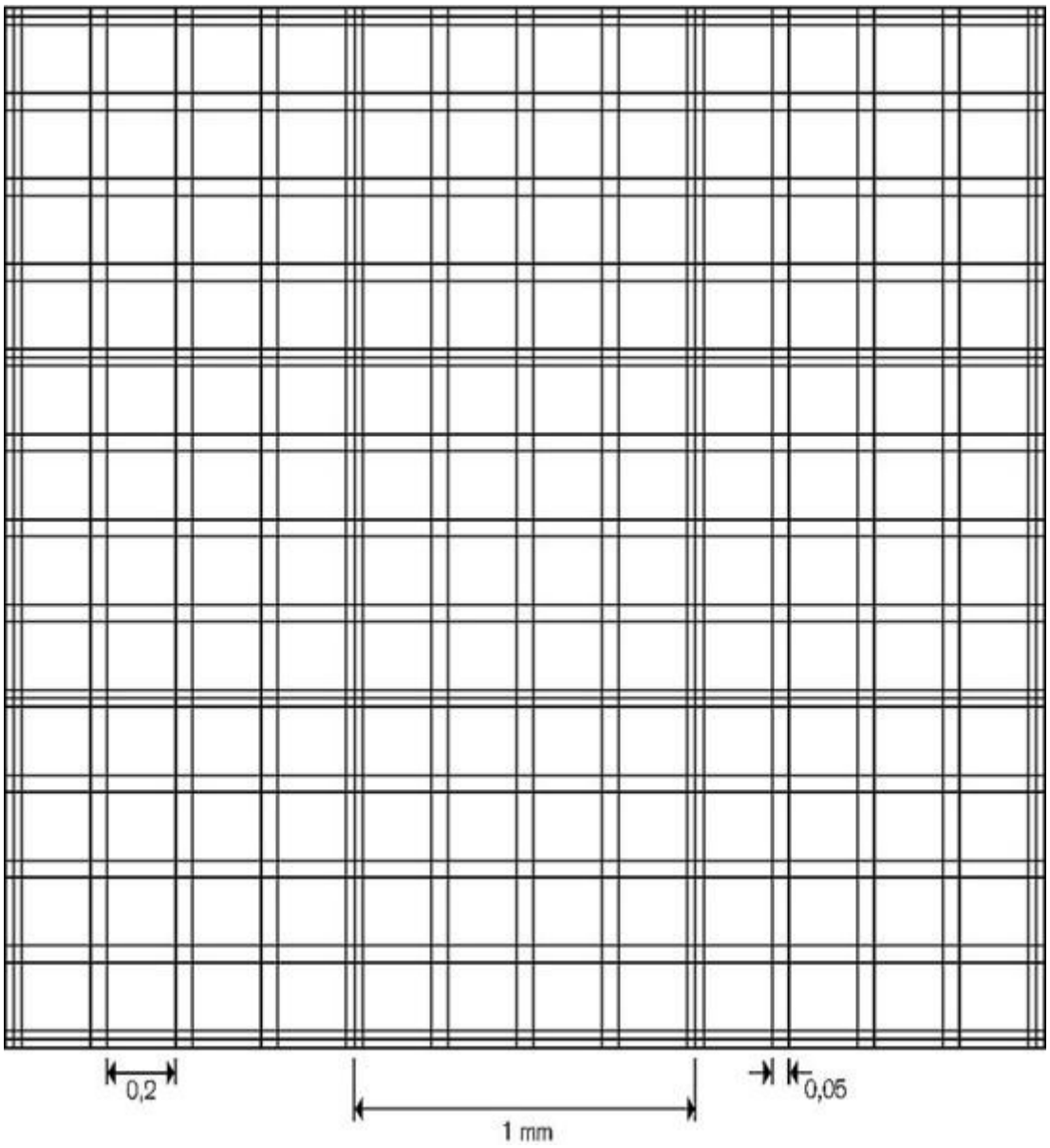
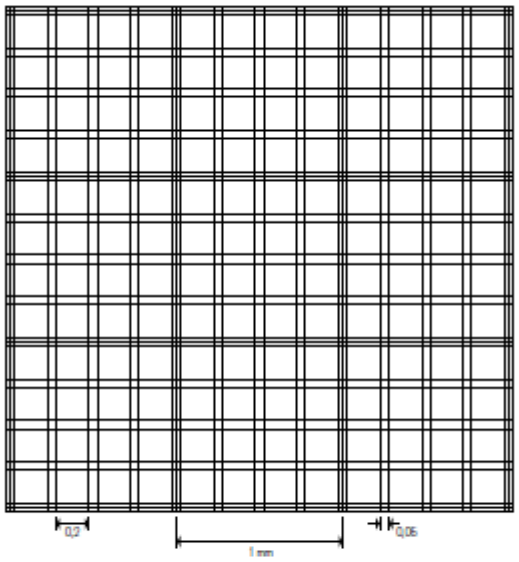
THC *Procambarus clarkii*

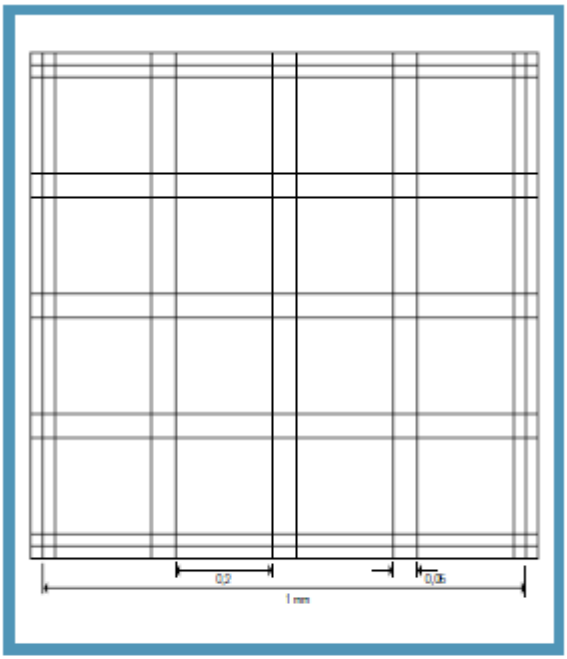


Counting chamber BLAUBRAND® Bürker pattern, with clips double rul.

Cat. No.: 718920
EAN: 4033378317803

Product description
The ruling shows 9 large squares of 1 mm^2 each. These are used for counting leucocytes. Each large square is subdivided by double lines (0.05 mm apart) into 16 group squares with 0.2 mm sides. The group squares correspond in size to the Neubauer counting chamber, but have no further subdivisions. They are used for counting thrombocytes and erythrocytes. The double lines from mini squares with an area of 0.0025 mm2.
CE-marked according to IVD-Directive 98/79 CE.





Bürker

Largo quadrato centrale

Bürker, doppio reticolo

La profondità della camera è di 0,1 mm. Il reticolo è formato da 9 larghi quadrati ciascuno con una superficie di 1 mm^2. Essi servono per il conteggio dei leucociti. Ogni largo quadrato è suddiviso da una doppia linea (distanti 0,05mm) in altri gruppi di 16 quadrati con lati di 0,2 mm. Questi corrispondono come formato al modello Neubauer, ma non hanno ulteriori suddivisioni. Sono usati per il conteggio dei trombociti ed eritrociti. Le doppie linee formano dei mini quadrati con un'area di 0,0025 mm^2.

Modello	Idoneo per certificazione ufficiale Codice	Ufficialmente certificato Codice
Senza pinzette	7189 05	7189 06
Con pinzette	7189 20	–

OBIETTIVO → CONTARE LE CELLULE CIRCOLANTI IN 1 MI DI EMOLINFA

1- riempire una siringa sterlie con 100 µl di anticoagulante sterile;
2- prelevare e immobilizzare l'animale, con una siringa da 1 mL (muovere lo stantuffo), dalla membrana articolare tra il 1° e il 2° tergite del pleon eseguire un

prelievo di 100µL – mescolare molto bene girando più volte la siringa;
3- togliere l'ago e mettere una goccia di emolinfa sulla scanalatura al centro della camera Bunker, il rimanente va lentamente estruso in un tubo da 1.5 mL e messa in ghiaccio;
4- eseguite le THC per ogni esemplare;
5- alla fine dei prelievi centrifugare i 10 tubi per 5' a 1000 RPM e con una P200 prelevare il plasma e trasferirlo in nuovi tubi da 1.5 mL sempre in ghiaccio;

Contando le cellule comprese in un quadrato 1 mm x 1 mm si hanno le cellule in 0.1 mm^3 o $1x10^{-4}$ cm^3 (0.1 µL).
Per avere le cellule per mL basta quindi moltiplicare per $1x10^4$ x 2 (anticoagulante).

Noi conteremo almeno 5 quadrati 0,25x0,25 mm (conta le cellule che toccano: ✓ bordo superiore ✓ bordo sinistro, non contare quelle che toccano: ✗ bordo inferiore ✗ bordo destro), poi mediamo e trasformiamo in cell/1 mm^2 ad ex → 20/5*16 = 64 poi 64*2*104 = 1.280.000 cell / mL

OBIETTIVO → VALUTARE LA COMPETENZA IMMUNITARIA

PROTOCOLLO PER pro-/FENOLOSSIDASI *Procambarus clarkii*

Attività della pro-/fenolossidasi emolinfatica

Preparazione reagenti

1. **Tampone fosfato PBS**

Disciogliere 1 tablet in 200 mL di acqua distillata, si ottiene una soluzione 0.01M=10mM. pH 7.4 a 25°C. Autoclavare.

2. **DL_DOPA 3 mg/mL in PBS**

Volume finale: 20 ml (180 µl x 96 pozzetti)
DL-DOPA non è solubile in acqua, quindi si discioglie 60 mg in 2 mL di acetone e si porta a volume con 18 mL tampone fosfato PBS.
Per MADOPAR® (100mg) → in 3 mL acetone e si porta a 33 mL di tampone fosfato PBS
Tenere in foglio di alluminio al buio!

3. **SDS**

Preparare una soluzione con 0.6 mg/mL di SDS in PBS.
Volume finale: almeno 30 µL x 96 pozzetti
→ 15 mg di SDS in 25 mL PBS

Attività pPO

- 10 µL di plasma sono miscelati a 10 µL di SDS (0,6 mg/mL) nel pozzetto della micro piastra su ghiaccio.
- Aggiungere 180 µL di DL-DOPA
- Si registra l'incremento lineare in assorbanza a 492 nm per un periodo di 30 min (o 60 minuti) ad intervalli di 5 min (o 10 min) a 25° C

Attività PO

- 20 µl di plasma nel pozzetto della micro piastra su ghiaccio.
- Aggiungere 180 µl di DL-DOPA
- Si registra l'incremento lineare in assorbanza a 492 nm per un periodo di 60 min ad intervalli di 2 min a 25° C (o 60 minuti)

L'attività enzimatica è espressa come au/min/µL (assorbance units)

REFS

Radha, Sankaranarayanan, Periasamy Mullainadhan, and Munusamy Arumugam. 2013. "Detection of Two Distinct Types of Hemolymphatic Prophenoloxidase and Their Differential Responses in the Black Tiger Shrimp, *Penaeus Monodon*, upon Infection by White Spot Syndrome Virus." Aquaculture 376–379 (February): 76–84. doi:10.1016/j.aquaculture.2012.11.017.

ELENCO MATERIALE E REAGENTI PER 10 *P. clarkii* CTRL e 10 SPERIMENTALI

- bilancia per pesare gli animali;
- pennarello indelebile per marcare gli animali;
- 2 vasche da 40L (20°C e 32°C);
- 40 siringhe da insulina con ago staccabile;
- 1 Burker;
- 40 tubi eppendorf 1.5 mL;
- 2 cestello ghiaccio fondente;
- 2 panetto -20°C;
- **1 pipetta multicanale da 200 µL;**
- **2 vaschette per pipetta multicanale;**
- 1 micropiastra da 96 wells;
- PBS 1x, 1 pastiglia Sigma in 200 mL acqua milliQ;
- SDS 0.6 mg/mL;
- L-DOPA 3 mg/mL;
- acetone.

SKETCH ESP32

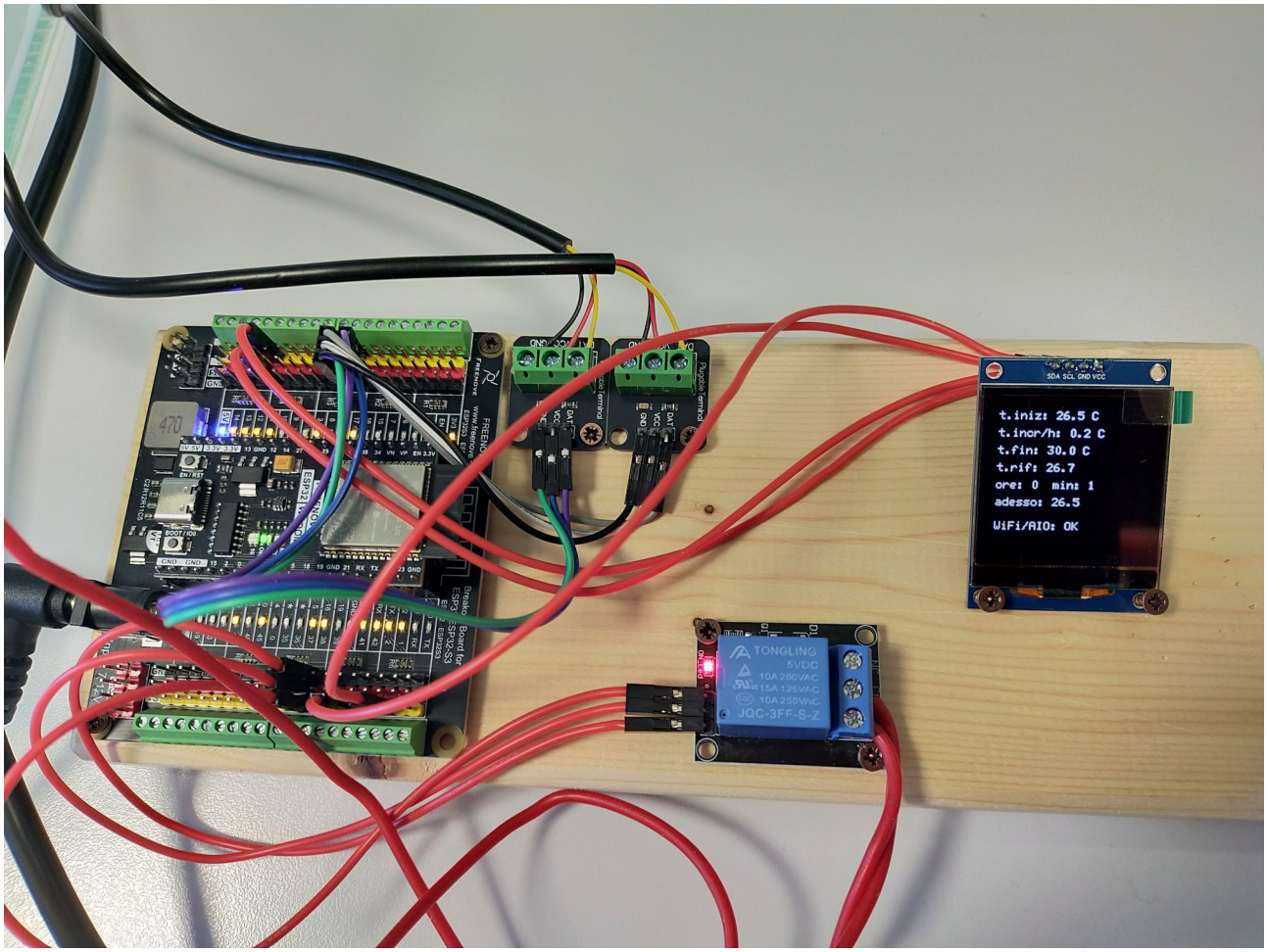
```
// ====================================
// Ondata di calore - ESP32 + 2x DS18B20
// Invio temperatura acquario ad Adafruit IO
// Relè attivo HIGH
// OLED SH1107 128x128 I2C
//
// Collegamenti:
// - DS18B20 sonda 1: DATA -> GPIO25
// - DS18B20 sonda 2: DATA -> GPIO26
// - Ogni DS18B20: resistenza pull-up 4.7 kOhm tra DATA e 3.3V
// - Relè riscaldatore: IN -> GPIO18
// - OLED SH1107 I2C: SDA -> GPIO21, SCL -> GPIO22
// - Alimentazione sensori: 3.3V e GND
// ====================================

#include "esp_task_wdt.h"
#include "esp_system.h"

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

#include <Wire.h>
#include <Adafruit_SH110X.h>
#include <U8g2_for_Adafruit_GFX.h>

// ----------------------------
// WIFI / ADAFRUIT IO
// ----------------------------
#include <WiFi.h>
#include "esp_wifi.h"
#include "AdafruitIO_WiFi.h"

// Credenziali Adafruit IO
#define IO_USERNAME "*******"
#define IO_KEY "*******"

// Rete locale WiFi
#define WIFI_SSID "IoThings"
#define WIFI_PASS "*******"

// Se la rete IoThings riconosce il dispositivo tramite MAC,
// puoi lasciare USE_CUSTOM_MAC a true.
// Se non ti serve, metti false.
#define USE_CUSTOM_MAC true

// MAC locale/unicast già usato nel tuo altro sketch.
// Deve essere applicato prima di io.connect().
uint8_t newMAC[6] = { 0xB6, 0xC5, 0xB6, 0x7C, 0x89, 0x88 };

AdafruitIO_WiFi io(IO_USERNAME, IO_KEY, WIFI_SSID,
WIFI_PASS);

// Feed Adafruit IO dove invio la temperatura finale dell'acquario.
// Deve esistere su Adafruit IO oppure verrà creato/gestito dalla
libreria.
AdafruitIO_Feed *temp_acq_lab = io.feed("temp_acq_lab");

// Invio ogni 15 minuti
const unsigned long AIO_SEND_INTERVAL_MS = 15UL * 60UL *
1000UL;

// Se Adafruit IO è offline, non stampo/tento a ogni loop ma al
massimo ogni minuto
const unsigned long AIO_RETRY_OFFLINE_MS = 60UL * 1000UL;

unsigned long lastAioSend = 0;
unsigned long lastAioAttempt = 0;
bool firstAioSendDone = false;

// ----------------------------
// WATCHDOG
// ----------------------------
#define WDT_TIMEOUT 40  // secondi

// ----------------------------
// PIN
// ----------------------------
// I2C principale per OLED SH1107
#define I2C_SDA 21
#define I2C_SCL 22

// DS18B20 su due bus OneWire separati
// GPIO25 e GPIO26 sono scelti per evitare GPIO2 LED onboard,
// GPIO16 RGB, e GPIO4/GPIO5.
#define ONE_WIRE_1 25
#define ONE_WIRE_2 26

// Relè attivo HIGH:
// LOW  = relè spento
// HIGH = relè acceso
const int RELAY_PIN = 18;

// ----------------------------
// DS18B20 su due bus separati
// ----------------------------
OneWire oneWire1(ONE_WIRE_1);
OneWire oneWire2(ONE_WIRE_2);

DallasTemperature sensor1(&oneWire1);
DallasTemperature sensor2(&oneWire2);

// ----------------------------
// OLED SH1107
// ----------------------------
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 128
#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C

Adafruit_SH1107 display(SCREEN_WIDTH, SCREEN_HEIGHT,
&Wire, OLED_RESET);
U8G2_FOR_ADAFRUIT_GFX u8g2;

// ----------------------------
// Parametri lettura temperatura
// ----------------------------
const int N_READS = 6;

// Se le due sonde differiscono più di 0.5 °C,
// uso la temperatura più alta e segnalo ALERT.
const float MAX_DIFF = 0.5f;

// Isteresi controllo relè
const float HYST = 0.2f;

// Tempo minimo tra due commutazioni del relè.
// Evita accensioni/spegnimenti troppo ravvicinati.
const unsigned long MIN_SWITCH = 30000UL;

// ----------------------------
// Programma setpoint ondata di calore
// ----------------------------
byte ora = 0;

// Temperatura iniziale: viene poi aggiornata con la prima lettura
valida
float temp_i = 20.0f;

// Incremento orario
float temp_d = 0.2f;

// Temperatura finale massima
float temp_f = 32.0f;

// Temperatura di riferimento corrente
float temp_r = temp_i + temp_d;

// Temperatura attuale finale/mediata
float temp_a = NAN;

// ----------------------------
// Timer programma
// ----------------------------
```

```
unsigned long temporif = 0;
unsigned long pausa = 60UL * 60UL * 1000UL;  // 1 ora
int minpass = 0;

// ------------------------------
// Stato relè
// ------------------------------
bool StatoRele = false;
unsigned long lastSwitch = 0;

// ------------------------------
// Display
// ------------------------------
unsigned long screenTimer = 0;
bool screenMode = false;

// ------------------------------
// Utilità: nome reset
// ------------------------------
const char *nomeReset(esp_reset_reason_t reason) {
  switch (reason) {
    case ESP_RST_UNKNOWN: return "UNKNOWN";
    case ESP_RST_POWERON: return "POWERON";
    case ESP_RST_EXT: return "EXT RESET";
    case ESP_RST_SW: return "SOFTWARE";
    case ESP_RST_PANIC: return "PANIC";
    case ESP_RST_INT_WDT: return "INT WDT";
    case ESP_RST_TASK_WDT: return "TASK WDT";
    case ESP_RST_WDT: return "WDT";
    case ESP_RST_DEEPSLEEP: return "DEEPSLEEP";
    case ESP_RST_BROWNOUT: return "BROWNOUT";
    case ESP_RST_SDIO: return "SDIO";
    default: return "UNDEFINED";
  }
}

// ------------------------------
// Utilità: controllo lettura DS18B20
// ------------------------------
static inline bool validDS18(float t) {
  if (isnan(t)) return false;
  if (t == DEVICE_DISCONNECTED_C) return false;  // -127
  if (t == 85.0f) return false;                  // valore spurio tipico
  if (t < -55.0f || t > 125.0f) return false;
  return true;
}

// Risoluzione 9 bit: conversione circa 94 ms.
// Uso 110 ms per margine.
static inline uint16_t convDelayMs9bit() {
  return 110;
}

// ------------------------------
// Applica MAC WiFi personalizzato
// ------------------------------
void applicaMACWiFi() {
  if (!USE_CUSTOM_MAC) return;

  WiFi.mode(WIFI_STA);
  WiFi.persistent(false);
  WiFi.setAutoReconnect(true);

  delay(100);

  // Nessuna connessione deve essere già attiva.
  WiFi.disconnect(false, false);
  delay(100);

  esp_err_t err = esp_wifi_set_mac(WIFI_IF_STA, newMAC);

  if (err == ESP_OK) {
    Serial.print("MAC WiFi impostato: ");
    Serial.println(WiFi.macAddress());
  } else {
    Serial.print("ERRORE impostazione MAC. Codice: ");
    Serial.println(err);
    Serial.print("MAC attuale: ");
    Serial.println(WiFi.macAddress());
  }
}

// ------------------------------
// Connessione ad Adafruit IO
// ------------------------------
void connectAdafruitIO() {
  applicaMACWiFi();

  Serial.print("Connessione WiFi/Adafruit IO a SSID: ");
  Serial.println(WIFI_SSID);

  Serial.print("MAC prima della connessione: ");
  Serial.println(WiFi.macAddress());

  io.connect();

  unsigned long t0 = millis();

  while (io.status() < AIO_CONNECTED && millis() - t0 < 30000UL) {
    io.run();
    Serial.print(".");
    delay(500);
  }

  if (io.status() >= AIO_CONNECTED) {
    Serial.println("\nAdafruit IO connesso");
    Serial.print("IP: ");
    Serial.println(WiFi.localIP());
    Serial.print("MAC in uso: ");
    Serial.println(WiFi.macAddress());
  } else {
    Serial.println("\nTimeout connessione Adafruit IO");
    Serial.print("WiFi.status = ");
    Serial.println(WiFi.status());
    Serial.print("IP = ");
    Serial.println(WiFi.localIP());
    Serial.print("MAC in uso = ");
    Serial.println(WiFi.macAddress());
  }
}

// ------------------------------
// Media di N letture valide.
// Ritorna NAN se nessuna lettura è valida.
// ------------------------------
float readMeanTemp(DallasTemperature &sens) {
  float sum = 0.0f;
  int ok = 0;

  for (int i = 0; i < N_READS; i++) {
    // NON bloccante perché in setup() imposto
setWaitForConversion(false)
    sens.requestTemperatures();

    delay(convDelayMs9bit());

    float t = sens.getTempCByIndex(0);

    if (validDS18(t)) {
      sum += t;
      ok++;
    }

    // Piccola pausa fra campioni
    delay(40);
  }

  if (ok == 0) return NAN;
  return sum / ok;
}

// ------------------------------
// Decide la temperatura finale.
// Se entrambe le sonde sono valide e vicine: media.
// Se sono valide ma troppo diverse: usa la più alta e attiva alarm.
// Se una sola è valida: usa quella.
// ------------------------------
void chooseTfinal(float T1, float T2, float &Tfinal, bool &alarm) {
  Tfinal = NAN;
  alarm = false;
```

```
  if (!isnan(T1) && !isnan(T2)) {
    float diff = fabsf(T1 - T2);

    if (diff <= MAX_DIFF) {
      Tfinal = (T1 + T2) / 2.0f;
    } else {
      Tfinal = (T1 > T2) ? T1 : T2;
      alarm = true;
    }

  } else if (!isnan(T1)) {
    Tfinal = T1;

  } else if (!isnan(T2)) {
    Tfinal = T2;
  }
}

// ------------------------------
// Stampa temperatura o ERR sul display
// ------------------------------
void printTempOrErr(float t, uint8_t decimals = 1) {
  if (!isnan(t)) {
    u8g2.print(t, decimals);
  } else {
    u8g2.print("ERR");
  }
}

// ------------------------------
// Invio temperatura acquario ad Adafruit IO
// ------------------------------
void inviaTemperaturaAdafruitIO(float Tfinal) {
  unsigned long nowMs = millis();

  // Invio:
  // - subito alla prima lettura valida
  // - poi ogni 15 minuti
  bool due = (!firstAioSendDone) || (nowMs - lastAioSend >=
AIO_SEND_INTERVAL_MS);

  if (!due) return;

  // Se offline, ritento al massimo ogni minuto
  if (lastAioAttempt > 0 && nowMs - lastAioAttempt <
AIO_RETRY_OFFLINE_MS) {
    return;
  }

  lastAioAttempt = nowMs;

  if (isnan(Tfinal)) {
    Serial.println("Adafruit IO: temperatura non valida, non invio");
    return;
  }

  if (io.status() >= AIO_CONNECTED) {
    Serial.print("Invio Adafruit IO temp_acq_lab = ");
    Serial.println(Tfinal, 2);

    temp_acq_lab->save(Tfinal);

    lastAioSend = nowMs;
    firstAioSendDone = true;

  } else {
    Serial.println("Adafruit IO non connesso: invio saltato");
  }
}

// ------------------------------
// Messaggio boot su display
// ------------------------------
void showBootInfoOnDisplay(const char *line1, const char *line2) {
  display.clearDisplay();

  u8g2.setFont(u8g2_font_6x12_tf);

  u8g2.setCursor(0, 18);
  u8g2.print(line1);

  u8g2.setCursor(0, 36);
  u8g2.print(line2);

  display.display();
}

// ------------------------------
// SETUP
// ------------------------------
void setup() {
  Serial.begin(115200);
  delay(200);

  // ------------------------------
  // Relè
  // ------------------------------
  // Relè attivo HIGH:
  // LOW = spento all'avvio, condizione sicura.
  pinMode(RELAY_PIN, OUTPUT);
  digitalWrite(RELAY_PIN, LOW);
  StatoRele = false;

  // ------------------------------
  // I2C + display
  // ------------------------------
  // SDA = GPIO21
  // SCL = GPIO22
  Wire.begin(I2C_SDA, I2C_SCL);
  Wire.setClock(100000);
  Wire.setTimeOut(50);

  if (!display.begin(SCREEN_ADDRESS, true)) {
    Serial.println("Display SH1107 NON trovato");
  } else {
    Serial.println("Display SH1107 OK");
  }

  display.clearDisplay();
  display.display();

  u8g2.begin(display);
  u8g2.setForegroundColor(SH110X_WHITE);
  u8g2.setBackgroundColor(SH110X_BLACK);

  // ------------------------------
  // Reset reason
  // ------------------------------
  esp_reset_reason_t rr = esp_reset_reason();

  Serial.print("BOOT reset_reason = ");
  Serial.print((int)rr);
  Serial.print(" - ");
  Serial.println(nomeReset(rr));

  showBootInfoOnDisplay("BOOT OK", nomeReset(rr));
  delay(1200);

  // ------------------------------
  // DS18B20
  // ------------------------------
  sensor1.begin();
  sensor2.begin();

  // 9 bit: più veloce, sufficiente per controllo termico dinamico.
  sensor1.setResolution(9);
  sensor2.setResolution(9);

  // Fondamentale: evita blocchi lunghi dentro
requestTemperatures().
  sensor1.setWaitForConversion(false);
  sensor2.setWaitForConversion(false);

  // ------------------------------
  // WiFi / Adafruit IO
  // ------------------------------
  connectAdafruitIO();

  // ------------------------------
  // Lettura iniziale setpoint
  // ------------------------------
  float T1 = readMeanTemp(sensor1);
  float T2 = readMeanTemp(sensor2);
```

```
  // Doppia lettura per evitare letture parassite iniziali.
  delay(1000);

  T1 = readMeanTemp(sensor1);
  T2 = readMeanTemp(sensor2);

  float Tfinal;
  bool alarm;

  chooseTfinal(T1, T2, Tfinal, alarm);

  if (!isnan(Tfinal)) {
    temp_i = Tfinal;
    temp_r = fminf(Tfinal + temp_d, temp_f);
  }

  Serial.print("Temperatura iniziale = ");
  Serial.println(temp_i, 2);

  Serial.print("Setpoint iniziale = ");
  Serial.println(temp_r, 2);

  // ------------------------------
  // Watchdog
  // ------------------------------
  esp_task_wdt_config_t wdt_config = {
    .timeout_ms = WDT_TIMEOUT * 1000,
    .idle_core_mask = (1 << 0) | (1 << 1),
    .trigger_panic = true
  };

  esp_err_t wdt_err = esp_task_wdt_init(&wdt_config);

  if (wdt_err == ESP_OK) {
    Serial.println("WDT init OK");
  } else if (wdt_err == ESP_ERR_INVALID_STATE) {
    Serial.println("WDT gia' inizializzato dal core: continuo");
  } else {
    Serial.print("Errore init WDT: ");
    Serial.println(wdt_err);
  }

  wdt_err = esp_task_wdt_add(NULL);

  if (wdt_err == ESP_OK) {
    Serial.println("WDT add loop OK");
  } else if (wdt_err == ESP_ERR_INVALID_STATE) {
    Serial.println("Task loop gia' registrato nel WDT: continuo");
  } else {
    Serial.print("Errore add loop al WDT: ");
    Serial.println(wdt_err);
  }

  temporif = millis();
  screenTimer = millis();

  showBootInfoOnDisplay("SETUP COMPLETO", "Avvio controllo");
  delay(1000);
}

// ------------------------------
// LOOP
// ------------------------------
void loop() {
  esp_task_wdt_reset();

  // Mantiene viva la connessione Adafruit IO.
  // La libreria gestisce anche eventuali riconnessioni.
  io.run();

  // ------------------------------
  // 1) Lettura sonde
  // ------------------------------
  float T1 = readMeanTemp(sensor1);

  esp_task_wdt_reset();
  io.run();

  float T2 = readMeanTemp(sensor2);

  esp_task_wdt_reset();
  io.run();

  // ------------------------------
  // 2) Temperatura finale
  // ------------------------------
  float Tfinal;
  bool alarm;

  chooseTfinal(T1, T2, Tfinal, alarm);
  temp_a = Tfinal;

  // ------------------------------
  // 3) Invio temperatura acquario ad Adafruit IO
  // ------------------------------
  inviaTemperaturaAdafruitIO(Tfinal);

  // ------------------------------
  // 4) Timer ore/minuti programma termico
  // ------------------------------
  minpass = (millis() - temporif) / 60000UL;

  if (ora < 100 && (millis() - temporif > pausa)) {
    temporif = millis();
    ora++;

    temp_r += temp_d;
    if (temp_r > temp_f) temp_r = temp_f;
  }

  // ------------------------------
  // 5) Alterna schermata ogni 5 secondi
  // ------------------------------
  if (millis() - screenTimer > 5000UL) {
    screenTimer = millis();
    screenMode = !screenMode;
  }

  // ------------------------------
  // 6) Display
  // ------------------------------
  display.clearDisplay();

  if (!screenMode) {
    // Schermata principale: temperatura attuale
    u8g2.setFont(u8g2_font_logisoso24_tf);

    if (!isnan(Tfinal)) {
      u8g2.setCursor(0, 32);
      u8g2.print(Tfinal, 1);
      u8g2.print(" C");
    } else {
      u8g2.setCursor(10, 32);
      u8g2.print("ERR");
    }

    u8g2.setFont(u8g2_font_6x10_tf);

    u8g2.setCursor(0, 55);
    u8g2.print("T1: ");
    printTempOrErr(T1, 1);

    u8g2.setCursor(0, 67);
    u8g2.print("T2: ");
    printTempOrErr(T2, 1);

    u8g2.setCursor(0, 79);
    u8g2.print("T.rif: ");
    u8g2.print(temp_r, 1);

    if (alarm) {
      u8g2.setCursor(85, 67);
      u8g2.print("ALERT");
    }

    u8g2.setCursor(0, 96);
    u8g2.print("AIO: ");
    u8g2.print(io.status() >= AIO_CONNECTED ? "OK" : "OFF");

    u8g2.setCursor(0, 110);
    u8g2.print("RELE: ");
    u8g2.print(StatoRele ? "ON" : "OFF");

  } else {
    // Schermata programma
```

```
u8g2.setFont(u8g2_font_6x12_tf);

u8g2.setCursor(0, 18);
u8g2.print("t.iniz: ");
u8g2.print(temp_i, 1);
u8g2.print(" C");

u8g2.setCursor(0, 33);
u8g2.print("t.incr/h: ");
u8g2.print(temp_d, 1);
u8g2.print(" C");

u8g2.setCursor(0, 48);
u8g2.print("t.fin: ");
u8g2.print(temp_f, 1);
u8g2.print(" C");

u8g2.setCursor(0, 63);
u8g2.print("t.rif: ");
u8g2.print(temp_r, 1);

u8g2.setCursor(0, 78);
u8g2.print("ore: ");
u8g2.print(ora);
u8g2.print("  min: ");
u8g2.print(minpass);

u8g2.setCursor(0, 93);
u8g2.print("adesso: ");
printTempOrErr(temp_a, 1);

if (alarm) {
  u8g2.setCursor(0, 114);
  u8g2.print("!!! ALERT SONDE !!!");
} else {
  u8g2.setCursor(0, 114);
  u8g2.print("WiFi/AIO: ");
  u8g2.print(io.status() >= AIO_CONNECTED ? "OK" : "OFF");
  }
 }

 display.display();

 // ------------------------------
 // 7) Controllo relè riscaldatore
 // ------------------------------
 // Sicurezza:
 // se non ho temperatura valida, spengo il relè.
 if (!isnan(Tfinal)) {
   if (!StatoRele && (Tfinal < (temp_r - HYST)) && (millis() -
lastSwitch > MIN_SWITCH)) {

     StatoRele = true;
     digitalWrite(RELAY_PIN, HIGH);
     lastSwitch = millis();

     Serial.println("RELE ON");

   } else if (StatoRele && (Tfinal > (temp_r + HYST)) && (millis() -
lastSwitch > MIN_SWITCH)) {

     StatoRele = false;
     digitalWrite(RELAY_PIN, LOW);
     lastSwitch = millis();

     Serial.println("RELE OFF");
   }

 } else {
   StatoRele = false;
   digitalWrite(RELAY_PIN, LOW);
 }

 esp_task_wdt_reset();

 // Aggiornamento non troppo lento.
 delay(500);
}
```


SDA SCL GND VCC
t.iniz: 26.5 C
t.incr/h: 0.2 C
t.fin: 30.0 C
t.rif: 26.7
ore: 0 min: 1
adesso: 26.5
WiFi/AIO: OK
TONGLING
5VDC
10A 250VAC
15A 125VAC
10A 250VAC
JQC-3FF-S-Z

SCRIPT R x THC

```
# source("THC.R")
# install.packages("cowplot")

library("cowplot")
library(ggplot2)
library(ggpubr)

thc<-read.table("THC.csv", dec = ".", header = TRUE, sep = "$", quote = "&", na.strings = "NA")

thc0<-thc[thc$tempo=="T0",]

thc1<-thc[thc$tempo=="T1",]

t1 <- ggboxplot(thc, "trattamento", "THC", ggtheme = theme_light(base_size=10), add = "jitter", xlab = "temperatura", ylab = "cells/µL")

t2 <- ggboxplot(thc, "tempo", "THC", ggtheme = theme_light(base_size=10), add = "jitter", xlab = "tempo (h)", ylab = "cells/µL")

t3 <- ggboxplot(thc0, "trattamento", "THC", ggtheme = theme_light(base_size=10), add = "jitter", xlab = "temperatura", ylab = "cells/µL")

t4 <- ggboxplot(thc1, "trattamento", "THC", ggtheme = theme_light(base_size=10), add = "jitter", xlab = "temperatura", ylab = "cells/µL")

plot_grid(t1, t2, t3, t4, labels = c("Tot", "Tot", "T0", "T1"), nrow = 2, ncol = 2, label_size = 12)

ggsave("THC.pdf", plot = last_plot(), device = "pdf")

# confronto tempo 0, parametrico
t.test(thc0$THC ~ thc0$trattamento)

# confronto tempo 1, parametrico
t.test(thc1$THC ~ thc1$trattamento)

# confronto tempo 0, non parametrico
wilcox.test(thc0$THC ~ thc0$trattamento)

# confronto tempo 1, non parametrico
wilcox.test(thc1$THC ~ thc1$trattamento)

# confronto tempi, parametrico
t.test(thc$THC ~ thc$tempo)

# confronto tempi, non parametrico
wilcox.test(thc$THC ~ thc$tempo)
```

SCRIPT R x pPO

```
# source("pPO.R")
# install.packages("cowplot")

library("cowplot")
library(ggplot2)
library(ggpubr)

PO<-read.table("26PO.csv", dec = ".", header = TRUE, sep = "$", quote = "&", na.strings =
"NA")

pPO0<-26PO[26PO$tempo=="T0",]

pPO1<-26PO[26PO$tempo=="T1",]

t1 <- ggboxplot(26PO, "trattamento", "pPO", ggtheme = theme_light(base_size=10), add =
"jitter", xlab = "temperatura", ylab = "cells/µL")

t2 <- ggboxplot(26PO, "tempo", "pPO", ggtheme = theme_light(base_size=10), add =
"jitter", xlab = "tempo (h)", ylab = "cells/µL")

t3 <- ggboxplot(pPO0, "trattamento", "pPO", ggtheme = theme_light(base_size=10), add =
"jitter", xlab = "temperatura", ylab = "cells/µL")

t4 <- ggboxplot(pPO1, "trattamento", "pPO", ggtheme = theme_light(base_size=10), add =
"jitter", xlab = "temperatura", ylab = "cells/µL")

plot_grid(t1, t2, t3, t4, labels = c("Tot", "Tot", "T0", "T1"), nrow = 2, ncol = 2, label_size =
12)

ggsave("pPO.pdf", plot = last_plot(), device = "pdf")

# confronto tempo 0, parametrico
t.test(pPO0$pPO ~ pPO0$trattamento)

# confronto tempo 1, parametrico
t.test(pPO1$pPO ~ pPO1$trattamento)

# confronto tempo 0, non parametrico
wilcox.test(pPO0$pPO ~ pPO0$trattamento)

# confronto tempo 1, non parametrico
wilcox.test(pPO1$pPO ~ pPO1$trattamento)

# confronto tempi, parametrico
t.test(26PO$pPO ~ 26PO$tempo)

# confronto tempi, non parametrico
wilcox.test(26PO$pPO ~ 26PO$tempo)
```