# Séances 8 et 9 : chauffage d'appoint pour les abeilles “BeeWarm”

## Introduction :

Pendant les séances 8 et 9, j’ai travaillé sur la mise en place du système de chauffage d’appoint pour la ruche. J’ai développé et structuré le programme de l’ESP32 afin de gérer la mesure de température, la commande des résistances par PWM et la surveillance de la puissance via le INA219. L’objectif était d’obtenir un fonctionnement chérent et prêt à être testé après la fabrication du circuit imprimé.

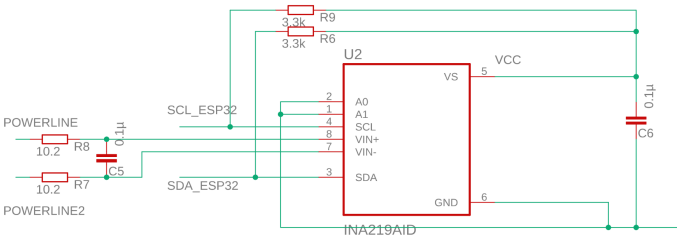
### 1) Proposition du code avec régulation thermique par PWM :

Pour rappel, la mesure de température est réalisée à l’aide de capteurs DS18B20 connectés sur un bus OneWire. L’initialisation se fait avec les bibliothèques OneWire et DallasTemperature, puis le programme détecte automatiquement les capteurs présents. À chaque cycle de la boucle principale, une requête de mesure est envoyée et la température de la ruche est stockée dans la variable correspondante.

```
// Lire la valeur du capteur de temperature
sensors.requestTemperatures(); // envoie la command pour avoir les temperatures
// affiche les donnees en temperature
tVeg = 0.0;
tRuche = 0.0;
for(int i=0;i<numberOfDevices; i++) {
  // Search the wire for address
  if(sensors.getAddress(tempDeviceAddress, i)){
    Serial.print("Capteur: ");
    Serial.println(i,DEC);
    float tempC = sensors.getTempC(tempDeviceAddress);
    if (tempC == DEVICE_DISCONNECTED_C) {
      Serial.println("Capteur de temp deconnecte !");
      return;
    }
    if (memcmp(tempDeviceAddress, capteurRuche, 8) == 0) {
      tRuche = tRuche + tempC;
    }
    else {
      tVeg = tVeg + tempC;
    }
    Serial.print(tempC);
    Serial.println("°C");
  }
}
tVeg = tVeg/(numberOfDevices-numberOfCapteurRuche); // moyenne des mesures de température pour la végétaline
```

Cette valeur est ensuite comparée à la température de consigne définie dans le programme.

La supervision énergétique est assurée par le capteur INA219, connecté en I²C sur les broches SDA (GPIO 21) et SCL (GPIO 22). le INA219 mesure la tension aux bornes d'une résistance shunt placée en série avec l'alimentation des résistances chauffantes. À partir de cette chute de tension, il calcule le courant traversant la charge, puis la puissance consommée.




Dans le code, les fonctions ***ina219.getCurrent_mA()*** et ***ina219.getPower_mW()*** permettent de récupérer ces grandeurs. Cette mesure permet de surveiller la consommation réelle du chauffage et d'éviter une surcharge ou une dérive de puissance. Elle pourra également être utilisée pour limiter la puissance maximale disponible selon les contraintes énergétiques du système.

```
String message = "DATA;" + String(tRuche, 1) + "°C;" + String(tVeg, 1) + "°C";
SerialBT.println(message);
Serial.println(message);

// ----- Lecture INA219 -----
float shuntVoltage = ina219.getShuntVoltage_mV();
float busVoltage   = ina219.getBusVoltage_V();
float current_mA   = ina219.getCurrent_mA();
float power_mW     = ina219.getPower_mW();

Serial.print("Bus Voltage:   "); Serial.print(busVoltage);   Serial.println(" V");
Serial.print("Shunt Voltage: "); Serial.print(shuntVoltage); Serial.println(" mV");
Serial.print("Current:       "); Serial.print(current_mA);   Serial.println(" mA");
Serial.print("Power:         "); Serial.print(power_mW);     Serial.println(" mW");
Serial.println("-------------------------------------");
```

Le contrôle du chauffage repose sur la génération d'un signal PWM par l'ESP32. Le module matériel LEDC intégré permet de produire un signal rectangulaire dont le rapport cyclique est ajustable.

Les fonctions ***ledcSetup()***, ***ledcAttachPin()*** et ***ledcWrite()*** permettent respectivement de configurer la fréquence et la résolution du PWM, d'associer un canal interne à une broche physique (SIGNAL1 et SIGNAL2), puis de fixer le duty cycle.

Le signal PWM est appliqué à la grille des MOSFETs de puissance, qui agissent comme des interrupteurs électroniques rapides. Lorsque la grille reçoit un signal haut, le MOSFET devient conducteur et le courant traverse la résistance chauffante ; lorsqu'elle est basse, le courant est coupé.

La régulation thermique repose sur le calcul de l'écart entre la température mesurée et la consigne. Pour cela, j'ai utilisé l'hystérésis pour activer le chauffage en dessous d'un seuil et le couper au-dessus d'un autre.

La configuration du PWM est réalisée dans le **setup()**. Deux canaux matériels sont utilisés :

```
// PWM (LEDC)
ledcSetup(ch1, pwm_freq, pwm_res);
ledcAttachPin(SIGNAL1, ch1);

ledcSetup(ch2, pwm_freq, pwm_res);
ledcAttachPin(SIGNAL2, ch2);

ledcWrite(ch1, 0);
ledcWrite(ch2, 0);
```

La fonction ***ledcSetup()*** définit la fréquence du signal PWM (1000 Hz) ainsi que la résolution (8 bits). Une résolution de 8 bits signifie que le rapport cyclique peut varier de 0 à 255. La fréquence de 1 kHz est adaptée à une charge résistive, car elle est suffisamment élevée pour éviter toute variation thermique perceptible. La fonction ***ledcAttachPin()*** relie chaque canal PWM interne à une broche physique (SIGNAL1 et SIGNAL2). Ces broches sont connectées aux grilles des MOSFETs qui pilotent les résistances chauffantes.

Dans la boucle principale, le rapport cyclique est déterminé par la variable ***duty***. La fonction ***ledcWrite()*** applique la valeur du duty cycle au canal PWM correspondant. Si duty = 0, le MOSFET reste bloqué et aucune puissance n'est envoyée aux résistances. Si duty = 255, la sortie reste à l'état haut en permanence, ce qui correspond à une puissance maximale. Une valeur intermédiaire produit une puissance moyenne proportionnelle.

```
int duty = chauffe ? puissanceDeChauffe : 0;
ledcWrite(ch1, duty);
ledcWrite(ch2, duty);

Serial.print("Etat chauffe: ");
Serial.print(chauffe ? "ON" : "OFF");
Serial.print(" | duty=");
Serial.println(duty);

delay(3000);
```

Remarque : éventuellement, une approche plus avancée consiste à appliquer une loi proportionnelle dans laquelle le rapport cyclique du PWM est proportionnel à l'erreur thermique. Plus l'écart est important, plus la puissance envoyée aux résistances est élevée.

Le calcul de la valeur **puissanceDeChauffe** est effectué à partir de la puissance mesurée par le INA219. Après, la variable booléenne **chauffe** est déterminée par une régulation à hystérésis. Si la température est inférieure à la consigne moins la bande de tolérance, le chauffage est activé. Si elle dépasse la consigne plus la bande, il est désactivé. Cette méthode évite les commutations rapides autour de la consigne.

```
const int MPPT_mW = 4000; // puissance max dispo estimée (à ajuster après tests PCB)
int puissanceDeChauffe = (int)((power_mW * 255.0) / MPPT_mW);

if (puissanceDeChauffe < 0) puissanceDeChauffe = 0;
if (puissanceDeChauffe > 255) puissanceDeChauffe = 255;

Serial.print("PWM base (0-255) calcule: ");
Serial.println(puissanceDeChauffe);

// ----- Hystérésis température -----
if (tRuche < (temperatureConsigne - bande)) {
  chauffe = true;
} else if (tRuche > (temperatureConsigne + bande)) {
  chauffe = false;
}
```

Cette version du code est une première base de fonctionnement. Les réglages comme la fréquence du PWM, la bande d'hystérésis, la gestion de la puissance ou encore les seuils de température devront être ajustés après la fabrication et les premiers tests du circuit imprimé. Des essais en conditions réelles permettront de vérifier que le chauffage réagit correctement, que la température reste stable et que la consommation électrique est adaptée. Ces tests serviront à affiner les paramètres pour obtenir un système fiable et efficace.

## Conclusion :

En conclusion, ces séances ont permis de finaliser la partie logicielle du système de chauffage et de valider le principe de régulation thermique par PWM ainsi que la supervision énergétique via le INA219. Prochainement, nous entamerons la phase de conception du layout du circuit imprimé.