ZORGANE Yassin
ETTOUHFY Imane
ELSE 4 – Semestre 8

Systèmes Embarqués & Communicants
Polytech Nice-Sophia Année

COMPTE RENDU DE PROJET

Module : Projet électronique

Thème : Création et développement d'une installation thermique sur la ruche

ZORGANE Yassin
ETTOUHFY Imane
ELSE 4 – Semestre 8

Table des matières

I. Présentation du projet 3
II. Les tâches effectuées 3
Mise en place de la carte électronique 3
2.1 Prise en main du PCB 3
2.2 Prise en main de l'ESP 32 4
2.3 Test de la maquette – Capteur de température 4
2.3 Test de la maquette – Ventilateurs 5
2.3 Test de la maquette – Résistance 7
2.3 Test de la maquette – Utilisation de deux coeurs 10
III. Les problèmes rencontrés 11
Dysfonctionnement de la carte ESP32 11
Problème de capteurs de température 11
Fonctionnement efficace d'un seul côté de la maquette 11
IV. Ce qu'il reste à faire 11
Conclusion 12
Points positifs 12
Points négatifs 12

ZORGANE Yassin
ETTOUHFY Imane
ELSE 4 – Semestre 8

I. Présentation du projet

Le projet thermique, initié par le professeur M. Peter, vise à aider ses abeilles en contrôlant la température de la ruche. Les abeilles peuvent supporter des températures allant jusqu'à 50°C, tandis que les varroas ne survivent pas au-delà de 38°C et les frelons ne résistent pas au-delà de 44°C. Ce contrôle de la température est crucial pour la santé de la ruche.

Pour ce projet, une maquette a été utilisée, comportant deux côtés, avec quatre résistances THS50, trois capteurs de température DS1820B, et trois ventilateurs.

II. Les tâches effectuées

Mise en place de la carte électronique

L'objectif principal était de suivre le plan électrique pour positionner chaque composant sur la carte.

L'installation de la carte électronique a impliqué plusieurs étapes critiques :

- **Placement des composants** : Les résistances, les capteurs et d'autres composants ont été positionnés sur la carte selon le schéma électrique fourni.
- **Refusion** : La carte a été placée dans un four de refusion pour fusionner les composants à sa surface, assurant une adhérence solide sans nécessiter de soudure préliminaire.
- **Soudage** : Les composants restants, y compris les connexions critiques et les composants plus grands, ont été soudés manuellement pour garantir leur bon fonctionnement.

2.1 Prise en main du PCB

La prise en main du PCB (Printed Circuit Board) a consisté à installer et vérifier tous les composants sur la carte électronique. Cela a permis de s'assurer que chaque composant était correctement connecté et opérationnel.

2.2 Prise en main de l'ESP 32

Le projet utilisait l'ESP32 pour le contrôle et la communication avec les capteurs et actuateurs. Cela incluait :

- **Prise en main de la maquette / carte** : Installation de huit résistances chauffantes, six capteurs de température et six ventilateurs. Le but était de rétablir la connexion entre la maquette et une carte avec des transistors pour communiquer avec l'ESP32.

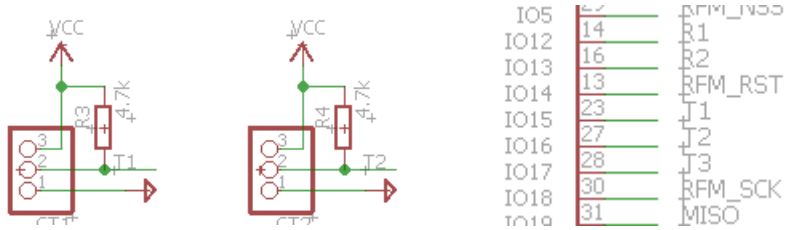
- **Utilisation de trois capteurs de température DS18B20 simultanément** : Test de la fiabilité et de la précision des capteurs. Les capteurs ont été câblés correctement sur la plaque d'essai et testés pour leur précision en les exposant à la même température ambiante et en enregistrant les lectures simultanément.

2.3 Test de la maquette – Capteur de température

```
#include <DallasTemperature.h>
// UTILISER PIN 15 16 17 POUR RELEVER LA TEMPERATURE

// GPIOs where the DS18B20 sensors are connected to
const int oneWireBus1 = 15; // Connect the first DS18B20 sensor to pin 15
//const int oneWireBus2 = 16; // Connect the second DS18B20 sensor to pin 16


// Configurer les instances OneWire pour la communication avec les capteurs
OneWire oneWire(oneWireBus1);
DallasTemperature sensors(&oneWire);
```

Le programme utilise un microcontrôleur Arduino pour interagir avec des capteurs de température DS18B20 via le protocole 1-Wire. Pour cela, il s'appuie sur deux bibliothèques externes : OneWire et DallasTemperature. La bibliothèque OneWire permet la communication avec les capteurs via le protocole 1-Wire, tandis que la bibliothèque DallasTemperature fournit des fonctions pour gérer les capteurs de température spécifiques.

La broche à laquelle est connecté le bus 1-Wire est définie comme constante ONE_WIRE_BUS. Dans ce programme, elle est configurée sur la broche 15, mais vous pouvez la modifier en fonction de la broche que vous utilisez sur votre carte Arduino.



En l'occurrence ici, on remarque sur le schéma de la carte qu'il s'agit bel et bien de la broche 15 de l'ESP 32.

La classe OneWire est utilisée pour créer un objet permettant la communication avec les capteurs en utilisant la broche définie précédemment. Cette classe fait partie de la bibliothèque OneWire.

Ensuite, la classe DallasTemperature est instanciée avec l'objet OneWire créé précédemment. Cette classe gère la communication avec les capteurs DS18B20 via le bus 1-Wire et fournit des fonctions pour lire les températures des capteurs.

La fonction setup () est exécutée une seule fois au démarrage du programme. Elle initialise la communication série à une vitesse de 115200 bits par seconde à l'aide de Serial.begin(115200). De plus, elle initialise la communication avec les capteurs en appelant sensors.begin().

```
void loop() {
  // Lecture de la requete pour les deux bus
  sensors.requestTemperatures();
  //sensors2.requestTemperatures();


  // Read temperatures from both sensors
  float temperatureC1 = sensors.getTempCByIndex(0);
  float temperatureC2 = sensors.getTempCByIndex(1);
  float temperatureC3 = sensors.getTempCByIndex(2);
  float temperatureC4 = sensors.getTempCByIndex(3);
  float temperatureC5 = sensors.getTempCByIndex(4);
  float temperatureC6 = sensors.getTempCByIndex(5);



  // Print temperature readings for the first sensor
  Serial.print("Capteur 1: ");
  Serial.print(temperatureC1);
  Serial.print("°C");
```

La fonction loop() est la boucle principale du programme, elle est exécutée en continu après l'exécution de la fonction setup(). À chaque itération de la boucle, le programme demande la conversion des températures sur tous les capteurs connectés en appelant sensors.requestTemperatures().

Ensuite, il lit la température de chaque capteur en utilisant la fonction sensors.getTempCByIndex(), où l'index correspond à l'identifiant du capteur sur le bus 1-Wire.

Les températures lues sont ensuite affichées sur le moniteur série via Serial.print() et Serial.println(), avec une étiquette indiquant le numéro de chaque capteur.

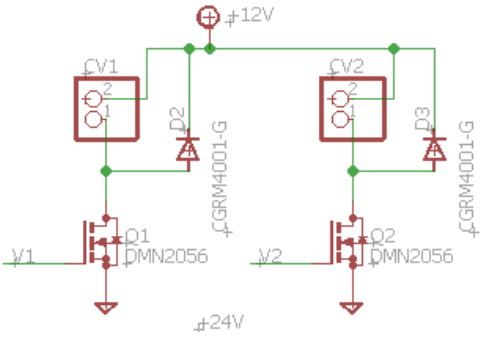
Un délai d'une seconde (delay(1000)) est ajouté à la fin de la boucle pour éviter des lectures trop fréquentes des capteurs.

2.3 Test de la maquette – Ventilateurs

Le second programme que nous avons réalisé permet de contrôler deux ventilateurs connectés à un Arduino. Voici une explication détaillée :

ZORGANE Yassin
ETTOUHFY Imane
ELSE 4 – Semestre 8

- Définition des broches : Les broches utilisées pour contrôler les ventilateurs sont définies en tant que constantes. Dans ce programme, le premier ventilateur est contrôlé par la broche 2 et le deuxième ventilateur par la broche 4. Ces broches ont été choisis en fonctions que celles déclarées sur le schéma.



- Configuration des broches en sortie : Dans la fonction setup(), les broches des ventilateurs sont configurées en tant que sorties à l'aide de la fonction pinMode(). Cela permet de spécifier que ces broches seront utilisées pour envoyer des signaux de commande aux ventilateurs.

- Initialisation de la communication série : La communication série est initialisée à une vitesse de 250000 bits par seconde à l'aide de Serial.begin(250000). Cela permet de communiquer avec un moniteur série pour le débogage ou pour afficher des informations.

-

```
// Définition des broches
const int ventilateurPin1 = 2; // Broche de commande du pre
const int ventilateurPin2 = 4; // Broche de commande du deu

void setup() {
  // Configuration des broches des ventilateurs en sortie
  pinMode(ventilateurPin1, OUTPUT);
  pinMode(ventilateurPin2, OUTPUT);
  Serial.begin(250000);
}
```

- Boucle principale : La fonction loop() est exécutée en continu après la fonction setup(). Dans cette boucle :

    - Les deux ventilateurs sont activés en envoyant un signal HIGH aux broches de contrôle correspondantes à l'aide de digitalWrite(). Cela les fait fonctionner à pleine puissance.
    - Un message "Ca tourne" est envoyé au moniteur série avec Serial.print() pour indiquer que les ventilateurs sont en marche.
    - Un délai de 5 secondes (delay(5000)) est introduit pour maintenir les ventilateurs allumés pendant cette durée.
    - Ensuite, les deux ventilateurs sont désactivés en envoyant un signal LOW à leurs broches de contrôle, ce qui les arrête.

```
void loop() {
  // Faire fonctionner les deux ventilateurs à pleine puissance
  digitalWrite(ventilateurPin1, HIGH); // Allumer le premier ventilate
  digitalWrite(ventilateurPin2, HIGH); // Allumer le deuxième ventilat
  Serial.print("Ca tourne");
  delay(5000); // Garder les ventilateurs allumés pendant 5 secondes

  // Arrêter les deux ventilateurs
  digitalWrite(ventilateurPin1, LOW);
  digitalWrite(ventilateurPin2, LOW);
  delay(2000); // Attendre 2 secondes avant de recommencer
```

Ce programme assure ainsi un contrôle basique des ventilateurs en les faisant fonctionner à pleine puissance pendant une période déterminée, puis en les arrêtant pour une courte période, et en répétant ce cycle indéfiniment.

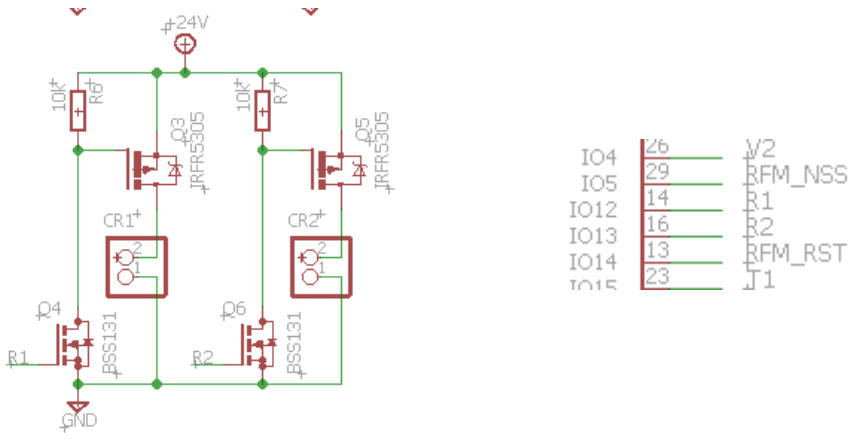
On remarque sur le schéma que les ventilateurs sont alimentés en 12V, d'où l'utilisation sur la carte de transistor permettant cette alimentation, car la carte elle sera alimenté en 30V.

2.3 Test de la maquette – Résistance

Le troisième programme utilise un contrôleur PID (Proportional-Integral-Derivative) pour maintenir la température à un certain niveau en contrôlant deux résistances chauffantes. Voici une explication détaillée :

- Inclusion des bibliothèques : Le programme inclut la bibliothèque PID_v1.h pour le contrôleur PID et la bibliothèque DallasTemperature.h pour la gestion des capteurs de température DS18B20.

- Définition des broches : Les broches utilisées sont définies comme constantes. relayPin et relayPin2 sont les broches de commande des résistances chauffantes, et oneWireBus1 est la broche des capteurs de température. On remarque sur le schéma qu'il s'agit des pins 12 et 13.



- Paramètres PID : Les constantes PID (Kp, Ki, Kd) sont définies pour régler les coefficients proportionnel, intégral et dérivé du contrôleur. setpoint est la température de consigne en degrés Celsius.

- Initialisation des objets PID : Un objet PID est créé avec les adresses des variables d'entrée (input) et de sortie (output), ainsi que la température de consigne et les paramètres PID définis précédemment.

```
// Constantes PID
double setpoint = 35.0; // Température de consigne en degrés
double input, output;
double Kp = 68.11, Ki = 0.054, Kd = 0; // Paramètres du PID

// Initialisation des objets PID
PID myPID(&input, &output, &setpoint, Kp, Ki, Kd, DIRECT);
//register thermistor(thermistorPin);
```

- Initialisation dans la fonction setup() :
  - Le port série est initialisé pour afficher les données de température et de sortie PID.
  - Les broches de commande des résistances chauffantes sont configurées en sortie.
  - La communication avec les capteurs de température est démarrée.

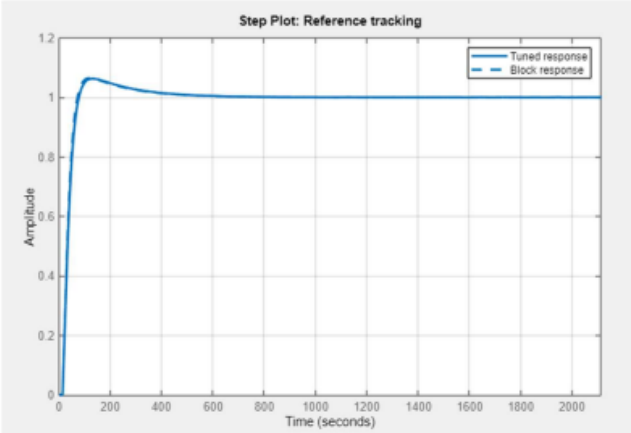
- Boucle principale loop() :
  - Les températures des capteurs sont demandées à l'aide de sensors.requestTemperatures().
  - La sortie du PID est calculée à l'aide de myPID.Compute(), en utilisant la différence entre la température mesurée (input) et la température de consigne (setpoint).
  - Les valeurs de température mesurée et de sortie PID sont affichées sur le moniteur série.
  - Les résistances chauffantes sont contrôlées en fonction de la sortie du PID :
    - Si la sortie est supérieure à zéro, les résistances chauffantes sont allumées.
    - Sinon, les résistances chauffantes sont éteintes.
  - Un court délai (delay(1000)) est introduit pour espacer les mesures.
  -

```
// Contrôle de la résistance chauffante en fonction de la sortie du PI
if (output > 0) {
  digitalWrite(relayPin, HIGH); // Allumer la résistance chauffante
  digitalWrite(relayPin2, HIGH); // Allumer la résistance chauffante 2

} else {
  digitalWrite(relayPin, LOW); // Éteindre la résistance chauffante
  digitalWrite(relayPin2, LOW); // Allumer la résistance chauffante 2

}
```

ZORGANE Yassin
ETTOUHFY Imane
ELSE 4 – Semestre 8

Ce programme permet ainsi de maintenir la température autour de la température de consigne en ajustant la puissance des résistances chauffantes en fonction de la différence entre la température mesurée et la température de consigne.

Il faut noter que les paramètres du PID ont été choisis grâce à une simulation effectué sur MATLAB. En effet, ayant déjà travailler sur ce concept il y a quelques années, nous avions les résultats d'une simulation d'une précédente plaque chauffante à disposition. Cette plaque devait aussi faire l'objet d'une régulation et nous avions utilisé un PID ajustable par la méthode d'Autotuning sur MATLAB. C'est-à-dire que nous personnalisons la courbe que l'on souhaite avoir et MATLAB nous donne les paramètres pour l'obtenir.



Bien entendu, le résultat n'est pas parfait lorsque l'on passe de la simulation à l'essai réel, nous conseillons donc la méthode par essai erreur, afin de tester plusieurs valeurs de manière pratique pour en tirer le meilleur fonctionnement.

On peut cependant commencer à travailler avec ces paramètres.

ZORGANE Yassin
ETTOUHFY Imane
ELSE 4 – Semestre 8

2.3 Test de la maquette – Utilisation de deux coeurs

Le quatrième programme utilise la bibliothèque Multicore pour exécuter des tâches sur différents cœurs du microcontrôleur ESP32. Voici une explication détaillée :

```
#include <Arduino.h>
#include "Multicore.h"

const int greenLedPin = 26; // Pin de la LED verte
const int blueLedPin = 27; // Pin de la LED bleue

// Fonction à exécuter sur le cœur 0
void core0Task(void * parameter) {
  pinMode(blueLedPin, OUTPUT);
  while(true) {
    digitalWrite(blueLedPin, HIGH); // Allumer la LE
    delay(500);
    digitalWrite(blueLedPin, LOW); // Éteindre la LE
    delay(500);
  }
}

// Fonction à exécuter sur le cœur 1
void core1Task(void * parameter) {
  pinMode(greenLedPin, OUTPUT);
  while(true) {
    digitalWrite(greenLedPin, HIGH); // Allumer la L
    delay(1000);
    digitalWrite(greenLedPin, LOW); // Éteindre la L
    delay(1000);
  }
}
```

- Inclusion des bibliothèques : Le programme inclut les bibliothèques nécessaires, notamment la bibliothèque Arduino.h pour les fonctions Arduino de base et la bibliothèque Multicore.h pour la gestion des tâches sur plusieurs cœurs.

- Définition des broches : Les broches utilisées pour les LED sont définies comme constantes. greenLedPin est la broche de la LED verte et blueLedPin est la broche de la LED bleue.

- Fonctions des tâches par cœur :
  - core0Task() : Une fonction qui sera exécutée sur le cœur 0. Elle initialise la broche de la LED bleue en sortie, puis allume et éteint la LED bleue en boucle avec un délai de 500 millisecondes entre chaque état.
  - core1Task() : Une fonction qui sera exécutée sur le cœur 1. Elle initialise la broche de la LED verte en sortie, puis allume et éteint la LED verte en boucle avec un délai de 1000 millisecondes entre chaque état.

- Initialisation dans la fonction setup() :
  - La bibliothèque Multicore est initialisée avec Multicore::init().
  - Les tâches sont lancées sur chaque cœur avec Multicore::launchCoreTask(). La fonction core0Task est lancée sur le cœur 0 et la fonction core1Task est lancée sur le cœur 1. Aucun paramètre supplémentaire n'est passé (NULL), et les priorités des tâches ne sont pas spécifiées (0 par défaut).

- Boucle principale loop() : La boucle principale peut être vide car les tâches sont gérées par les cœurs individuels. Dans ce cas, la boucle loop() est inutilisée car les tâches sur les cœurs sont autonomes et s'exécutent en parallèle.

Ce programme permet ainsi de tirer parti de la capacité multi-cœur du microcontrôleur ESP32 pour exécuter des tâches simultanément sur différents cœurs, ce qui est utile pour des applications nécessitant un traitement parallèle ou une réactivité accrue. Ce qui est le cas dans notre cas de figure

où il sera intéressant de contrôler les deux côtés de la maquette de manière parallèle. Nous avons jusqu'ici uniquement testé les Leds pour vérifier si le multicore fonctionne.

Nous avons commencé à implémenter tous les autres programmes de sorte à contrôler les résistances et ventilateurs en fonction de la température relevé par les capteurs sur le multicore mais aucun résultat n'a été concluant pour l'instant.

III. Les problèmes rencontrés

Dysfonctionnement de la carte ESP32

La carte ESP32 a dû être réinitialisée et plusieurs tests ont été effectués pour assurer son bon fonctionnement. Ce dysfonctionnement a causé des retards dans l'avancement du projet.

Problème de capteurs de température

Lors de tests, un problème a été rencontré où le deuxième capteur montrait une valeur stable alors que les autres capteurs réagissaient aux changements de température. Cela était dû à l'utilisation d'un seul bus pour six capteurs, nécessitant une réadaptation du code pour récupérer correctement les adresses des capteurs.

Fonctionnement efficace d'un seul côté de la maquette

Ce problème persiste et nécessite une solution pour assurer un fonctionnement équilibré des deux côtés de la maquette. Il est nécessaire d'améliorer la conception et le câblage pour garantir le bon fonctionnement des deux côtés de la maquette.

IV. Ce qu'il reste à faire

- Assembler les tests individuels pour une intégration complète.
- Vérifier l'alimentation de la maquette pour éviter des courts-circuits.
- Isoler correctement les câblages pour améliorer la fiabilité.
- Finaliser les tests des ventilateurs et des résistances avec des paramètres PID optimisés.
- Résoudre les problèmes restants pour assurer un fonctionnement efficace des deux côtés de la maquette.

Conclusion

Points positifs

- Plaisir et satisfaction à avancer le projet.
- Développement de nombreuses solutions innovantes.
- Liberté et flexibilité dans le travail.

Points négatifs

- Retards accumulés en début de projet.
- Obstacles techniques rencontrés tout au long du projet.

Ce rapport décrit les étapes clés, les défis rencontrés et les tâches restantes pour le projet thermique visant à contrôler la température dans une ruche pour aider les abeilles du professeur M. Peter.