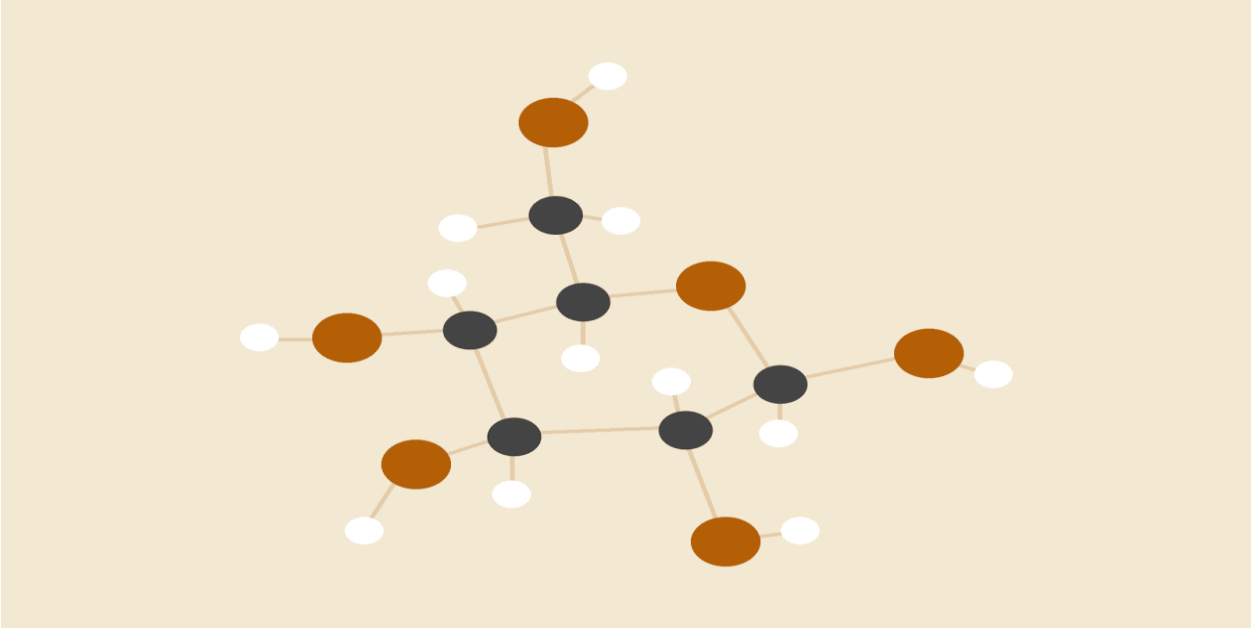
Université du Québec en outaouais

DEPARTEMENT DE L'INFORMATIQUE

COURS INF1633 : PROGRAMMATION DE SYSTEMES EMBARQUES EN C/C++

RAPPORT DU PROJET
DE RÉGULATION DE TEMPÉRATURE EMBARQUÉ



Présenté par TRESOR MEGANE TAMBAT (TAMT79360604)

Sous la supervision de Mustapha Bennai

lundi le 23 décembre 2024

UQO
UNIVERSITÉ
DU QUÉBEC
EN OUTAOUAIS

AVANT-PROPOS

Ce projet a été réalisé dans le cadre du cours **INF1633 - Programmation de Systèmes Embarqués en C/C++** sous la supervision de Monsieur **Mustapha Bennaï**.

Le code du projet a été réalisée par **TRESOR MEGANE TAMBAT** et **NYAMA KOUMBA, AUDE GUYSLIAH MAËL.**

Il porte sur le développement d'un système de surveillance et de régulation de température, intégrant la programmation en c, de l'électronique appliquée, et des communications série via le protocole I2C.

À travers ce travail, l'objectif principal a été de concevoir et implémenter un système embarqué capable de lire et traiter des donné du capteur BMP280 a l'aide de notre carte FRDMKL28Z et de l'afficher sur l'ecran LCD 2004A

Il a nécessité :

- **Utilisation des protocoles de communication I2C** pour interfacer le capteur BMP280 et l'écran LCD.
- **Gestion des GPIO (General Purpose Input/Output)** pour contrôler les LED et le moteur.
- **Programmation en temps réel** à l'aide du **module SysTick** pour gérer les délais
- **Intégration de transistors** pour piloter le moteur, assurant une commande sûre et la protection
- **Manipulation des bibliothèques C/C++** pour la lecture et le traitement des données provenant du capteur.

Ce travail a également mis en lumière l'importance de l'électronique dans les systèmes embarqués, notamment avec l'utilisation de **transistors** pour gérer la commande de dispositifs comme le moteur, tout en protégeant les circuits sensibles contre les surcharges électriques.

Remerciements

Je tiens à exprimer ma gratitude à Monsieur **Mustapha Bennai** pour son accompagnement et les directives technique qui nous a permis de réaliser ce projet, Karin le charger de travails dirigés pour ses multiples conseils ainsi que les Chargers de laboratoire Ali et Abdoukarim pour la mise à disposition du matériel de travail et le branchement des périphériques sur la tension approprie.

Je tiens également à remercier mes amis Loïc, Mohamed, Nathan, Daniel et bien d'autre dont les échanges et conseils nous ont permis de mener à bien ce travail

Table des matières

INTRODUCTION

Domaine monde actuel, le control de température est devenu crucial dans le sens ou la bonne gestion de la température est nécessaire pour palier à de nombreux problèmes. Ce projet vise à concevoir un prototype de système embarqué, permettant de contrôler la température de manière autonome dans un milieu .il sera question pour nous de réguler la température, et d'indiquer les différents états du système à travers les leds, et d'activer un ventilateur en cas de surchauffe. Ce projet est d'une grande nécessité car il nous permet non seulement d'apprendre comment programmer un système embarquer en c mais également d'apprendre comment adapter les éléments mis à notre disposition pour pouvoir effectuer nos taches.

2. Objectifs

Les objectifs du projet sont les suivants :

- Lire les données de température à partir du capteur BMP280 avec une précision fiable.
- Afficher les données sur un écran LCD de manière claire et réactive.
- Indiquer l'état de la température à l'aide de LEDs.
- Activer un moteur via un transistor en cas de surchauffe (supérieur à 50°C).
- Assurer une communication fluide entre les composants via le protocole I2C.

3. Matériels et Logiciel

Matériels

- **Carte FRDM-KL28Z** : cette carte est utilisée comme notre microcontrôleur dans ce projet. Elle est équipée d'un processeur arm cortex-M0+ qui a pour caractéristique une faible consommation d'énergie. Cette carte est idéale pour ce projet car elle est compatible avec plusieurs langages de programmation tel que le C, et elle dispose des périphériques comme les GPIO, l'I2C… qui nous permet de connecter votre carte avec des interfaces de communication.
- **Capteur BMP280** : Capteur combiné de température et de pression barométrique, communiquant via le protocole I2C, Il est utilisé pour surveiller en continu la température. Il a été choisi pour sa facilité d'utilisation avec notre microcontrôleur.
- **LCD 2004A** : Écran LCD avec interface I2C pour afficher les données.
- **LEDs (rouge, verte, bleue)** : utiliser comme indicateur visuel de l'état du système.
- **Moteur** : Activé via un transistor pour indiquer une surchauffe.
- **Transistor** : Agit comme un interrupteur électronique pour contrôler le moteur, garantissant une protection contre les courants excessifs.
- **Résistances** : Limitent le courant pour protéger les LEDs.

Logiciel

- **IDE : MCUXpresso** : Environnement de développement intégré pour programmer et déboguer la carte FRDM-KL28Z.

4. Méthodologie

Analyse et Planification

- **Lecture des besoins** : Identifier les données nécessaires (température) et les actions associées (affichage, activation des LEDs et du moteur).
- **Design modulaire** : Diviser le système en fonctions indépendantes pour la lecture, l'affichage et le contrôle.
- **Câblage et protection** : Utiliser un transistor pour activer le moteur et des résistances pour protéger les LEDs.

Description de la solution

Notre but est de programmer un système capable de mesurer la température à l’aide d’un capteur, d’utiliser un microcontrôleur pour lire la température et indiquer l'état du système grâce au leds puis activer un ventilateur si la température est en dessous du seuil.

- Spécifications du système.

- Configuration des broches
- Initialisation des leds
- Initialisation du ventilateur
- Lecture en continue de la température

Si la température est supérieure au seuil, allumer la led rouge, allumer le ventilateur, allumer la led verte.

Si non éteindre la led rouge, arrêter le ventilateur, éteindre la led verte et allumer la led bleu.

- Schéma fonctionnel détaillé. :

Le capteur de température : a pour rôle de mesurer la température, et d’envoyer les données à notre microcontrôleur.

Le microcontrôleur utiliser pour lire la température.

L’unité de contrôle situer dans le microcontrôleur a pour rôle de vérifier la température lue, et d’activer ou de désactiver les led et le ventilateur.

- Configuration logiciel:

- Nous avons sélectionné aléatoirement des registres sur notre carte (D0, D1, D2, D3) et nous les avons configurés dans l’option « configTools »de notre IDE en mode I2C.
- Après la configuration de l’horloge, nous l’avons donné une fréquence dans « ocloks » permettant la bonne synchronisation du transfert de nos donnes.
- Nous avons également configuré les pins du SDA et du SCL en mode i2c pour permettre le branchement physique sur notre carte et effectuer la communication en utilisant le protocole i2c.

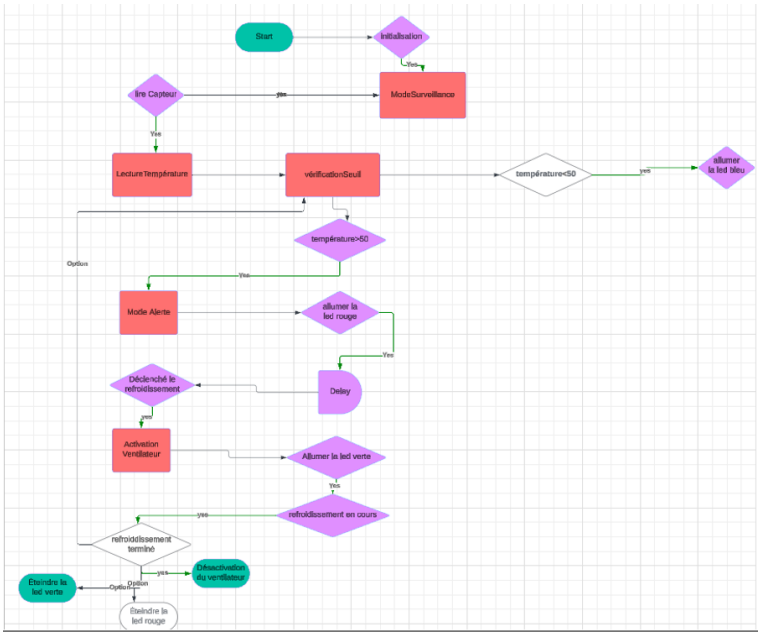
Remarque : la fréquence de notre microcontrôleur est de 96MH. Cette fréquence est supérieure à celle des autres composant. Il est important de donner une fréquence a notre horloge pour pouvoir s’assurer que l’information a été envoyer, réceptionner et analyser par l’autre dispositif.

Également utiliser la fonction PRINTF pour afficher un message à l’écran après chaque action (initialisation, fin de l’initialisation, valeur reçu valeur calculer) pour s’assurer que l’action désirer a été effectuer avec succès. Ceci est l’action principale qui nous a permis de debugger notre code.

La lecture de la documentation du dispositif que nous utilisons est importante parce qu’elle nous permet de connaitre les particularités de celui-ci.

Organigrammes :

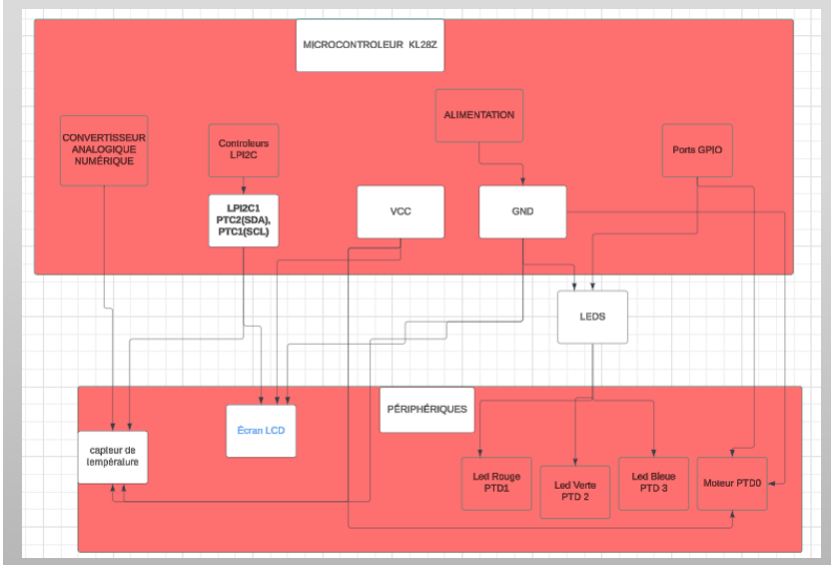
- Initialisation du système.
- Lecture de la température à l’aide du capteur.
- Vérification de la valeur de la température.
- État de la led en fonction de la valeur de la température.
- Action (éteindre ou allumer le ventilateur).
- État de la led après l’action.
- Boucle infinie de régulation de la température.



Schémas bloc :

Capteur de température -> microcontrôleur -> unité de contrôle -> led -> ventilateur

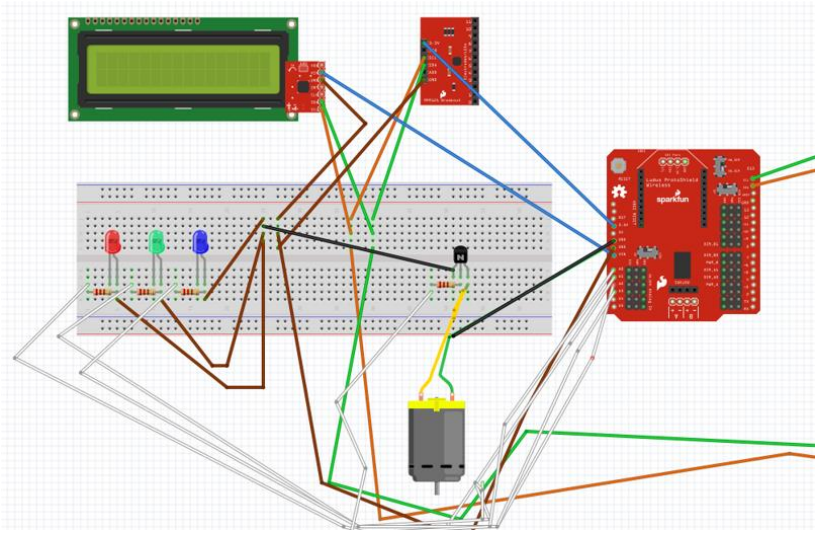
- Le bus de donne SDA connecte au PTC1et l'horloge SCL connecte sur le PTC0 sont en parallèle à tous nos dispositifs pour pouvoir assurer le bon transfert de donnes et une communication synchrone.
- Nous avons utilisé les ports D0, D1, D2, D3 pour les led et le moteur.
- Le moteur est connecté au vcc 5v de notre carte.
- Le VCC du capteur est connecté au vcc 3.3v de notre carte.
- Le VCC du LCD est connecté au vin de notre carte.
- Le GND est connecter en parallèle à tous les dispositifs de notre prototype (LCD, BMP, LEDS, MOTEURS).



Schémas de connexion :

Les connexions de nos diffèrent composant ont été fait judicieusement :

Nous avons choisi des couleurs particulière (marron : GND, vert : SDA, orange : SCL, bleu : tension électrique, blanc : leds physique, noir(pour distinguer de manière particulière le branchement du moteur) pour chaque fil pour pouvoir mieux repérer les erreurs de branchement.



5. Description Détaillée des Fonctions et Bibliothèques

Bibliothèques

- **fsl_lpi2c.h** : Configure et gère les communications via I2C entre la carte FRDM-KL28Z, le BMP280 et l'écran LCD.
- **bmp280.h** : Fournit des fonctions pour :
    1. Initialiser le capteur.
    2. Lire les données brutes.
    3. Appliquer des calculs de compensation pour obtenir la température.
- **LiquidCrystal_I2C.h** : Permet de contrôler l'écran LCD pour afficher du texte de manière ciblée et éviter les clignotements.
- **fsl_gpio.h** : Utilisée pour configurer les GPIO pour activer les LEDs et envoyer des signaux de commande au transistor.
- **fsl_debug_console.h** : Affiche des messages de débogage utiles pour surveiller les valeurs et diagnostiquer les problèmes.

Fonctions Principales

- **BMP280_ReadTemperature ()** :
    1. **Rôle** : Lire la température en interrogeant le BMP280 via I2C.
    2. **Détails** :
        1. Utilise des fonctions de lecture I2C pour récupérer les données brutes.
        2. Convertit les données en degrés Celsius grâce aux fonctions de compensation.
- **print_temperatureAt (int x, int y, float temperature)**:
    1. **Rôle** : Afficher la température sur l'écran LCD à une position spécifique.
    2. **Détails** :
        1. Positionne le curseur en fonction de (x, y).
        2. Convertit la température en texte formaté avant de l'afficher.
- **process_temperature()** :
    1. **Rôle** : Gérer la logique principale du système.
    2. **Détails** :

1. Lit la température via BMP280_ReadTemperature.
2. Active les LEDs et le moteur selon les seuils définis.
3. Met à jour l'affichage LCD.

- **activation_alerte()** :
    1. **Rôle** : Activer le moteur via le transistor et allumer la LED rouge en cas de surchauffe.
    2. **Détails** :
        1. Envoie un signal à la base du transistor pour activer le moteur.
        2. Allume la LED rouge pour signaler une alerte.
- **desactivation_alerte()** :
    1. **Rôle** : Désactiver le moteur et remettre les LEDs à l'état normal.
    2. **Détails** :
        1. Éteint la LED rouge et le moteur.
        2. Allume la LED bleue pour indiquer un retour à la normale.
- **SysTick_Handler ()** :
    1. **Rôle** : Gérer les délais pour garantir le fonctionnement en temps réel.
    2. **Détails** :
        1. Décrémente un compteur pour synchroniser les actions périodiques.
- **LPI2C1_Init ()** :
    1. **Rôle** : Configurer le bus I2C pour communiquer avec le capteur et l'écran LCD.
    2. **Détails** :
        1. Initialise les broches SDA et SCL.
        2. Définit la fréquence de communication.
- **Hardware_Init()** :
    1. **Rôle** : Initialiser tous les composants matériels.
    2. **Détails** :

1. Configure les GPIO pour les LEDs et le moteur.
2. Active les horloges nécessaires.

6. Résultats

Problèmes :

Le tableau suivant présente les problèmes et les solutions rencontrer dans notre projet

Catégorie	difficulté	Solution
Difficultés Techniques	• Problèmes de communication I2C : erreurs de lecture/écriture avec le capteur BMP280	• Consultation des datasheets pour valider l'I2C, les broches et l'horloge.
	• Affichage LCD instable : clignotements lors de la mise à jour des valeurs	• - Mettre à jour la température uniquement si la valeur change pour réduire les clignotements.
Erreurs de Branchement	• Mauvais branchement du BMP	• - Vérification des branchements et choix des couleurs de câbles pour limiter les confusions.
Configuration Incorrecte	• Configuration Incorrecte	- Lecture des **notes de cours** et des datasheets pour comprendre les connexions et configurations requises.
Utilisation des Bibliothèques	Mauvaise compréhension des bibliothèques utilisées	• Analyse des documentations et notes de cours pour maîtriser les bibliothèques.
	Mauvaise utilisation des fonctions propres à l'IDE (e.g., printf/PRINTF)	- Familiarisation avec l'IDE pour maîtriser les fonctions fournies.
Problèmes liés à l'Affichage	Difficulté d'afficher la température en décimal sur l'écran	• - .Affichage des valeurs avec les notes de cours et conversion en décimal si besoin.
Problèmes de Débogage	Difficulté d'identifier la source des erreurs	• - Débogage via l'affichage des valeurs attendues dans le terminal.
Activation des Composants	Difficulté à activer le moteur	• -Vérification des branchements et ajout de résistances pour protéger les composants.

Tests et Observations

- **Lecture des températures**
- **Affichage** : Stable et sans clignotement.
- **LEDs et moteur** : Fonctionnent correctement selon les seuils définis.
- Exemple :

  - À 25 °C : LED bleu allumer et le moteur éteint.

  - À 30°C : LED rouge allumer, LED verte allumer et moteur en marche.

Le système conçu permet de régulariser la température

Les alertes se déclenche respectivement en fonction de la valeur de la température prédéfini

Conclusion

Les objectifs du projet ayant été atteint, nous avons proposé un système embarqué fiable, réactif et capable de surveiller la température ambiante. Grâce à l'intégration du capteur BMP280 pour la collecte des données, de l'écran LCD 2004A pour l'affichage, et des leds pour fournir un retour visuel clair, le système gère efficacement les seuils critiques de température. De plus, l'utilisation de transistors pour activer le moteur garantit une commande sécurisée et protège les circuits sensibles contre les surcharges électriques.

Ce projet met également en avant l'interaction entre les aspects matériels et logiciels des systèmes embarqués, en s'appuyant sur des techniques avancées de programmation en C. L'intégration des protocoles I2C pour la communication avec les périphériques, combinée à la gestion efficace des GPIO pour le contrôle des leds et du moteur, illustre parfaitement la capacité des systèmes embarqués à répondre à des besoins précis.

Enfin Comme autres perspectives d'amélioration, le projet pourrait être étendu pour intégrer des fonctionnalités supplémentaires et répondre à des besoins plus avancés, notamment :

- **Connexion à distance via XBee** : L'ajout de modules XBee permettrait de transmettre les données de température sans fil, offrant ainsi une surveillance à distance fiable et en temps réel. Cette fonctionnalité serait particulièrement utile pour des environnements industriels ou des applications nécessitant un suivi à distance.
- **Création d'une application mobile** : Une application mobile pourrait être développée pour recevoir des alertes en cas de problème, comme une surchauffe. Cette interface utilisateur apporterait une meilleure accessibilité, permettant aux utilisateurs de surveiller et d'interagir avec le système directement depuis leur smartphone.
- **Utilisation de FreeRTOS pour la gestion des tâches** : L'intégration d'un système d'exploitation temps réel comme FreeRTOS faciliterait la gestion des tâches concurrentes (lecture de la température, affichage sur LCD, activation des LEDs/moteur, transmission des données). Cela simplifierait le code tout en garantissant une meilleure réactivité et une optimisation des ressources.
- **Simplification et modularisation du code** : Une refactorisation du code pourrait être réalisée pour rendre le système plus modulaire, maintenable et extensible. Cela inclurait par exemple l’utilisation de bibliothèques dédiées pour chaque composant ou périphérique.

Ces améliorations renforceraient la robustesse, la flexibilité et les possibilités d’utilisation du système, tout en ouvrant la voie à des applications plus complexes et connectées.

Références

- Documentation du capteur BMP280.
- Bibliothèque LiquidCrystal_I2C.
- Bibliothèque BMP280.
- Documentation officielle de la carte FRDM-KL28Z.
- Notes de cours INF1633, Automne 2024.
- Documentation carte FRDMKL228Z
- Chat gpt, pour la reformulation de certaine partie du rapport
- Chat gpt pour la compréhension des erreurs du code et pour de potentiels correction
- Chat gpt pour commenter le code
- Code du TD4 (affichage d'un message à l'écran) du cours INF1633 modifier et compléter.