Lycée Alphonse BENOÎT
Cours Victor HUGO
84800 L'ISLE SUR LA SORGUE
Téléphone : 04-90-20-64-20

ACADÉMIE
D'AIX-MARSEILLE
Liberté
Égalité
Fraternité

BTS SN

Épreuve E6.2 – Session 2024

Dossier de revue n°2

CapTemp

SERVELEC
Boutique & Atelier

- Lachiver Christopher(IR)
- Soualmia Jessim (EC1)
- Fleury Benjamin (EC2)

Table des matières

Table des matières....2
Partie commune....5
1. Expression du besoin....5
1.1 Introduction et Problématique du Projet....5
1.2 Importance du Suivi de Température pour l'Entreprise....5
2. Contexte de Réalisation....5
2.1 Origine de l'Idée du Projet....5
2.2 Objectifs du projet du point de vue de SERVELEC....5
3. Présentation des Équipes de Projet....6
3.1 Composition des Équipes....6
4. Client/Donneur d'ordre....6
5. Cahier des Charges - Expression du Besoin....7
5. 1 Description détaillée des exigences du client:....7
6. partie commune - projet final 8
Partie personnelle Lachiver Christopher....11
7. Présentation de ma partie:....11
8. Mon travaille à faire:....11
9. Mon travaille fait:....12
9.1 Exemples de classes....13
9.2 L'interface graphique....20
9.3 Le serveur FTP....26
10 finalisation de l'interface graphique sur la Raspberry 29
11 Diagramme de Gantt....34
11.1 présentation de la partie physique....36
la sonde pt100....36
Principe de mesure :....36
Conclusion....37
Partie personnelle Soualmia Jessim....39
12. Présentation de ma partie :....39
13. Diagramme de blocs :....40
14. Diagramme de Gantt....41
15. Mise en œuvre du capteur DS18B20....42
15. Mise en œuvre de la Sonde PT100....46
16. Mise en œuvre du DS3231MZ....60
Partie personnelle FLEURY Benjamin....66
Présentation de ma partie....66
17. Diagramme de blocs....67
18. Diagramme de gantt....67
19. Présentation du matériel....69
20. Mise en œuvre du projet....71

21. Mise en œuvre du capteur DS18B20.....71
21.1 .Relevé de trame sur SCANA STUDIOS 72
23.2. Photo du résultat.....74
22. Mise en œuvre de la sonde PT100.....74
22.1. Schéma Fritzing de l'ESP32-S3 avec le DS18B20 et la sonde PT100.....77
23. Mise en œuvre du breakout micro SD..... 78
23.1 Schéma fritzing du câblage de l'ESP32-S3 et carte SD et TC74.....78
24. Routage de la carte fille..... 79
25. Procédure de mise en oeuvre de la carte..... 83
Annexe.....88
26. lien du compte rendu de la réunion.....88
27. Note d'étude du projet.....88

Partie commune

Partie commune

1. Expression du besoin

1.1 Introduction et Problématique du Projet

Dans le cadre de notre collaboration avec l'entreprise SERVELEC, nous nous sommes engagés dans un projet visant à mettre en place un système de suivi de température. Afin d'aider SERVELEC à assurer les bonnes températures de fonctionnement de leurs appareils électroménagers.

1.2 Importance du Suivi de Température pour l'Entreprise

Le suivi précis de la température revêt une importance capitale pour SERVELEC, car il influe directement sur la qualité et la sécurité des produits livrés à ses clients. En garantissant des conditions de conservation optimales tout au long de la chaîne logistique, SERVELEC aspire à renforcer la confiance de ses clients et à maintenir sa réputation d'excellence dans son domaine d'activité.
Cependant, la mise en place d'un tel système de suivi de température soulève plusieurs défis techniques et organisationnels. Tout d'abord, il est crucial de concevoir des dispositifs de mesure de température fiables et précis. Ensuite, il convient de développer une infrastructure logicielle robuste pour collecter, stocker et analyser les données de température de manière efficace et sécurisée.

2. Contexte de Réalisation

2.1 Origine de l'Idée du Projet

L'idée de ce projet est née de la collaboration entre SERVELEC et le lycée Alphonse Benoît, dans le cadre d'un partenariat visant à améliorer les processus logistiques et la qualité des services offerts par l'entreprise.

2.2 Objectifs du projet du point de vue de SERVELEC

Le système à développer a pour fonction principale de mesurer la température dans des réfrigérateurs et des fours.
Le matériel actuellement utilisé pour la mesure de température dans un four est un PeakTech 4975 et un thermomètre à dilatation.
Pour les réfrigérateurs l'entreprise utilise un thermomètre AT10 qui ne permet d'effectuer qu'une seule mesure à la fois.
Comme les températures sont dans des gammes extrêmes, le capteur pour mesurer les températures froides (entre -30° et 20°C)

sera différent du capteur pour mesurer les températures chaudes (20° et 400°C).
La précision dans les deux cas sera de 1°C.
Pour un réfrigérateur l'entreprise souhaite pouvoir mesurer simultanément la température de la partie congélateur et celle de la
partie réfrigérateur.
Les températures seront mesurées automatiquement selon un intervalle et une durée d'acquisition prédéfinis :
- réfrigérateur/congélateur :
- intervalle de temps entre mesures : 5 minutes
- durée totale de la capture : 1 semaine.
- four :
- intervalle de temps entre mesures : 5 secondes
- durée totale de la capture : 2h.
Les données acquises seront stockées dans un fichier au format CSV, sur une carte SD amovible.
En fin d'acquisitions, le technicien enlèvera la carte SD et la placera dans un lecteur de carte SD d'un ordinateur, afin
d'exploiter les données à l'aide d'un logiciel tableur.

3. Présentation des Équipes de Projet

3.1 Composition des Équipes

Le système sera développé par trois étudiants.
Un premier système sera développé par l'étudiant alternant ayant la spécialité EC, conformément à l'expression du besoin de SERVELEC.
Ce système sera développé autour d'une carte ESP32-S3 T-Display Lilygo.
Un autre système sera développé par deux étudiants (1 IR, 1 EC), avec une carte PI zéro ou équivalent, avec en plus la possibilité de transmettre les données acquises dans un cloud configuré. Un point d'accès WIFI devra exister dans l'environnement du réfrigérateur au four.
Les solutions devront proposer au technicien un affichage local de la température en temps réel. L'alimentation de ces systèmes se fera par bloc secteur. L'objectif est d'avoir une solution compacte pouvant être fixée sur ou à proximité de l'appareil à tester (par aimantation ou autre).

4. Client/Donneur d'ordre

Entreprise ou organisme commanditaire :
Nom : SERVELEC
Adresse : 136, Avenue de Roumanille, 84400 APT
Contact : Claude MAUREL
Adresse site : https://www.servelec-apt.fr/
Tél. : 04 90 05 68 38

5. Cahier des Charges - Expression du Besoin

5. 1 Description détaillée des exigences du client:

Le cahier des charges établi par SERVELEC spécifie clairement les exigences du client :

- Objectif Général :
    - Le système doit permettre la surveillance précise et continue de la température dans les réfrigérateurs, congélateurs et fours, afin d'assurer le bon fonctionnement et la sécurité des produits stockés ou manipulés.

Fonctionnalités Requises :

- Mesure précise de la température dans une plage allant de -30°C à 400°C.
- Utilisation de capteurs adaptés aux températures extrêmes.
- Acquisition automatique des données selon un intervalle et une durée prédéfinie.
- Stockage des données sur une carte SD amovible au format CSV.
- Consultation des données en temps réel via une interface utilisateur conviviale.
- Alerte sonore en cas de dépassement de seuil de température prédéfini.
- Alimentation électrique par bloc secteur pour une utilisation continue.

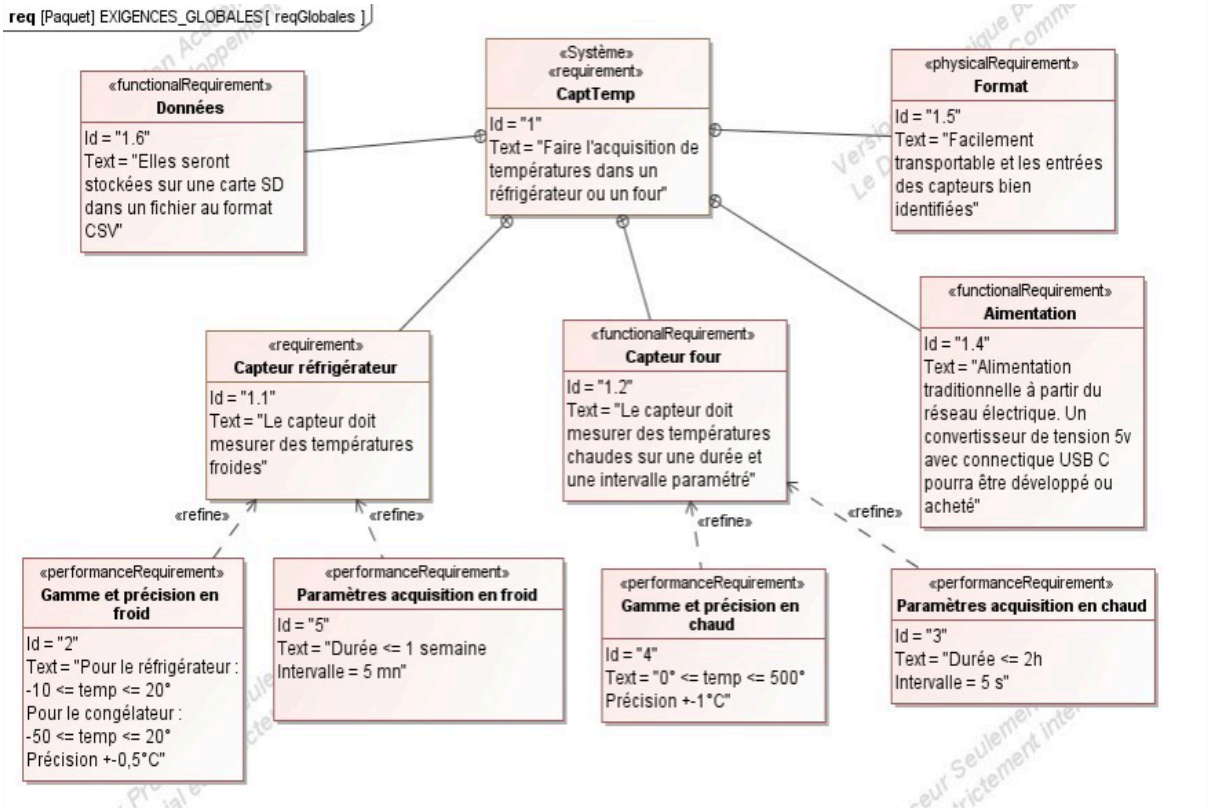
Spécifications Techniques :

- Utilisation de cartes matérielles telles que ESP32-S3 T-Display Lilygo et Raspberry Pi Zero.
- Intégration de capteurs de température DS18B20 pour les températures froides et PT100 pour les températures chaudes.
- Fréquence d'acquisition des données :
    - Réfrigérateurs/congélateurs : toutes les 5 minutes pendant 1 semaine.
    - Fours : toutes les 5 secondes pendant 2 heures.
- Affichage en temps réel des données de température sur l'écran de la carte.
- Stockage des données sur carte SD au format CSV pour analyse ultérieure.
- Conception d'une interface utilisateur intuitive pour la consultation des données et la configuration des paramètres.
- Documentation complète du système pour faciliter la maintenance et le dépannage.

Contraintes :

- Respect des normes de sécurité et de qualité en vigueur.
- Disponibilité des ressources matérielles et logicielles nécessaires.
- Respect des délais de livraison et de mise en service du système.

Ce cahier des charges servira de référence tout au long du développement du système, garantissant ainsi la satisfaction des besoins et des attentes du client.
cette expression du Besoin peut être mise sous la forme du diagramme suivant



6. partie commune - projet final

Pour ce projet nous avons donc conçu un système de capture de température miniaturisé sur une raspberry qui sera finalement une raspberry pi3 car une pi0 est pas assez puissante pour faire fonctionner toute l'"application.

Les capteurs chaud et froid captent des températures qui sont affichées sur l'écran de la raspberry lorsque le client choisit un capteur et lance l'application.

Ces données de température sont enregistrées sur la carte sd de la raspberry dans un fichier que le client choisit au format FTP.

De plus ces données sont aussi sauvegardées sur un serveur FTP ou l'espace de stockage est bien supérieur afin d'avoir une historique plus longue des données.

Ce système est miniaturisé et peut donc être placé ou le client souhaite même dans des endroits avec de l'espace restreint, ou même être attaché à l'appareil.

Le système rendu correspond aux attente du client avec des fonctionnalités ajoutés.

Partie Personnelle
Lachiver Christopher

(IR)

Partie personnelle Lachiver Christopher

7. Présentation de ma partie:

Dans le cadre de du projet, mon rôle est de contribuer au développement d'une solution permettant de mesurer et d'enregistrer les températures dans des appareils. Plus précisément, je suis chargé de la mise en œuvre du système de mesure sur une carte Raspberry Pi Zero, ainsi que de la configuration d'un serveur FTP pour le stockage des données.

8. Mon travaille à faire:

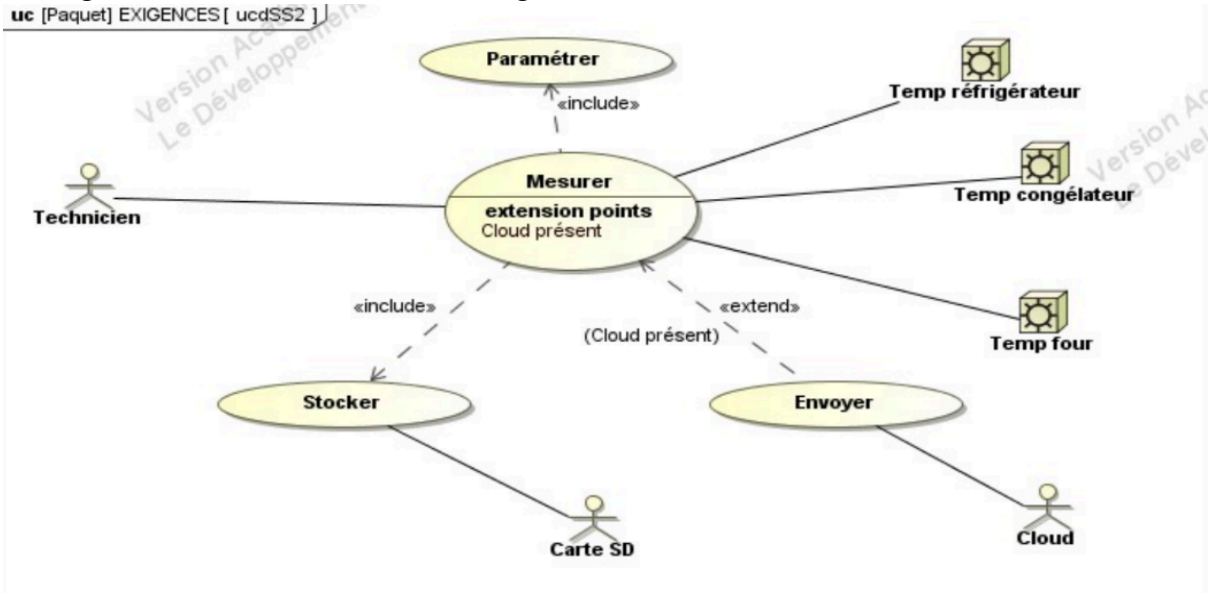
dans le projet suis chargé de concevoir et de mettre en œuvre le système de mesure de température sur une carte Raspberry Pi Zéro. Cette tâche implique plusieurs étapes:
Essais de mesure de Température :
Je vais effectuer des essais de mesure de température à l'aide de différents capteurs adaptés aux températures froides et chaudes. Pour cela, je vais utiliser le DS18B20 pour les températures froides et une sonde PT100 pour les températures élevées. Ce projet sera en lien avec l'étudiant EC.
Configuration de la Raspberry Pi Zéro :
Une fois les capteurs sélectionnés, je vais configurer et programmer la Raspberry Pi Zéro pour qu'elle puisse acquérir les données de température à partir de ces capteurs. Je vais m'assurer que les ports GPIO sont correctement initialisés, que les données des capteurs sont lues de manière précise et que les communications avec les capteurs sont efficaces.

Affichage des Données :
Après avoir acquis les données de température, je vais les afficher de manière lisible sur un écran IPS LCD display HAT pour Raspberry Pi ou Afficheur LCD / LCD Display connecté à la Raspberry Pi Zéro. Cela permettra aux utilisateurs de visualiser facilement les températures mesurées en temps réel.
Enregistrement des Données : Je vais également mettre en place un système d'enregistrement des données sur une carte SD insérée dans la Raspberry Pi Zéro. Les données de température seront stockées dans un fichier au format CSV pour permettre une analyse ultérieure.
Gestion des Alarmes : Enfin, je vais mettre en place un système d'alarme pour alerter en cas de dépassement de seuil de température. Cela peut être réalisé en déclenchant une alarme sonore ou en affichant un avertissement visuel sur l'écran TFT.
En ce qui concerne l'intégration avec un serveur FTP, je vais également m'en charger :
Installation du Serveur FTP :
Je vais installer et configurer un serveur FTP sur un ordinateur ou un serveur dédié. Cela implique l'installation du logiciel serveur FTP et la configuration des paramètres de sécurité.
Gestion des Autorisations d'Accès :
Une fois le serveur FTP opérationnel, je vais définir les autorisations d'accès aux répertoires de stockage des données de température. Je veillerai à ce que seules les personnes autorisées puissent accéder et modifier ces données.

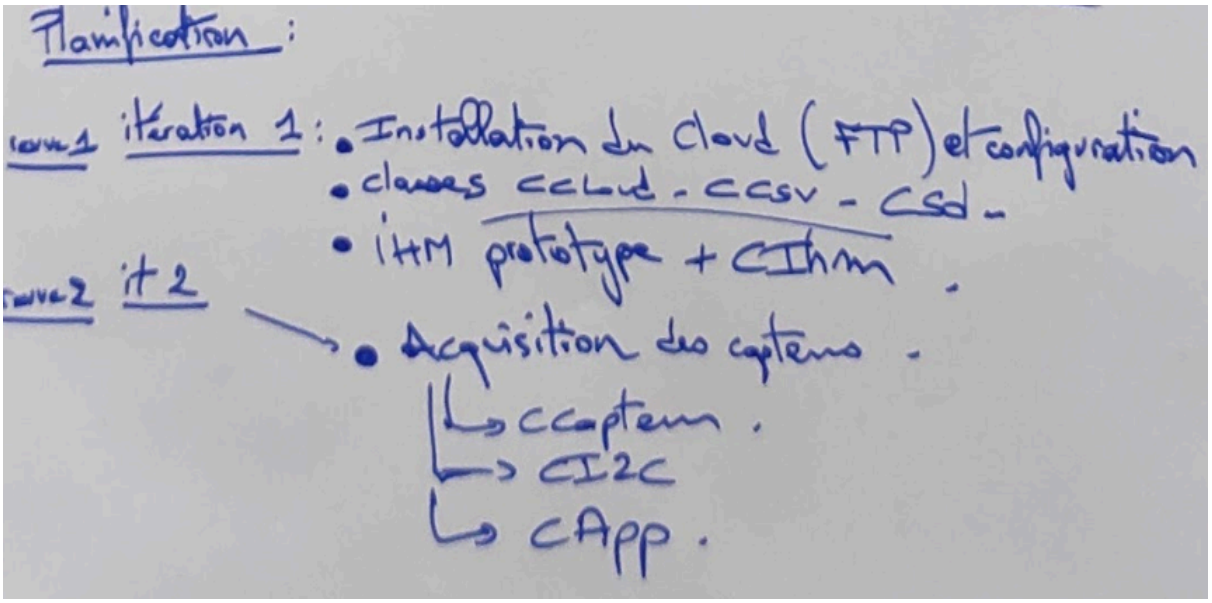
Mise en Place de l'Authentification Sécurisée :
Enfin, je vais mettre en place un système d'authentification sécurisée pour garantir que seuls les utilisateurs autorisés puissent accéder au serveur FTP et aux données de température stockées.
Ces travaux seront en lien avec l'étudiant EC afin de rendre un travail bien fait et correct au niveau des exigences.

9. Mon travaille fait:

Le diagramme ci-dessous montre les exigences finales demandées à l'étudiant IR.



Durant la réunion de projet, nous avons séparé le travail en plusieurs parties.
Pour la première itération je vais devoir mettre en place et configurer le serveur FTP en y incluant un utilisateur et mettre la bonne configuration.
Mais aussi créer, instancier et coder les classes App/CCsv/Cihm/Vsd/et CStop sans y intégrer une classe CCloud finalement.



9.1 Exemples de classes

classe App.h

```
#ifndef APP_H
#define APP_H

#include <QMainWindow>
#include <QObject>
#include <QWidget>
#include <CIhm.h>

class App
{
public:
    App();
    void runApp();

private:
    CIhm ihm;
};
#endif // APP_H
```

Classe app.cpp

```
#include "app.h"
App::App()
{}
void App::runApp(){
    ihm.askForSaveLocation();
}
```

Cette classe est la partie la plus importante de l'application.

Elle initialise et lance l'application en appelant la méthode runApp();.

La méthode runApp(); de cette classe peut invoquer d'autres classes et méthodes pour démarrer les fonctionnalités de l'application, telles que la gestion de l'interface utilisateur.

classe CCsv.h

```
#ifndef CCSV_H
#define CCSV_H

#include <QString>

class CCsv
{
public:
```

```
    CCsv(const QString &filename);

    QString readFromCsv();

    bool appendToCsv(const QString &data);

    static void setSaveLocation(const QString location);

    static QString getSaveLocation();

private:
    QString m_filename;

    static QString saveLocation;
};
#endif // CCSV_H
```

classe CCsv.cpp

```
#include "ccsv.h"
#include <QFile>
#include <QTextStream>
CCsv::CCsv(const QString &filename): m_filename(filename){
}
QString CCsv::readFromCsv(){
    QFile file(m_filename);
    if(!file.open(QIODevice::ReadOnly |QIODevice::Text))
        return "text écrit";

    QByteArray data = file.readAll();
    file.close();
    return QString::fromUtf8(data);
   }

bool CCsv::appendToCsv(const QString &data){
    QFile file(m_filename);
    if(!file.open(QIODevice::Append | QIODevice::Text))
        return false;
    QTextStream out(&file);
    out << data <<"\n";
    file.close();
    return true;
}

QString CCsv::saveLocation ="";

void CCsv::setSaveLocation(const QString location){
    saveLocation = location;
}
```

```
QString CCsv::getSaveLocation(){
    return saveLocation;
}
```

Le constructeur CCsv prend en paramètre le nom du fichier CSV avec lequel travailler.
La méthode readFromCsv() est utilisée pour lire les données à partir du fichier CSV spécifié. Elle ouvre le fichier en mode lecture, lit les données et les retourne sous forme de chaîne de caractères.
La méthode appendToCsv() permet d'ajouter des données à un fichier CSV existant. Elle ouvre le fichier en mode ajout, écrit les données fournies à la fin du fichier et retourne true si l'opération a réussi, sinon false.
La méthode statique setSaveLocation() est utilisée pour définir l'emplacement de sauvegarde des fichiers CSV. Elle prend en paramètre l'emplacement de sauvegarde et le stocke dans une variable statique saveLocation.
La méthode statique getSaveLocation() retourne l'emplacement de sauvegarde des fichiers CSV défini précédemment.

classe CIhm.h

```
#ifndef CIHM_H
#define CIHM_H
#pragma once
#include <QMainWindow>
#include <QWidget>
#include <QFileDialog>
#include <csd.h>
#include <ccsv.h>
#include <cstop.h>
#include <QDebug>

QT_BEGIN_NAMESPACE
namespace Ui { class Cihm; }
QT_END_NAMESPACE

class CIhm : public QMainWindow

{
    Q_OBJECT

public:

    explicit CIhm(QWidget *parent = nullptr);
    void askForSaveLocation();
    ~CIhm();

private slots:
    void on_goButton_clicked();
    void askSaveLocation();
    void on_stopButton_clicked();
    void HandleRecordingStopped();
```

```
private:
    Ui::Cihm *ui;
    Ui::Cihm *wdg;
    QFileDialog * fileDialog;
    CStop *cStopInstance;
};

#endif // CIHM_H
```

classe CIhm.cpp

```
 #include "CIhm.h"
#include "ui_cihm.h"
#include "csd.h"
#include "QWidget"
#include <QTimer>
#include <QMessageBox>
#include <QThread>

CIhm::CIhm(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::Cihm)
{
    ui->setupUi(this);
    cStopInstance = new CStop(this) ;
    fileDialog = new QFileDialog(this);
    connect(fileDialog, &QFileDialog::fileSelected, this, &CIhm::askSaveLocation);
    connect(ui->stopButton , &QPushButton::clicked, this ,
&CIhm::on_stopButton_clicked);
    connect(cStopInstance, &CStop::recordingStoped, this,
&CIhm::HandleRecordingStopped);
    QTimer::singleShot(0, this, &CIhm::askSaveLocation);
}

CIhm::~CIhm()
{
    delete ui;
    delete cStopInstance;
}


void CIhm::on_goButton_clicked()
{
    QMessageBox::StandardButton reply;
    reply= QMessageBox::question(this, "confirmation", "etes vous sur de vouloir lancer
l'application ?");

    if(reply== QMessageBox::Yes){
        QMessageBox::information(this,"information", "opération en cour");
```

```
        QThread::msleep(1000);

    }
}


void CIhm::askForSaveLocation(){
    QString saveLocation = QFileDialog::getExistingDirectory(nullptr,
QObject::tr("choisir un emplacement de sauvegarde"), QDir::homePath());

    Csd::setSaveLocation(saveLocation);
    CCsv::setSaveLocation(saveLocation);

}

void CIhm::askSaveLocation(){
    QString saveLocation = fileDialog->getSaveFileName(this,tr("choisir l'emplacement
de sauvegarde"), "", tr("fichiers CSV(*.csv)"));

    Csd::setSaveLocation(saveLocation);
}

void CIhm::on_stopButton_clicked(){
    cStopInstance->stopRecording();
}

void CIhm::HandleRecordingStopped(){
    QMessageBox::information(this,"information", "enregistrement terminé");
    QThread::msleep(500);
}
```

Constructeur et Destructeur :
Le constructeur CIhm initialise l'interface utilisateur en chargeant le fichier d'interface utilisateur (UI) et en connectant les signaux et les slots nécessaires pour interagir avec les autres composants de l'application.
Le destructeur ~CIhm libère la mémoire utilisée par l'interface utilisateur et d'autres composants associés.
Méthodes de Gestion des Événements :
La méthode on_goButton_clicked() est appelée lorsque le bouton "Go" est cliqué. Elle affiche une boîte de dialogue de confirmation et exécute une opération lorsque l'utilisateur confirme.
Les méthodes askForSaveLocation() et askSaveLocation() sont utilisées pour demander à l'utilisateur de sélectionner un emplacement de sauvegarde pour les fichiers CSV. La première méthode ouvre une boîte de dialogue pour sélectionner un répertoire, tandis que la seconde ouvre une boîte de dialogue pour sélectionner un fichier CSV spécifique.
La méthode on_stopButton_clicked() est appelée lorsque le bouton "Stop" est cliqué. Elle arrête l'enregistrement en cours en appelant la méthode stopRecording() de l'instance de la classe CStop.
La méthode HandleRecordingStopped() est appelée lorsque l'enregistrement est terminé. Elle affiche une boîte de dialogue pour informer l'utilisateur que l'enregistrement est terminé.
Variables Membres :

ui : Pointeur vers l'objet d'interface utilisateur généré à partir du fichier UI.
wdg : Pointeur vers l'objet d'interface utilisateur.
fileDialog : Pointeur vers la boîte de dialogue de sélection de fichier.
cStopInstance : Pointeur vers l'instance de la classe CStop utilisée pour contrôler l'enregistrement.

classe Csd.h

```
#ifndef CSD_H
#define CSD_H

#include <QString>
class Csd
{
public:
    Csd(const QString &filename);

    //méthode d'écriture du dossier
    bool writeToFile(const QString &data);

    static QString getSaveLocation();
    static void setSaveLocation(const QString& location);
private:
    QString m_filename;
    static QString saveLocation;
};

#endif // CSD_H
```

classe Csd.cpp

```
#include "csd.h"
#include <QFile>
#include <QTextStream>


Csd::Csd(const QString &filename) : m_filename(filename)
{

}
bool Csd::writeToFile(const QString &data){
    QFile file(m_filename);
  if(!file.open(QIODevice::WriteOnly | QIODevice::Text))
        return false;

    QTextStream out(&file);
    out << data;
    file.close();
    return true;
}
```

```
QString Csd::saveLocation = "";

void Csd::setSaveLocation(const QString& location){
   saveLocation = location;
}

QString Csd::getSaveLocation(){
   return saveLocation;
}
```

Constructeur :
Le constructeur Csd initialise un objet Csd avec le nom du fichier spécifié.
Méthode d'Écriture dans le Fichier :
La méthode writeToFile() est utilisée pour écrire des données dans le fichier spécifié. Elle prend en paramètre les données à écrire dans le fichier et retourne vrai si l'écriture est réussie, sinon elle retourne faux.
Méthodes Statiques pour la Gestion de l'Emplacement de Sauvegarde :
Les méthodes statiques setSaveLocation() et getSaveLocation() sont utilisées pour définir et obtenir l'emplacement de sauvegarde des fichiers. Elles permettent de spécifier l'emplacement où les données doivent être sauvegardées et de récupérer cet emplacement lorsque nécessaire.
Variables Membres Statiques :
m_filename : Chaîne de caractères contenant le nom du fichier dans lequel les données doivent être écrites.
saveLocation : Chaîne de caractères statique contenant l'emplacement de sauvegarde des fichiers.

classe CStop.h

```
#ifndef CSTOP_H
#define CSTOP_H

#include <QMainWindow>
#include <QObject>
#include <QWidget>

class CStop: public QObject{
   Q_OBJECT
public:
   CStop();
   explicit CStop(QObject *parent = nullptr);

public slots:
   void stopRecording();

signals:
   void recordingStoped();
};

#endif // CSTOP_H
```

classe CStop.cpp

```
#include "cstop.h"
#include <QDebug>

CStop::CStop(QObject *parent) : QObject(parent){
}
void CStop::stopRecording(){
   emit recordingStoped();
}
```

Constructeurs :
La classe CStop possède deux constructeurs, un par défaut et un autre qui prend un pointeur vers un parent QObject en paramètre. Le constructeur par défaut initialise un nouvel objet CStop.
Méthode stopRecording() :
La méthode stopRecording() est définie comme un slot public. Lorsqu'elle est appelée, elle émet un signal recordingStoped().
Signal recordingStoped() :
Le signal recordingStoped() est émis par la méthode stopRecording() lorsqu'elle est appelée. Ce signal indique que l'enregistrement a été arrêté avec succès et affiche le message.

Tout ce code permet un début d'application qui sera amélioré pour l'itération n°2 avec l'ajout d'un lien entre les données du capteur et l'application.

Ces classes sont des exemples il existe dans le projet d'autres classes qui permettent à l'application de fonctionner:

```h
// calarme.h

#ifndef CALARME_H
#define CALARME_H


#include <QObject>
#include <QMessageBox>
#include <QMediaPlayer>

class Calarme : public QObject
{
    Q_OBJECT
public:
    explicit Calarme(QObject *parent = nullptr);
    ~Calarme();

    void verifierTemperature(int temperature);

signals:
    void alarmeActive();

private slots:
    void arreterAlarme(int result);

private:
    QMediaPlayer *player;
    QMessageBox *messageBox;
    int seuilChaud = 400;
    int seuilFroid = -30;
};

#endif // CALARME_H

```

```cpp
// calarme.cpp

#include "Calarme.h"
#include <QMessageBox>
#include <QMediaPlayer>


Calarme::Calarme(QObject *parent) : QObject(parent)
{
    player = new QMediaPlayer();
    player->setMedia(QUrl::fromLocalFile("/home/pi/Music/imperial-alert.mp3"));
    messageBox = new QMessageBox();
    messageBox->setWindowTitle("Alarme");
    messageBox->setText("La température a dépassé le seuil !");
    messageBox->setStandardButtons(QMessageBox::Ok);
    connect(messageBox, &QMessageBox::finished, this, &Calarme::arreterAlarme);
}

Calarme::~Calarme()
{
    delete player;
    delete messageBox;
}

void Calarme::verifierTemperature(int temperature)
{
    if (temperature > seuilChaud || temperature < seuilFroid)
    {
        emit alarmeActive();
        messageBox->show();
        player->play();
    }
}

void Calarme::arreterAlarme(int result)
{
    if (result == QMessageBox::Ok)
    {

        player->stop();
    }
}

```

la classe CAlarme permet d'émettre un son lorsque la température atteint un seuil définit

```
#ifndef CI2C_H
#define CI2C_H

#include <QObject>
#include <QMutex>
#include <QMutexLocker>
#include <QDebug>
#include <stdio.h>
#include <unistd.h>             //Needed for I2C port
#include <fcntl.h>              //Needed for I2C port
#include <sys/ioctl.h>          //Needed for I2C port
#include <linux/i2c-dev.h>      //Needed for I2C port
#include <strings.h>

class CI2c : public QObject
{
    Q_OBJECT

public:
    // creation destruction de l'objet
    static CI2c *getInstance(QObject *parent = 0, char no = '1');
    static void freeInstance();
    int lire(unsigned char addr, unsigned char *buffer, int lg);
    int ecrire(unsigned char addr, unsigned char *buffer, int lg);

private:
    explicit CI2c(QObject *parent = 0, char noBus = '1');
    int init();
    QObject *m_parent;
    QMutex m_mutexI2c;
    int getNbLink();
    int m_addr;   // Adresse du composant I2C
    char m_noBus;   // No d'accès au fichier /dev
    int m_fileI2c;  // descripteur du fichier i2C
    int m_nbLink;
    static CI2c *m_singleton;

signals:
    void sigErreur(QString msg);
};

#endif // CI2C_H

```

```cpp
#include "ci2c.h"

CI2c::CI2c(QObject *parent, char noBus) :
    QObject(parent)
{
    m_parent = parent;
    m_noBus = noBus;
    m_nbLink=0;
} // constructeur

CI2c * CI2c::m_singleton = NULL;

int CI2c::lire(unsigned char addr,  unsigned char *buffer, int lg)
{
    if(ioctl(m_fileI2c, I2C_SLAVE, addr)!=0) {  // Règle le driver I2C sur l'adresse.
        QString mess="CI2c::lire Erreur ioctl acces au bus I2C";
//        qDebug() << mess;
        emit sigErreur(mess);
        return -1;
    } // if ioctl
    bzero(buffer, lg);                                          The bzero() function is obsoleted
    QMutexLocker lock(&this->m_mutexI2c);  // verrouillage du mutex. Il est libéré en sortie de méthode

    int nb=read(m_fileI2c, buffer, lg);
//    qDebug() << "CI2c:lire: " << buffer[0] << " " << buffer[1] << buffer[2] << " " << buffer[3] << buffer[4] << '
    return nb;
} // lire

int CI2c::ecrire(unsigned char addr, unsigned char *buffer, int lg)
{
    if(ioctl(m_fileI2c, I2C_SLAVE, addr)!=0) {  // Règle le driver I2C sur l'adresse.
        QString mess="CI2c::ecrire Erreur ioctl acces au bus I2C";
//        qDebug() << mess;
        emit sigErreur(mess);
        return -1;
    } // if ioctl

    QMutexLocker lock(&this->m_mutexI2c);  // verrouillage du mutex. Il est libéré en sortie de méthode

    int nb=write(m_fileI2c, buffer, lg);
//    qDebug() << "CI2c:ecrire: nb=" << nb << " : " << buffer[0] << " " << buffer[1] << buffer[2];
    return nb;
} // ecrire

int CI2c::init()
```

```
int CI2c::init()
{
    char filename[20];
    sprintf(filename, "/dev/i2c-%c",m_noBus);
    if((m_fileI2c=open(filename, O_RDWR))==-1) {  // ouvre le fichier virtuel d'accès à l'I2C
        QString mess="CI2c::init Erreur ouverture acces au bus I2C";
//        qDebug() << mess;
        emit sigErreur(mess);
        return -1;
    } // if open
    return m_fileI2c;
} // init

int CI2c::getNbLink()
{
    return m_nbLink;
} // getNbLink

CI2c *CI2c::getInstance(QObject *parent, char no)
{
    if (m_singleton == NULL)
    {
        qDebug("L'objet CI2c est créé");
        m_singleton =  new CI2c(parent, no);
        m_singleton->init();
        m_singleton->m_nbLink=1;
    }
    else
    {
        m_singleton->m_nbLink++;
        qDebug("singleton est déjà existant");
    }
    return m_singleton;
} // getInstance

void CI2c::freeInstance()
{
    if (m_singleton != NULL)
      {
            m_singleton->m_nbLink--;
            if (m_singleton->m_nbLink==0) {
                close(m_singleton->m_fileI2c);
                delete m_singleton;
                m_singleton = NULL;
            } // if mNbLink
      } // if null
} // freeInstance

```

```h
#ifndef CPT100_H
#define CPT100_H

#include <QObject>
#include "ci2c.h"

#define NO_ESCLAVE 8

class CPt100 : public QObject

{
    Q_OBJECT
public:
    explicit CPt100(QObject *parent = nullptr);
    ~CPt100();

    float lireTemp();

private:
    CI2c *_i2c;


signals:

};

#endif // CPT100_H

```

```cpp
#include "cpt100.h"

CPt100::CPt100(QObject *parent) : QObject(parent)
{
    _i2c = CI2c::getInstance();

}

CPt100::~CPt100()
{
    CI2c::freeInstance();
}

float CPt100::lireTemp()
{
    float temp;
    unsigned char trame[5];
    _i2c->lire(NO_ESCLAVE, trame, 2);

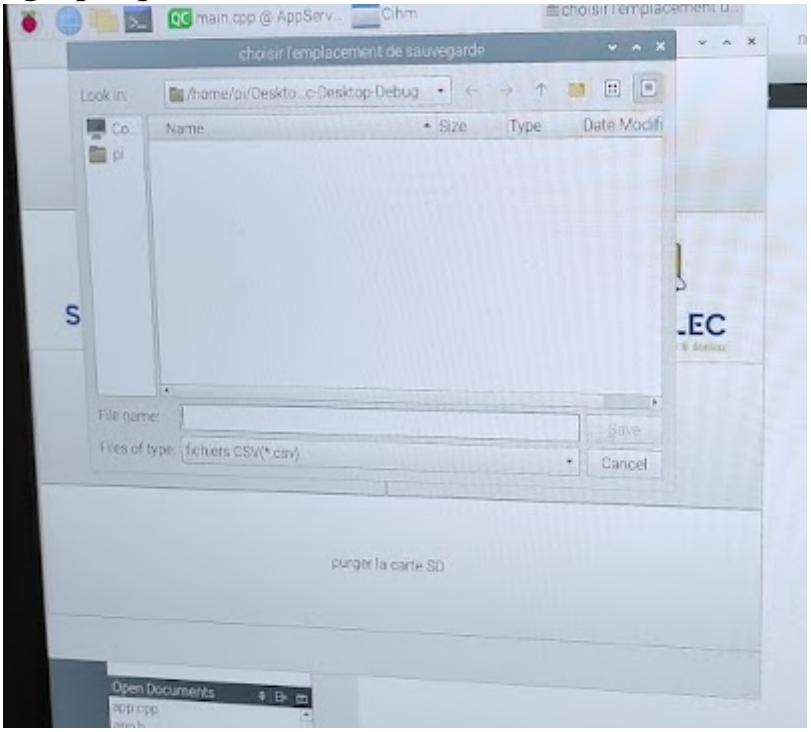
    temp = trame[0]*10 + trame[1];
    return temp;
}

```

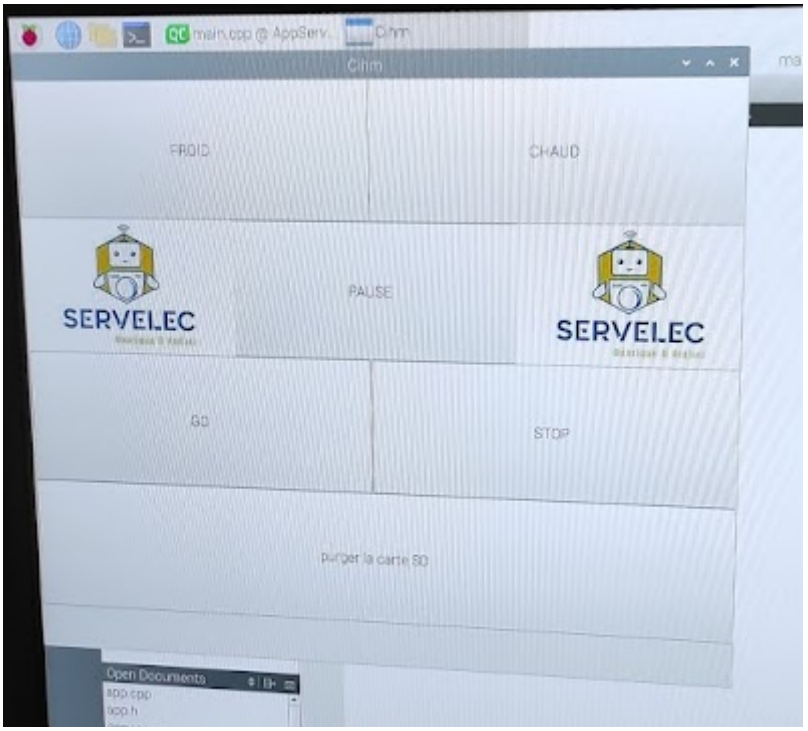
les classes ci2c ds18b20 et cpt100 permettent de capter la température des capteur en i2c pour la pt100 et en onewire pour la ds18b20

9.2 L’interface graphique

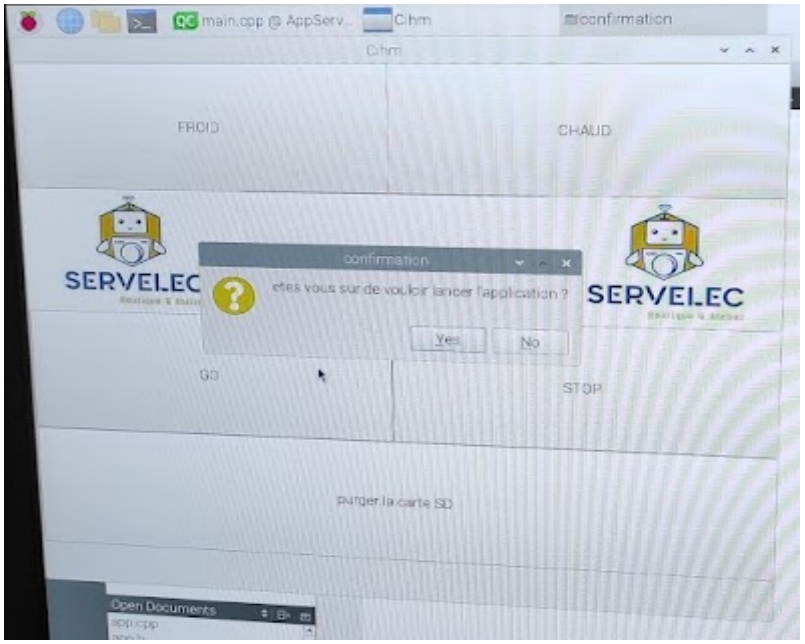
L’application graphique est actuellement la suivante:



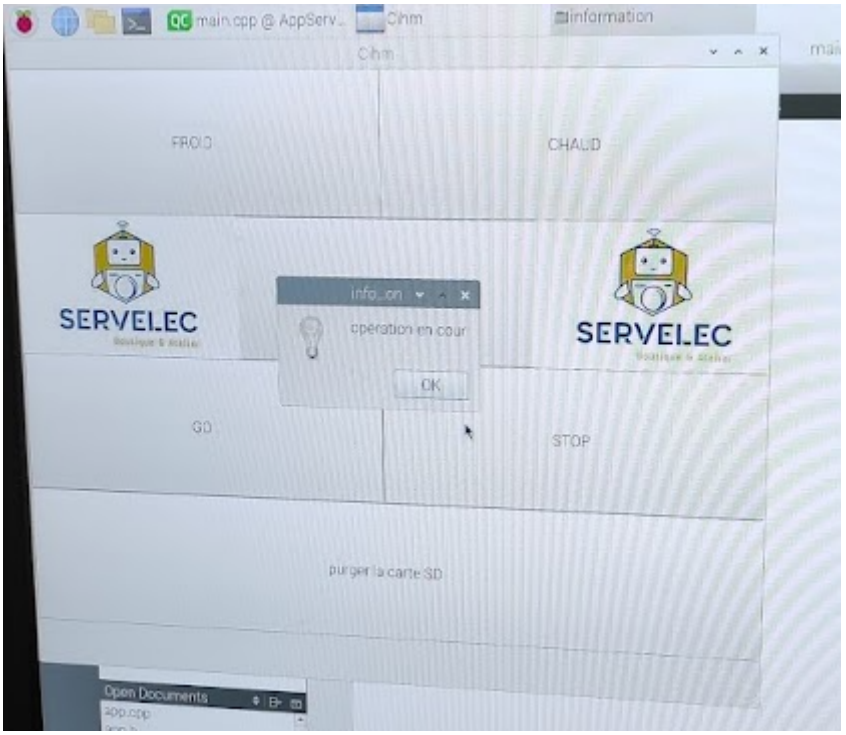
Tout d’abord l’application demande un emplacement de sauvegarde des fichiers au format CSV “normalement”.



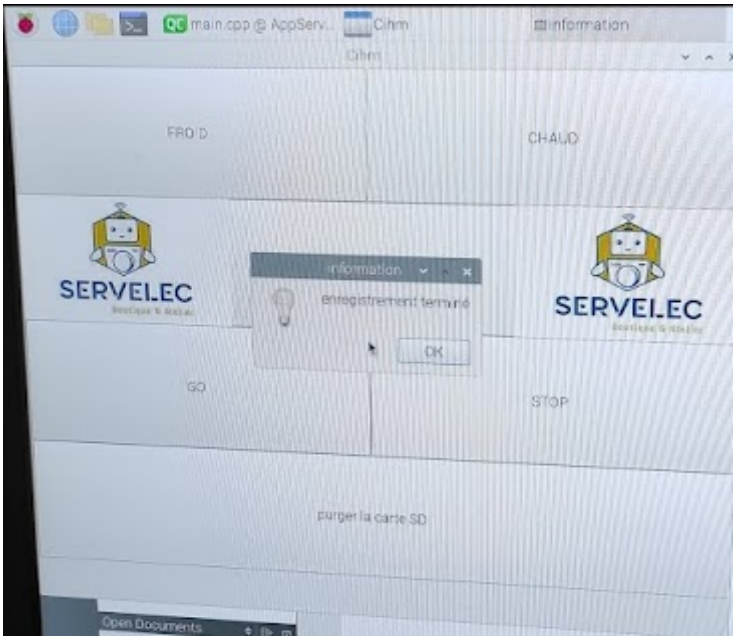
Puis plusieurs options s’offrent à nous celles faites en grandes parties sont:



GO qui demande une confirmation de lancement de l'application



Puis qui affiche que l'application est bien lancée.



Et le bouton stop qui stop l'acquisition des températures,

De plus j'ai réglé le problème du double cliquage des push button (voir note d'étude de projet) en mettant en commentaire les connect des boutons ce qui causait un double affichage des informations des boutons.

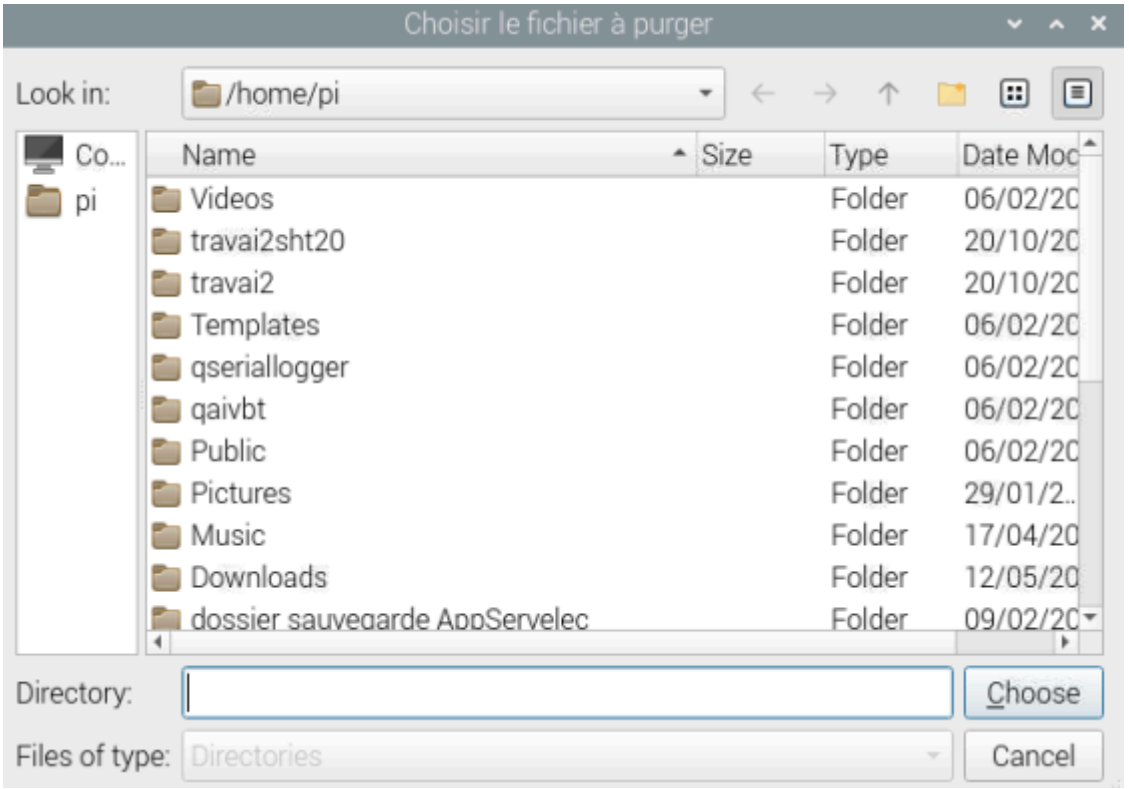
```
    connect(fileDialog, &QFileDialog::fileSelected, this, &CIhm::askSaveLocation);
  //  connect(ui->stopButton, &QPushButton::clicked, this, &CIhm::on_stopButton_clicked);
//    connect(ui->pauseButton, &QPushButton::clicked, this, &CIhm::on_pauseButton_clicked);
 //   connect(ui->purgeButton, &QPushButton::clicked, this, &CIhm::on_purgeButton_clicked);
    connect(cStopInstance, &CStop::recordingStoped, this, &CIhm::handleRecordingStopped);
    connect(cPauseInstance, &CPause::recordingPaused, this, &CIhm::recordingPaused);
  //  connect(ui->goButton, &QPushButton::clicked, this, &CIhm::on_goButton_clicked); // Connexion du signal au slot

    // Connexion du QTimer au slot afficherTemp
    connect(tempTimer, &QTimer::timeout, this, &CIhm::afficherTemp);
```

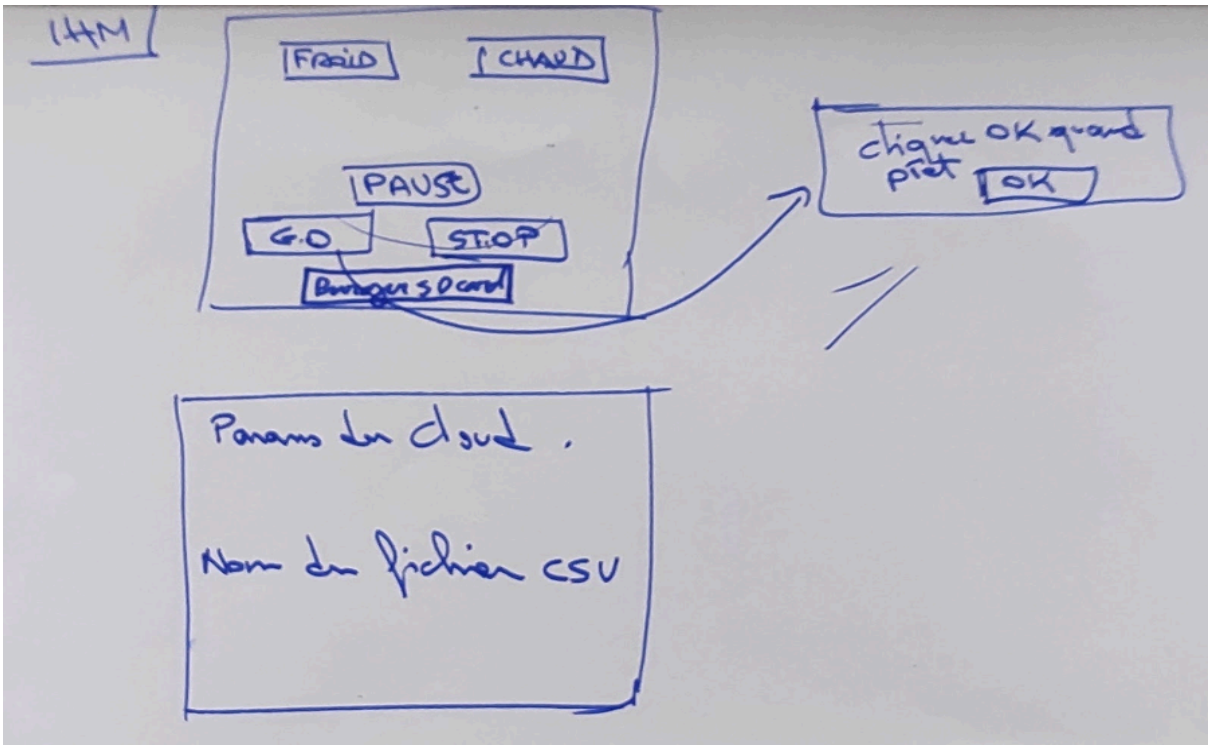
mais j'ai aussi codé le choix des capteurs froids et chaud avec une sécurité qui empêche de choisir les deux capteurs et une autre qui empêche le lancement de la capture sans avoir choisis de capteur afin d'éviter une enregistrement de valeurs null qui prennent de la place. lorsque l'on choisis un capteur celui ci est en couleurs et la température est affiché au centre de l'interface graphique



Lorsque l'application est lancée mettre pause coupe la capture sans l'arrêter et appuyer sur stop coupe la capture et l'arrête.
le bouton purger la carte sd permet de supprimer un fichier choisi par l'utilisateur et de le supprimer.

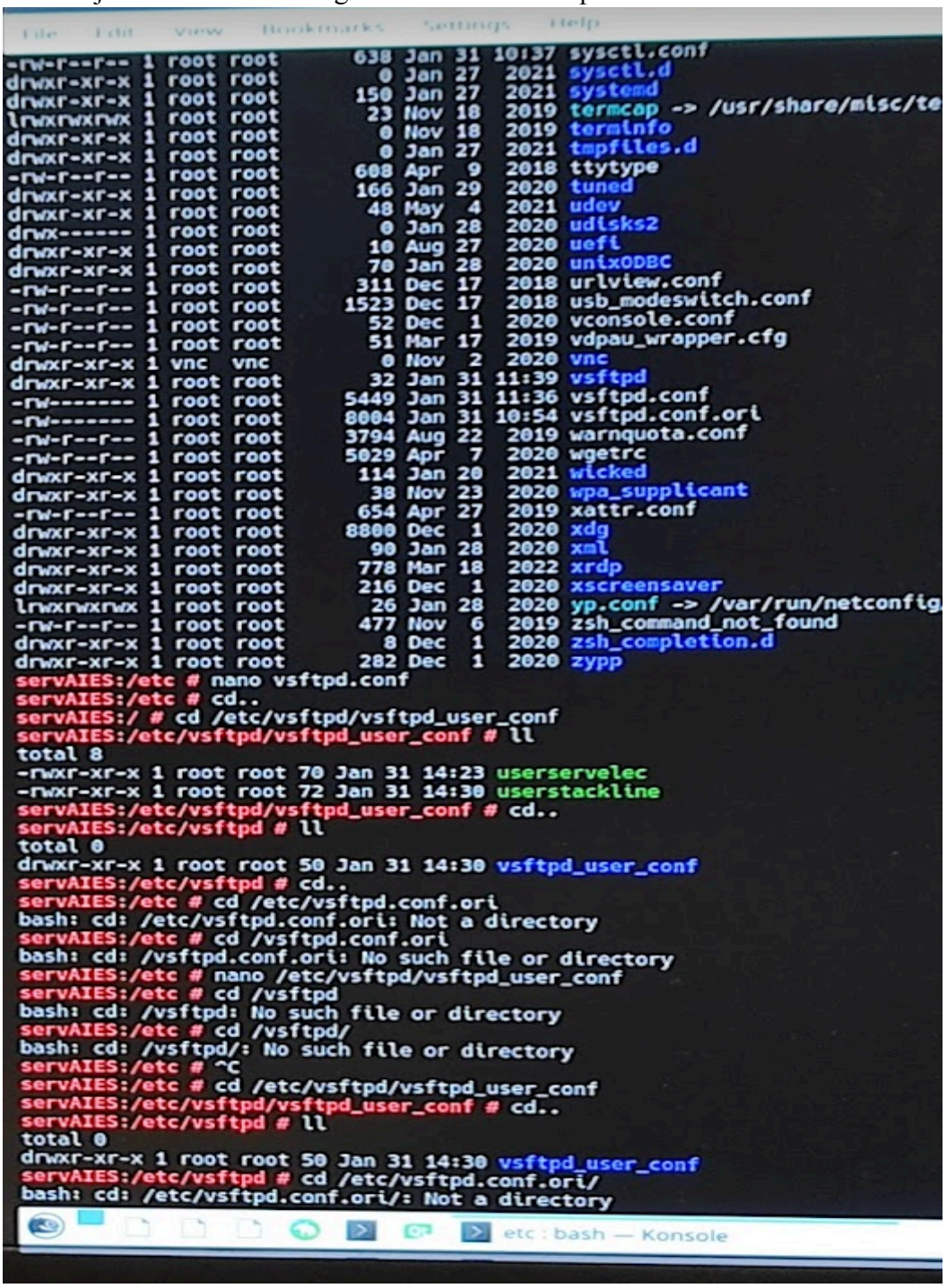


- Cet application correspond à l'affichage exigé durant la réunion de projet

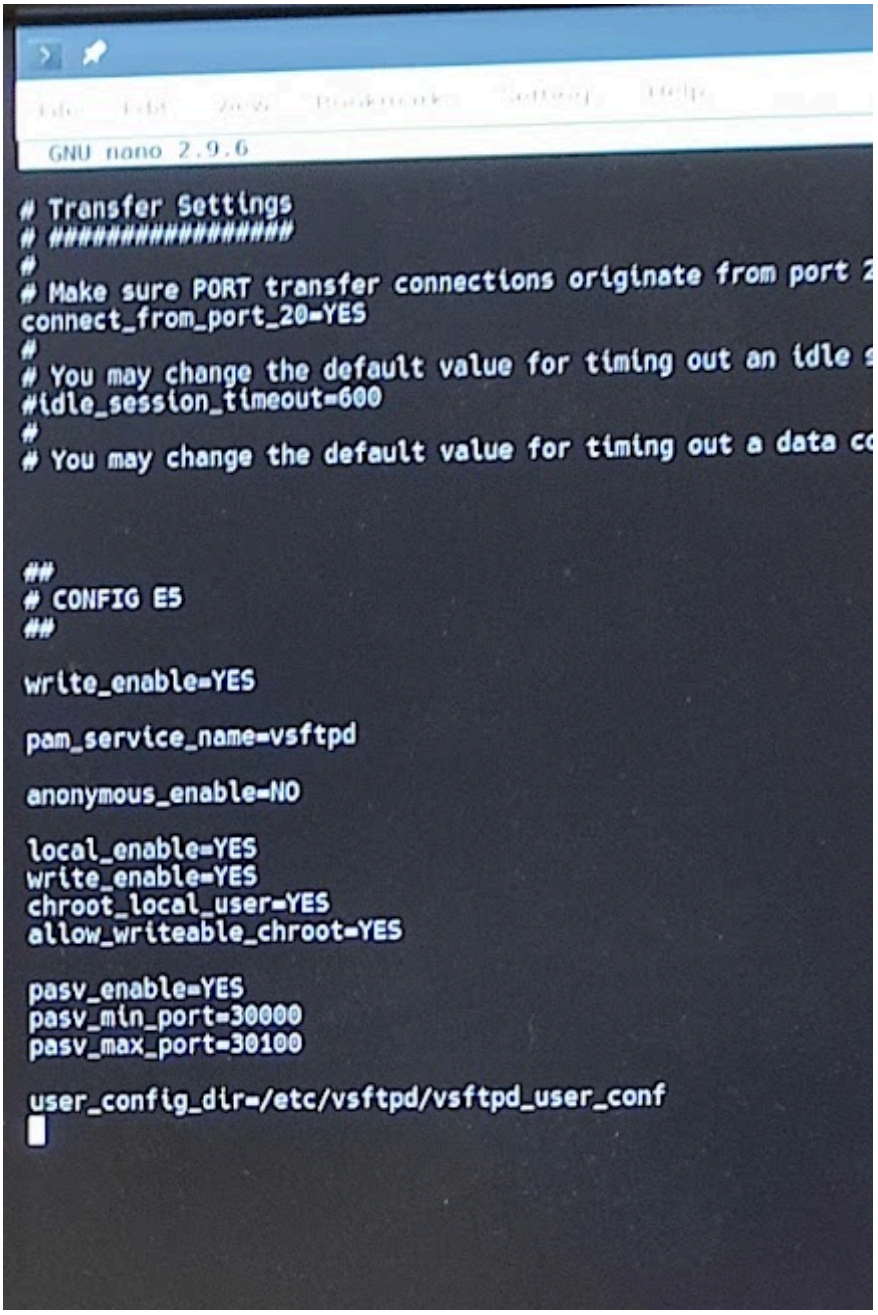


9.3 Le serveur FTP

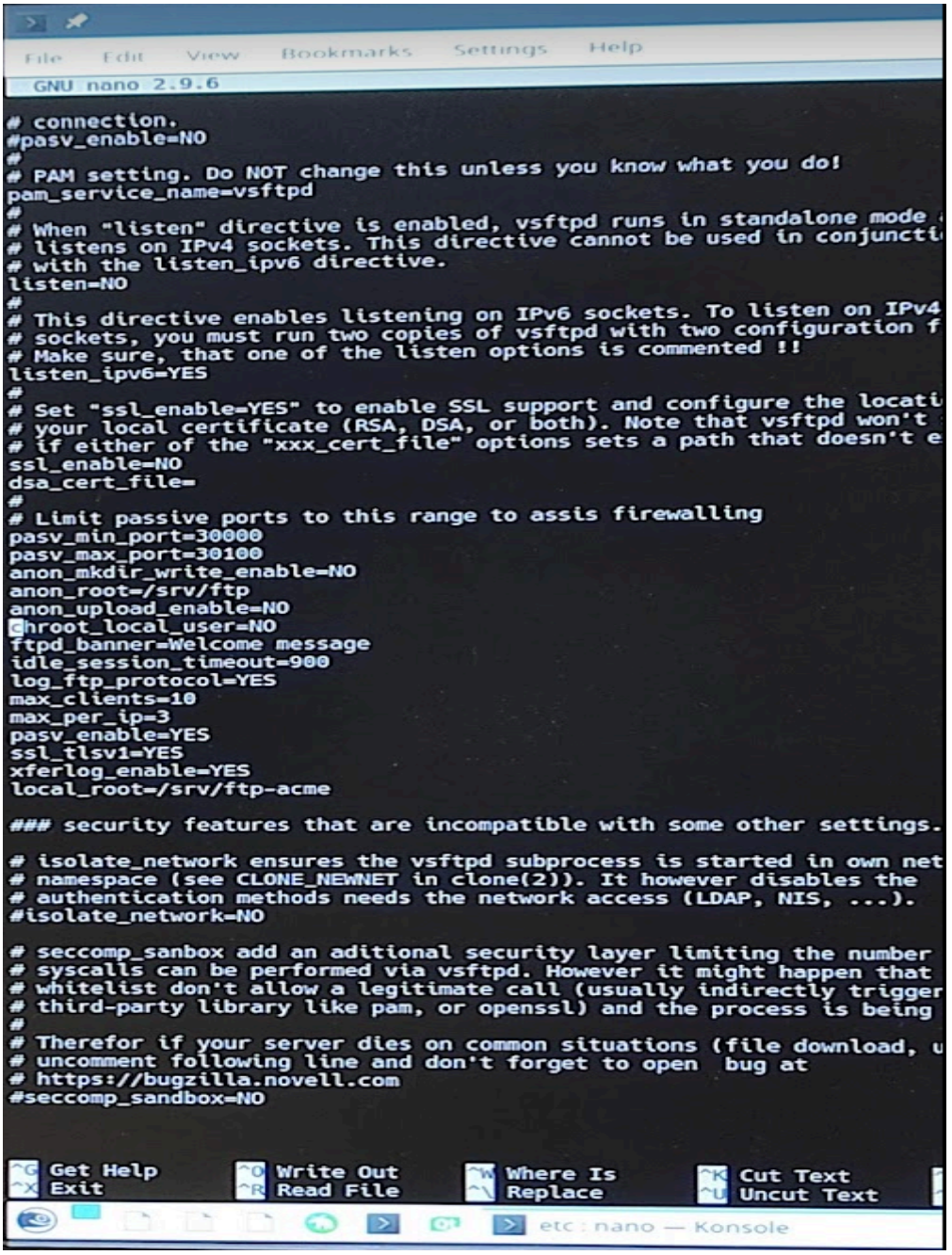
Pour terminer j'ai initialisé la configuration sans mot de passe de l'utilisateur du serveur FTP:



ici nous voyons les deux fichier des utilisateurs de mon projet et d'un autre dans le fichier vsftpd_user.conf.



Voici le fichier vsftpd_user.conf.
Qui gère les paramètres utilisateurs.



Et voici le fichier vsftpd_user.conf.ori.
Qui gère les paramètres du serveur.

Dans le but de notre projet, le serveur ftp permet de sauvegarder sur celui-ci les données de température sur une plus grande durée, de les historiser et de les sécuriser .

Il permet de remplacer le cloud qui lui aurait été payant ce qui dans le cadre du projet n'était pas ce qu'il fallait et donc le serveur ftp permet d'émuler le cloud ftp.

10 finalisation de l’interface graphique sur la Raspberry

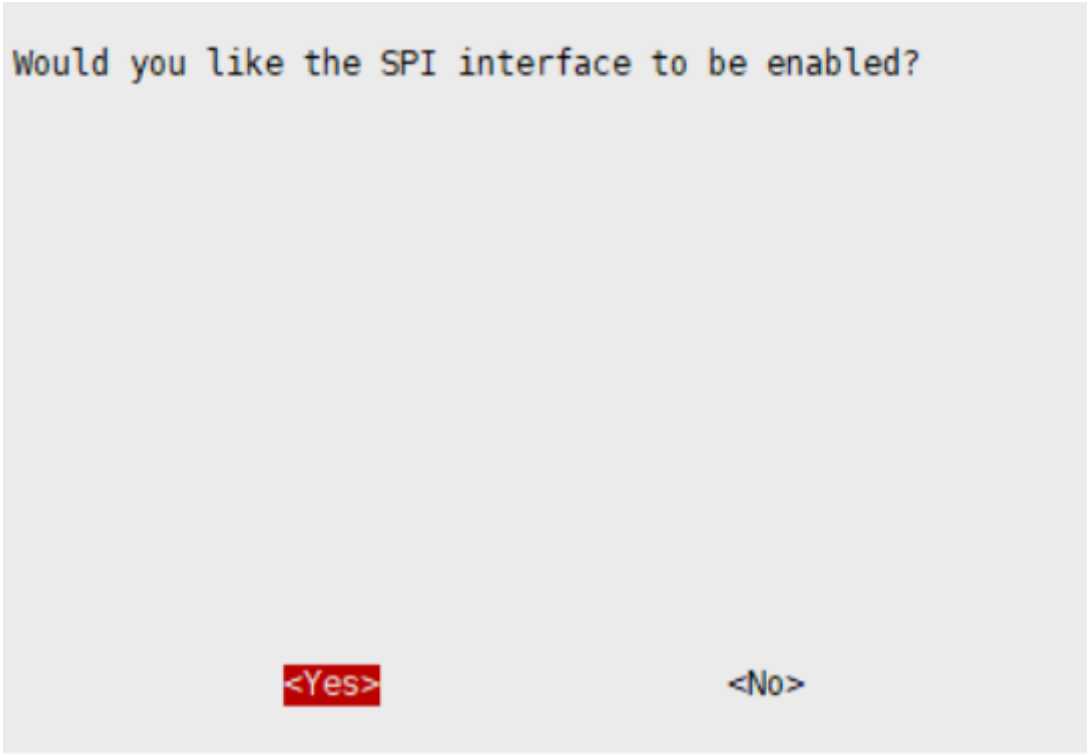
Lors de la deuxième partie du projet mon objectif était de permettre un affichage complet et de qualité de l’interface graphique sur l’écran GPIO de la raspberry.

J’ai donc commencé à activer la communication SPI

```
sudo raspi-config
Choose Interfacing Options -> SPI -> Yes  to enable SPI interface
```

```
1 Change User Password Change password for the current user
2 Network Options      Configure network settings
3 Boot Options         Configure options for start-up
4 Localisation Options Set up language and regional settings to match your location
5 Interfacing Options  Configure connections to peripherals
6 Overclock            Configure overclocking for your Pi
7 Advanced Options     Configure advanced settings
8 Update               Update this tool to the latest version
9 About raspi-config   Information about this configuration tool
```

```
P1 Camera      Enable/Disable connection to the Raspberry Pi Camera
P2 SSH         Enable/Disable remote command line access to your Pi using SSH
P3 VNC         Enable/Disable graphical remote access to your Pi using RealVNC
P4 SPI         Enable/Disable automatic loading of SPI kernel module
P5 I2C         Enable/Disable automatic loading of I2C kernel module
P6 Serial      Enable/Disable shell and kernel messages on the serial connection
P7 1-Wire      Enable/Disable one-wire interface
P8 Remote GPIO Enable/Disable remote access to GPIO pins
```



```
sudo reboot
```

Puis j'ai installé toutes les librairies (commandes prises sur le site du fabriquant)

```
#Open the Raspberry Pi terminal and run the following command
wget http://www.airspayce.com/mikem/bcm2835/bcm2835-1.71.tar.gz
tar zxvf bcm2835-1.71.tar.gz
cd bcm2835-1.71/
sudo ./configure && sudo make && sudo make check && sudo make install
# For more, you can refer to the official website at: http://www.airspayce.com/mikem/bcm2835/
```

```
#Open the Raspberry Pi terminal and run the following command
cd
sudo apt-get install wiringpi
#For Raspberry Pi systems after May 2019 (earlier than that can be executed without), an upgrade may be required:
wget https://project-downloads.drogon.net/wiringpi-latest.deb
sudo dpkg -i wiringpi-latest.deb
gpio -v
# Run gpio -v and version 2.52 will appear, if it doesn't it means there was an installation error

# Bullseye branch system using the following command:
git clone https://github.com/WiringPi/WiringPi
cd WiringPi
. /build
gpio -v
# Run gpio -v and version 2.70 will appear, if it doesn't it means there was an installation error
```

```
#Open the Raspberry Pi terminal and run the following command
wget https://github.com/joan2937/lg/archive/master.zip
unzip master.zip
cd lg-master
sudo make install

# You can refer to the official website for more: https://github.com/gpiozero/lg
```

```
sudo apt-get update
sudo apt-get install python3-pip
sudo apt-get install python3-pil
sudo apt-get install python3-numpy
sudo pip3 install spidev
```

Après les avoir installées je les ai exécuté afin de tester si l'exemple fonctionnait

```
cd c
make clean
make
sudo ./main
```

Cependant lorsque je lançait les démos rien ne s'affichait j'ai donc malgré ceci continué l'installation car je pensais que c'était la démo qui ne fonctionnait pas.

```
cd ~
sudo apt-get install cmake -y
sudo apt-get install p7zip-full -y
wget https://files.waveshare.com/upload/f/f9/Waveshare_fbcp.7z
7z x Waveshare_fbcp.7z -o./waveshare_fbcp
cd waveshare_fbcp
mkdir build
cd build
```

```
cmake -DSPI_BUS_CLOCK_DIVISOR=20 -DWAVESHARE_1INCH3_LCD_HAT=ON -DBACKLIGHT_CONTROL=ON -DSTATISTICS=0 ..
```

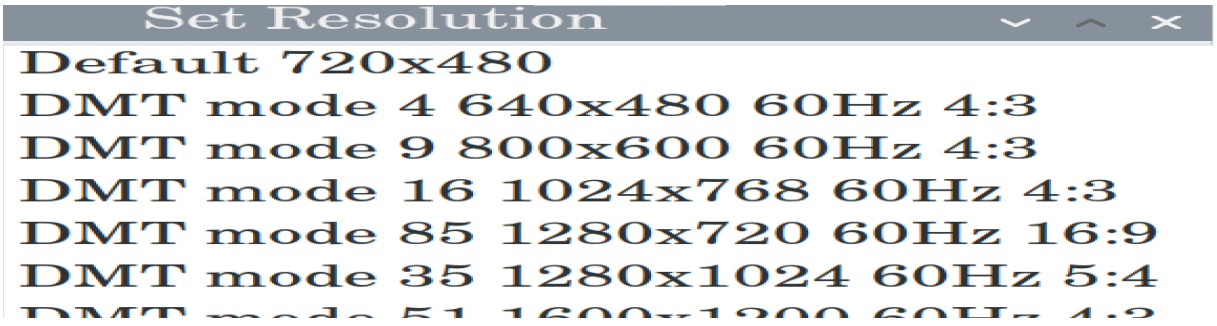
```
make -j
sudo ./fbcp
```

Après l'exécution de l'application, rien ne s'affichait. J'ai donc fait des recherches pour savoir ce qui ne fonctionnait pas car ce n'était pas la démo qui avait un problème.

J'ai donc modifié la résolution en tâtonnant afin de trouver celle qui convient le mieux.

```
# uncomment if hdmi display is not detected and composite is being output
hdmi_force_hotplug=1
hdmi_cvt=600 600 60 1 0 0 0
hdmi_group=2
hdmi_mode=87
display_rotate=0
```

Je l'ai modifié dans le sudo nano /boot/config.txt car la changer dans les paramètres de la raspberry n'était pas possible en raison du peu d'options de personnalisation.



Mais avant de fonctionner j'ai désactivé les communication infrarouge

```
# Uncomment this to enable infrared communication.
#dtoverlay=gpio-ir,gpio_pin=17
#dtoverlay=gpio-ir-tx,gpio_pin=18
```

J’ai activé les pines GPIO.

```
[all]
gpio = 6 19,5,26,13,21,20,16 = pu
dtoverlay=w1-gpio
enable_uart=0
scaling_kernel=8
```

J’ai désactivé des drivers.

```
# Enable DRM VC4 V3D driver
#dtoverlay=vc4-kms-v3d
max_framebuffers=2
```

Et j’ai activé l’interface SPI manuellement et désactivé L’interface I2C et I2S.

```
# Uncomment some or all of these to enable the optional hardware interfaces
dtparam=i2c_arm=off
dtparam=i2s=off
dtparam=spi=on
```

J’ai aussi boosté l’affichage

```
# uncomment to increase signal to HDMI, if you have interference, blanking, or
#no display
config_hdmi_boost=4
```

Tous ces paramètres ont permis l’affichage de linux sur la Raspberry. Or celui-ci était trop zoomé et décalé du bord de l’écran.

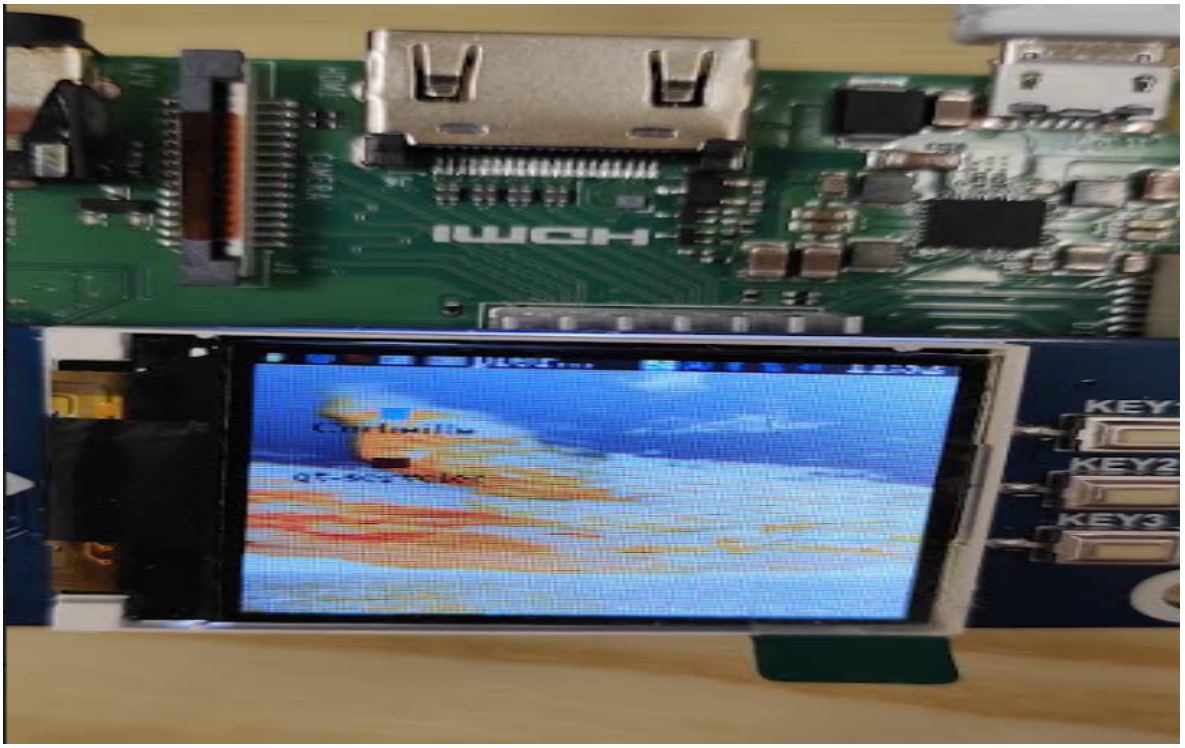
J’ai donc suivi un tutoriel sur plusieurs sites internet jusqu’à trouvé sur un forum la solution qui était de modifier la taille de linux et activer les paramètres d'overscan qui fonctionnent pas uniquement sur la console mais sur linux aussi ce qui a permis de recentrer l’affichage après tâtonnement.

```
# uncomment the following to adjust overscan. Use positive numbers if console
# goes off screen, and negative if there is too much border
overscan_left=0
overscan_right=275
overscan_top=0
overscan_bottom=275

# uncomment to force a console size. By default it will be display's size minus
#overscan.
framebuffer_width=600
framebuffer_height=600
```

Après avoir tout bien paramétré j'ai modifié la tailles des icônes sur le bureau et de la souris et redimensionné l'interface.

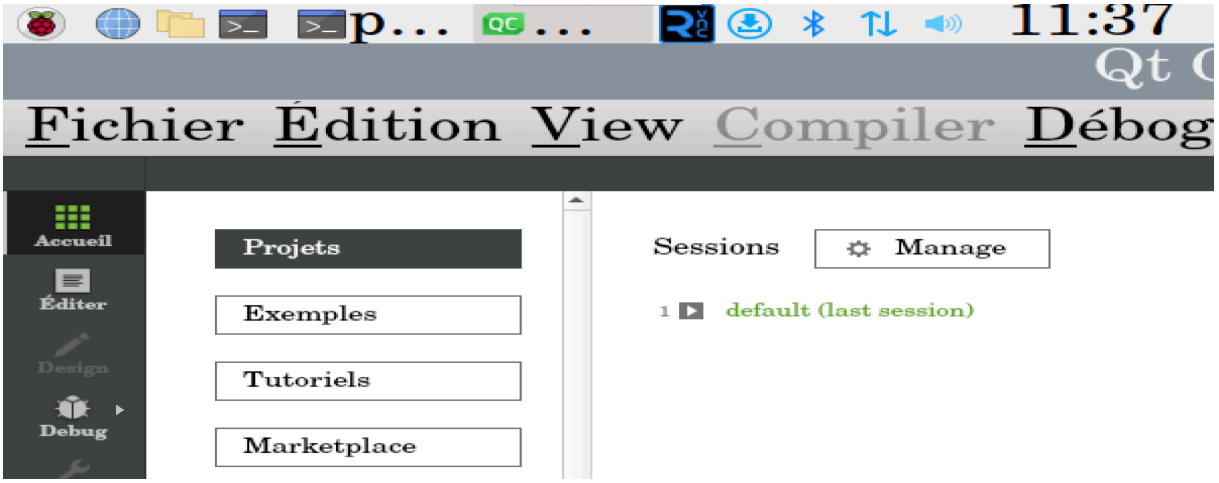
L'interface finale est la suivante.



L'interface finale est celle souhaitée et ne possède plus de pixel non rafraîchis qui étaient causés par la taille supérieur de l'interface réel par rapport à celle de l'écran.

Cependant il demeure des problèmes moindres mais toujours présents qui est celui de l'inversement de la couleur grise et blanche et des applications système qui s'ouvrent avec un format supérieur à celui de l'écran.

Exemple avec qt.

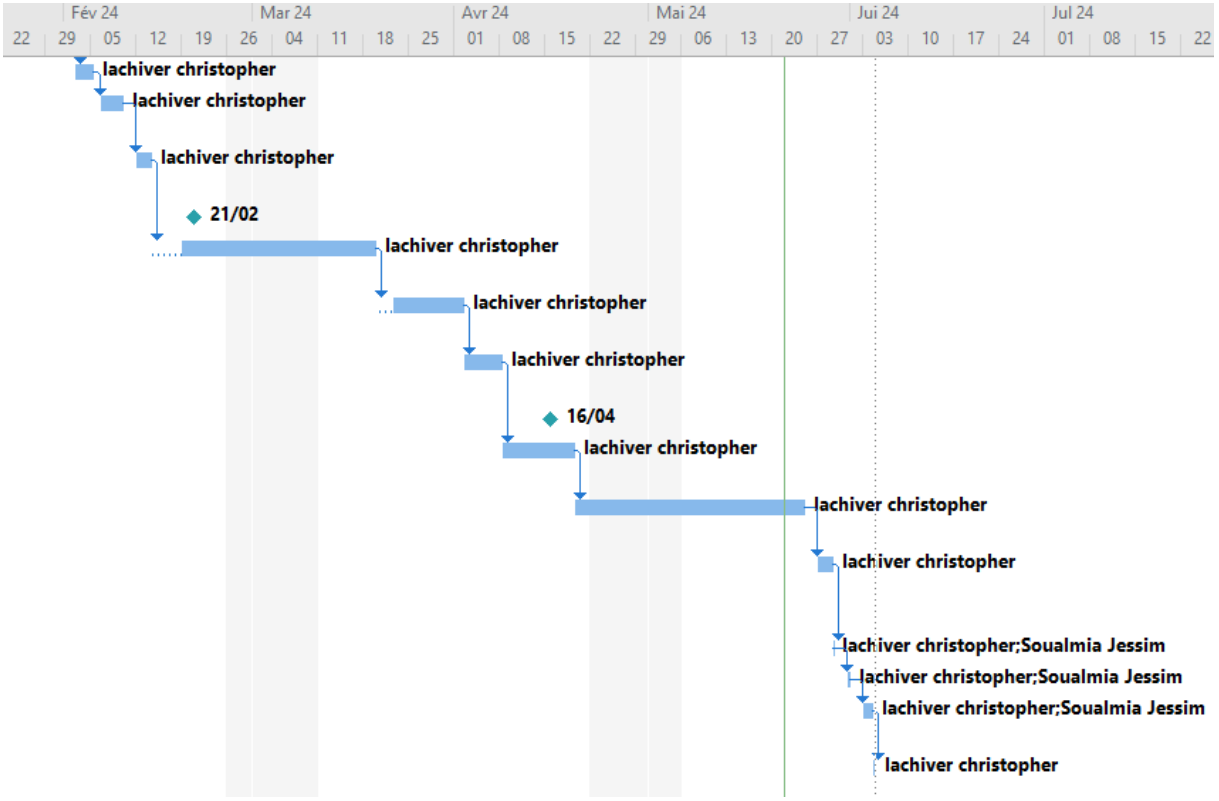


11 Diagramme de Gantt

voici le planning de mon projet pour l'itération finale:

DIAGRAMME DE GANTT

	Mode Tâche	Nom de la tâche	Durée	Début	Fin	Prédéces	Noms ressources
7		Fiches de recette	4 h	Ven 02/02/24	Lun 05/02/24	6	lachiver christopher
8		Rédaction du dossier de projet 1	10 h	Mar 06/02/24	Ven 09/02/24	7	lachiver christopher
9		création de l'interface graphique	8 h	Lun 12/02/24	Mer 14/02/24	8	lachiver christopher
10		Revue de projet 1	0 h	Mer 21/02/24	Mer 21/02/24		
11		mise en place de l'interface graphique	30 h	Mer 14/02/24	Mar 19/03/24	9	lachiver christopher
12		paramétrage de l'application	15 h	Mer 20/03/24	Mar 02/04/24	11	lachiver christopher
13		Rédaction du dossier de projet 2	10 h	Mar 02/04/24	Lun 08/04/24	12	lachiver christopher
14		Revue de projet 2	0 h	Mar 16/04/24	Mar 16/04/24		
15		Sauvegarde des données dans un fichier csv	20 h	Lun 08/04/24	Ven 19/04/24	13	lachiver christopher
16		acquisition des données des capteurs	25 h	Ven 19/04/24	Ven 24/05/24	15	lachiver christopher
17		finalaisation de tout le projets et résolution de bugs	8 h	Lun 27/05/24	Mer 29/05/24	16	lachiver christopher
18		test du projet	2 h	Mer 29/05/24	Mer 29/05/24	17	lachiver christopher;Soualmia Jessim
19		recette	3 h	Ven 31/05/24	Ven 31/05/24	18	lachiver christopher;Soualmia Jessim
20		finalisation du dossier de projet	4 h	Lun 03/06/24	Mar 04/06/24	19	lachiver christopher;Soualmia Jessim
21		Préparation de l'oral jusqu'à l'épreuve le 13/06/2024	3 h	Mar 04/06/24	Mar 04/06/24	20	lachiver christopher



11.1 présentation de la partie physique

Les capteurs de températures sont des dispositifs essentiels dans de nombreuses applications scientifiques, industrielles et quotidiennes. Ils permettent de mesurer la température et de convertir ces mesures en signaux compréhensibles et exploitables. Les principes physiques qui font fonctionner ces capteurs sont nombreux et adaptés à différents besoins.

la sonde pt100

La sonde Pt100 est un capteur de température qui est utilisé dans le domaine industriel (agroalimentaire, chimie,raffinerie…). Ce capteur est constitué d'une résistance en Platine. La valeur initiale du Pt100 est de 100 ohms correspondant à une température de 0°C.

Principe de mesure :

L'équation de calcul « linéaire » de la résistance d'une sonde Pt100 en fonction de la température est :

$R(\theta) = R_0 + \alpha . \theta$

avec R(θ) = résistance à θ °C, Ro = résistance à 0°C (100 W), α = coefficient de température de la sonde (0,385W/°C) et θ = la température en °C.

Exemple :

R(0°)=100 W

R(100°)=100+100*0,385 = 138,5 W

Pour obtenir une tension image de la température, on injecte un courant constant I_0.

$U=R(\theta)*I_0$

$U= (100+0,385*\theta)*I_0$

On mesure U, on en déduit θ

$$\theta = \frac{\frac{U}{I0} - 100}{0,385}$$

Exemple : considérons un courant $I_0 = 0,1$ mA, si on mesure une tension U = 11 mV cela correspond à une température θ :

$$\theta = \frac{\frac{11.10^{-3}}{0,1.10^{-3}} - 100}{0,385} = 25,97\ soit\ 26\ °C$$

Conclusion

Les équations linéaires sont approximatives et s'appliquent dans une plage de température restreinte, tandis que les relations dites “quadratiques” ou plus complexes sont nécessaires pour une plus large gamme de températures.

Dans le cas de notre projet les équations linéaires sont largement suffisantes car nous n’avons pas besoin d’une très grande précision.

Partie Personnelle
Soualmia Jessim

(EC 1)

Partie personnelle Soualmia Jessim

12. Présentation de ma partie :

Ma partie me concernant sur le Projets SERVELEC consiste à développer un système qui a pour fonction principale de mesurer la température dans des réfrigérateurs ou des fours et d'afficher la température sur petit écran qui sera sur la carte.

Les températures seront mesurées automatiquement selon un intervalle et une durée d'acquisition prédéfinis:

- réfrigérateur/congélateur :
- intervalle de temps entre mesures : 5 minutes
- durée totale de la capture : 1 semaine.

- four :
- intervalle de temps entre mesures : 5 secondes
- durée totale de la capture : 2h.

Les données relevées seront stockées dans un fichier au format CSV et sur une carte SD.

Pour cela je dispose du matériel suivant :

- Capteur de température DS18B20 (sonde)
- Sonde PT 100
- Raspberry Pi 0
- Breakout Max31865 de chez Adafruit
- Horloge temps réel DS3231MZ

Mon objectif sera de rassembler les programmes des 2 capteurs de température pour le froid et le chaud afin d'être capable d'afficher la température sur l'un des afficheurs.

On prévoit de faire une réalisation constituée 3 cartes empilables :

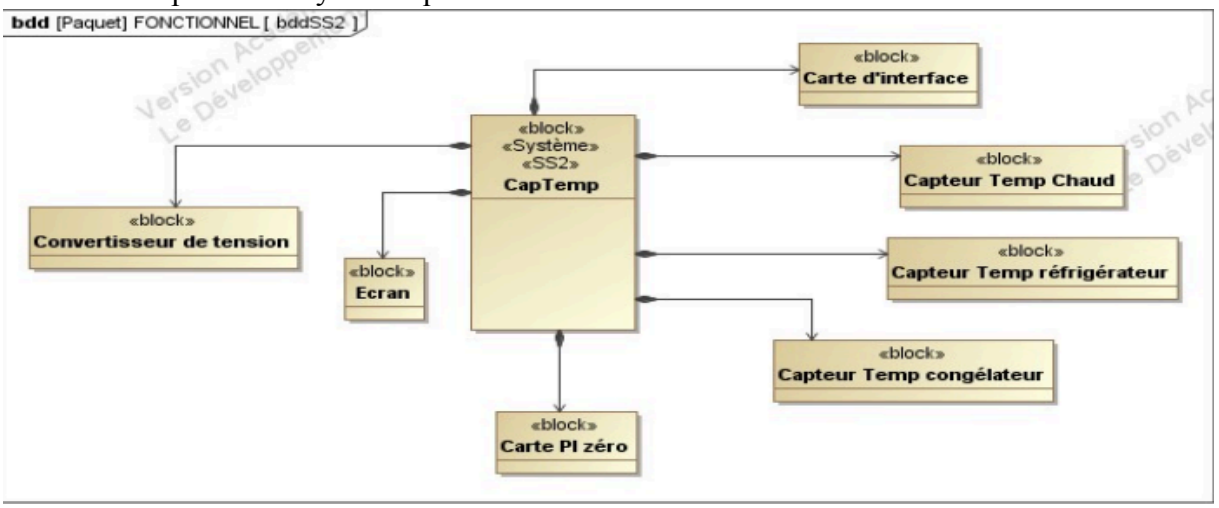
1ère partie qui sera l'afficheur pour la température

2ème partie qui seras le circuit imprimé rassemblant le DS18B20 et la PT100 et autres

3ème partie la Raspberry Pi0

13. Diagramme de blocs :

Ce diagramme de bloc représente l'architecture de la carte à mettre au point rassemblant différent composant du système qui sont :



- **Écran** : pour afficher la température.

- **Carte PI zéro** : Cela pourrait faire référence à une carte Raspberry Pi Zéro, qui est un petit ordinateur utilisé pour le projet. .

- **CapTemp** : relié à trois autres blocs qui représente les capteurs :

  - Capteur Temp Chaud : Un capteur mesurant la température dans un environnement chaud.
  - Capteur Temp réfrigérateur : Un capteur conçu pour mesurer la température à l'intérieur d'un réfrigérateur.
  - Capteur Temp congélateur : Un capteur pour mesurer la température dans un congélateur.

  L'utilisation de deux modèles de capteurs différents, le DS18B20 et la PT100, se justifie par leurs spécificités et leur complémentarité dans la mesure de températures dans des plages variées. Le DS18B20 est idéal pour les environnements froids (réfrigérateurs et congélateurs) avec sa plage de température de -55°C à +125°C, tandis que la PT100 est utilisée pour les mesures de température plus élevées (fours), grâce à sa plage étendue de -100°C à 200°C. Cette diversité permet une couverture complète des besoins de température du projet.

Avant de commencer à travailler les projets j'ai d'abord analysé tous les documents qui m'ont été fournis par l' enseignant puis j'ai effectué un diagramme de Gantt prévisionnel afin de terminer toutes les étapes que j'ai effectué durant les projets.

14. Diagramme de Gantt

Mode Tâche	Nom de la tâche	Durée	Début	Fin
	etude du dossier	1 h	Mar 09/01/2	Mar 09/01/2
	creation d'un drive plus document de travail	20 min	Mar 09/01/24	Mar 09/01/24
	planifiaction sur project sur 2h	1 jr	Mar 09/01/24	Ven 12/01/24
	préparation du matériel	30 min	Mar 09/01/2	Mar 09/01/2
	charte de présentation des documents et logo (fait tout au long de l'activitée)	200 h	Mar 09/01/24	Mer 05/06/24
	installation des librairie rpi 0	2 h	Ven 12/01/24	Lun 15/01/24
	mise en oeuvre des capteurs de température	50 h	Lun 15/01/24	Mar 13/02/24
	test	2 h	Mar 13/02/2	Mar 13/02/2
	mise en oeuvre de L'IPS LCD	25 h	Mar 13/02/24	Mer 13/03/24
	Fabrication de la carte	40 h	Mer 13/03/2	Ven 05/04/2
	kicad	10 h	Mer 13/03/2	Mar 19/03/2
	kicad	10 h	Mar 19/03/2	Lun 25/03/2
	kicad	10 h	Lun 25/03/2	Ven 29/03/2
	kicad	8 h	Ven 29/03/2	Mer 03/04/2
	bon de commande des composant	2 h	Ven 05/04/24	Lun 08/04/24
	correction des erreurs eventuelles	10 h	Lun 08/04/24	Mar 16/04/24
	test du projet	2 h	Mar 16/04/2	Ven 19/04/2
	correction erreurs eventuelles	20 h	Ven 19/04/24	Mer 22/05/24
	finalisation du dossier	4 h	Mer 22/05/2	Lun 27/05/2

15. Mise en œuvre du capteur DS18B20

Tout d'abord avant de commencer la mise en œuvre du capteur sur raspberry Pi0
J'ai terminé un tp qui concernait le DS18B20 ou sa mise en œuvre se faisait sur Arduino Uno.

Le capteur DS18B20 est un capteur de température communiquant sur le bus OneWire qui est un protocole de communication qui nécessite seulement un fil pour la transmission des données en plus du fil de masse et d'alimentation.

Caractéristique du capteur :



- Plage d'alimentation : 3V à 5V
- Plage de température : -55°C à +125°C
- Valeur température : 2 Bytes, nombre signé
- Family code : 28h
- Précision de ±0,5°C sur la plage de température de -10°C à +85°C

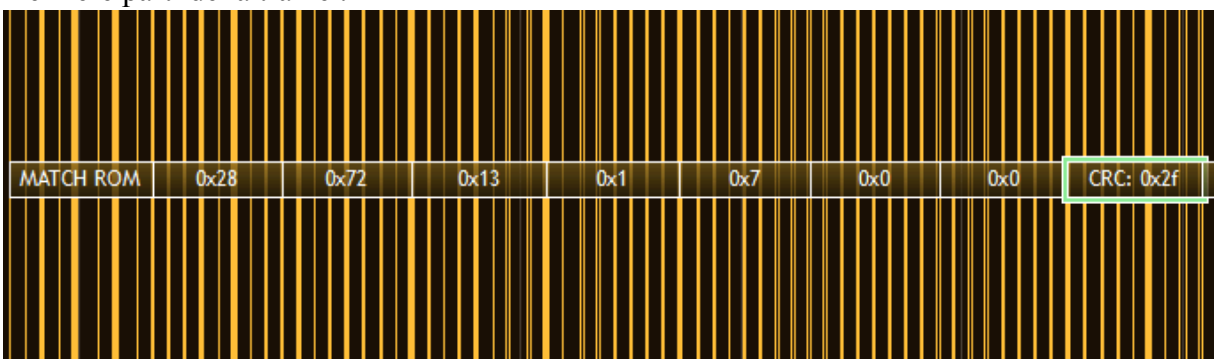
Ce capteur sera utilisé pour le congélateur et le réfrigérateur.

Après m'être documenté pour le capteur j'ai fait la mise en œuvre sur Arduino en téléchargeant la librairie et en utilisant l'exemple fournis afin de faire afficher la température.
Voici le résultat attendu du programme :

```
ROM = 28 72 13 1 7 0 0 2F
  Chip = DS18B20
  Data = 1 A4 1 4B 46 7F FF C 10 DA  CRC=DA
  Temperature = 26.25 Celsius, 79.25 Fahrenheit
```

Le rom représente l'identificateur du capteur
La data est la trame de données brute reçue du capteur. Elle contient plusieurs parties, y compris la valeur de température mesurée qui sont les 2 premier octets.
79.25 Fahrenheit représente la température mais cette échelle de température est généralement utilisée au Etats-Unis

J’ai ensuite un relevé de trame sur le logiciel SCANA STUDIOS :
Première parti de la trame :

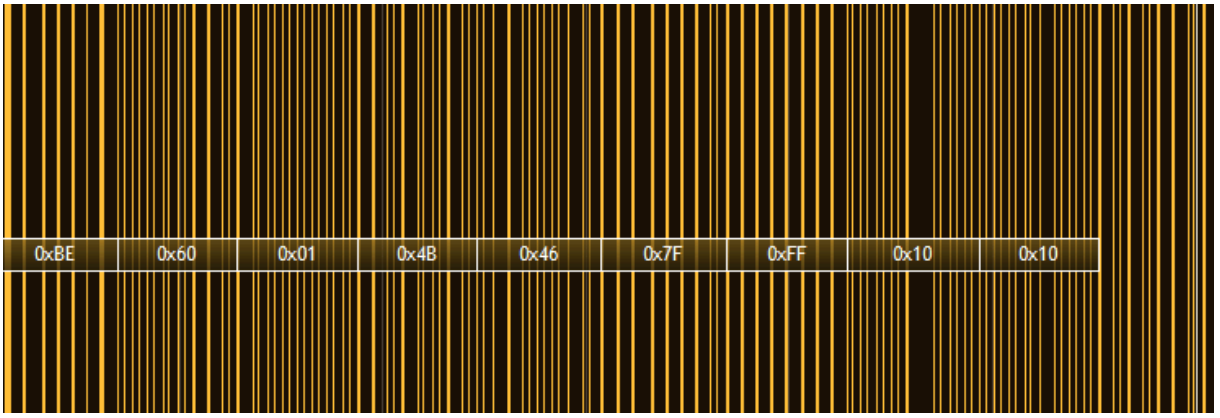


Voici une description plus détaillée des éléments de la trame en tenant compte que cela provient d'un capteur DS18B20 :

1. **MATCH ROM (0x55) :** Dans le protocole 1-Wire, la commande MATCH ROM permet de sélectionner un dispositif particulier pour communiquer avec lui. Chaque DS18B20 possède un code ROM unique de 64 bits qui peut être utilisé pour s'adresser directement à lui lorsqu'il y a plusieurs capteurs sur le même bus 1-Wire.

2. **Codes hexadécimaux :** Après le Match Ces chiffres en hexadécimal représentent l'adresse du capteur spécifique. Par exemple, 0x28 est le code de famille pour les capteurs DS18B20, et le reste est le numéro de série qui est 0x000007011372.

3. **CRC (0x2F) :** Il s'agit d'une forme de vérification d'erreur utilisée dans les communications numériques pour détecter les erreurs accidentelles dans les données transmises. Le "0x2F" est la valeur calculée du CRC pour les données précédentes.
   Voici la deuxième partie de la trame :



4. **0xBE** : READ SCRATCHPAD permet de lire les donnée (octets+CRC)

5. **0x60,0x01** : Ses deux codes hexadécimaux représente la température
   Pour trouver la température il suffit d'inverser ses deux octets donc ça nous donne 0x0160 puis on le convertit en décimal. On inverse ses 2 octets car 0x01 l’octet de poids fort 0x60 est octets de poids faible.

Après on converti en décimal les 2 octets ça nous donne :

0x0160 = 352

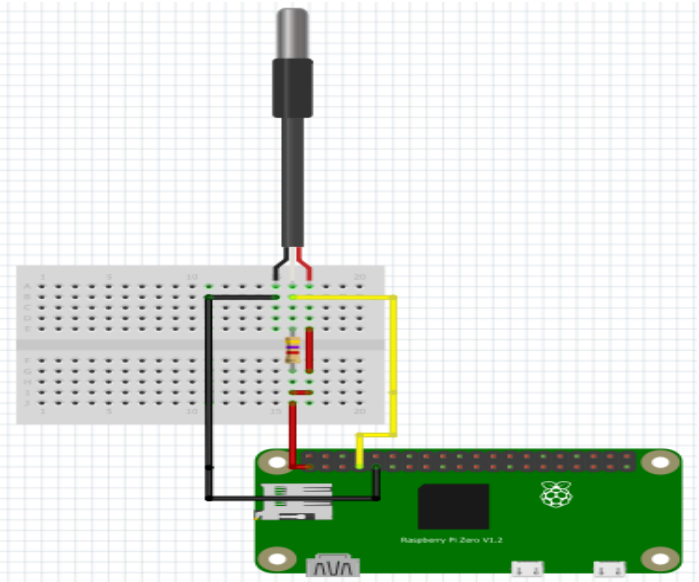
Puis on le multiplie par son coefficient de température qui est de 0.0625°C.

Temp = 352*0.0625 = 22°C
On utilise ce coefficient de température car sa résolution est de 12 bits par défauts comme nous l'indique sa documentation :

Operation—Measuring Temperature

The core functionality of the DS18B20 is its direct-to-digital temperature sensor. The resolution of the temperature sensor is user-configurable to 9, 10, 11, or 12 bits, corresponding to increments of 0.5°C, 0.25°C, 0.125°C, and 0.0625°C, respectively. The default resolution at power-up is 12-bit. The DS18B20 powers up in a low-

Après avoir fait le travail sur ARDUINO j'ai dû mettre en œuvre le capteur sur la raspberry Pi0 mais j'ai d'abord effectué un schéma fritzing :



Après validation du professeur j'ai effectué le câblage et l'installation des librairies sur la RPI afin de pouvoir faire fonctionner le capteur. Pour cela voici comment j'ai procédé à l'installation :

J'ai câblé le capteur sur le GPIO4

Configurer Le bus 1-wire dans : Préférences → Configuration du Raspberry Pi → Interfaces

Installer les librairies :

- build-essential et cmake
- sudo apt-get install build-essential
- sudo apt-get install cmake
- Installer la librairie bcm2835 en suivant la procédure d'installation :
    - tar zxvf bcm2835-1.xx.tar.gz
    - cd bcm2835-1.xx
    - ./configurer
    - faire
    - sudo faire un chèque
    - sudo make install
- Installer la Librairie DS18B20 en suivant la procédure d'installation :
    - Git clone https://github.com/nilshenrich/DS18B20_cpp
    - cd DS18B20_cpp
    - mkdir build
    - cd build/ c
    - make ..
    - make sudo make install
    - ./compileTest.sh (pour exécuter le programme)
    - cd build/ ./DS18B20 (pour lancer le programme)

Tester le capteur avec le programme suivant :

```
#include "ds18b20.h"

#include <bcm2835.h>
#include <iostream>

using namespace std;

int main()
{
    // Initialize bcm2835 library
    if (!bcm2835_init())
    {
        cerr << "Initialize BCM2835 failed!" << endl;

        return -1;
    }

    // Set GPIO 4 to input with pullup
    bcm2835_gpio_fsel(4, BCM2835_GPIO_FSEL_INPT);
    bcm2835_gpio_set_pud(4, BCM2835_GPIO_PUD_UP);

    DS18B20 ds18b20;

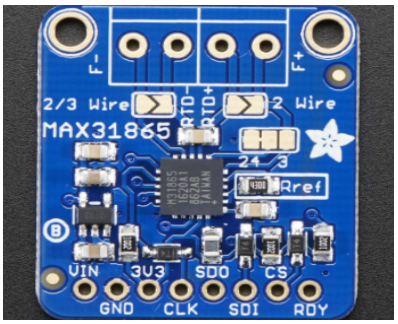
while(1)
{
    cout << "Temperature = " << ds18b20.readTemp() << " degrees Celsius" << endl;
    delay(100);
}
    return 0;
}
```

15. Mise en œuvre de la Sonde PT100

Les capteurs de type PT100 sont des dispositifs de mesure de température très précis et fiables. Le terme PT100 fait référence à un type spécifique de thermistance à résistance de platine dont la résistance est de 100 ohms à 0°C. Possède une plage de température plus grande que DS18B20 qui est de -100°C à 200°C donc ce capteur serait utilisé pour le chaud.

Pour les applications nécessitant la mesure de températures particulièrement élevées, une sonde PT100 est préférée en raison de sa capacité à opérer à des températures beaucoup plus élevées et de sa précision supérieure. En revanche, pour des applications nécessitant une intégration numérique facile, un coût plus bas, et des mesures dans une plage de température plus modeste, le DS18B20 pourrait être plus approprié.

Pour sa mise en œuvre je suis fournit de la breakout MAX31865 qui est une carte qui va me permettre d'utiliser la sonde PT 100 en 3 fil sur le bus SPI. Cette carte contient le circuit intégré MAX31865 qui est un convertisseur analogique numérique (ADC). La breakout peut être câblé soit en 2 fils, 3 fils, 4 fils.



Caractéristiques des broches :

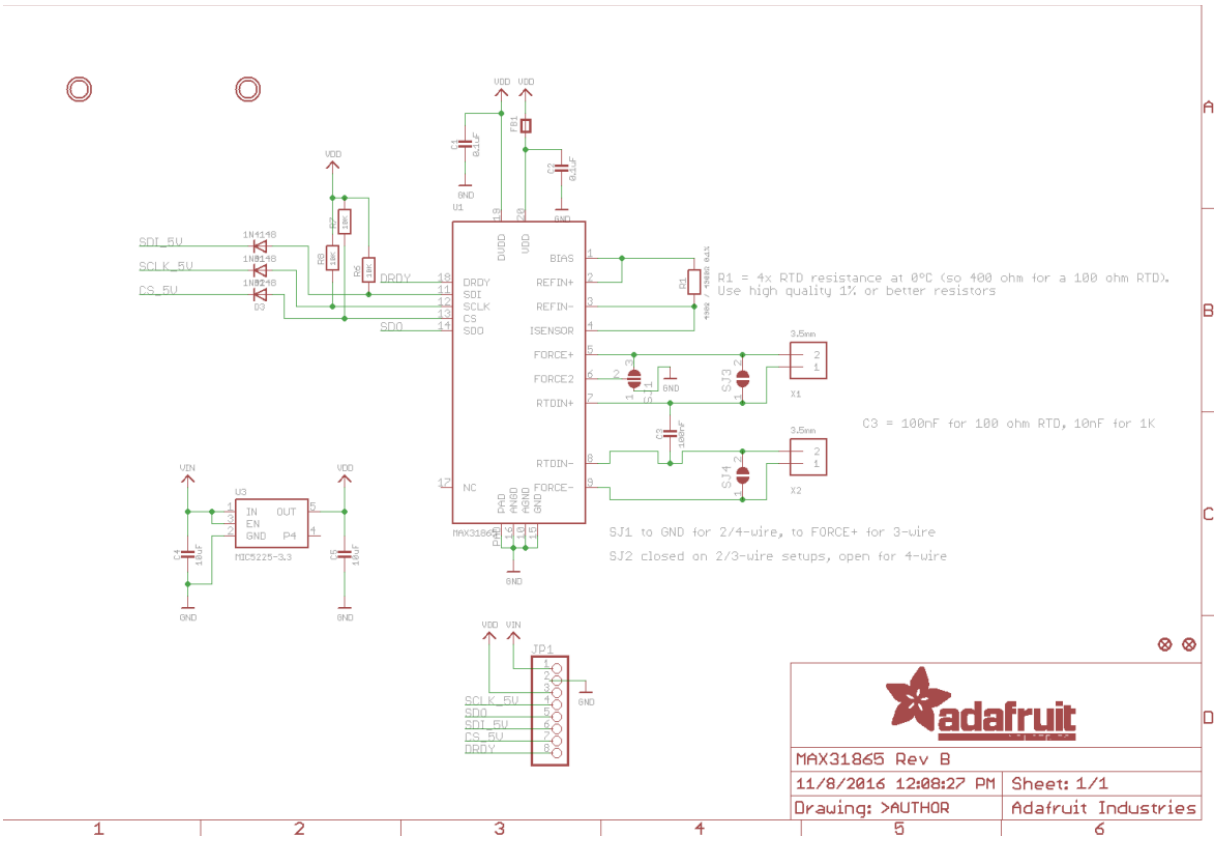
- **Vin** - c'est la broche d'alimentation. Étant donné que la puce utilise 3 VDC, inclus un régulateur de tension intégré qui prendra 3 à 5 VDC et le convertira en toute sécurité.
- **3Vo** - c'est la sortie 3,3 V du régulateur de tension.
- **GND** - terrain d'entente pour la puissance et la logique.
- **SCK** - Il s'agit de la broche de verrouillage S PI C, c'est une entrée dans la puce.
- **SDO** - il s'agit de la broche de sortie
  de données série / entrée de capteur du microcontrôleur,
- **SDI** - il s'agit de la sortie du microcontrôleur, pour les données envoyées depuis votre processeur vers le MAX31865.
- **CS** - c'est la broche d'élection du Chip S

On a choisi le fils car ces configurations dépendent principalement de la balance entre le coût, la distance de câblage et la précision requise. Une connexion à 2 fils peut donner des résultats acceptables lorsque la RTD est située près du MAX31865. Notez que, pour une PT100, une résistance en série de 0,4 Ω cause une erreur d'environ 1°C. Par conséquent, à mesure que la longueur du câble augmente, l'erreur due à la résistance du câble peut devenir excessive.

Tandis que celle à 4 fils offre la meilleure précision mais à un coût plus élevé.

La configuration à 3 fils offre un compromis pratique entre les deux, fournissant une précision suffisante pour la plupart des applications industrielles sans le coût supplémentaire de la configuration à 4 fils.

Avant de pouvoir souder la carte et la mettre en œuvre j'ai étudié le schéma structurel de la carte :



Sur le schéma il y a :

1- R1 est la résistance de réfférence Reff d'une valeur 430Ω

2- **Le convertisseur MAX31865** en U1 qui convertit la résistance variable du RTD possédant plusieurs broches de connexions permettant de communiquer en SPI (SDI, SDO, SCLK, CS) et les broches de connexion (RTDIN+, RTDIN-, FORCE+, FORCE-).

3- **JP1** est un connecteur qui permet de connecter les broches SPI

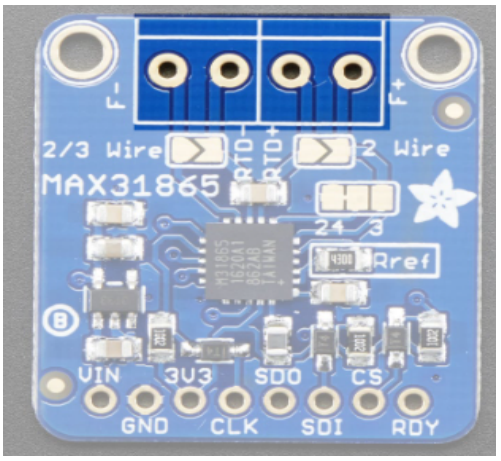
4- **U3** est un régulateur de tension

5- **SJ1 et SJ3** sont des points sur le circuit imprimé qui va nous permettre de souder les cavaliers pour configurer le capteur en 2 fils, 3fils et ,4 fils. Pour la PT100 elle sera en 3 fils. Dans notre cas x1 seras souder afin de mettre le troisième fils.

Après m'être renseigner sur la breakout j'ai souder la carte afin de pouvoir la mise en œuvre du capteur :

Pour cela j'ai suivi les étapes sur le site d' Adafruit :

Tout d'abord j'ai commencé par souder les borniers de capteurs afin de pouvoir relier la sonde PT 100 qui seras câbler en 3 fils.



Il y a *quatre* contacts, pour utiliser des capteurs à 2, 3 ou 4 fils. Pour le projet j'ai utilisé celui à 3 fils car elle permet de compenser la résistance d'un des fils de connexion, ce qui améliore la précision

Ensuite pour que le câble marche en 3 fils j'ai soudé la carte pour fermer le cavalier 2/3 Wire et fermer la soudure à droite sur le chiffre 3 afin qu'elle soit configuré en 3 fils.

Couper la fine trace entre le cavalier bidirectionnel sur le côté droit de la carte**,** puis souder la goutte sur le côté droit**.**

Après câblage j'ai testé la carte sur Arduino en utilisant le programme qui ma était fournis sur le site et voici le résultat attendu :

```
Adafruit MAX31865 PT100 Sensor Test!
RTD value: 8336
Ratio = 0.25439453
Resistance = 109.38964843
Temperature = 24.08
```

Pour trouver la valeur de la résistance le calcul et le suivant :

On prend la valeur du RTD value : 8336

On le divise par 32768 ce qui donne le ratio qui est de 0.025439453

Cette valeur on le multiplie par Reff qui est de 430 Ω.

Donc la formule est :

R= (RTD value /32768) *430

= 8336/32768*430= 109.389683Ω.

Pour trouver la température j'utilise la valeur de la résistance trouver et je cherche cette valeur dans un tableau de température.

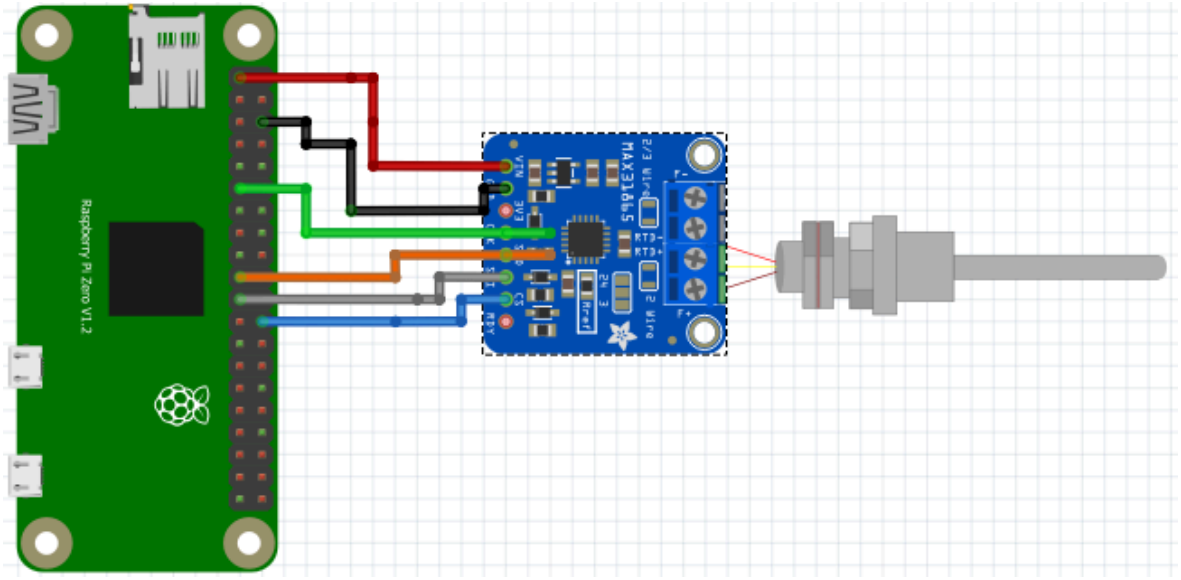
Sinon il existe une autre façon de trouver la température mais que si on la valeur de la résistance qui est afficher donc le calcul est le suivant :

$$T = \frac{valeur\ de\ la\ résistance - résistance\ à\ 0°C}{coefficient\ de\ température * 100\Omega}$$

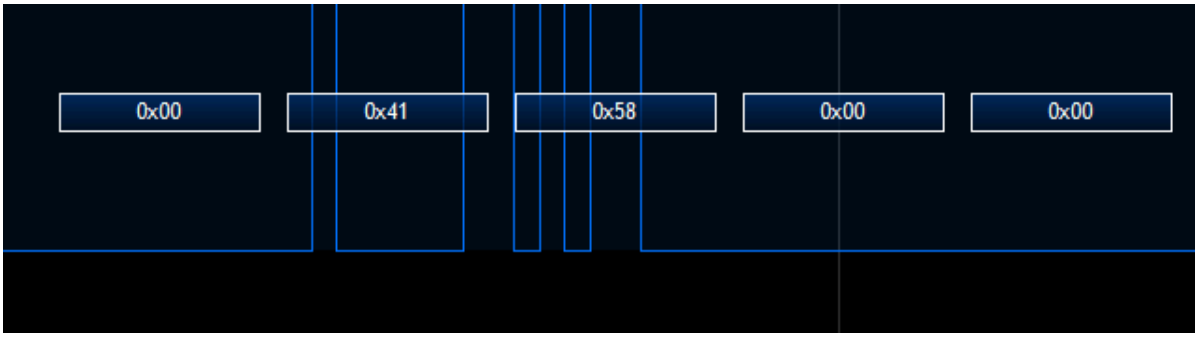
$$= \frac{109.38-100}{0.0039083 * 100} \qquad = 24.024°C$$

Utilisant cette formule on constate qu'il y a une petite marge d'erreur de 0.05 mais dans notre cas ceci n'est pas vraiment important car on ne nous demande pas une grande précision.

Après avoir fait la mise en œuvre sur Arduino je me suis redirigée sur la raspberry Pi 0 afin de tester le programme qui mon été fournis par le professeur mais avant ça j'ai le montage de la carte sur fritzing :



Après avoir tester le programme j'ai utilisé un analyseur logique afin de relever une trame :



Les deux octets qui sont 0x41 et 0x58 représente la valeur RDT du capteur donc pour la trouver il suffit de convertir c'est 2 octets en format binaire :

0x41 = 0100 0001

0x58= 0101 1000

Ce qui donne si on les rassemble :

0x4158= 0100 0001 0101 1000 d'une valeur de 16728 en décimal

Pour avoir la valeur RTD ils suffit décaler d'un bit vers la droite.

0100 0001 0101 1000 = 0010 0000 1010 1100

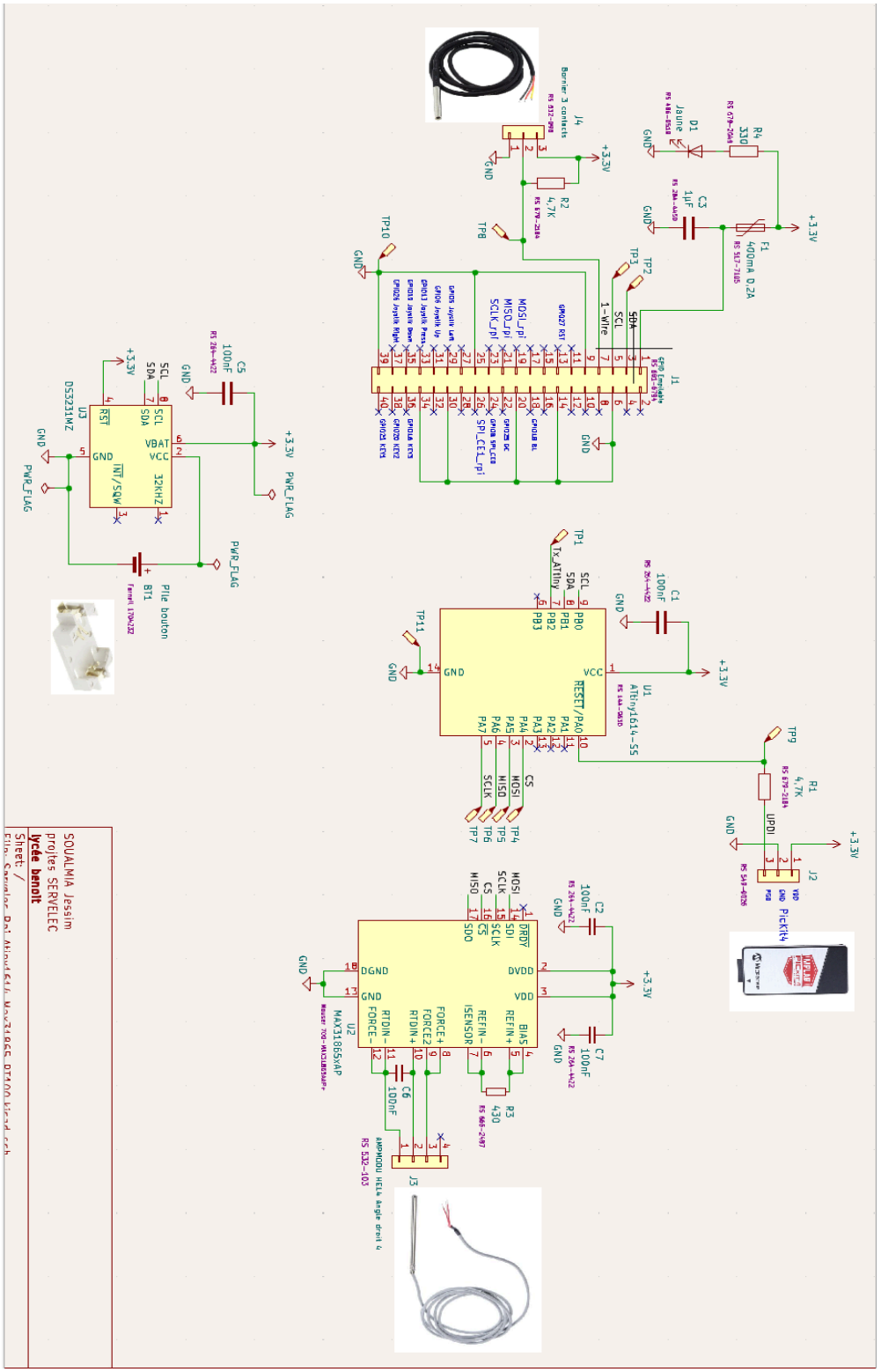
Après avoir décalé le bit

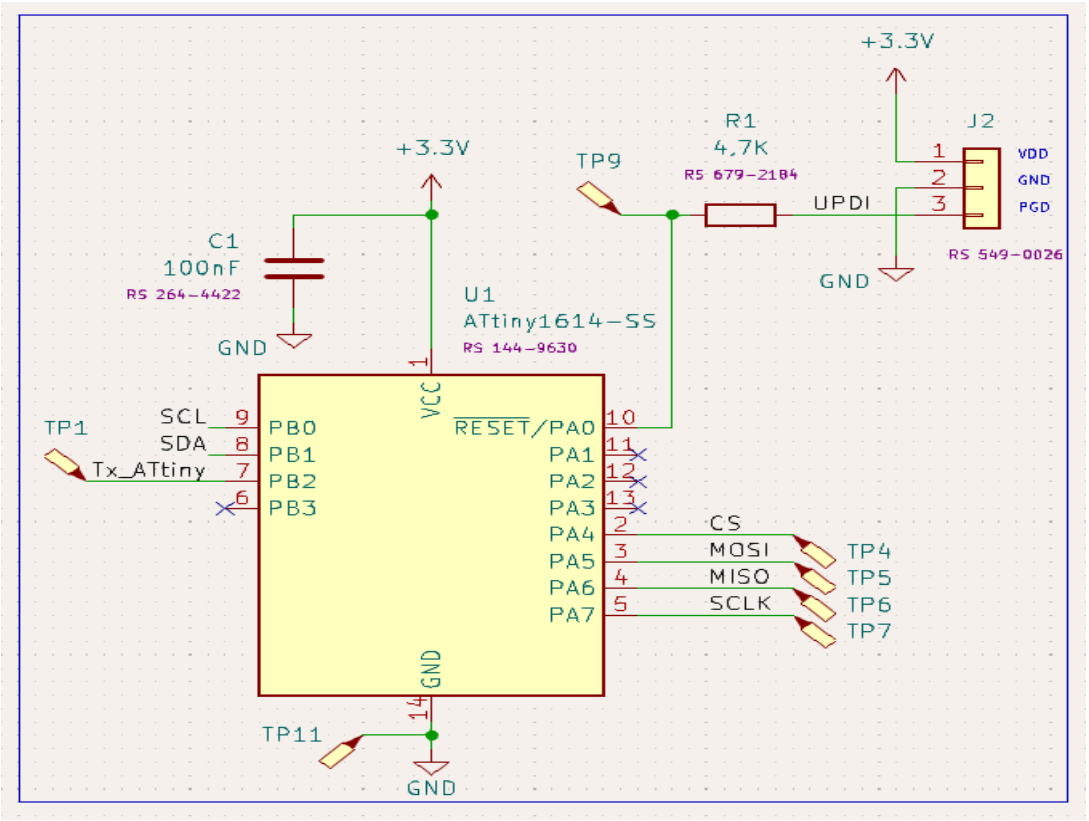
La valeur décimale devient 8364 soit la valeur RTD relevée quand on teste le programme.

Le décalage à droite lors de l'interprétation des données du capteur RTD est une étape nécessaire pour convertir les données brutes en une valeur utilisable. Ce décalage ajuste la précision des mesures en alignant correctement les bits de données pour le calcul de la résistance. Ce processus est crucial pour obtenir une mesure précise de la température à partir de la valeur de résistance calculée du capteur RTD.

Pour donner suite aux projets j'ai effectué un schéma structurel sur le logiciel Kicad. Ce schéma représente le circuit électronique de la carte que je dois mettre aux points incluant les deux capteurs de température ainsi qu'un Attiny 1614, la Raspberry PI0 et l'horloge temps réel.

Voici le schéma structurel :

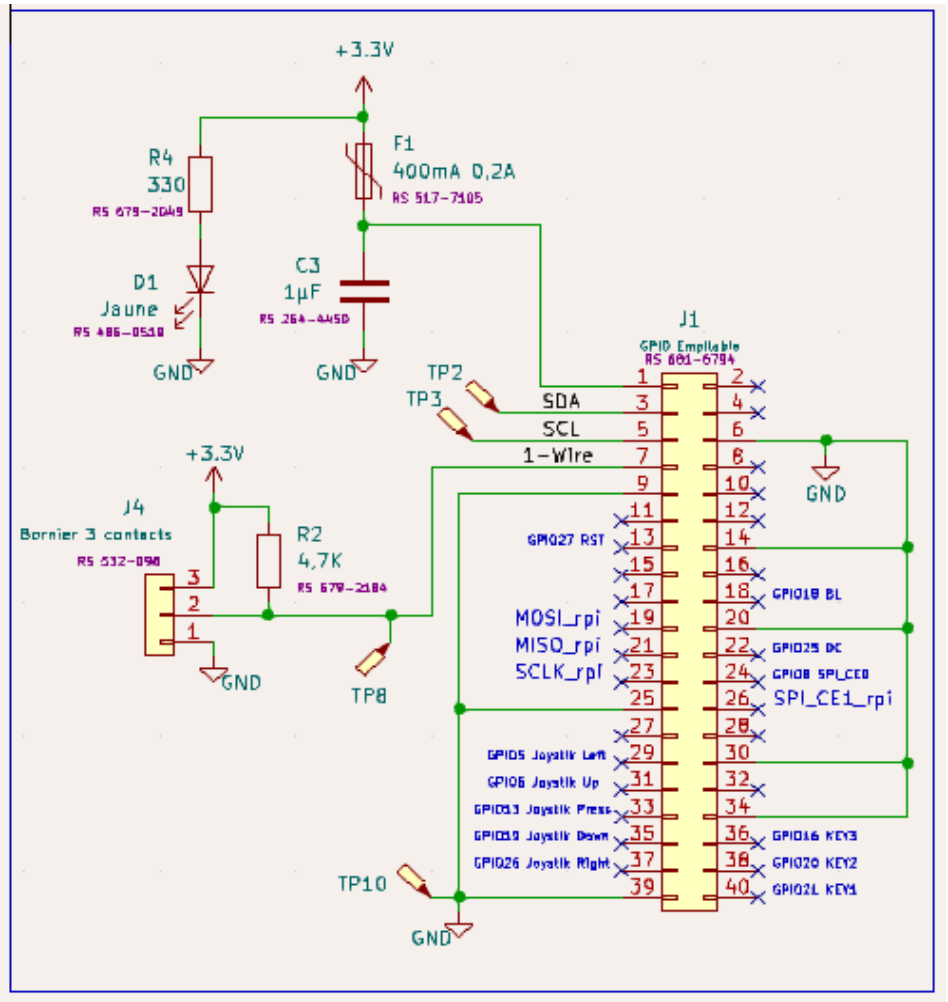




En se servant de l'ATtiny1614 comme cerveau de notre carte, nous sommes capables de transformer les données capturées par la sonde PT100 au travers du MAX31865, utilisant le protocole SPI, accessibles via le bus I2C. Ce microcontrôleur va agir non seulement comme un processeur, mais aussi comme un traducteur entre différents langages de bus de communication.

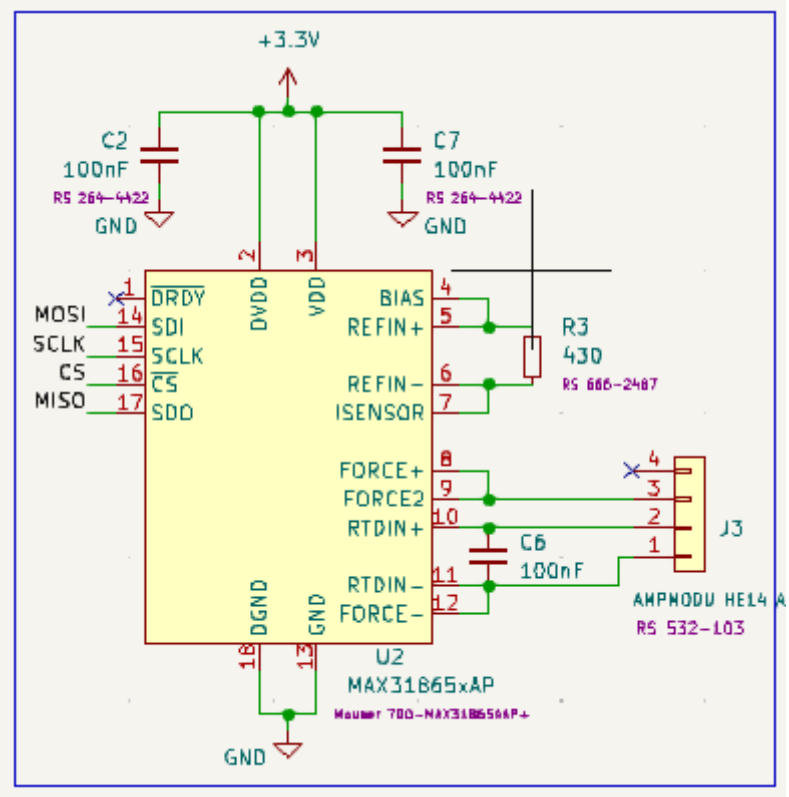
On a opté pour cette solution afin d'éviter d'utiliser trop de gpio en spi sur Raspberry PI 0.

- **Résistance (R1)** : La résistance de 4.7kΩ est utilisée ici comme résistance de pull-up pour la ligne UPDI, qui est une interface pour la programmation et le débogage des microcontrôleurs de la série ATtiny. Elle permet de maintenir la ligne à un niveau logique haut par défaut, garantissant ainsi des conditions stables pour les communications et évitant les états flottants qui pourraient causer des erreurs de communication ou un comportement indéterminé.
- **Condensateur (C1)** : Le condensateur de 100nF est un condensateur de découplage, placé très près de l'alimentation du microcontrôleur ATtiny 1614. Son rôle est d'agir comme une petite réserve d'énergie pour compenser les variations rapides de la demande en courant par le microcontrôleur. Il aide à stabiliser la tension d'alimentation du microcontrôleur en filtrant le bruit électrique et les pics de tension qui peuvent survenir sur la ligne d'alimentation.



Dans ce schéma électronique, la résistance et le condensateur ont des fonctions spécifiques qui sont essentielles pour assurer la performance et la fiabilité du circuit :

- **Résistance (R2)** : La résistance de 4.7kΩ est une résistance de pull-up pour la ligne pour la ligne 1-Wire. Ces résistances sont utilisées pour assurer que les lignes de données sont maintenues à un niveau logique haut quand aucun dispositif n'est en train de les tirer à un niveau bas.
- **Condensateur (C3)** : Le condensateur de 1µF est utilisé pour stabiliser la tension d'alimentation auxiliaire ou comme condensateur de découplage pour un sous-système particulier du circuit.
- **Fusible (F1)** : Ce fusible de 400 mA est utilisé pour protéger le circuit contre les surintensités qui pourraient endommager les composants.
- **Connecteur (J1)** : Il s'agit du GPIO empilable qui seras sur la carte afin de la connecter sur la Raspberry PI 0 et l'écran IPS que le candidat IR mettra en œuvre
- **Bornier (J4)** : C'est un bornier à trois contacts qui nous serviras à câbler le DS18B20

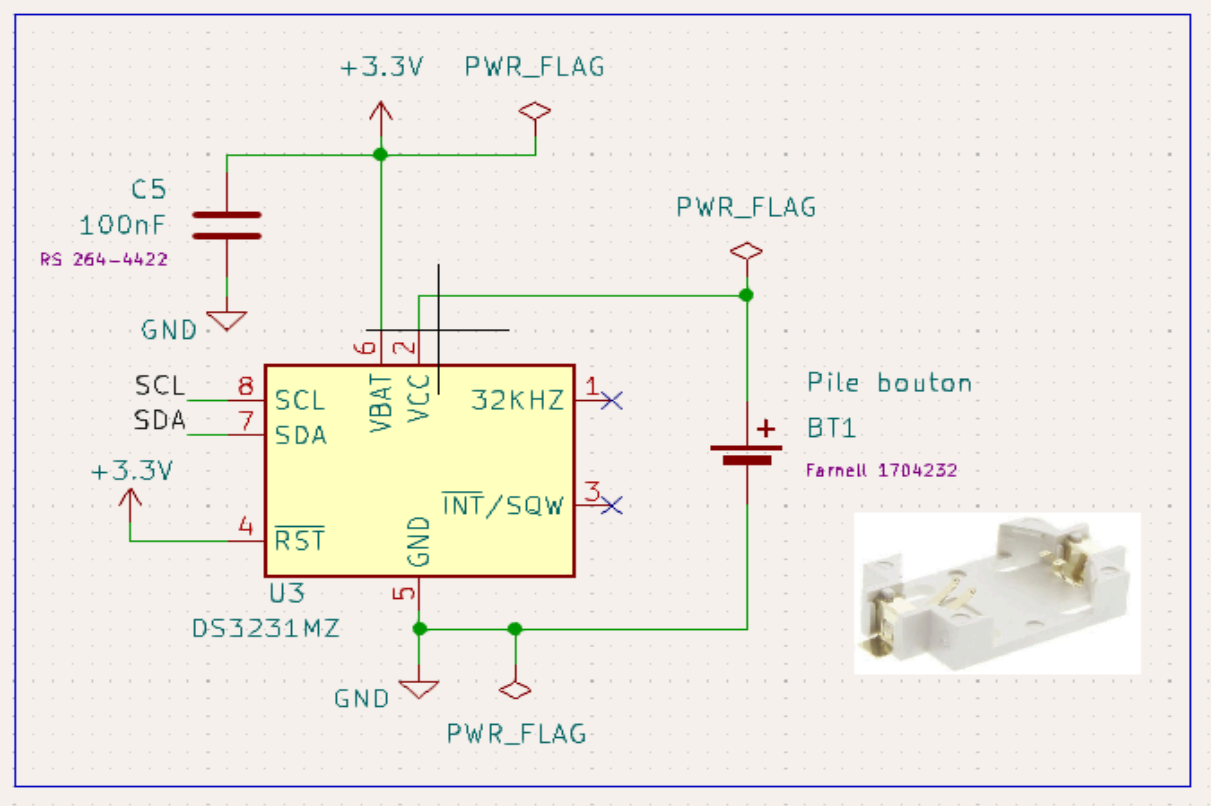


- **MAX31865** : Ce circuit intégré est au cœur du module et s'occupe de convertir la résistance variable du capteur RTD en un signal numérique que le microcontrôleur peut lire. Il est conçu spécifiquement pour les capteurs RTD et gère toute la complexité de la lecture de la résistance précise.

- **Résistance (R3)** : La résistance de référence 430Ω est utilisée pour fixer la référence de courant pour le capteur RTD. Dans le cas des capteurs PT100, c'est une partie importante du circuit car elle influence directement la précision des mesures de température.

- **C2** et **C7** sont des condensateurs de découplage de 100nF utilisés pour filtrer le bruit de l'alimentation électrique et assurer un fonctionnement stable du MAX31865.

- **C6** est un condensateur de 100nF utilisé pour la stabilité du filtre interne du MAX31865.

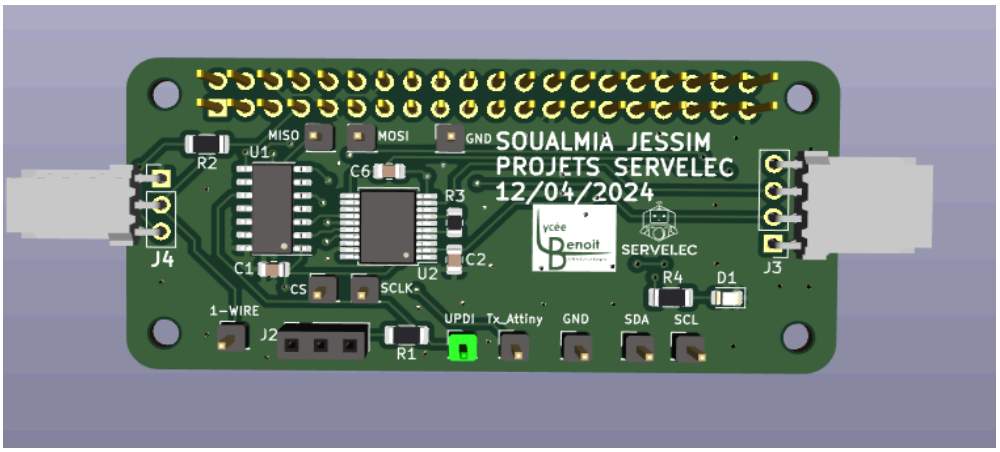


Ce schéma présente une section d'un circuit impliquant un module de gestion du temps et d'interface utilisateur, en l'occurrence une horloge temps réel (RTC) et un bouton poussoir. Voici une explication des composants clés :

- **Horloge Temps Réel (RTC) DS3231MZ** : U3 est le circuit intégré DS3231MZ, une horloge temps réel de précision avec une compensation de température intégrée. Elle est utilisée pour maintenir l'heure exacte dans une large gamme de températures et est souvent utilisée dans des applications nécessitant une horloge précise, comme les systèmes d'enregistrement de données ou les réveils.

- **Condensateur (C5)** : Le condensateur de 100nF est un condensateur de découplage destiné à stabiliser l'alimentation du DS3231MZ en filtrant le bruit et en fournissant une réserve d'énergie pour les courts moments où la demande en courant peut dépasser ce que l'alimentation peut fournir immédiatement.

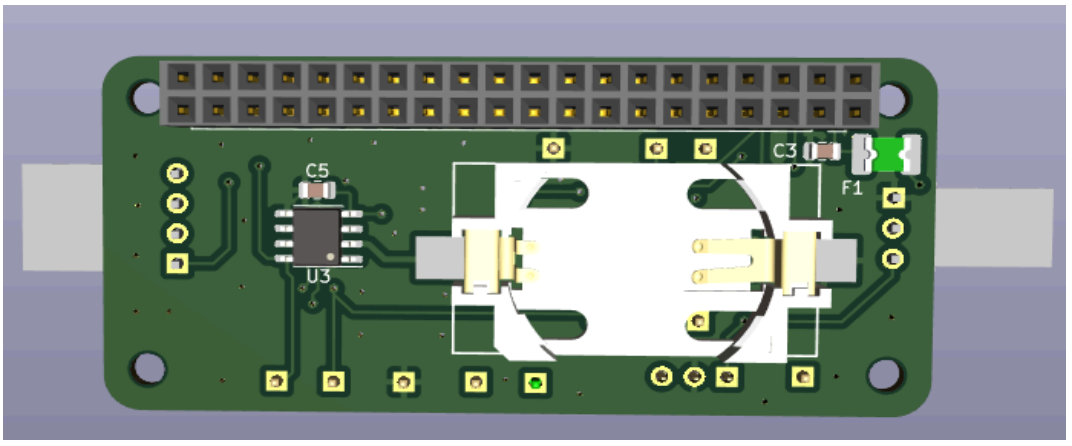
Pour donner suite au schéma structurel de la carte j'ai dû effectuer son routage :



Du côté top de la carte nous avons placé les tous les composant qui concerne le relevé de température comme :

- L'ATiny 1614 et le max31865 cote a cote et proche du GPIO SDA et SCL
- Les borniers pour la Pt 100 et le DS18B20
- Le bornier DS18B20 qui à gauche qui est proche du gpio du bus 1-wire
- Les points test pour les analyseurs logique
- Les connecteurs femelles qui permet de programmer l'ATiny
- Le condensateur de découplage proche des microcontrôleurs

Et du côté BOTOM nous avons placé le reste des composants :



- L'horloge temps réels DS3231MZ ainsi que son condensateur de découplage
- La pile bouton

Après routage de la carte j'ai ensuite généré les fichiers gerber et essayer le fichier sur JLC PCB pour voir si le fichier était conforme.

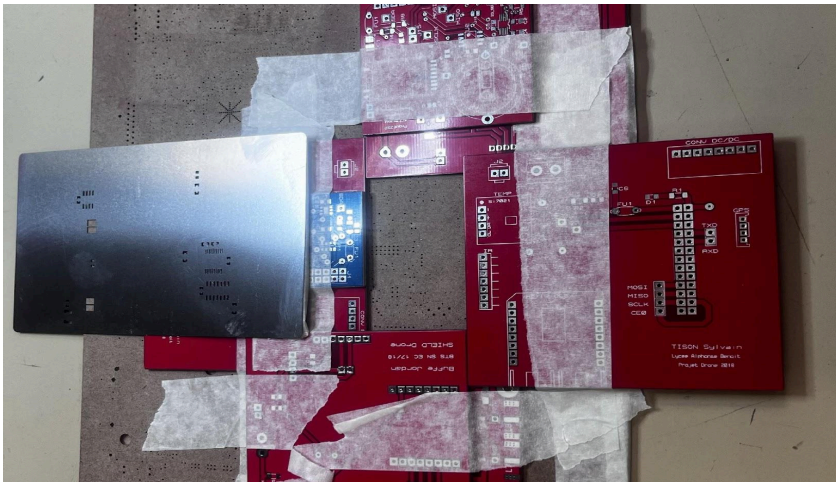
Pour donner suite à ça j'ai effectué le bon de commande des composant de la carte :

Lycée A. Benoit B.P. 118 84803 L'Isle sur la Sorgue cedex Tél. 04 90 20 64 20 Fax 04 90 20 57 20	FOURNISSEUR Mouser Farnell RS Composant Fax : 0 825 345 000	BON DE COMMANDE Référence à rappeler sur la facture et le bon de livraison		BON Cpte	N° T N°
Désignation	**Code commande**	**Qté**	**PUHT**	**PTHT**	
Point test (1*36)	Gotronic 08000	1	0,500	0,500	
Résistance 430 ohm (R3)	RS-666-2487	1	0,468	0,468	
Résistance 330 ohm (R4) 1*50	RS-679-2049	1	0,029	0,029	
Résistance 4,7k ohm (R1, R2) 1*50)	RS-679-2184	2	0,029	0,058	
Bornier 3 contact (J4)	RS-532-096	1	0,479	0,479	
Bornier 4 contact (J3)	RS-532-103	1	0,584	0,584	
J2	RS-549-0026	1	5,176	5,176	
GPIO empilable (J1)	Rs 681-6794	1	5,700	5,700	
Polyfuse (F1) 1*10	RS-517-7105	1	0,369	0,369	
Condensateur de 1µF (C3) 1*25	RS-264-4450	1	0,028	0,028	
Condensateur de 100 nF (C1-C7) 1*25	RS-264-4422	5	0,116	0,580	
Led Jaune (D1) 1*25	RS-486-0519	1	0,244	0,244	
ATiny1614-SS (U1) 1*10	RS-144-9630	1	0,835	0,835	
MAX31865 (U2)	Mouser 700-MAX31865AAP+	1	7,530	7,530	
DS3231MZ (U3)	Farnell 2515487	1	8,790	8,790	
Pile bouton (BT1)	Farnell 1704232	1	3,130	3,130	
TOTAL HT				**34,5**	
BTS SN2			**TOTAL TTC**	**41,4**	
Le 14 / 05 /2024					
L'Isle sur la Sorgue					
L'Intendant ,					
	Date limite de livraison ou d'exécution :				

Après avoir effectué le bon de commande avant les vacances à la rentrée nous avons reçu les composants et nos cartes donc j'ai pu commencer à câbler ma carte.

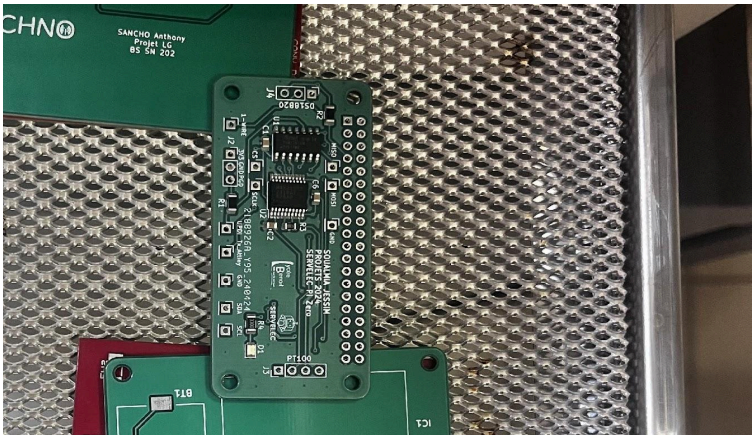
Pour commencer à câbler la carte j'ai d'abord mis de la pâte à brasser du côté Bottom car c'est le côté qui possède le moins de composant. Pour appliquer la pâte correctement pour les composants cms j'ai dû utiliser un stencil.

Mais j'ai dû faire en sorte de maintenir la carte appliquer la pâte à braser :



Après avoir appliqué la pâte et les composant j'ai mis la carte à four à fusion afin de les souder

Après avoir utilisé le four j'ai attendu que la carte refroidisse et j'ai appliqué la pâte du côté top ainsi que ces composants et remettre la carte aux fours.

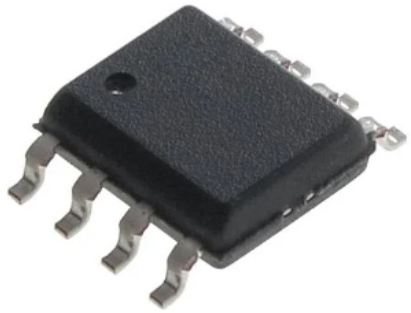


Pour souder les composants du côté top nous avons mis la carte en hauteur dans le four afin que le côté botom ne soit pas impacté par la chaleur et dessouder les composants. Pour donner suite j'ai pu mettre en œuvre la carte mais avant de tester les capteurs etc... J'ai d'abord testé les alims de la carte ensuite j'ai câblé la PT100 et le DS18B20 et tout fonctionne correctement.

Après avoir souder les cms j'ai ensuite souder les traversant et le support de pile de la pile bouton. En soudant la pile bouton j'ai remarqué que juste en dessous il y avait un condensateur de découplage donc on a décidé de l'enlever car le circuit possède déjà un autre condensateur de découplage donc dans notre utilisation cela n'aller endommager la carte.

16. Mise en œuvre du DS3231MZ

Le DS3231M est une horloge temps réel (RTC) I2C très précise et à faible coût souvent utilisée dans les applications nécessitant une gestion exacte du temps, comme les horloges, les systèmes de synchronisation, et les dispositifs enregistrant des données temporelles. La RTC maintient les informations de secondes, minutes, heures, jour, date, mois et année.



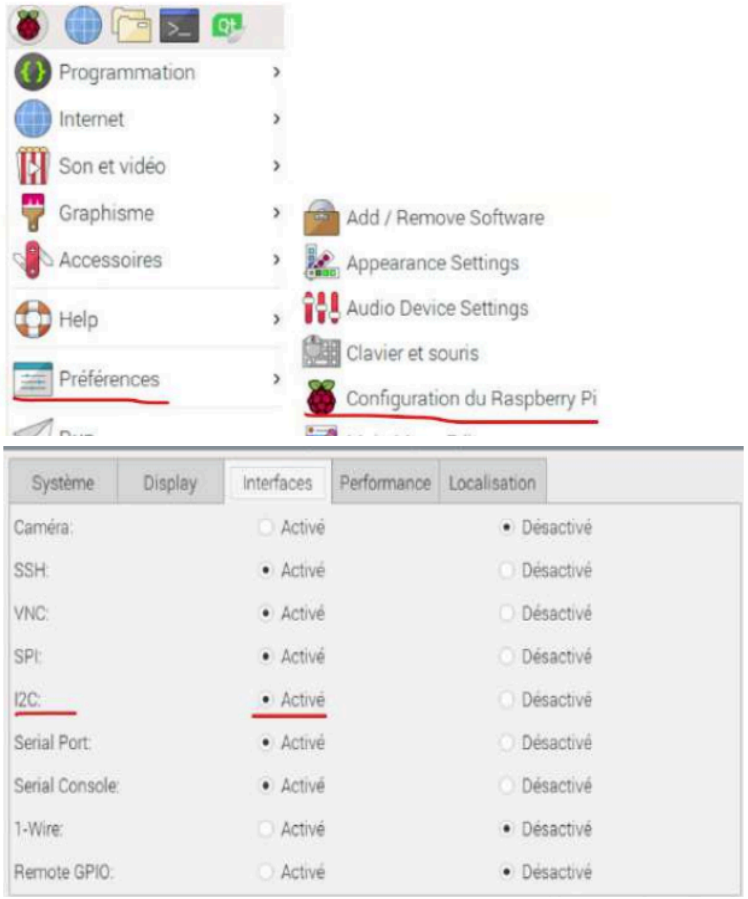
Caractéristique de la RTC :

- **Fonctionnalité complète de l'horloge et du calendrier** :
  - Secondes, minutes, heures, jour, date, mois, année
  - Compensation des années bissextiles jusqu'en 2100
- **Précision de chronométrage** :
  - ±5 ppm (±0,432 seconde/jour) de -45°C à +85°C
- **Interface de communication** :
  - Interface I2C
- **Tension d'alimentation** :
  - +2,3V à +5,5V
- **Boîtiers disponibles** :
  - Boîtiers SO à 8 broches ou 16 broches (dans notre cas on utilise celle a 8 broches)
- **Broches** :
  - VCC : Tension d'alimentation
  - GND : Masse
  - SDA : Données I2C
  - SCL : Horloge I2C
  - SQW : Sortie de signal carré
  - RST : Entrée de réinitialisation
  - BAT : Entrée de la pile bouton
  - INT/SQW : Sortie d'interruption ou de signal carré
-

En ce qui concerne la RTC sur la carte nous l'avons équipé d'une pile bouton afin d'avoir un chronométrage continu.

Juste après cela, j'ai effectué les branchements nécessaires entre ma carte électronique disposant de la RTC et la Raspberry Pi 0 pour effectuer la programmation dans ce test.

Avant de passer au programme, vu que l'horloge en temps réel fonctionne en I2C, je devais m'assurer que la liaison soit établie entre la carte RPI et la RTC pour permettre une communication pour cela j'ai activé l'interface I2C sur Raspberry PI.



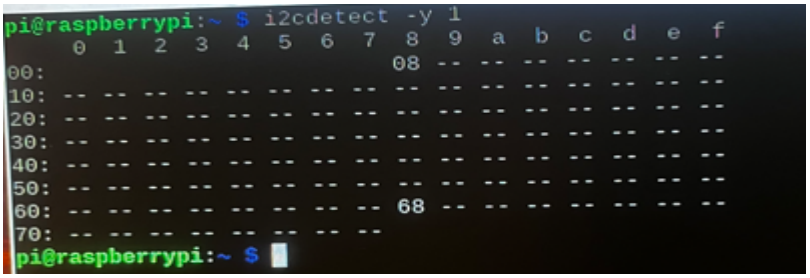
Après ça j'ai dû activer allez dans le terminal de commande afin de mettre à jours les paquets en utilisant la commande :

```
Sudo apt-get update
Sudo apt-get upgrade
```

Pour mettre à l'heure l'horloge, j'ai suivi un tutoriel en ligne dont voici les étapes que j'ai suivis :

```
sudo i2cdetect -y 1
```

Cette instruction nous retourne les infos suivantes, ici le RPI à détecter un composant à l'adresse hexadécimal 0x68 qui d'après sa documentation, est l'adresse par défaut de la RTC.



Après ça j'ai ajouté la gestion du module en éditant le fichier de configuration de boot.

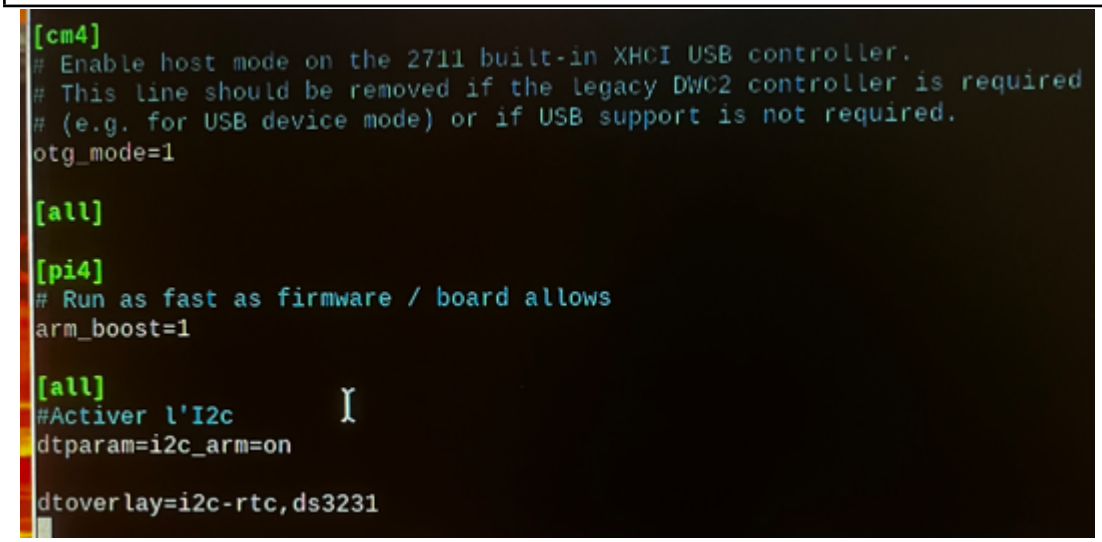
```
sudo nano /boot/config.txt
```

Auquel on va ajouter un la ligne suivante à la fin du fichier :

```
dtoverlay=i2c-rtc, ds3231
```

Cependant même en mettant cette commande j'ai remarqué que je ne pouvais toujours mettre à jour l'heur et la date car la commande I2C ne marche pas donc j'ai dû l'activer en ajoutant la ligne de commande suivante :

```
dtparam=i2c_arm=on
```



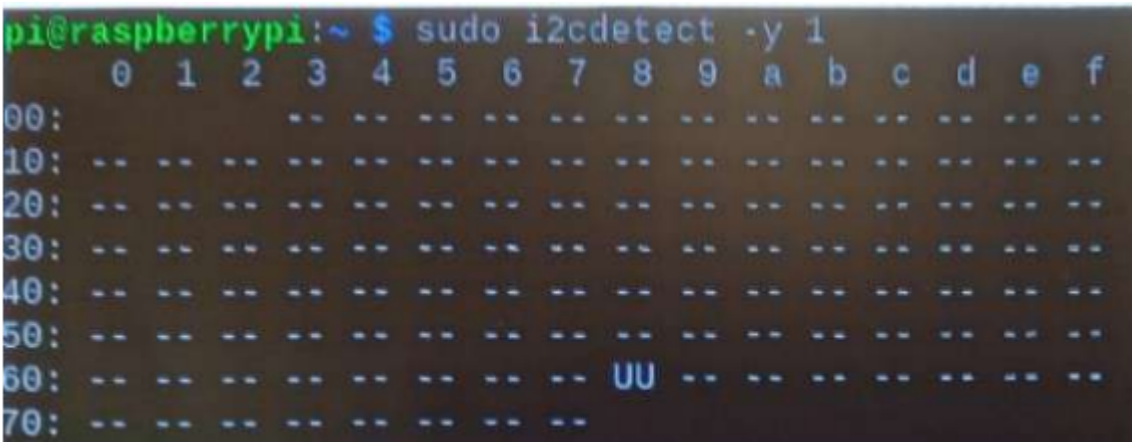
Ensuite j'ai effectué un reboot de la carte avec la commande :

```
sudo reboot
```

Une fois le système de nouveau actif, on peut exécuter de nouveau la commande :

```
sudo i2cdetect -y 1
```

On voit bien que le composant à changer d'adresse pour UU signifie que la RTC est prête à être mise à l'heure.



Avant dernière étape, on doit désormais désactiver la fausse horloge intégrée dans la RTC :

```
sudo apt-get y remove fake-hwclock

sudo update-rc.d f fakehwclock remove
```

Enfin, on peut désormais afficher l'heure actuelle avec la commande suivante :

```
sudo hwclock -D -r
```

Partie Personnelle
Fleury Benjamin

(EC 2)

Partie personnelle FLEURY Benjamin

Présentation de ma partie

Le projet de l'entreprise Servelec est de mettre en œuvre un afficheur sur lequel seraient transmises les valeurs en degrés Celsius de deux capteurs permettant de récupérer les températures des fours et des réfrigérateurs/congélateurs pour connaître l'état de leurs bons fonctionnements ou non.

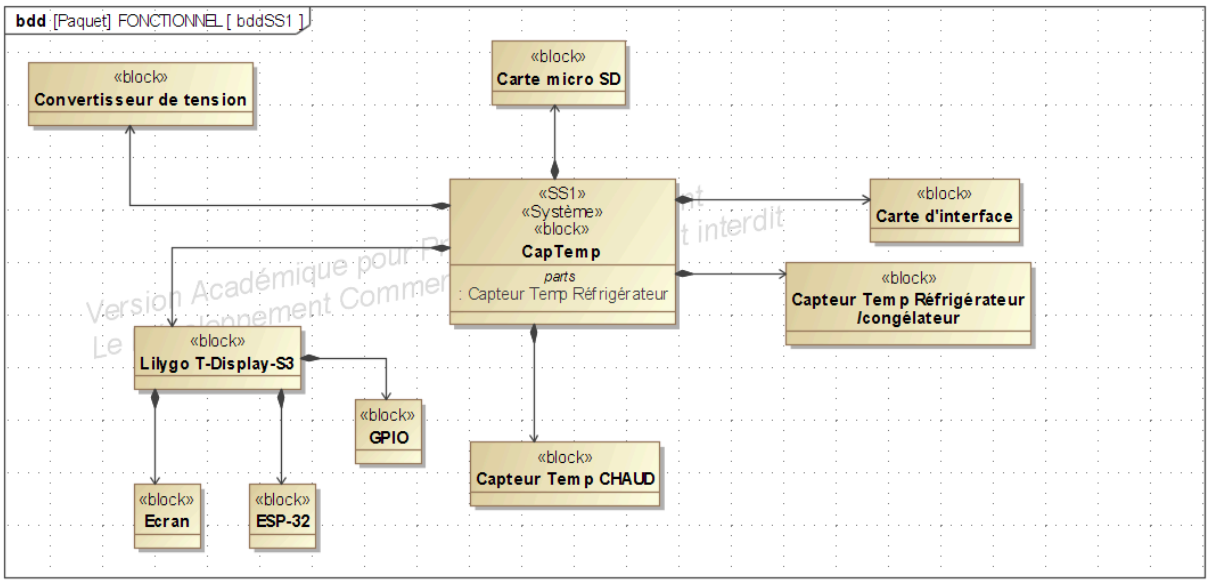
Pour les réfrigérateurs/congélateurs le principe serait de placer le capteur à l'intérieur pendant une durée d'environ 7 jours, durant cette période les valeurs seront relevés toutes les 5 minutes puis seront stockées dans un dossier sur une carte SD afin de pouvoir en faire un graphique par la suite qui permettrait à l'entreprise de vérifier si la température se régule correctement et si elle correspond bien au données du constructeur, soit environ 4 degrés pour les réfrigérateurs et environ -18 degrés pour les congélateurs. Après la récupération du capteur de température et vérification du graphique, une intervention sur l'appareil pourra être mise en place, si bien évidemment l'appareil est réparable.

Pour les fours, étant donnés qu'ils chauffent à partir de la température ambiante de l'environnement ou il se situe jusqu'à environ 400 degrés maximum, pour un four domestique, le but avec ceux-ci est de vérifier en premier temps si il chauffe bien à la température choisie et si elle reste constante. La température sera relevée toutes les 5 secondes pendant une durée de 2 heures et sera affichée sur l'afficheur et pourra être consultée directement par le technicien, il déterminera la pièce défectueuse de l'appareil et pourra faire la réparation si bien évidemment dans ce cas aussi l'appareil est réparable.

Mon but est en premier temps est de récupérer les températures des capteurs afin qu'ensuite je puisse programmer la carte de l'afficheur, compatible avec les librairies Arduino, pour les retranscrire dessus, je pourrais ensuite stocker ces données sur la carte micro SD. Une fois fait je pourrais router une carte fille sur laquelle sera placé l'écran que j'aurai programmé au préalable afin qu'il affiche en temps réel les températures des capteurs.

Les données qui seront relevées puis enregistrées sur la carte SD seront au format CSV.

17. Diagramme de blocs



Ce diagramme de bloc représente l'architecture de la carte à mettre en oeuvre rassemblant différent composant du système qui sont :

- **Lilygo T-Display-S3** : elle intègre un écran pour afficher les températures, des broches d'alimentation en 3,3V et 5V, des broches GPIO sur lesquelles seront branchés les différents capteurs, ainsi que la carte SD.

- **CapTemp** : relié à deux autres blocs qui représente les capteurs :

  - Capteur Température Chaud : Un capteur mesurant la température dans un environnement chaud.
  - Capteur Température réfrigérateur/congélateur :Un capteur mesurant la température dans un environnement froid.

18. Diagramme de gantt

Pour toute la durée du projet, j'ai réalisé un diagramme de Gantt prévisionnel :

1			**Evaluations du projet**	**Lun 19/02/24**	**Ven 14/06/24**	
2			Revue 1	Lun 19/02/24	Ven 23/02/24	
3			Revue 2	Lun 15/04/24	Ven 19/04/24	
4			Dépôt dossier technique	Jeu 23/05/24	Jeu 23/05/24	
5			Soutenance finale E62 (Revue 3)	Lun 10/06/24	Ven 14/06/24	17
6			**Epreuve E4**	Mer 15/05/24	Mer 15/05/24	
7			**Epreuve E1**	Mar 14/05/24	Mar 14/05/24	
8			Vacances d'hiver	Lun 26/02/24	Dim 10/03/24	
9			Vacances printemps	Lun 22/04/24	Dim 05/05/24	16
10			**Projet**	**Lun 08/01/24**	**Ven 14/06/24**	
11			Specifications générales, prise de connaissance des consignes de travail, du dossier du projet technique et planification sur microsoft project.	Lun 08/01/24	Mar 09/01/24	
12			Présentation des solutions actuelles pour la mesure de température sur un réfrigérateur/congélateur et sur un four. Lecture des documentations des capteurs, sonde de température PT100 pour les fours et sonde de température DS18B20 pour les réfrigérateurs.	Lun 15/01/24	Mar 16/01/24	
13			Prise en main des appareils à utiliser, arduino uno et ESP32S3 Lilygo et mise en œuvre des sondes DS18B20 sur arduino uno et du DS18B20 par l'intermédiaire d'un MAX31865 et de la librairie arduino proposée par Adafruit également sur arduino uno.	Lun 22/01/24	Mar 30/01/24	12
14			Transposer et modifier en fonction le travail fait sur arduino uno sur ESP32S3 Lilygo et essais de bon fonctionnement.	Lun 05/02/24	Mar 13/02/24	13
15			Comparaison des résultats des deux sondes avec un capteur de référence.	Lun 19/02/24	Mar 20/02/24	
16			Prototypage rapide et proposition du schéma structurel d'une carte fille, dressage de la liste des composants avec code commande chez les ditributeurs et vérification de la disponibilité des composants et du conditionnement. Routage d'un circuit imprimé.	Lun 11/03/24	Mar 16/04/24	8

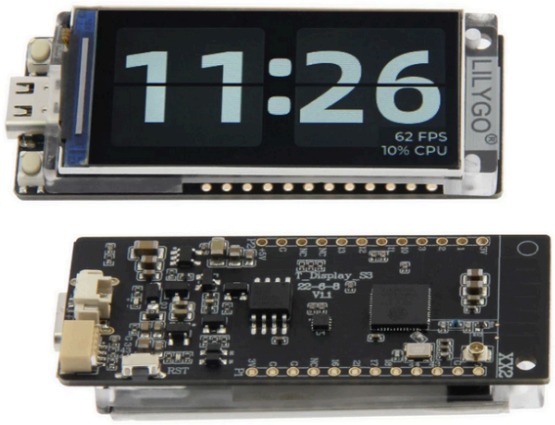
17			Production des fichiers de fabrication et commande de la carte, après réception, câblage de celle-ci puis essais de bon fonctionnement de la carte avec les capteurs.	Lun 06/05/24	Mar 04/06/24	9
18			Fichier de fabrication PCB finalisés	Mar 16/04/24	Mar 16/04/24	
19			Rédaction du dossier	Lun 08/01/24	Mer 22/05/24	
20			1er BTS Blanc	Jeu 22/02/24	Jeu 22/02/24	
21			2nd BTS Blanc	Jeu 18/04/24	Jeu 18/04/24	
22			Soutenance E61	Lun 18/03/24	Mar 19/03/24	

En tout premier temps j'ai réalisé ce diagramme de Gantt qui me permet de planifier ce que dois faire et le temps que j'ai pour le réaliser.

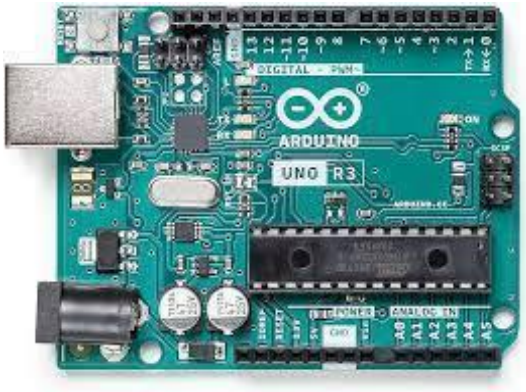
19. Présentation du matériel

Pour réaliser ce projet je dispose du matériel suivant :

• Carte ESP32-S3 Lilygo



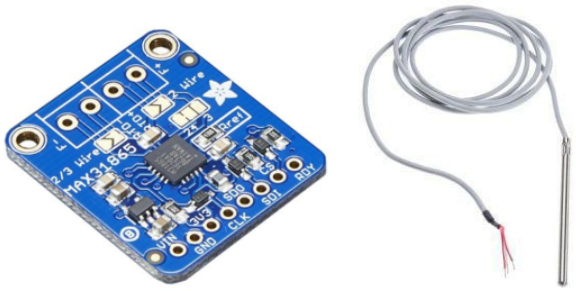
• Carte Arduino Uno



• Capteur DS18B20



• Capteur PT100 + breakout MAX31865



• Breakout Micro-SD Card



• Régulateur de tension 5V vers 3,3V **LP2981AIM5-3.3** uniquement pour la carte SD



• Horloge temps réel



20. Mise en œuvre du projet

Je commence par prendre connaissance des documentations des différents appareils et capteurs que je devrais utiliser pour mettre en œuvre pour ce projet.

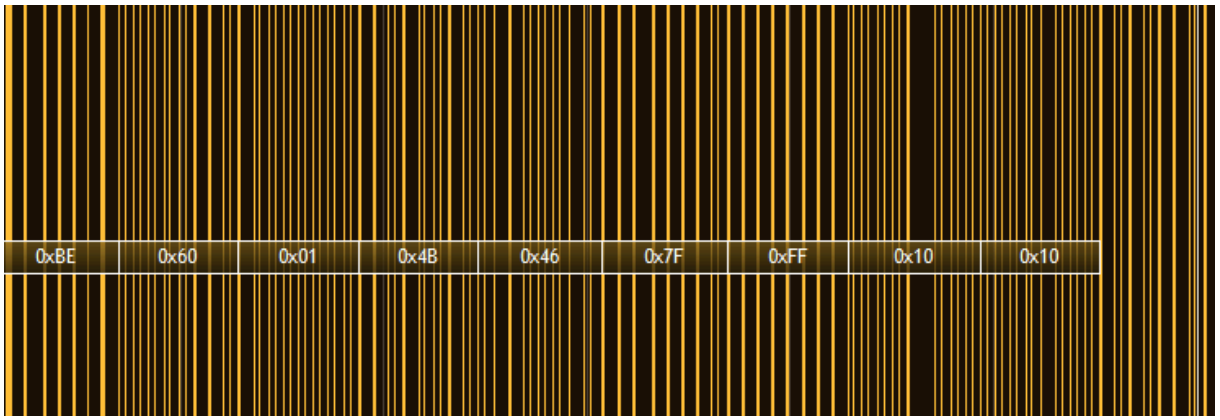
Avec le travail fait en amont pendant les cours je mets en œuvre le capteur DS18B20 sur un Arduino Uno avec affichage de la température dans le moniteur série de celle-ci.

21. Mise en œuvre du capteur DS18B20

Ce capteur de température communique sur le bus OneWire qui est un protocole de communication qui nécessite seulement un fil pour la transmission des données en plus du fil de masse et d'alimentation, il peut être alimenté entre 3V et 5V, et il peut fonctionner entre -55°C et +125°C avec une précision de +/-0,5°C et il est étanche à l'eau et à la poussière, il sera donc utilisé pour les réfrigérateurs/congélateurs.

J’ai fait la mise en œuvre sur Arduino en téléchargeant la librairie et en utilisant l’exemple fournis pour faire afficher la température dans le moniteur série.

21.1 .Relevé de trame sur SCANA STUDIOS



0x60,0x01 : Représentation de la température en hexadécimal.
Pour avoir la température en degrés Celsius il faut inverser ses deux octets puis les convertir en décimal.

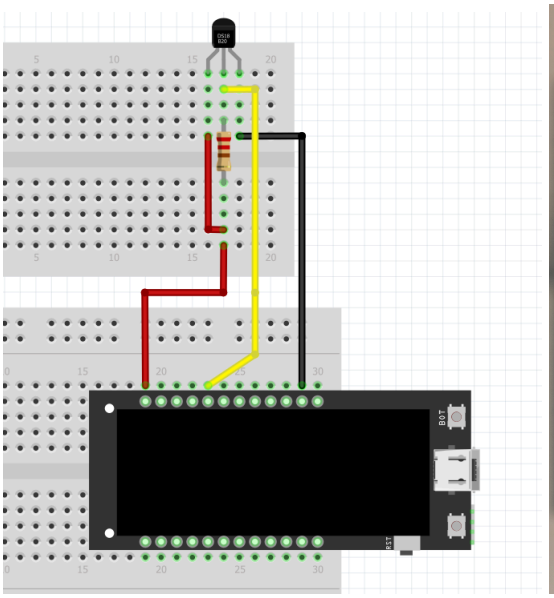
Cela nous donne : 0x0160 = 352, suite à ça on le multiplie par son coefficient de température qui est de 0.0625°C, valeur qui à été trouvée dans la documentation, ce qui nous donne 352*0.0625 = 22°C.

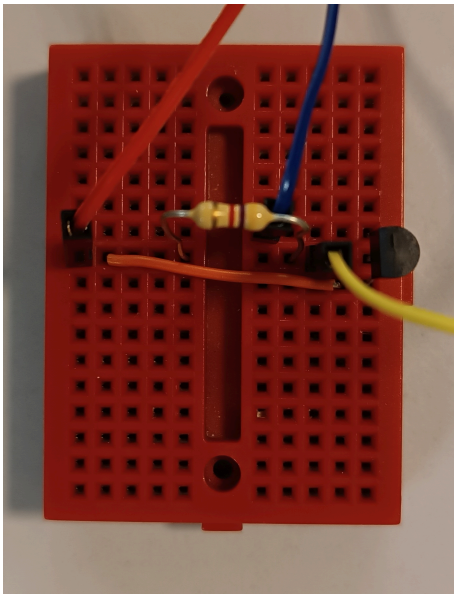
Je prends ensuite en main l’ESP32-S3 Lilygo, les spécificités et les codes pour utiliser les broches et l’afficheur. Une fois fait, je récupère le code de l’Arduino Uno pour le transposer sur l’ESP32-S3 en modifiant et ajoutant des lignes de codes pour arriver à avoir la température écrite directement sur l’afficheur, le résultat ci-dessous :



L'afficheur de l'ESP32-S3 est de 320*170 pixels, 320 correspond à la largeur de l'écran et 170 à sa hauteur, j'ai du donc créé une zone de texte dans ces cotes pour pouvoir écrire à l'intérieur avec la commande
« sprite.createSprite(320,25);», avec cette commande je sélectionne toute la largeur de l'écran et 25 pixels dans la hauteur. Dans cette zone j'ai fait écrire la température avec la commande
« sprite.drawString("Temp.Frigo:"+String(celsius)+"`C",150,13,4) ; » 150 correspond à la largeur, car le point de départ se situe sur le point milieu de la largeur de l'écriture, soit à peu près sur la lettre O, 13 correspond à la hauteur du texte dans la zone, il se situe sur le point en bas à gauche de l'écriture, soit en bas de la lettre T et 4 correspond au format de l'écriture.

Voici une image du câblage du capteur DS18B20 pour les test et le schéma Fritzing de l'ESP32-S3 avec le DS18B20 :

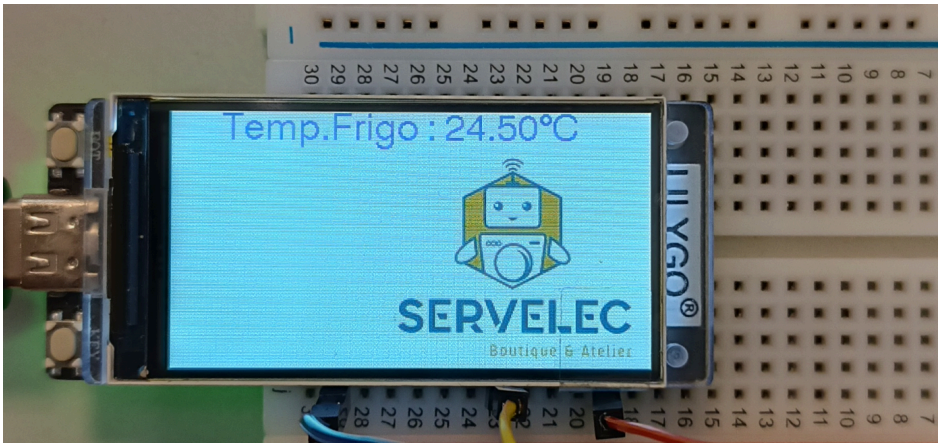




Le fil rouge est l'alimentation de 3,3V de l'ESP32-S3, le bleu est la masse et le jaune la donnée de la température reilé sur une broche GPIO.

Sur le reste de l'écran j'en ai profité pour insérer le logo de l'entreprise, il me restait plus que 145 sur la hauteur pour placer le logo j'ai converti l'image d'origine qui était de 1051*685 sur l'application paint, au format 158*135 pour qu'elle rentre correctement sur l'afficheur, avant de la mettre sur le site http://www.rinkydinkelectronics.com qui convertit l'image au format hexadécimal que je télécharge et ouvre dans un traitement de texte comme bloc-notes que je copie et je créer une nouvelle page dans le logiciel Arduino et je colle ce texte dans cette nouvelle page. Avec la ligne de commande tft.pushImage(150,30,158,134,Servelec_158x135); j'introduis l'image sur l'écran.

23.2. Photo du résultat



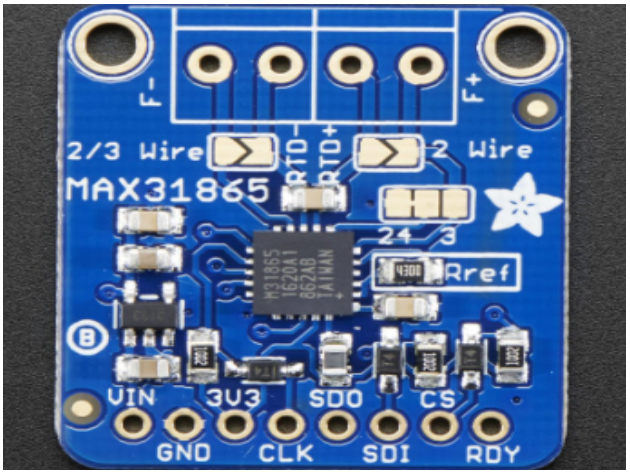
22. Mise en œuvre de la sonde PT100

Par la suite, je prends connaissance de la documentation sur la sonde PT100.

La sonde PT100 possède une plage de température plus grande que DS18B20 qui est de -50°C à +250°C donc ce capteur sera utilisé pour la mesure des fours.

Le terme PT100 fait référence à un type spécifique de thermistance à résistance de platine dont la résistance est de 100 ohms à 0°C.

Pour sa mise en œuvre je possède un breakout MAX31865 qui est un convertisseur analogique numérique (ADC). La breakout peut être câblé soit en 2 fils, 3 fils, 4 fils.

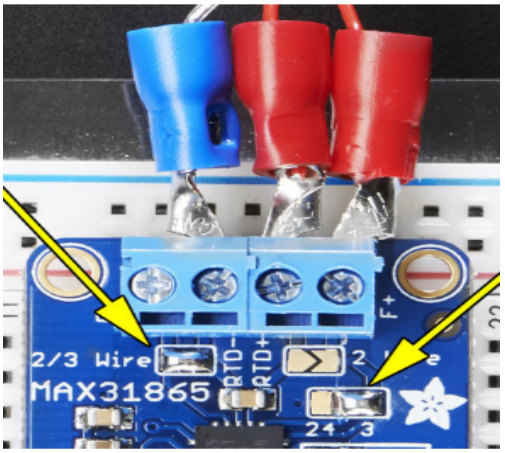


Description des broches utilisés pour le bus SPI :

- **Vin** - broche d'alimentation en 5V.
- **GND** - masse.
- **CLK** - il s'agit de la broche d'horloge, c'est une entrée dans la puce.
- **SDO** - il s'agit de la broche de sortie de données série des portes logiques.
- **SDI** - il s'agit de l'entrée du microcontrôleur, pour les données envoyées depuis votre processeur vers le MAX31865.
- **CS** - c'est la broche de sélection de contrôle. C'est une entrée dans la puce.

Il y a quatre contacts, pour utiliser des capteurs à 2, 3 ou 4 fils. Pour le projet j'ai utilisé celui à 3 fils car elle permet de compenser la résistance d'un des fils de connexion, ce qui améliore la précision

Ensuite pour que le câble marche en 3 fils j'ai soudé la carte pour fermer le cavalier "2/3 Wire" et fermer la soudure à droite sur le chiffre 3 qui correspond au nombre de fils de la sonde, elle est donc configurée en 3 fils.



Après câblage j'ai testé la carte sur Arduino Uno en utilisant le programme que j'ai récupérer sur le site et voici le résultat attendu :

```
Adafruit MAX31865 PT100 Sensor Test!
RTD value: 8336
Ratio = 0.25439453
Resistance = 109.38964843
Temperature = 24.08
```

Pour trouver la valeur de la résistance le calcul et le suivant :

On prend la valeur du RTD value : 8336 que l'on divise par 32768 ce qui donne un ratio de 0.025439453

Cette valeur on le multiplie par Reff qui est de 430 Ω.

Donc la formule est :

R= RTD value /32768*430

= 8336/32768*430= 109.389683Ω.

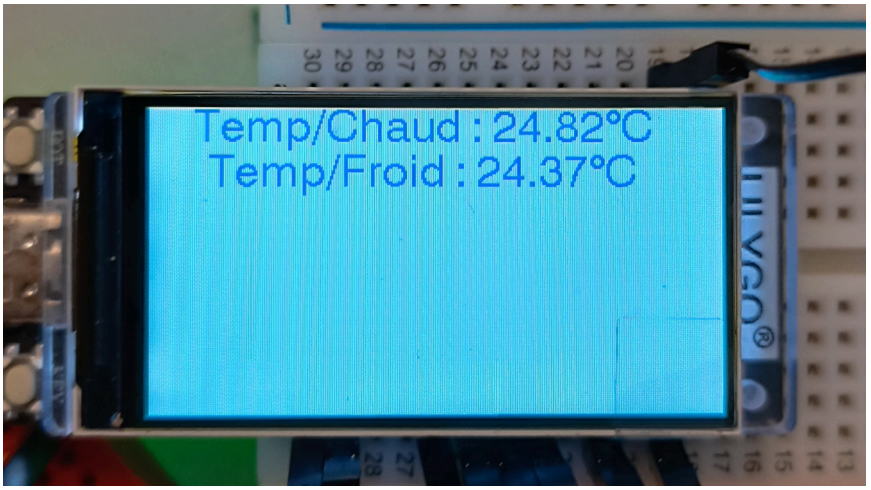
Pour trouver la température je met un multimètre en position ohmmètre sur la résistance, le calcul est le suivant :

$$T = \frac{valeur\ de\ la\ résistance - résistance\ à\ 0°C}{coefficient\ de\ température * 100\Omega} = \frac{109.38-100}{0.0039083 * 100} = 24.24°C$$

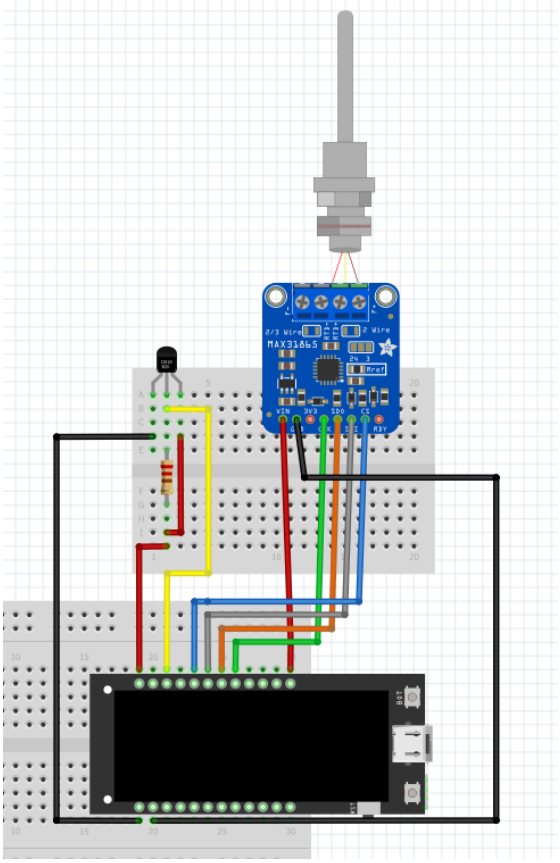
En utilisant cette formule on constate qu'il y a une petite marge d'erreur de 0.05 mais dans notre cas ce n'est pas vraiment important.

Pour avoir la température directement en degrés Celsius sur l'afficheur, j'inclus la formule de calcul dans le code ensuite même procédé que pour le capteur DS18B20, j'inclus le nouveau code dans celui existant afin d'obtenir les deux valeurs de température simultanément sur l'afficheur. Etant donné que les deux lignes d'écritures prennent de la place elles passent par-dessus le logo ce qui le fait disparaître.

Actuellement la sonde PT100 utilisée n'est pas adaptée à la prise de mesure désirée car sa plage va jusqu'à 200°C maximum et la température demandée est de 400°C.



22.1. Schéma Fritzing de l'ESP32-S3 avec le DS18B20 et la sonde PT100



23. Mise en œuvre du breakout micro SD



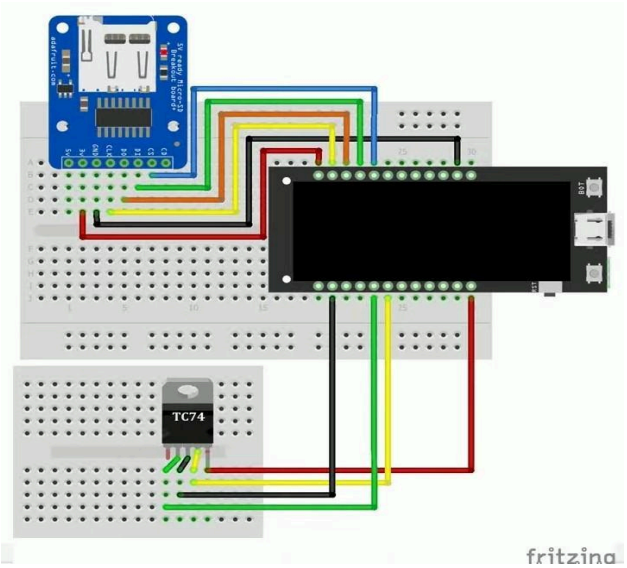
Description des broches utilisés pour le bus SPI :

5V - broche d'alimentation en 5V.

- **GND** - masse.
- **CLK** - il s'agit de la broche d'horloge, c'est une entrée dans la puce.
- **DO** - il s'agit de la broche de sortie de données série du microcontrôleur afin que l'ESP32-S3 puis lire la carte SD.
- **DI** - il s'agit de l'entrée du microcontrôleur, pour les données envoyées depuis l'ESP32-S3, les données de températures à écrire sur la carte SD.
- **CS** - c'est la broche de sélection de contrôle. C'est une entrée dans la puce.

Une fois les tests fait pour les prises de températures je passe ensuite à la mise en œuvre du breakout Micro-SD Card directement sur l'ESP32-S3, je commence par me documenter sur cette carte et j'effectue le câblage avec un capteur de température TC74.

23.1 Schéma fritzing du câblage de l'ESP32-S3 et carte SD et TC74



Je réalise ensuite le code permettant d'écrire un mot sur la carte SD, je vérifie qu'il est bien écrit puis je modifie le code pour pouvoir créer un dossier et pour mettre les valeurs en degrés Celsius dans un tableau excel au format CSV.

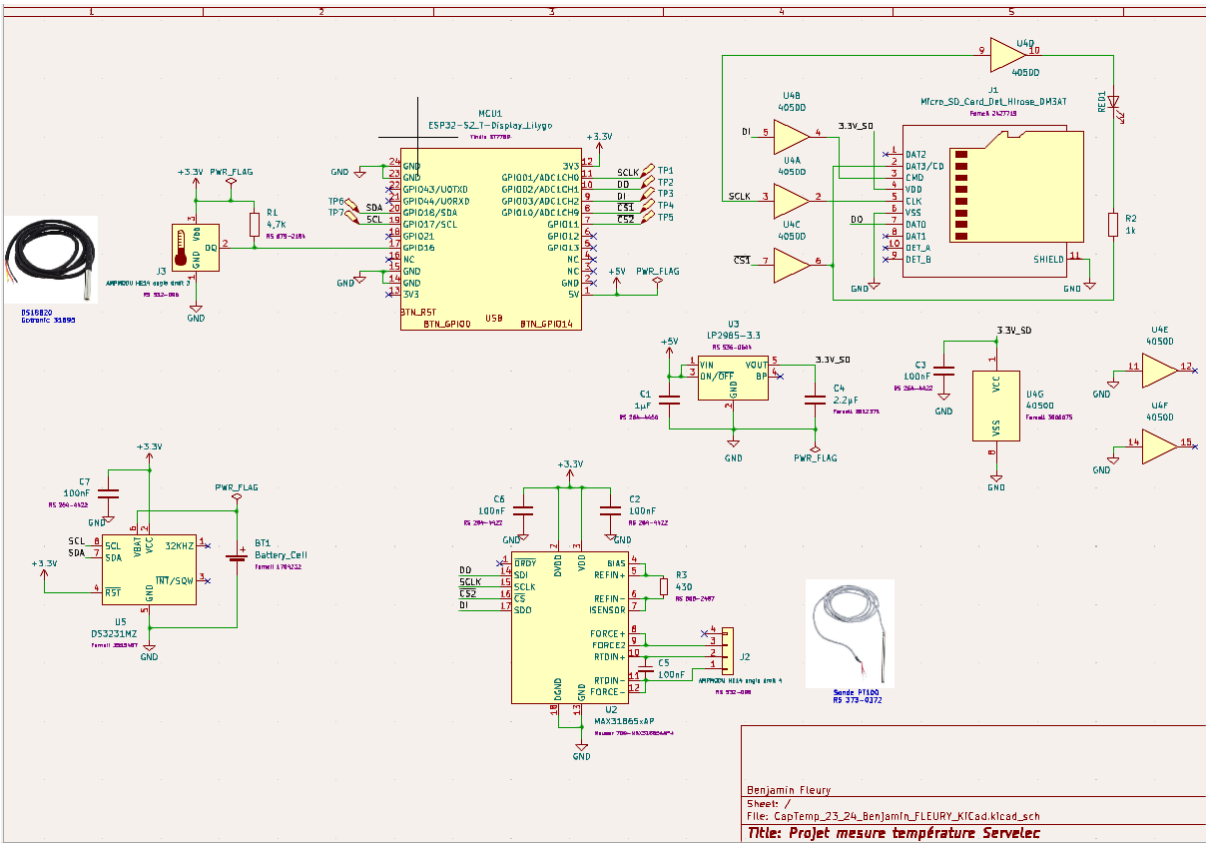
24. Routage de la carte fille

Un peu pressé par le temps je met de côté le code permettant d'écrire les données des sondes DS18B20 et PT100 sur la carte SD pour commencer à faire sur kiCad le schéma de la carte fille en plaçant tous les composants et en gardant seulement le nécessaire du breakout MAX31865 afin de gagner de la place sur ma carte fille d'autant plus que je n'aurais pas besoin de la totalité des composant du breakout.

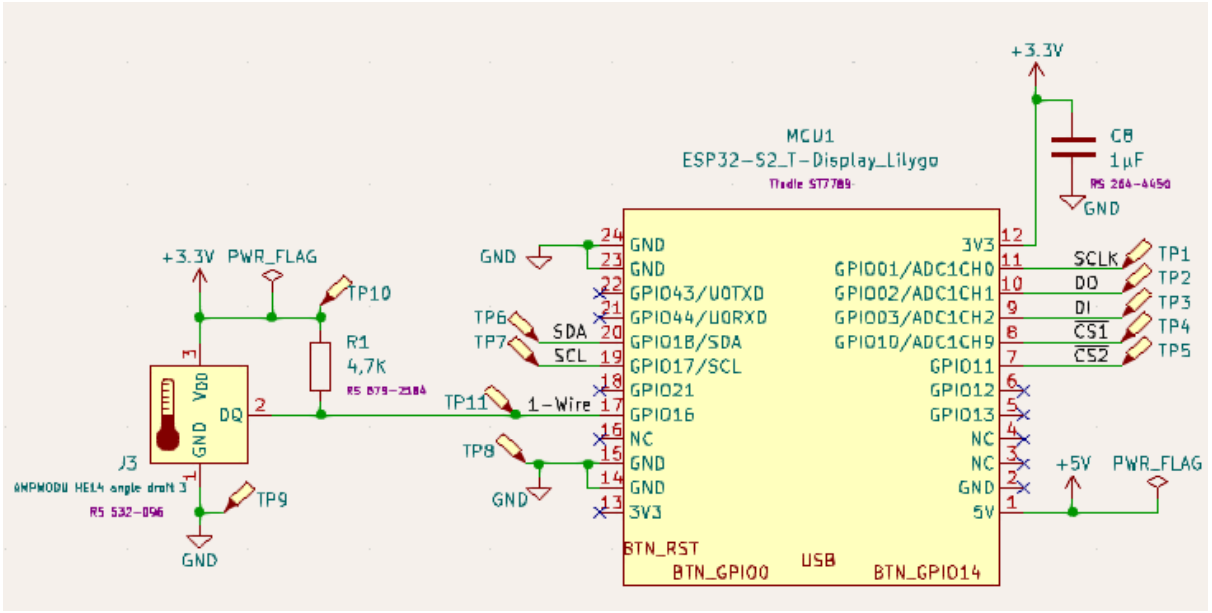
Quand au breakout micro SD j'ai besoin de la totalité des composants donc je les réorganise sur la carte fille.

J'ajoute en plus un support de pile afin de posséder une horloge temps réel qui nous permettra plus tard sur le relevé du tableau de la carte SD d'avoir la date et l'heure à laquelle les températures ont été prises.

Capture d'écran du schéma de la carte fille :



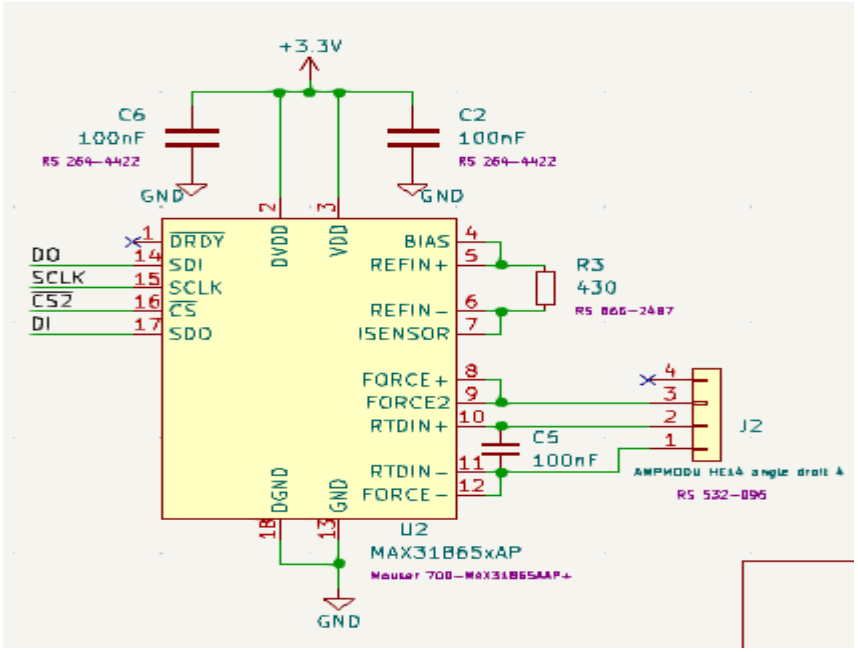
Plusieurs captures d'écrans des différentes parties du schéma :



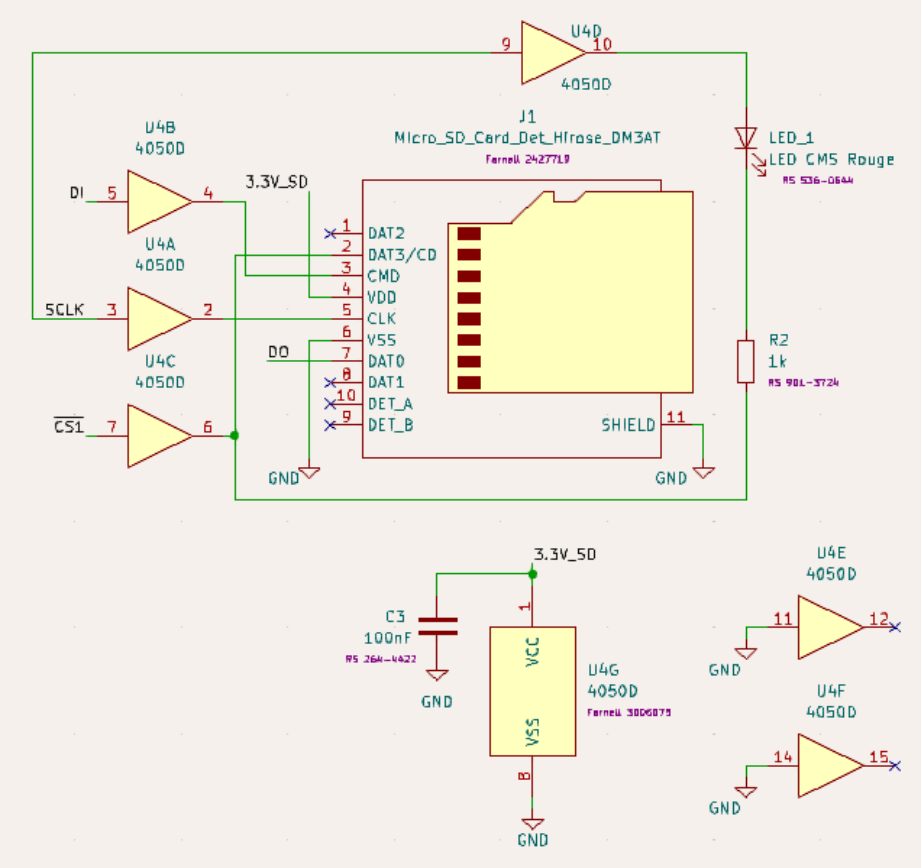
Ici nous avons l'ESP32-S3 reliée à une prise à gauche sur laquelle sera branchée la sonde DS18B20 ainsi qu'une résistance de pull'up, à droite les broches "SCLK, DO, DI, CS1 et CS2".

Les broches "SCLK, DO et DI" sont reliées à la fois sur le MAX31865 et sur le microcontrôleur de la carte SD.

CS1 est une broche de chip select qui permet de sélectionner le microcontrôleur de la carte SD.

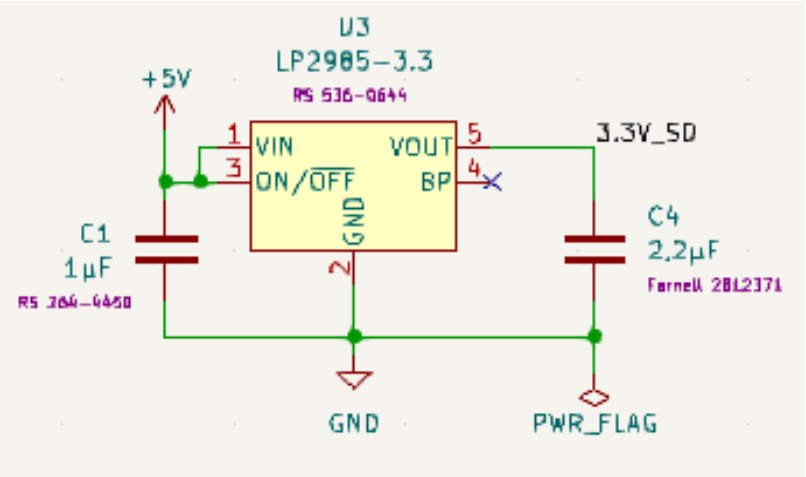


Le MAX31865 est relié sur les broches “SCLK, DO, DI et CS2”, au 3,3V avec deux condensateurs de découplage et à une prise sur laquelle sera branchée la sonde PT100.

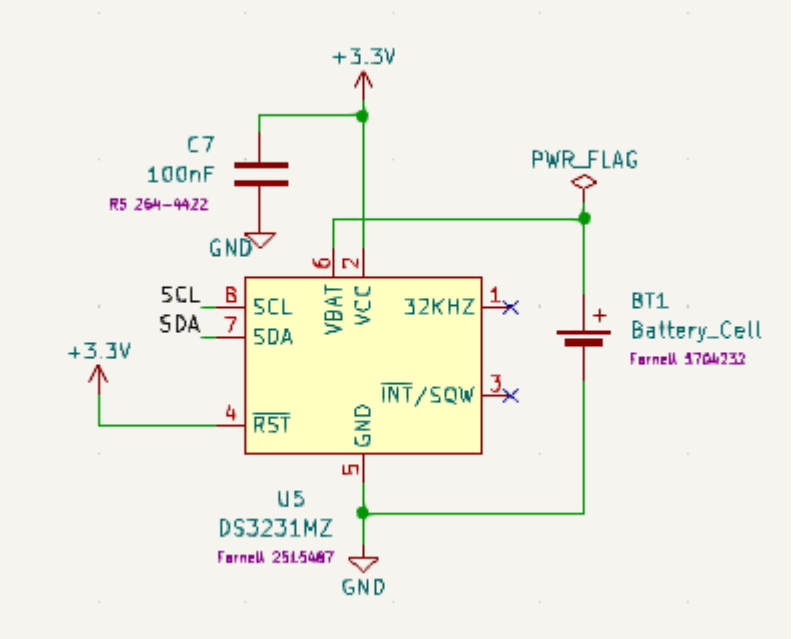


Le support de carte SD relié aux broches “SCLK, DO, DI et CS1”, il y a une led rouge qui indique quand la carte est lue et lorsqu’on écrit dessus.

Les 6 triangles et le rectangle du milieu sont un seul et même composant, ce sont des portes logiques non inversées de la carte SD.

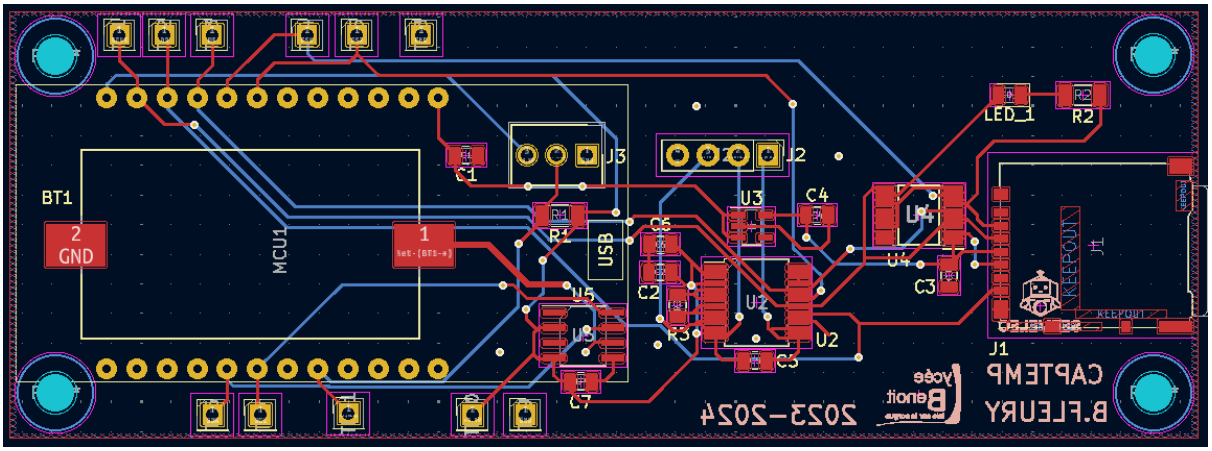


Ici nous avons le régulateur de tension.

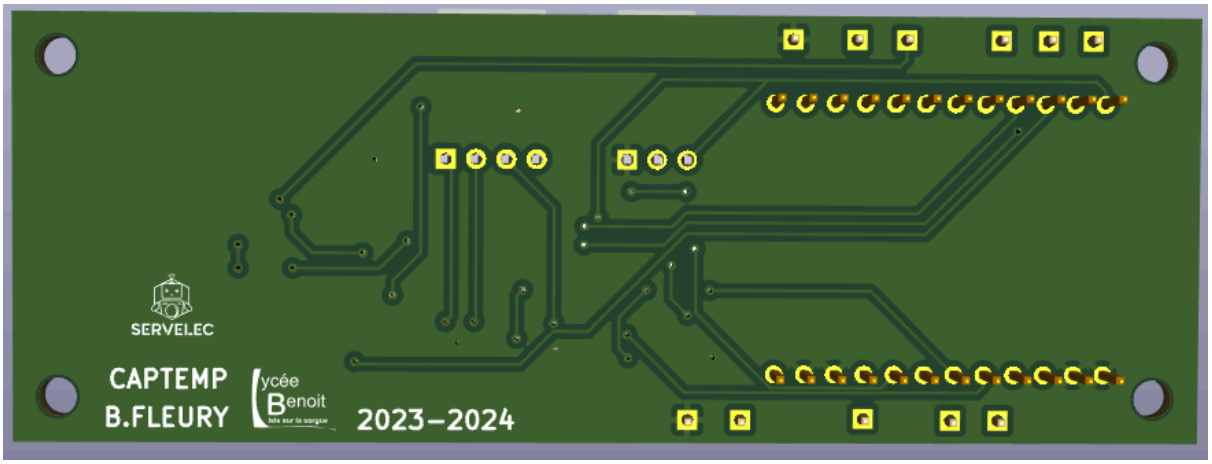


L'horloge temps réel est alimentée par une pile lorsque l'ESP32-S3 n'est pas utilisée.

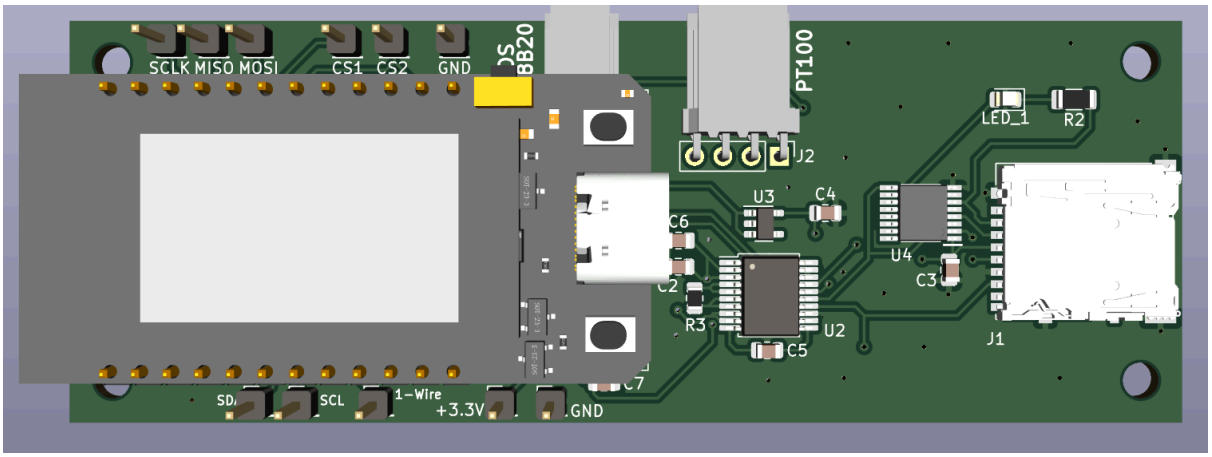
Une fois le schéma terminé, vérifié et corrigé par mon professeur, je m'attaque au routage de la carte fille, pour cela je me base sur des côtes de 100 mm par 35 mm, je place les composants sur la carte et voici le résultat :



Vue du routage de la carte avec des points de tests ajoutés.



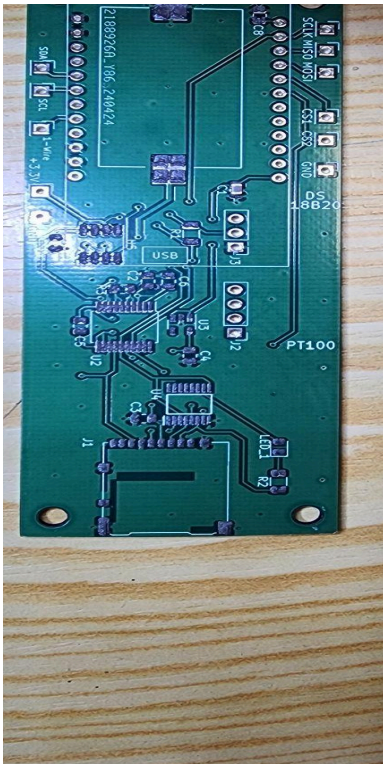
Vue en 3D du dos de la carte.

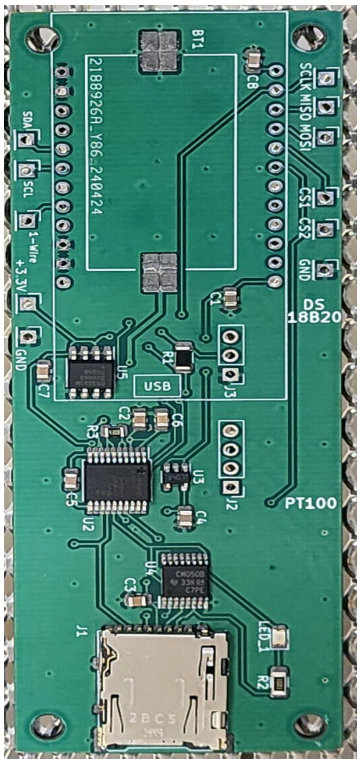


Vue en 3D de la face de la carte

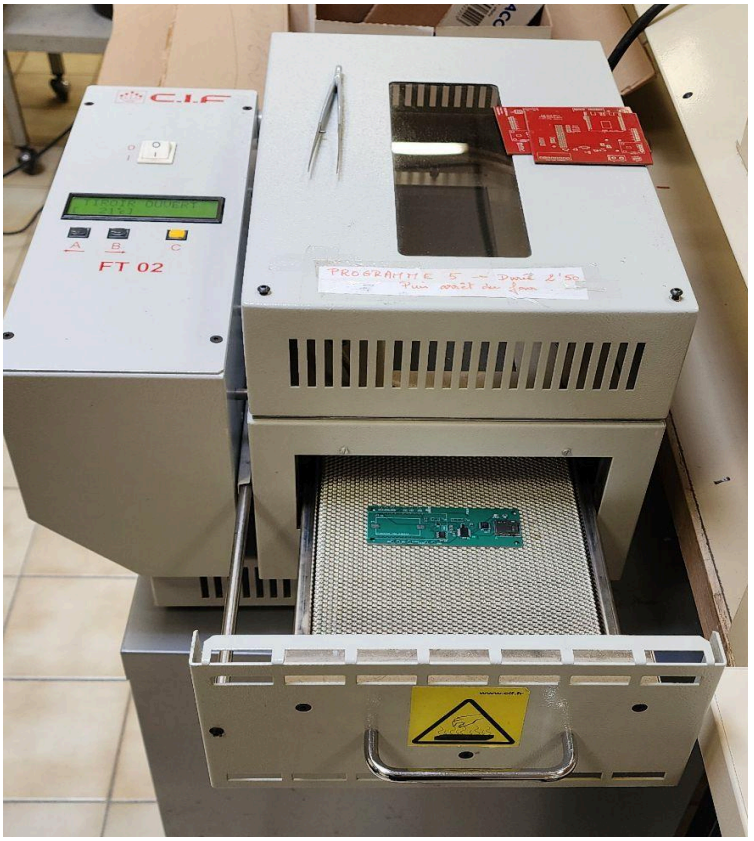
25. Procédure de mise en oeuvre de la carte

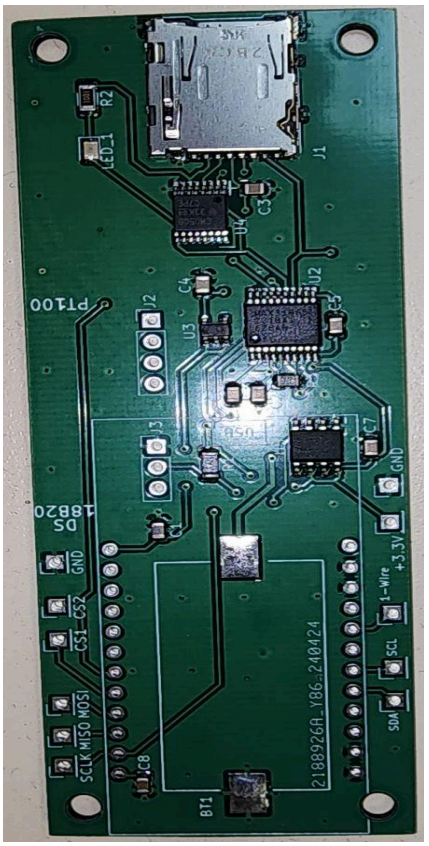
Après réception de la carte non câblée, je regarde si elle est bien conforme a mon schéma de câblage, je prends ensuite ma liste de composants avec les composants et je vérifie qu'il n'en manque aucun. Je prend ensuite le stencil fourni avec la carte que je positionne avec le support 3D fait par mon professeur. Je peux ensuite étalé la pâte à braser et placer les composants sur la carte au bon endroit et dans le bon sens avec l'aide du plugin Bom sur KiCad, ce plugin nous envois sur une page google sur laquelle se trouve la liste des composants et la vue top de la carte avec les composants et quand on sélectionne un composant dans la liste il est mis en évidence sur le schéma.

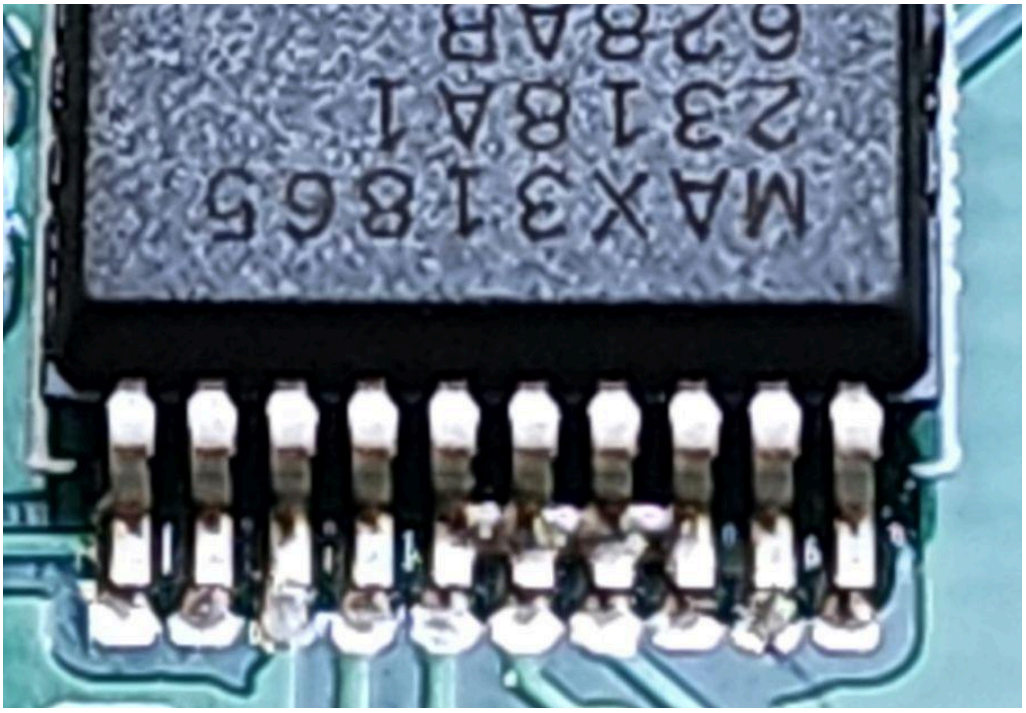




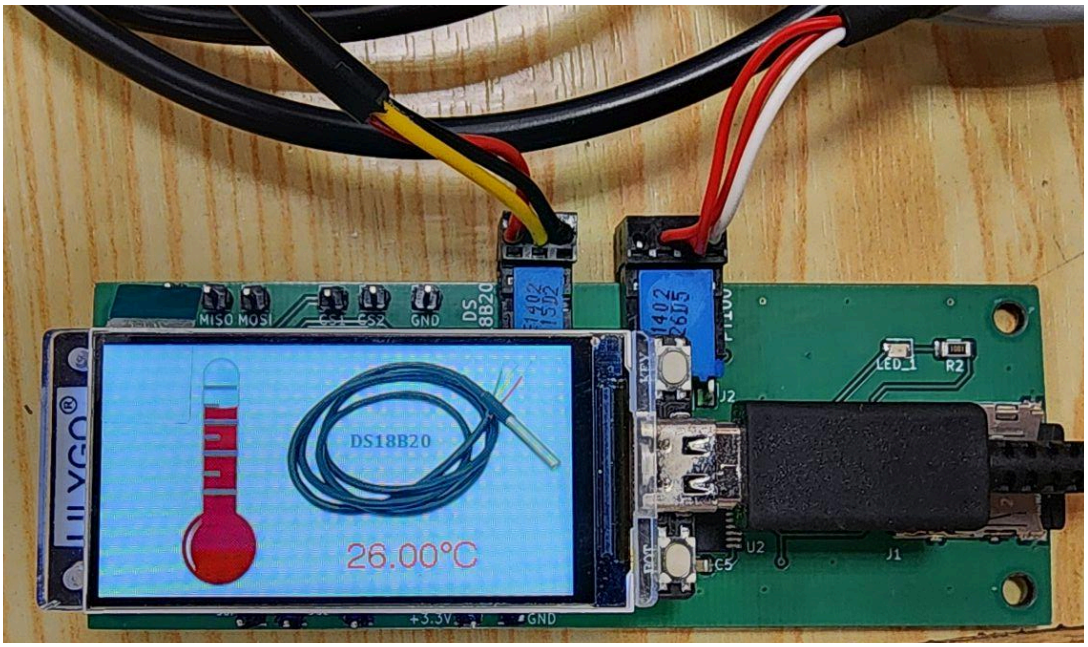
Une fois tous les composants à la bonne place et dans le bon sens je met la carte dans un four à refusion qui est à une température de 90°C et qui va chauffer la carte jusqu'à 250°C ce qui va faire fondre la pâte à braser et souder les composants que j'ai placer.
Je sors la carte et attend qu'elle refroidisse puis je contrôle que les soudures soit bien faites, je remarque que sur le MAX31865 il y'a 4 broches soudées ensemble je passe alors dessus avec un fer à souder est une panne fine afin de retirer l'excédent d'étain.





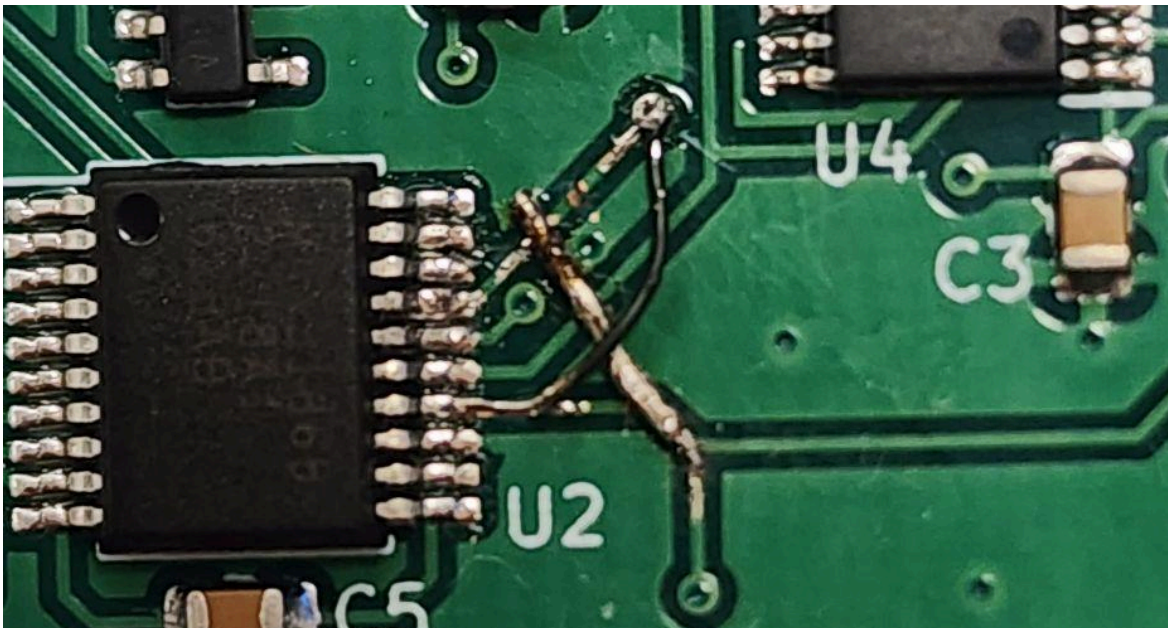


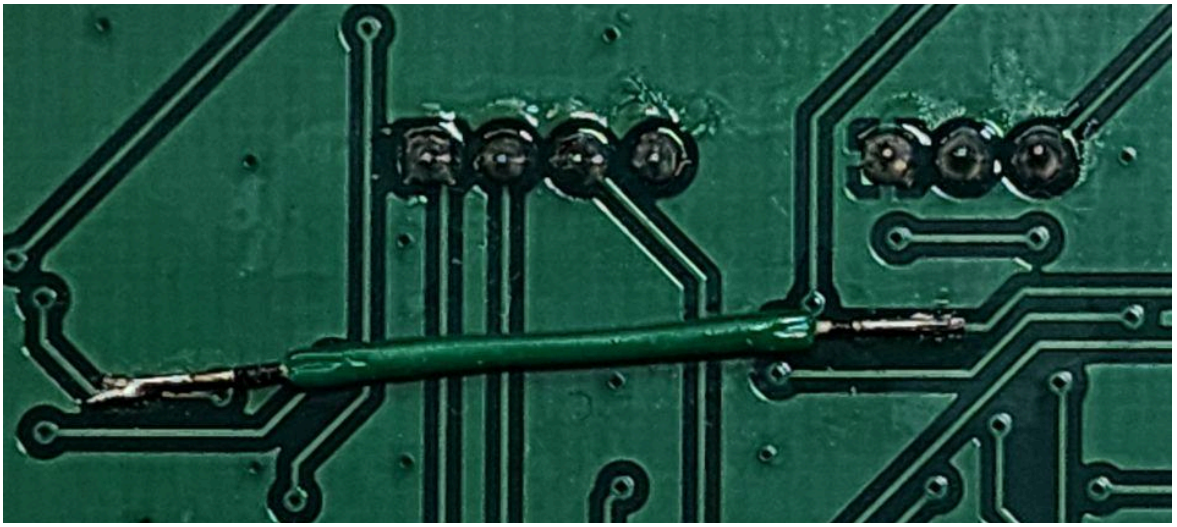
Je peux ensuite souder les composants traversants qui sont les points de test et les supports de l'ESP32-S3.
La partie soudure étant terminée je test la continuité des différentes broches et des points de test, la continuitée est bonne je peux donc placer l'ESP32-S3 sur son support et alimenter la carte.
Avec un multimètre je test mes alimentations 3,3V et 5V qui sont délivrées par l'ESP32-S3 et je téléverse en premier temps le code qui permet d'afficher la température de la sonde DS18B20 et ça fonctionne bien.



Je téléverse ensuite le code permettant d'afficher la température du DS18B20 et de l'écrire sur la carte Micro SD et il n'y a aucun problème de ce côté là, je téléverse le code pour lire la température de la sonde PT100 et je trouve des valeurs incohérentes, -204°C, avec la température approximative de la pièce qui était d'environ 20°C.

J’en parle donc à mon professeur qui a réussi à trouver l’origine du problème, sur le schéma de câblage j’ai inversé les broches DO et DI sur le MAX31865 qui correspondent à l’entrée et à la sortie des données. Mon professeur a donc dû couper le circuit imprimé pour inverser les deux fils et de ce fait les mettre dans le bon ordre.





Pour la revue finale je dois encore tester la carte en prenant la température de la sonde PT100 avec la modification faite et si tout fonctionne correctement je pourrais ensuite faire le code permettant de lire les valeurs des sondes DS18B20 et PT100 et qui les écrit sur la carte micro SD. Si tout fonctionne, je pourrais mettre en œuvre l’horloge temps réel et, si le temps me le permet, faire un modèle en 3D d’un boîtier pour contenir toute la carte.

Annexe

26. lien du compte rendu de la réunion

https://docs.google.com/document/d/1CUu-Meaxj2K2vg_b0RDlK1BUwYl-o_0-2UDyEGPtEIQ/edit?usp=drive_link

27. Note d’étude du projet

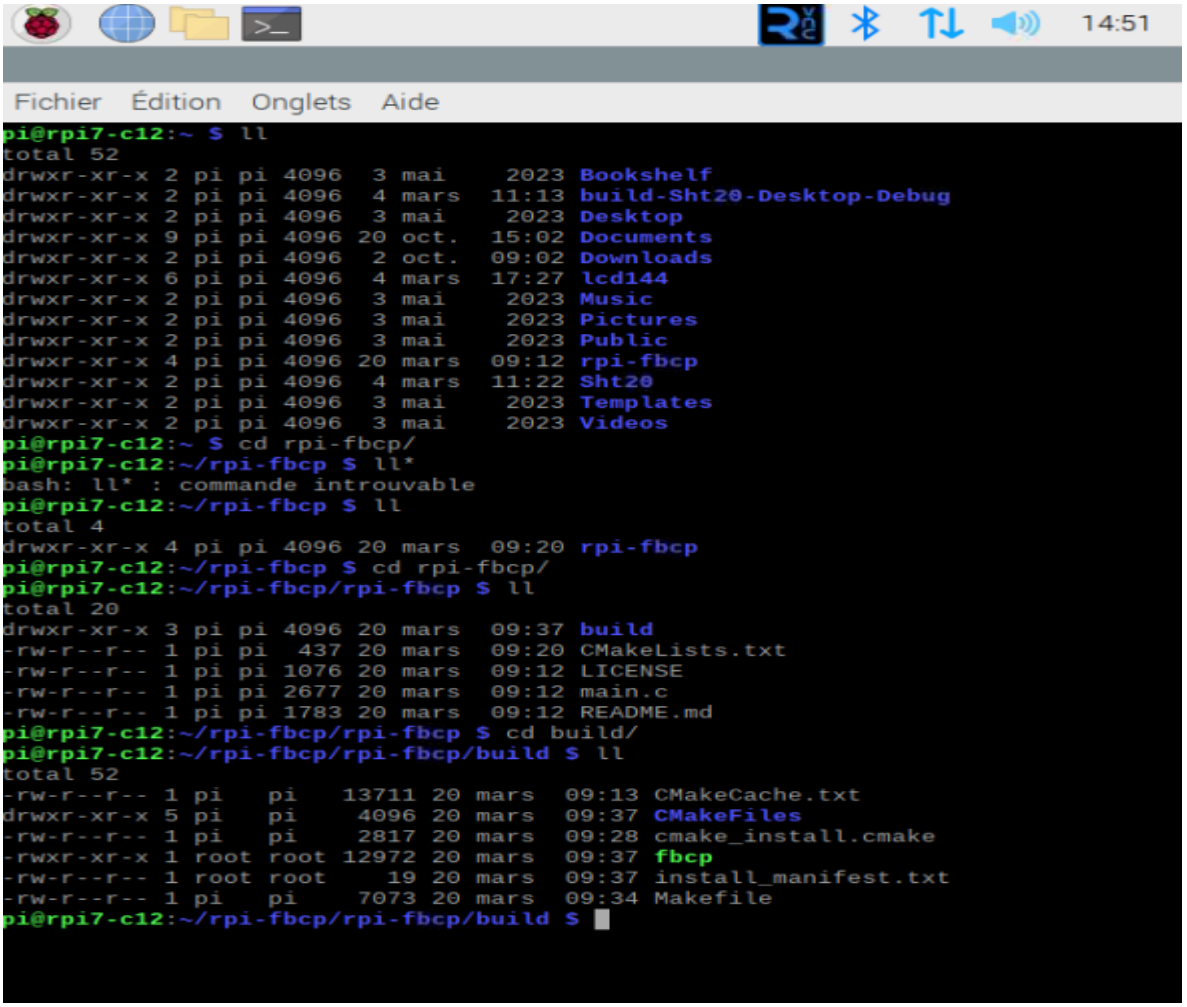
lors du projet j’ai écris lors des situation ou j’ai rencontré un problème une note d’étude du projet.

j’y ai décrit simplement le problème rencontré et la ou les solutions utilisées.

voici ci dessous celle écrite durant celui-ci.

objet: affichage de l’écran principale sur un écran GPIO de 1.44 pouces

travaille réalisé:

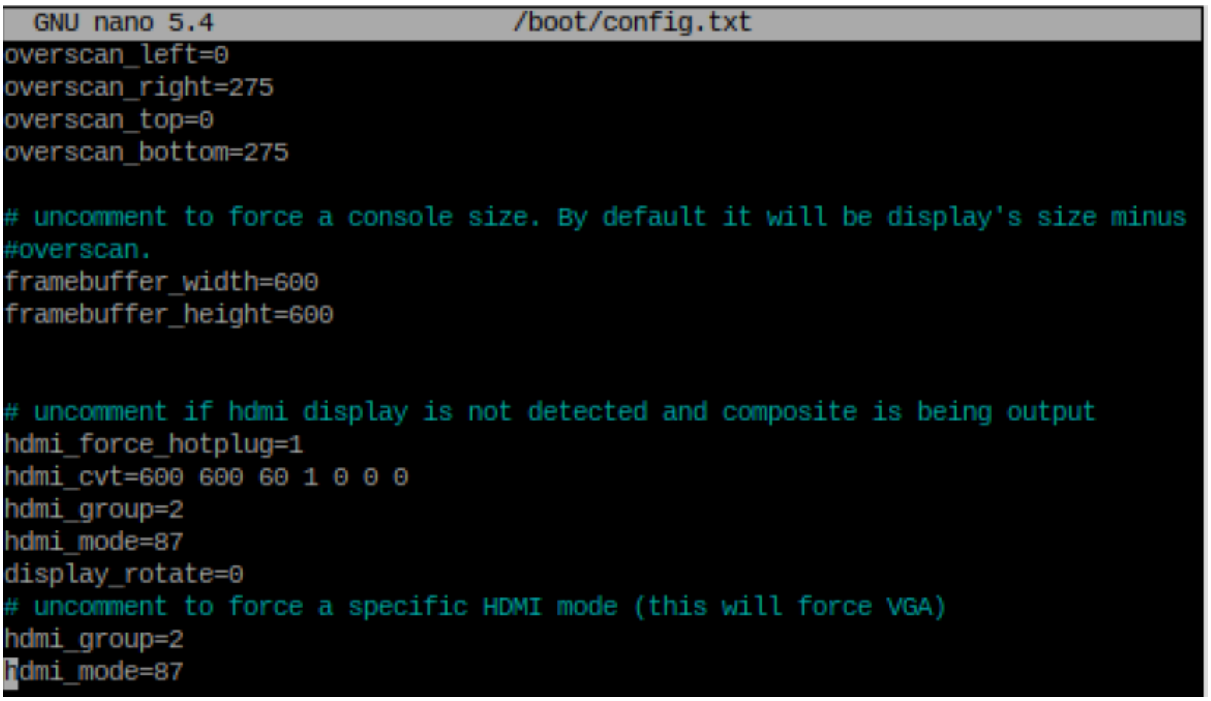


-installation d’une librairie permettant la transmission de l’image de l’hdmi sur l’écran GPIO
-modification de la résolution pour correspondre à celle de l’écran
-lancement du programme

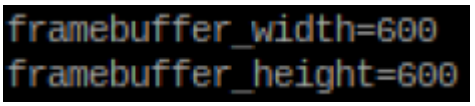
résultat:RIEN aucun changements après plusieurs heures de recherche

solution:suivre le tutoriel de la page suivante
https://www.waveshare.com/wiki/1.3inch_LCD_HAT

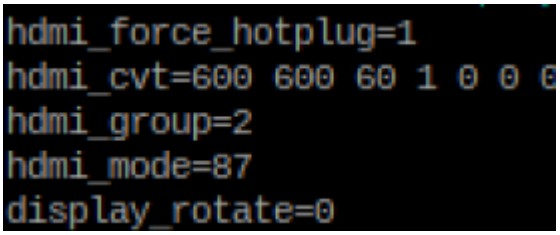
résultat : fonctionne aprés quelques mises au point



-modification de la taille de l’affichage



-modification de la résolution

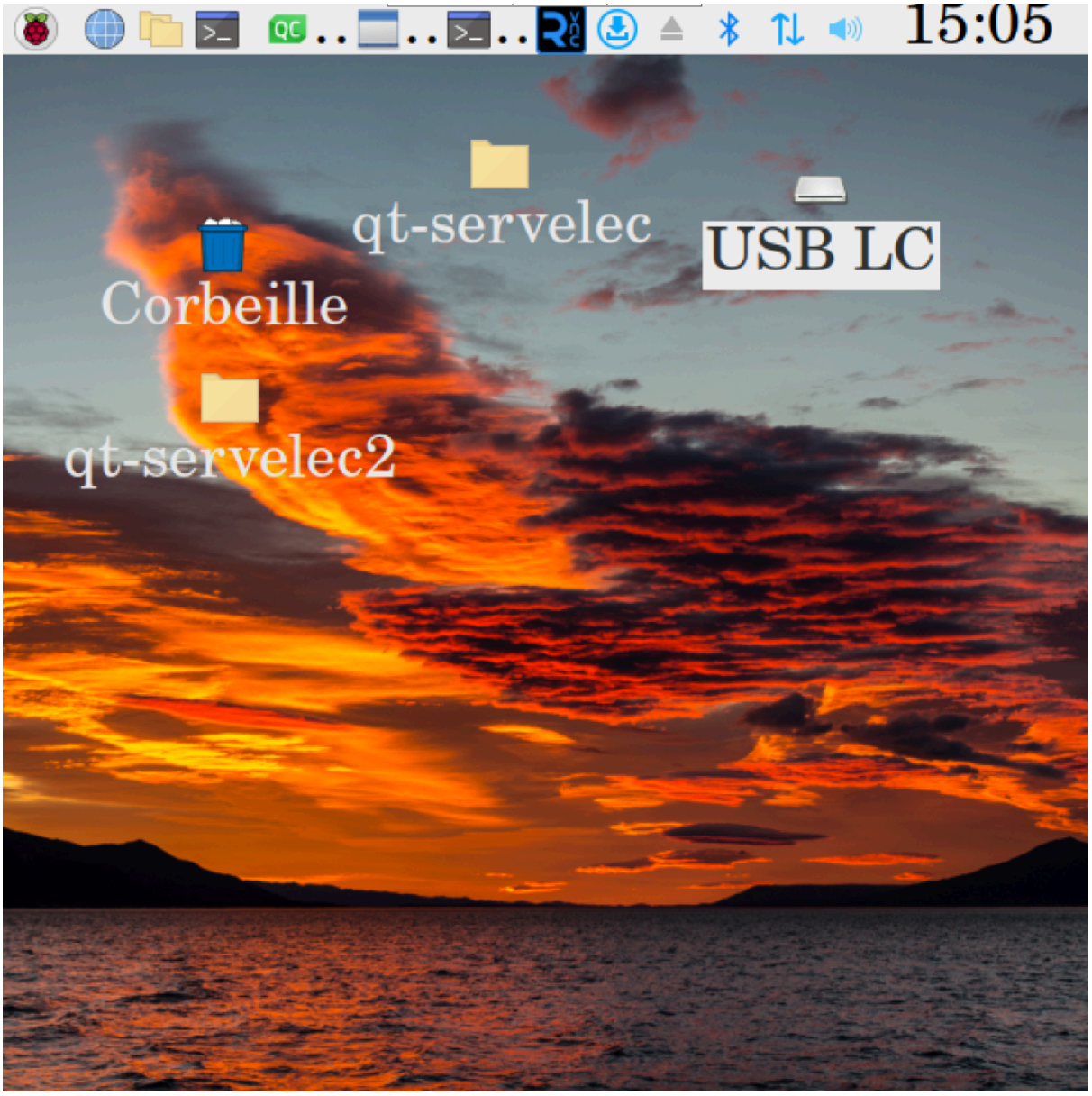


-décalage vers la gauche de 275 pixels

```
overscan_right=275
```

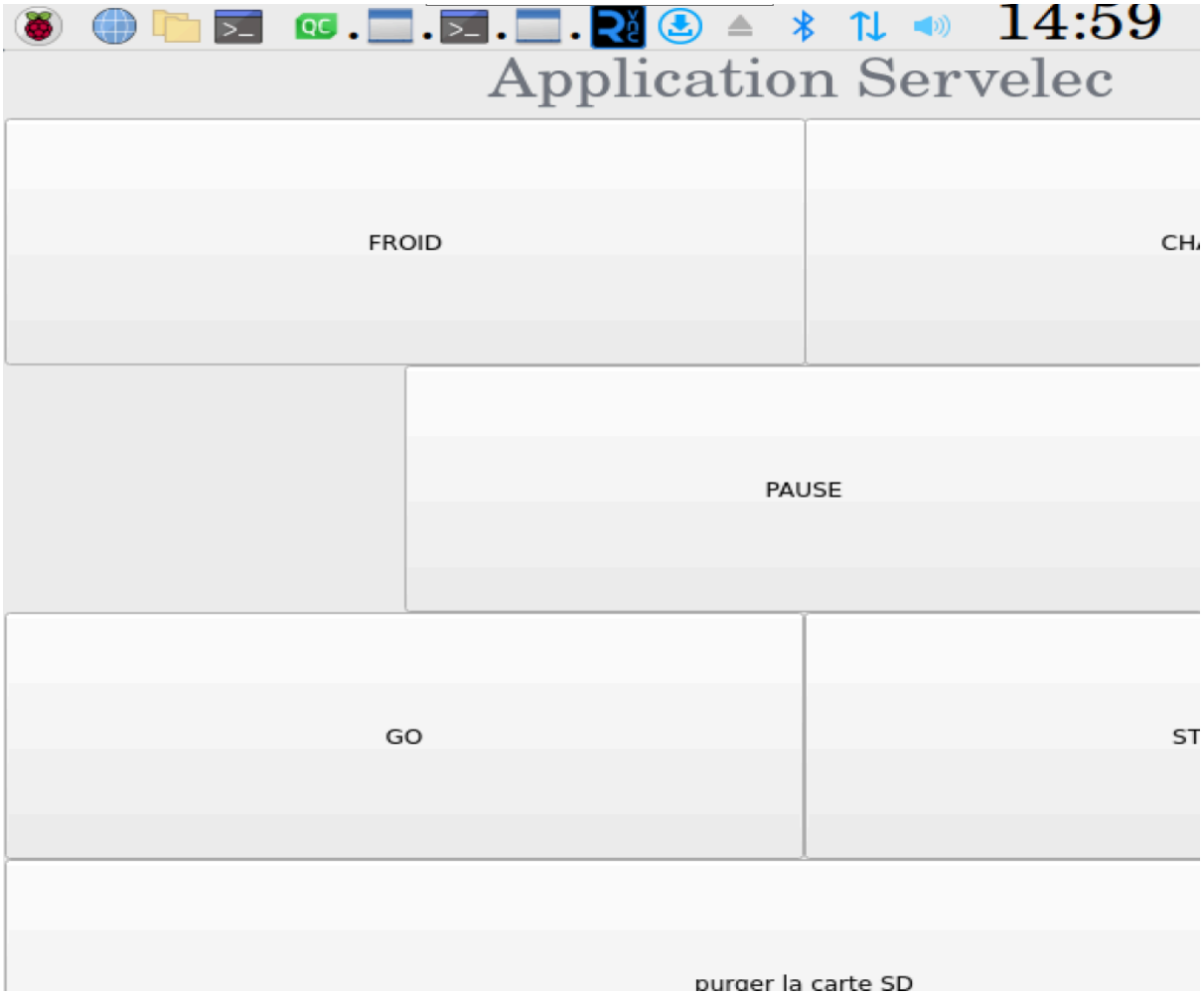
-décalage vers le haut de 275 pixels

```
overscan_bottom=275
```



objet: affichage de l'application sur l'écran principale sur un écran GPIO de 1.44 pouces.

Problème: mauvaise dimension de l'application après exécution sur ligne de commande.



solution possible modifier la taille d'affichage:

```
ui->setupUi(this);

resize(600, 600);

this->setWindowTitle("Application Servelec");
```

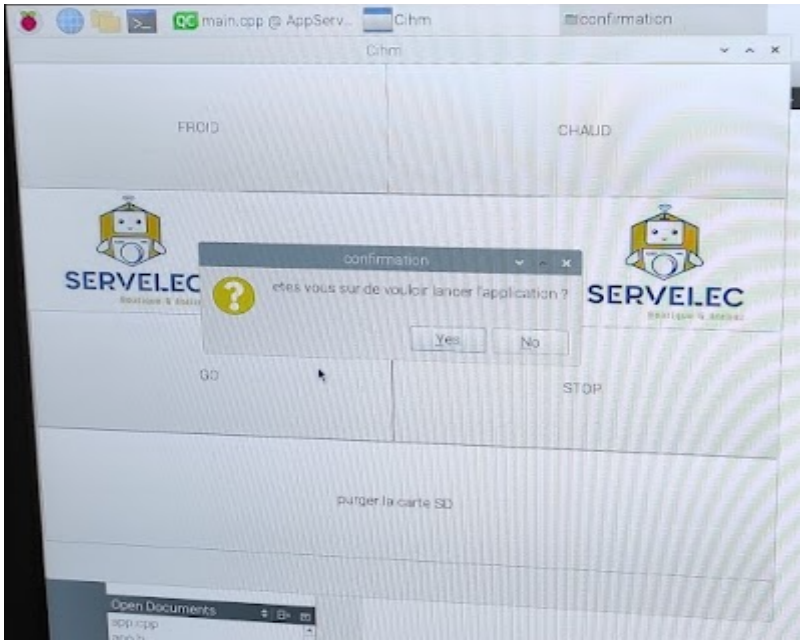
Résultat aucun changement sur invite de commande mais fonctionne après exécution sur Qt Creator.

solution actuelle: utiliser Qt Creator



objet:double cliquage du push button.

Lors du début du développement du projet, j'ai rencontré un problème au niveau de l'interface graphique de l'application.
lorsque je désirais appuyer sur un bouton de l'application celui-ci fonctionnait, cependant l'action s'exécutent à chaque fois deux fois ?
Par exemple , j'appuyais sur go et ca me disait que l'application est lancée 2 fois à la suite et ceci pour tous les boutons.



Après de nombreuses recherches et un peu d'aide j'ai découvert que lorsque que l'on mettait un push button sur une interface graphique et que l'on faisait (go to slot) l'IHM faisait automatiquement le connect alors que moi je le rajoutais ce qui causait 2 connect pour chaques fois que j'appuyais sur un bouton, d'où l'affichage du message 2 fois.
j'ai donc mis en commentaire les connect de la classe IHM et ceci a réglé le problème.

```
CIhm::CIhm(QWidget *parent) : QMainWindow(parent), ui(new Ui::CIhm), pauseCounter(0), isHotSensorSelected(false), isColdSensorSelected(false)
{
    ui->setupUi(this);

    resize(600, 600);
    this->setWindowTitle("Application Servelec");

    cStopInstance = new CStop(this);
    cPauseInstance = new CPause(this);
    fileDialog = new QFileDialog(this);
    pt100 = new CPt100();
    tempTimer = new QTimer(this); // Initialisation du QTimer

    connect(fileDialog, &QFileDialog::fileSelected, this, &CIhm::askSaveLocation);
  //  connect(ui->stopButton, &QPushButton::clicked, this, &CIhm::on_stopButton_clicked);
//    connect(ui->pauseButton, &QPushButton::clicked, this, &CIhm::on_pauseButton_clicked);
 //   connect(ui->purgeButton, &QPushButton::clicked, this, &CIhm::on_purgeButton_clicked);
    connect(cStopInstance, &CStop::recordingStoped, this, &CIhm::handleRecordingStopped);
    connect(cPauseInstance, &CPause::recordingPaused, this, &CIhm::recordingPaused);
  //  connect(ui->goButton, &QPushButton::clicked, this, &CIhm::on_goButton_clicked); // Connexion du signal au slot

    // Connexion du QTimer au slot afficherTemp
    connect(tempTimer, &QTimer::timeout, this, &CIhm::afficherTemp);

    // Désactiver le QTimer initialement
    tempTimer->stop();

    QTimer::singleShot(0, this, &CIhm::askSaveLocation);
}
```