# Documentation technique

# Projet 2IOTE NOKIA 3310

**Benjamin Maury – François-Xavier Leprovost de Saint Jean**

# SOMMAIRE

# 1. Introduction

La documentation technique fournit une description détaillée du projet, de ses objectifs et de son fonctionnement. Cette introduction vise à donner un aperçu général du projet, mettant en évidence ses principaux aspects et son contexte :

## 1.1 Objectif du projet :

L'objectif de ce projet est de simuler l'expérience du téléphone Nokia 3310 en utilisant une carte Arduino. Le projet vise à développer un programme qui permettra d'interfacer un écran LCD, un clavier et autres composants… Afin de pouvoir simuler une partie des fonctionnalités de ce téléphone.

L'objectif final est de créer une version fonctionnelle et ludique du Nokia 3310 qui peut être utilisée comme un téléphone simplifié pour Arduino ou un appareil de démonstration.

## 1.2 Description générale du système :

Le système utilise une combinaison de code logiciel et de matériel pour fournir les fonctionnalités requises. Le code logiciel exécuté sur les cartes Arduino Uno assure la gestion des contacts, la navigation dans les menus, l'affichage des informations sur les écrans LCD, la génération des tonalités des sonneries... Les composants matériels, tels que les écrans LCD I2C, les claviers et les buzzers, sont interfacés avec les cartes Arduino Uno pour permettre l'interaction avec les utilisateurs et la sortie audio des sonneries.

En combinant ces composants matériels et logiciels, le système offre une solution pratique pour l'utilisation des fonctionnalités du projet et des différents composants.

# 2. Configuration

## 2.1 Architecture Matérielle :

- Deux cartes Arduino Uno : Les cartes Arduino Uno servent de cerveau du système. Elles exécutent le code qui contrôle les fonctionnalités du système, y compris la gestion des contacts, l'affichage des informations sur les écrans LCD et la génération des tonalités des sonneries

- Deux écrans LCD I2C : Les écrans LCD I2C fournissent une interface d'affichage pour visualiser les contacts et les informations relatives aux sonneries. Ils utilisent une interface I2C pour communiquer avec les cartes Arduino Uno, ce qui permet de réduire le nombre de broches nécessaires pour la connexion.

- Deux claviers (keypad) : Les claviers sont utilisés pour la saisie des contacts et des commandes. Ils sont connectés aux cartes Arduino Uno et permettent aux utilisateurs d'entrer les informations nécessaires, telles que les noms et les numéros de téléphone des contacts, ainsi que les commandes de navigation dans le système.

- Deux buzzers : Les buzzers sont utilisés pour la génération des sonneries personnalisées. Ils produisent les tonalités spécifiées par l'utilisateur pour les différentes sonneries. Les cartes Arduino Uno contrôlent les buzzers en leur envoyant des signaux appropriés pour générer les notes musicales correspondantes.

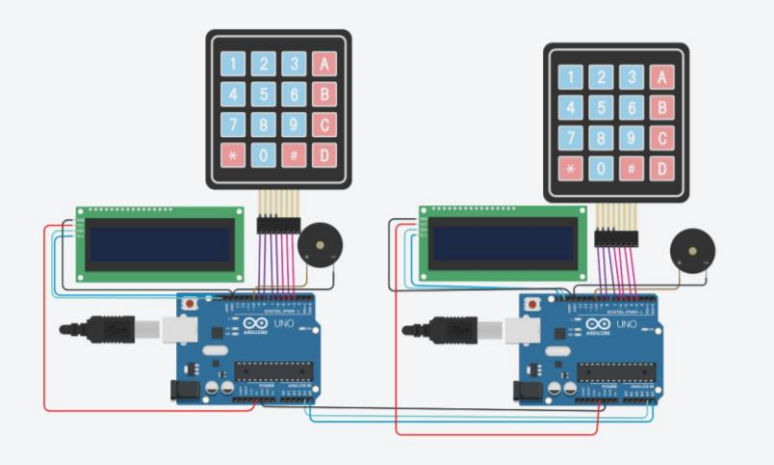
## 2.2 Architecture Logicielle :

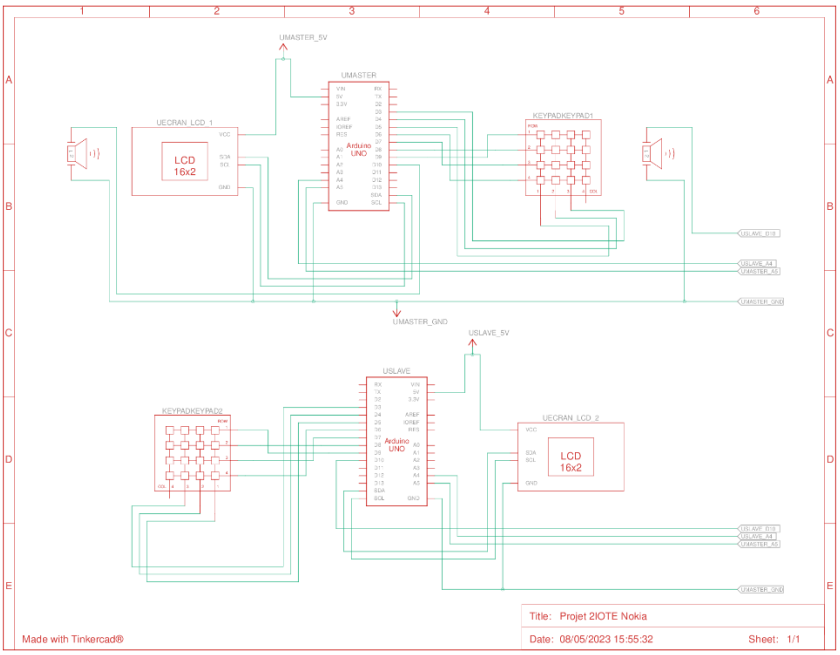
- Code logiciel Arduino : Le code logiciel est développé et téléversé sur les cartes Arduino Uno. Il est responsable de la gestion des fonctionnalités du système, telles que la gestion des contacts, l'affichage des informations sur les écrans LCD, la saisie au clavier, la génération des tonalités des sonneries, etc.

- Bibliothèques logicielles : Le code logiciel utilise des bibliothèques spécifiques, telles que la bibliothèque Wire pour la communication I2C avec les écrans LCD, la bibliothèque Keypad pour la gestion des claviers, etc. Ces bibliothèques facilitent le développement en fournissant des fonctions prédéfinies pour effectuer des opérations spécifiques.

- Communication entre les composants : Les cartes Arduino Uno interagissent avec les écrans LCD, les claviers et les buzzers via des connexions matérielles appropriées. Les informations sont échangées entre les composants pour permettre la saisie, l'affichage et la génération des tonalités des sonneries.

## 2.3 Schéma Architecture Matérielle :




1
2
3
4
5
6
A
B
C
D
E
UMASTER_5V
UMASTER
UECRAN_LCD_1
LCD
16x2
KEYPADKEYPAD1
USLAVE_D13
USLAVE_A4
UMASTER_A5
UMASTER_GND
UMASTER_GND
USLAVE_5V
USLAVE
KEYPADKEYPAD2
UECRAN_LCD_2
LCD
16x2
USLAVE_D13
USLAVE_A4
UMASTER_A5
UMASTER_GND
Made with Tinkercad®
Title: Projet 2IOTE Nokia
Date: 08/05/2023 15:55:32
Sheet: 1/1

# 3. Fonctionnalités du système

## 3.1 Ajout, modification et suppression d'un contact :

- L'ajout de contacts se fait par la fonction addContact qui demande à l'utilisateur de rentrer un nom et une adresse pour celui-ci (l'adresse représente l'adresse I2C du contact pour pouvoir envoyer les messages à celui-ci).

- La fonction editContact permet à l'utilisateur de modifié un contact si celui-ci est existant. L'utilisateur doit rentrer l'ID du contact (qui est l'ordre dans lequel ils ont été enregistrés : ex : le premier contact à l'ID 1).
  Il pourra alors changer son nom et son adresse.

- Enfin, la suppression d'un contact se fait par la fonction deleteContact où l'utilisateur doit remplir l'ID du contact à supprimer. Attention la suppression est définitive !

## 3.2 Gestion des sonneries :

- L'utilisateur peut sélectionner une sonnerie qui par la suite pourrait être la sonnerie du Nokia. Cette sonnerie il peut la jouer via le menu des musiques, en appuyant sur 1(« Play »). Il peut retourner au menu principal via la touche « # » du clavier

- Il peut aussi composer une musique avec la fonction composeMusic ou selon la touche appuyée cela va enregistrer un do, ré, mi, fa, sol, la, si. La musique enregistrée peut-être jouée en appuyant sur la touche « # »

## 3.3 Envoi de messages :

L'utilisateur peut envoyer un message à un de ses contacts. Il lui suffit de lancer la fonction sendMessage depuis le menu principal. Il doit rentrer l'ID du contact puis le message.
Enfin le message est envoyé automatiquement au contact souhaité.

## 3.4 Lecture et sauvegarde dans l'EEPROM :

Tout se fait automatiquement par deux fonctions :

- loadContact : qui charge les données des contacts depuis l'EEPROM
- saveContact : qui enregistre les données des contacts dans l'EEPROM

## 3.5 Gestion des paramètres de certains composants :

L'utilisateur peut modifier le volume du buzzer mais aussi la luminosité de l'écran LCD. Cependant, ces paramètres sont réinitialisés à chaque démarrage de l'Arduino

# 4. Description des fonctions principales

## 4.1 Fonction addContact :

La fonction addContact() permet d'ajouter un nouveau contact. Voici une description simple des étapes qu'elle suit :

- Déclarer les variables name et address pour stocker le nom et l'adresse du contact.
- Demander à l'utilisateur d'entrer le nom du contact.
- Vérifier que la longueur du nom ne dépasse pas la taille maximale autorisée.
- Stocker le nom dans la variable name.
- Demander à l'utilisateur d'entrer l'adresse du contact.
- Vérifier que l'adresse est valide (composée de chiffres et de la longueur appropriée).
- Stocker l'adresse dans la variable address.
- Copier le nom et l'adresse dans les tableaux contactNames et contactAddresses.
- Augmenter le nombre total de contacts.
- Stocker le nombre de contacts dans l'EEPROM.
- Déterminer l'adresse de départ pour les données de contact dans l'EEPROM.
- Parcourir les contacts et stocker les noms et adresses dans l'EEPROM.
- Afficher un message de confirmation de l'ajout du contact.
- Attendre pendant 2 secondes.
- Passer à la gestion du menu des contacts.

En résumé, la fonction permet à l'utilisateur d'ajouter un nouveau contact en saisissant son nom et son adresse, puis stocke les informations dans l'EEPROM et affiche un message de confirmation.

## 4.2 Fonction editContact :

La fonction editContact() permet de modifier les informations d'un contact existant dans la liste des contacts enregistrés. Voici une description détaillée de son fonctionnement :

- Charger les contacts depuis l'EEPROM.
- Demander à l'utilisateur d'entrer l'ID du contact à modifier.
- Vérifier la validité de l'ID saisi.
- Récupérer les informations du contact à partir des tableaux contactNames et contactAddresses.
- Afficher le nom actuel du contact sur l'écran LCD.
- Demander à l'utilisateur d'entrer le nouveau nom du contact.
- Mettre à jour le tableau contactNames avec le nouveau nom saisi.
- Afficher l'adresse actuelle du contact sur l'écran LCD.
- Demander à l'utilisateur d'entrer la nouvelle adresse du contact.
- Mettre à jour le tableau contactAddresses avec la nouvelle adresse saisie.
- Enregistrer les contacts modifiés dans l'EEPROM.
- Afficher un message de confirmation de la mise à jour du contact.
- Revenir au menu principal après un court délai.

En résumé, la fonction permet à l'utilisateur de modifier un contact en utilisant directement les tableaux contactNames et contactAddresses pour stocker les modifications.

## 4.3 Fonction deleteContact :

La fonction deleteContact() permet de supprimer un contact existant. Voici un résumé simple des étapes qu'elle suit :

- Charger les contacts à partir de l'EEPROM.
- Demander à l'utilisateur d'entrer l'ID du contact à supprimer.
- Vérifier la validité de l'ID saisi.
- Si l'ID est valide, réduire le nombre total de contacts et mettre à jour cette valeur dans l'EEPROM.
- Déplacer les contacts suivants vers l'emplacement du contact supprimé pour combler le vide.
- Afficher un message de confirmation de suppression du contact.
- Définir la variable de contrôle returnToMainMenu à true pour revenir au menu principal.
- Terminer la fonction.

En résumé, la fonction permet à l'utilisateur de supprimer un contact existant en entrant son ID, puis met à jour les informations des contacts restants dans l'EEPROM.

## 4.4 Fonction sendMessage :

La fonction sendMessage() permet d'envoyer un message à un contact spécifié. Voici une description simple des étapes qu'elle suit :

- Lire le nombre de contacts à partir de l'EEPROM.
- Vérifier s'il y a des contacts en vérifiant si le nombre de contacts est égal à zéro. Si c'est le cas, afficher un message indiquant l'absence de contacts, puis revenir au menu principal.
- Afficher un message demandant d'entrer l'ID du contact.
- Lire l'entrée de l'utilisateur en utilisant la fonction createMessage(), en s'assurant qu'il s'agit d'un nombre valide correspondant à l'ID d'un contact existant.
- Convertir l'ID du contact en entier et le stocker dans la variable selectedContact.
- Vérifier si l'ID du contact sélectionné est valide. Si ce n'est pas le cas, afficher un message d'erreur, puis revenir à l'étape 3 pour demander un nouvel ID.
- Charger les contacts à partir de l'EEPROM en appelant la fonction loadContacts().
- Récupérer l'adresse du contact choisi en utilisant l'ID du contact et soustraire 1 pour correspondre à l'indice du tableau.
- Afficher un message demandant d'entrer le message à envoyer.
- Lire le message de l'utilisateur en utilisant la fonction createMessage(), en s'assurant que sa longueur ne dépasse pas la taille maximale définie.
- Convertir l'adresse du contact en entier.
- Initialiser la communication avec le dispositif correspondant à l'adresse du contact en utilisant la bibliothèque Wire.
- Écrire le message dans le flux de données vers le dispositif cible à l'aide de la fonction Wire.write().
- Terminer la transmission de données en appelant la fonction Wire.endTransmission().
- Effacer l'entrée de l'utilisateur.
- Afficher un message indiquant que le message a été envoyé avec succès, puis revenir au menu principal.

En résumé, la fonction permet à l'utilisateur de sélectionner un contact existant, d'entrer un message et de l'envoyer à l'adresse correspondante du contact via la communication I2C.

## 4.5 Fonction playCurrentMusic :

La fonction playCurrentMusic() permet de jouer la sonnerie actuelle sélectionnée. Voici comment elle fonctionne :

- Elle appelle la fonction getRingtonePersonnalised() pour récupérer la sonnerie personnalisée actuelle.

- Elle lit la valeur de la sonnerie actuelle depuis la mémoire EEPROM en utilisant la fonction EEPROM.get().

- En fonction de la valeur de currentRingtone, elle récupère les notes et les durées correspondantes de la sonnerie à partir des tableaux notesRingtone1 à notesRingtone7 et durationsRingtone1 à durationsRingtone7.

- Ensuite, elle utilise une boucle pour jouer chaque note de la sonnerie en utilisant la fonction tone(). La durée de chaque note est calculée en utilisant la formule noteDuration = 1000 / currentRingtoneDurations[i].

- Après avoir joué chaque note, elle attend un court délai de noteDuration + 50 millisecondes pour créer une pause entre les notes.

- Ainsi, la fonction joue la sonnerie actuelle sélectionnée en utilisant le buzzer connecté au pin BUZZER_PIN.

En résumé, la fonction playCurrentMusic() permet de jouer la sonnerie actuelle sélectionnée à l'aide du buzzer. Elle récupère les notes et les durées de la sonnerie à partir des tableaux correspondants et les joue en utilisant la fonction tone().

## 4.6 Fonction handleCurrentRingtone :

La fonction handleCurrentRingtone() permet à l'utilisateur de sélectionner la sonnerie actuelle à partir d'une liste de sonneries disponibles. Voici comment elle fonctionne :

- Elle appelle la fonction getRingtonePersonnalised() pour récupérer la sonnerie personnalisée actuelle.
- Elle initialise une variable waitingRingtone à 0, qui sera utilisée pour indiquer si une sonnerie est en attente d'être sélectionnée.
- Ensuite, elle entre dans une boucle infinie qui affiche le menu "Select Ring 1 to 7" sur l'écran LCD.
- L'utilisateur peut appuyer sur les touches numériques de 1 à 7 pour sélectionner la sonnerie correspondante.
- Lorsqu'une touche est pressée, la variable currentRingtone est mise à jour avec le numéro de la sonnerie sélectionnée, et la variable waitingRingtone est définie sur 1 pour indiquer qu'une sonnerie est en attente d'être sélectionnée.
- Si l'utilisateur appuie sur la touche '#', la fonction handleMusicMenu() est appelée pour revenir au menu principal.
- Si l'utilisateur appuie sur une touche qui ne correspond à aucune sonnerie disponible, un message "Ringtone not defined" est affiché sur l'écran LCD.
- Enfin, la sonnerie sélectionnée est enregistrée dans la mémoire EEPROM en utilisant la fonction EEPROM.put().

En résumé, la fonction handleCurrentRingtone() permet à l'utilisateur de sélectionner la sonnerie actuelle parmi les sonneries disponibles, et la sonnerie sélectionnée est enregistrée dans la mémoire EEPROM.

## 4.7 Fonction composeMusic:

La fonction composeMusic() permet à l'utilisateur de composer une mélodie en utilisant le clavier du système. Voici comment elle fonctionne :

- Elle commence par appeler la fonction getRingtonePersonnalised() pour récupérer la sonnerie personnalisée actuelle.
- Elle définit quelques variables et affiche le menu "Compose Music" sur l'écran LCD.
- Ensuite, elle entre dans une boucle qui permet à l'utilisateur de sélectionner les notes de la mélodie.
- Lorsque l'utilisateur appuie sur une touche correspondant à une note de musique (de 1 à 9), la fréquence de la note est attribuée à la variable noteFreq.
- Ensuite, l'utilisateur appuie sur une autre touche pour définir la durée de la note. La durée est calculée en multipliant la valeur de la touche par 50.
- Les fréquences et les durées des notes sont enregistrées dans les tableaux notesRingtone et durationRingtone, respectivement, et la longueur de la mélodie (ringtoneLength) est augmentée.
- La note est jouée en utilisant la fonction tone() pour produire un son sur le buzzer.
- L'utilisateur peut sélectionner jusqu'à 20 notes pour composer sa mélodie.
- Une fois que la mélodie est composée, elle est enregistrée dans la mémoire EEPROM, selon certaines conditions.
- La fonction se termine.

En résumé, la fonction composeMusic() permet à l'utilisateur de composer une mélodie en sélectionnant des notes à l'aide du clavier, et la mélodie composée est enregistrée dans la mémoire EEPROM.

## 4.8 Fonction loadContact :

La fonction loadContacts() permet de charger les contacts à partir de l'EEPROM. Voici une description simple des étapes qu'elle suit :

- Initialiser l'EEPROM pour pouvoir y accéder.
- Lire le nombre total de contacts stockés dans l'EEPROM et le stocker dans la variable numContacts.
- Parcourir chaque contact en utilisant une boucle.
- Calculer l'adresse de chaque contact en fonction de son indice dans la boucle.
- Récupérer le nom du contact à partir de l'EEPROM et le stocker dans le tableau contactNames.
- Récupérer l'adresse du contact à partir de l'EEPROM et la stocker dans le tableau contactAddresses.

En résumé, la fonction charge les contacts précédemment enregistrés dans l'EEPROM en récupérant les noms et les adresses correspondants, puis les stocke dans les tableaux contactNames et contactAddresses respectivement.

## 4.9 Fonction saveContact :

La fonction saveContacts() permet de sauvegarder les contacts dans l'EEPROM. Voici une description simple des étapes qu'elle suit :

- Initialiser l'EEPROM pour pouvoir y accéder.
- Calculer le nombre total de contacts en divisant la taille du tableau contactNames par la taille d'un élément (sizeof(contactNames) / sizeof(contactNames[0])) et le stocker dans la variable numContacts.
- Enregistrer le nombre total de contacts dans l'EEPROM à l'adresse 0 à l'aide de la fonction EEPROM.put().
- Parcourir chaque contact en utilisant une boucle.
- Calculer l'adresse de chaque contact en fonction de son indice dans la boucle.
- Créer une chaîne de caractères contactData en concaténant le nom et l'adresse du contact, séparés par une virgule.
- Enregistrer la chaîne contactData dans l'EEPROM à l'adresse correspondante à l'aide de la fonction EEPROM.put().
- Terminer l'accès à l'EEPROM en appelant la fonction EEPROM.end().

En résumé, la fonction sauvegarde les contacts dans l'EEPROM en enregistrant le nombre total de contacts, suivi des données de chaque contact, sérialisées en une seule chaîne de caractères.

## 4.10 Fonction createMessage:

La fonction createMessage() permet de créer un message en utilisant un clavier matriciel (keypad) et d'afficher le message en temps réel sur un écran LCD. Voici une description simple des étapes qu'elle suit :

- Initialiser les variables nécessaires, telles que le message, la touche précédente (prevKey), le temps de pression précédent (prevKeyPressTime) et le compteur de touches (keyCount).

- Entrer dans une boucle qui se répète tant que la longueur du message est inférieure à la longueur spécifiée.

- Lire la touche actuellement pressée à l'aide de la fonction keypad.getKey().

- Vérifier si une touche a été pressée en comparant la touche actuelle à '\0' (caractère nul).

- Si une touche a été pressée, vérifier si c'est la même touche que la précédente et si le délai entre les pressions est inférieur ou égal à 1 seconde. Si c'est le cas, cela indique une pression répétée de la même touche dans un court laps de temps.

- Si le compteur de touches (keyCount) est supérieur ou égal à 5, le réinitialiser à 1 pour éviter de dépasser la limite de caractères pour une touche.

- Sinon, supprimer le dernier caractère ajouté au message, puis obtenir le caractère suivant de la touche avec le compteur de touches en utilisant la fonction makeKeypadToNokia3310(). Si le caractère obtenu n'est pas '\0' (caractère nul), l'ajouter au message. Ensuite, augmenter le compteur de touches.

- Si une nouvelle touche a été pressée (différente de la précédente) :

- Stocker la nouvelle touche comme la touche précédente.

- Enregistrer le temps de pression actuel comme le temps de pression précédent.

- Réinitialiser le compteur de touches à 1.

- Si la touche est '#', sortir de la boucle, car cela indique la fin de la saisie du message.

- Si la touche est '*' et que le message a une longueur supérieure à zéro, supprimer le dernier caractère du message.

- Sinon, obtenir le premier caractère correspondant à la touche unique en utilisant la fonction makeKeypadToNokia3310() avec un compteur de touches de 1. Si le caractère obtenu n'est pas '\0', l'ajouter au message.

- Effacer l'écran LCD, puis afficher le message actuel.

- Retourner le message créé une fois que sa longueur atteint la longueur spécifiée.

En résumé, la fonction permet à l'utilisateur d'entrer un message en utilisant un clavier matriciel, avec la possibilité de saisir des caractères répétés en maintenant une touche enfoncée. Le message est affiché en temps réel sur l'écran LCD.

## 4.11 Fonction makeKeypadToNokia3310:

La fonction makeKeypadToNokia3310() convertit une touche pressée sur un clavier matriciel en un caractère correspondant utilisé sur un téléphone Nokia 3310. Voici une description simple du fonctionnement :

- La fonction prend en entrée la touche pressée (key) et le compteur de touches (keyCount).
- Elle utilise une structure switch pour vérifier la valeur de key et déterminer quel caractère retourner en fonction de la touche et du compteur de touches.
- Selon la valeur de key, plusieurs cas sont évalués pour différents compteurs de touches (keyCount) :
- Chaque cas correspond à une combinaison spécifique de touche et de compteur de touches.
- Si la combinaison correspondante est trouvée, la fonction retourne le caractère correspondant pour cette combinaison.
- Si aucune combinaison ne correspond, la fonction retourne '\0' pour indiquer un caractère non pris en charge.
- Si key ne correspond à aucun des cas évalués, la fonction retourne également '\0'.

En résumé, la fonction permet de mapper les touches du clavier matriciel aux caractères correspondants du téléphone Nokia 3310, en prenant en compte le compteur de touches pour gérer les caractères multiples associés à une même touche.

Note : La fonction tient compte uniquement des touches numériques (de 0 à 9) et ne prend pas en charge les autres touches du clavier matriciel.

# 5. Mémoire et performance

Le code a été optimisé pour optimiser l'utilisation de la mémoire et maximiser l'efficacité du système. Les optimisations suivantes ont été implémentées :

- Utilisation de types de données appropriés : Pour économiser de la mémoire, les types de données les plus adaptés ont été utilisés. Par exemple, les variables qui stockent les identifiants des contacts sont déclarées en tant que type byte au lieu d'utiliser des types de données plus larges comme int.

- Gestion efficace de l'EEPROM : L'EEPROM est utilisée pour stocker les contacts. Plutôt que de stocker chaque contact en tant que structure de données séparée, les noms et adresses des contacts sont stockés de manière consécutive dans l'EEPROM, ce qui permet d'économiser de l'espace mémoire. De plus, lors de la sauvegarde des contacts, la fonction EEPROM.put() est utilisée pour écrire les données de manière optimisée en une seule opération, réduisant ainsi le nombre d'accès à l'EEPROM.

- Utilisation de tableaux : Les noms et adresses des contacts sont stockés dans des tableaux multidimensionnels. Cela permet de gérer les contacts de manière efficace et compacte. Les opérations de copie et de déplacement des contacts sont effectuées en utilisant des boucles, minimisant ainsi la consommation de mémoire.

- Suppression des contacts sans décalage : Lorsqu'un contact est supprimé, plutôt que de décaler tous les contacts suivants vers l'indice précédent, les noms et adresses des contacts sont simplement copiés de l'indice suivant vers l'indice actuel. Cela permet d'économiser du temps et de la mémoire, évitant ainsi le décalage de chaque élément.

- Gestion de la mémoire dynamique : L'utilisation de la bibliothèque String a été minimisée au profit de l'utilisation de tableaux de caractères (char) pour stocker les noms, les adresses et les messages. Les tableaux de caractères sont plus efficaces en termes d'utilisation de la mémoire, car ils sont statiques et n'allouent que la quantité de mémoire nécessaire pour stocker les données.

Ces optimisations de mémoire contribuent à rendre le système plus efficace et à réduire la consommation de ressources, ce qui est particulièrement important lorsque l'on travaille avec des dispositifs à mémoire limitée tels que les microcontrôleurs.

# 6. Conclusion

La documentation fournit des explications détaillées sur chaque fonction, en les décrivant de manière simple et compréhensible. Elle met en évidence les principaux éléments et mécanismes utilisés, tels que le chargement et la sauvegarde des contacts dans l'EEPROM, la création de messages à l'aide du clavier matriciel, l'envoi de messages via un bus I2C, et d'autres fonctionnalités essentielles du système.

En conclusion, cette documentation technique décrit un système de gestion de contacts utilisant un écran LCD, un clavier matriciel et une mémoire EEPROM. Les différentes fonctions implémentées permettent d'ajouter, supprimer et envoyer des messages à des contacts enregistrés. Elle fournit une ressource précieuse pour la compréhension et l'utilisation du système de gestion de contacts, offrant aux développeurs les connaissances nécessaires pour créer des applications similaires ou étendre les fonctionnalités existantes.