Département de génie informatique et génie logiciel

Cours INF1900:
Projet initial de système embarqué

Travaux pratiques 7 et 8

Production de librairie statique et stratégie de débogage

Par l'équipe

No 0314

Noms:

Sara Dakir
Asmaa Sabouri
Chaïmaa Mesbah
Imane Abdeljalil

Date:
28 octobre 2024

Partie 1 : Description de la librairie

Décrire la librairie construite et formée (définitions, fonctions ou classes, utilité, etc.) pour que cette partie du travail soit bien documentée pour la suite du projet au bénéfice de tous les membres de l'équipe.

Partie 2 : Décrire les modifications apportées au Makefile de départ

Décrire les quelques modifications apportées au Makefile de la librairie pour démontrer votre compréhension de la formation des fichiers. Faire de même pour les modifications apportées au Makefile du code (bidon) de test qui utilise cette librairie.

Le rapport total ne doit par dépasser 7 pages incluant la page couverture.

Barème: vous serez jugé sur:

- *La qualité et le choix de vos portions de code choisies (5 points sur 20)*
- *La qualité de vos modifications aux Makefiles (5 points sur 20)*
- *Le rapport (7 points sur 20)*
  - *Explications cohérentes par rapport au code retenu pour former la librairie (2 points)*
  - *Explications cohérentes par rapport aux Makefiles modifiés (2 points)*
  - *Explications claires avec un bon niveau de détails (2 points)*
  - *Bon français (1 point)*
- *Bonne soumission de l'ensemble du code (compilation sans erreurs …) et du rapport selon le format demandé (3 points sur 20)*

Partie 1 : Description de la librairie

Introduction

Notre librairie est un ensemble de classes C++ conçues pour faciliter le contrôle et l'interaction avec divers composants d'un robot basé sur un microcontrôleur AVR. Cette librairie offre une abstraction de haut niveau pour des fonctionnalités telles que le contrôle des DELs, la gestion des boutons, la conversion analogique-numérique, la gestion du temps, le contrôle des moteurs et la communication série.

Composants de la Librairie

1. Classe Del

Utilité : Contrôle d'une DEL bicolore.

Fonctions principales :

- allumer (Couleur couleur, uint16_t delai = 0) : Allume la DEL dans une couleur spécifique (ambre, rouge, vert). On a rajouté l'option délai pour éviter de les mettre dans le main question d'alléger notre code. Ce paramètre a un délai de 0 ms par défaut sauf si on désire le changer dépendamment du contexte.
- éteindre() : Éteint la DEL.
- clignoter (Couleur couleur, uint8_t nombreClignotements, uint16_t delai) : Fait clignoter la DEL avec une couleur, nombre de clignotements et délai spécifiés.

Utilisation : Permet de gérer facilement les signaux visuels du robot.

Pins utilisés : broches B0, B1

2. Classe Bouton

Utilité : La classe Bouton permet de gérer les interactions avec des boutons poussoirs connectés aux broches du microcontrôleur. Elle inclut des fonctions pour vérifier si un bouton est appuyé ou relâché, et elle gère le comportement des boutons.
Le **bouton blanc (sur le breadbord)**; lorsqu'il est appuyé, la lecture donne 0 et lorsqu'il est relâché, elle donne 1. Tandis que pour le bouton noir (sur la carte mere); lorsqu'il est appuyé la lecture donne 1 et lorsqu'il est relâché, elle donne 0.
Dans notre classe Bouton, un constructeur ; Bouton (**uint8_t** pinBouton) est mis en place pour appeler la logique qui doit etre utilisée avec les fonctions de la classe en changeant le parametre (pinBouton).

Fonctions principales :

- estAppuyer() : Vérifie si le bouton est appuyé.
- estRelacher() : Vérifie si le bouton est relâché.
- desactiverInteruption() : Désactive l'interruption associée au bouton. Utile pour éviter des déclenchements intempestifs lorsque l'interruption n'est plus nécessaire.

Utilisation : Facilite la détection des entrées utilisateur via les boutons.

Pins utilisés :

- **Bouton noir** : Connecté à la broche PD2.
- **Bouton blanc** : Connecté à la broche PD3

On a choisi les pins PD2 et PD3 car ils sont les seuls supportant les interruptions externes INT0/ INT1

3. Classe can (Convertisseur Analogique/Numérique)

Utilité : cette classe permet de convertir des signaux analogiques en valeurs numériques via le convertisseur analogique-numérique intégré du microcontrôleur AVR. Elle simplifie la lecture des tensions analogiques sur les broches du port A.
Le CAN convertit une tension analogique (0-5V) en une valeur numérique sur 10 bits (0-1023). Notre classe effectue automatiquement un décalage pour retourner une valeur sur 8 bits (0-255). Le constructeur initialise le CAN avec les paramètres appropriés, et le destructeur le désactive pour économiser l'énergie.

Fonction principale :

- lecture(uint8_t pin) : Lit une valeur analogique sur une broche spécifique (0-7).

Utilisation : Permet d'interfacer avec des capteurs analogiques.

Pins utilisés : Port A (0-7)

4. Classe Minuterie

Utilité : La classe Minuterie permet de gérer précisément les intervalles de temps en utilisant le Timer1 du microcontrôleur. Elle offre des fonctionnalités pour générer des interruptions à intervalles réguliers, essentielles pour les tâches nécessitant une temporisation précise comme le clignotement de DELs, la lecture périodique de capteurs ou la génération de signaux temporels.
Le Timer1 est configuré en mode CTC (Clear Timer on Compare Match), ce qui signifie que le compteur se réinitialise automatiquement lorsqu'il atteint la valeur de comparaison définie. Cette approche garantit une précision temporelle supérieure aux délais logiciels.
On a choisi le Timer1, car il est sur 16bits ce qui permet d'obtenir une résolution plus fine et des délais plus longs par rapport au Timer0.
Dans notre classe Minuterie, un constructeur Minuterie() est mis en place pour initialiser le Timer1 en mode CTC et configurer les interruptions nécessaires. La fréquence CPU de 8MHz sert de base pour tous les calculs temporels.

Fonctions principales :

- Minuterie() : Le constructeur de la classe initialise automatiquement le Timer1 en mode CTC et configure les interruptions. Cela nous permet de commencer à utiliser la minuterie immédiatement après la création de l'objet.
- demarrer(uint16_t periode, uint16_t prescaler = PRESCALER_1024) : Démarre la minuterie avec une période et un prescaler précis.

- arreter() : Arrête le minuterie.
- reinitialiser() : Réinitialise le minuterie.

Utilisation : Permet de créer des délais précis et de synchroniser des actions.

5. Classe Roue

Utilité : La classe Roue permet de contrôler le déplacement d'un robot en gérant deux moteurs via PWM. Elle offre un contrôle précis de la vitesse et de la direction de chaque roue, permettant des mouvements complexes comme l'avance, le recul, les virages et la rotation sur place.
Le Timer2 est configuré en mode Phase Correct PWM pour un contrôle fluide de la vitesse des moteurs, avec un prescaler de 64 pour une fréquence PWM optimale.
Dans notre classe Roue, le constructeur Roue() initialise les broches et le Timer2 pour la génération du signal PWM. La vitesse est contrôlée en pourcentage (0-100%) pour une utilisation intuitive.

Fonctions principales :

- Roue() : constructeur pour initialiser les paramètres nécessaires au contrôle des moteurs. Lorsequne instance de la classe `Roue` est créé, il configure les broches appropriees pour la direction et le controle de la vitesse des deux roues. Il initialse egalement le Timer2 en mode Phase Correct PWM.
- controlerRoue(Roues roue, Direction direction, uint8_t vitessePourcentage) : Cette fonction permet de contrôler une seule roue, en spécifiant sa direction (avant ou arrière) et sa vitesse en pourcentage.
- avancer (uint8_t vitesse), reculer(uint8_t vitesse) : Fait avancer ou reculer le robot.
- tournerGauche(uint8_t vitesse, facteurReduction = 2), tournerDroite(uint8_t vitesse, facteurReduction = 2) : Fait tourner le robot. Le paramètre facteurReduction détermine à quel point la vitesse de la roue en question sera diminuée par rapport à la vitesse initiale.
- arreter() : Arrête le mouvement du robot.
- accelerer(uint8_t roue, uint8_t vitesseCible, uint8_t increment) : Accélère progressivement une roue.
- ralentir(uint8_t roue, uint8_t vitesseCible, uint8_t increment) : Ralentit progressivement une roue.

Utilisation : Permet un contrôle précis des mouvements du robot.
Pins utilisés :

- Roue gauche : PD6 (Contrôle PWM de la vitesse), et PD4(contrôle de la direction).
- Roue droite : PD7 (Contrôle PWM de la vitesse), et PD5(contrôle de la direction).

6. Classe Usart

Utilité : La classe Usart gère la communication série (UART) entre le microcontrôleur AVR et un périphérique externe (PC, terminal série). Elle permet la transmission et réception de données en format série asynchrone, essentielle pour le débogage et le contrôle du système.
Fonctions principales :

- transmettre(uint8_t donnee) : Envoie un octet de données.
- transmettreDonne(const char *message) : Envoie une chaîne de caractères.
- transmettreDonne(uint8_t donnee) : Envoie un entier de 8 bits.

Nous avons donné à ces fonctions le même nom afin de pouvoir appeler ``une même fonction`` pour envoyer des chaines de caractères ou des entiers. Cela rend la programmation plus simple

Utilisation : Facilite la communication entre le robot et un ordinateur ou d'autres dispositifs.

Pins utilisés :

- TX (PD1) : Transmission des données
- RX (PD0) : Réception des données

7. Classe Debug

Utilité : cette classe facilite le débogage du code. Elle permet d'envoyer des messages de débogage via l'interface USART. Grace a cette classe, on pourra visualiser les valeurs des variables pendant l'exécution du programme.

Fonctions principales :

- debugPrintNum(uint_8t donnee) : transmet les donnes de débogage via l'interface USART. Elle crée une instance de la classe Usart et appelle la méthode `transmettreDonnee` qui prends un uint_8t comme argument pour envoyer des entiers sur 8 bits.

- debugPrintChar(const char *message) : Fonctionne de la même manière que debugPrintNum(uint_8t donnee), mais celle-ci appelle la méthode `transmettreDonnee` qui prends un argument de type const char * pour envoyer des chaines de caractères.

- DEBUG_PRINT_NUM(X) : Une macro qui simplifie l'appel a `debugPrintNum` dans le code. Si le mode de débogage est activé (defini par #define DEBUG), la fonction `debugPrintNum` est appelée. Sinon, l'appel est ignoré.

DEBUG_PRINT_CHAR(X) : Une macro qui simplifie l'appel à `debugPrintChar` dans le code. Si le mode de débogage est activé (défini par #define DEBUG), la fonction `debugPrintChar` est appelée. Sinon, l'appel est ignoré.

8. Classe mémoire24

Utilité : La librairie Memoire24 offre une interface simple et efficace pour l'accès aux mémoires EEPROM via I2C. Grâce à sa structure orientée objet, elle facilite la gestion des opérations de lecture et d'écriture tout en respectant les protocoles I2C standards.

Fonctions principales :

- init() : Initialise le port série et l'horloge de l'interface I2C.
- choisir_banc() : Permet de choisir un banc de mémoire.
- lecture() et ecriture() : Deux variantes pour la lecture et l'écriture, respectivement, soit une seule donnée, soit un bloc de données.

Partie 2 : Makefile de la librairie

Afin de pouvoir compiler tous les programmes de la librairie avec la commande ``make``, les modifications suivantes ont été apportées au Makefile fourni.

- **PROJECTNAME=librairie1900 :** Cette ligne défini le nom de la librairie.
- **PRJSRC= $(wildcard *.cpp) :** Cette ligne permet de compiler tous les fichiers .cpp du répertoire sans avoir à tous les lister.
- **TRG=lib$(PROJECTNAME).a:** Cette ligne permet de produire une biblioteque statique
- **L'abscence de la ligne AVRDUDE=avrdude:** Cette a été retirée car ce makefile n'a pas pour but de programmer le ATMega324, mais uniquement de compiler la librairie.
- **rm *.o :** Permet de supprimer les fichiers .o, générés par la compilation, afin de garder le répertoire propre.
- **rm *.d :** Permet de supprimer les fichiers .d, générés par la compilation, afin de garder le répertoire propre.

Makefile du fichier Exec

- **PRJSRC= $(wildcard *.cpp) :** Cette ligne permet de compiler tous les fichiers .cpp du répertoire sans avoir à tous les lister.
- **INC= -I ../lib :** Cette ligne permet au compilateur de repérer ou se situent les fichiers d'entêtes définis dans la librairie liée.
- **LIBS= -L ../lib -l librairie1900 :** Cette ligne permet au compilateur de trouver le répertoire de la librairie et de la lier au répertoire exec afin de permettre l'usage de fichiers d'entêtes se situant dans la librairie.
- **rm *.o :** Permet de supprimer les fichiers .o, générés par la compilation, afin de garder le répertoire propre.
- **rm *.d :** Permet de supprimer les fichiers .d, générés par la compilation, afin de garder le répertoire propre.
- **.PHONY: all install clean debug :** L'ajout de ``debug`` permet d'utiliser la commande ``make debug``
- **debug: CFLAGS += -DDEBUG -g**
  debug: $(TRG) $(HEXROMTRG) : Ces deux lignes permettent d'activer le mode deboggage, donc pouvoir appeler les macros de deboggage.

Conclusion

En résumé, notre librairie offre une base flexible pour notre projet, simplifiant le développement et la gestion des composants robotiques. Elle nous permettra de se concentrer sur l'innovation tout en assurant une collaboration efficace et une évolution aisée du projet.