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

27 octobre 2023

Équipe: 0626

ALCINDOR, Jodel
BLANCHARD, Noah
FOLEFACK TEMFACK, Samira
SAKA Ndzana, Saïd
ZAMA, Matei

Partie 1: Description de la librairie

La librairie ci-dessous est constituée de plusieurs classes facilitant le contrôle du robot. En effet, elle permet de contrôler les mécanismes de fonctionnement du robot les plus importants, dont la motricité au niveau des roues, la navigation de ce dernier, le contrôle de la Led bicolore présente sur le robot, mais également d'autres mécanismes d'interruptions et de communication avec l'extérieur.

Classe Bouton

La classe Bouton est conçue pour gérer les interruptions externes d'un microcontrôleur AVR. Elle offre la possibilité de configurer les interruptions pour être déclenchées sur un front montant, un front descendant ou les deux.

Attributs

- **_int_N :** un entier non signé sur 8 bits qui contient le numéro de l'interruption externe à gérer

Constructeur

- **Bouton(uint8_t int_N_p)**: construit un objet de type Bouton avec un numéro d'interruption spécifié.

Méthodes

- **setRisingEdge() :** permet d'activer l'interruption pour qu'elle soit déclenchée sur le front montant.
- **setFallingEdge() :** permet d'activer l'interruption pour qu'elle soit déclenchée sur le front descendant.
- **setAnyEdge() :** permet d'activer l'interruption pour qu'elle soit déclenchée sur les deux fronts (montants et descendants)
- **enableInterrupt() :** permet d'activer l'interruption spécifiée dans l'attribut **_int_N**
- **disableInterrupt() :** permet de désactiver l'interruption spécifiée dans l'attribut **_int_N**
- **reset () :** désactive les interruptions et réinitialise les paramètres à leur valeur par défaut

Classe LED

La classe LED est conçue pour gérer une LED bicolore (verte et rouge). Elle permet de contrôler l'état de la LED en la rendant verte, rouge, éteinte ou ambre (en alternant rapidement entre le vert

et le rouge).

Attributs

- **_port**: un pointeur vers le registre associé au port de sortie de la LED
- **_mode**: un pointeur vers le registre de mode (souvent le registre de direction) pour le port associé à la LED
- **_greenLed**: Un entier non signé sur 8 bits représentant la broche de la LED verte.
- **_redLed**: Un entier non signé sur 8 bits représentant la broche de la LED rouge.

Constructeur

- **LED (Register port, Register mode, uint8_t greenLed, uint8_t redLed)**: Construit un objet de type LED avec les paramètres spécifiés. Il configure également les broches spécifiées comme sorties.

Méthodes

- **turnOffLed():** Éteint la LED.
- **turnLedRed():** Allume la LED en rouge.
- **turnLedGreen():** Allume la LED en vert.
- **turnLedAmber():** Fait clignoter la LED en alternance entre vert et rouge pour donner l'impression qu'elle est de couleur ambre. La durée de chaque couleur est définie par la constante **DELAY_AMBER_COLOR**.

Classe Timer

La classe Timer offre un outil de configuration et de gestion des minuteries (timers). Cette classe permet de configurer, activer, désactiver et réinitialiser la minuterie.

Structure TimerConfig

- **timer**: Numéro de la minuterie (timer) à utiliser (0, 1 ou 2).
- **prescaler**: Facteur de division de la fréquence pour la minuterie.
- **delay_ms**: Durée du délai en millisecondes pour la minuterie.

Attributs

- **_config**: Stocke la configuration de la minuterie.
- **_timerFreq**: Fréquence de la minuterie après application du prescaler.
- **BASE_FREQ**: Fréquence de base du microcontrôleur, définie à 8 MHz.

Constructeur

- **Timer(TimerConfig config)**: Prend en paramètre une structure **TimerConfig** et initialise la minuterie avec la configuration fournie.

Méthodes

- **enable ():** Active les interruptions de la minuterie selon la configuration.
- **disable():** Désactive les interruptions de la minuterie selon la configuration.
- **reset ():** Réinitialise la valeur du compteur de la minuterie à 0.
- ***(static)* findPrescalerBits(uint16_t prescaler, uint8_t timer)**: Retourne les bits de configuration pour un prescaler donné et un numéro de minuterie spécifique.

Classe Communication

La classe Communication fournit des méthodes pour utiliser la communication (UART) sur le robot. Elle offre des méthodes pour envoyer et recevoir des données en utilisant le port série.

Constructeur

- **Communication ():** Initialise la communication série en configurant les registres nécessaires. Le baud rate est configuré pour une valeur spécifique, en fonction des valeurs définies dans **UBRR0H** et **UBRR0L.**

Méthodes

- **send(uint8_t data):** Envoie un octet via UART après vérification de la disponibilité du buffer.
- **sendString(const char* data):** Envoie une chaîne de caractères octet par octet via UART.
- **receive():** Attend la disponibilité d'un octet, puis le récupère via UART.
- **reinitialize():** Réinitialise les paramètres UART pour la transmission et la réception.

Classe CAN

La classe can offre une interface permettant d'accéder au convertisseur analogique/numérique du microcontrôleur, elle facilite la gestion des conversions A/N.

Constructeur

- **can ()**: initialise le convertisseur analogique/numérique pour qu'il soit prêt à effectuer des conversions en paramétrant les registres appropriés.

Méthodes

- **lecture (uint8_t pos):** Convertit la valeur analogique de l'entrée spécifiée (0-7) en une valeur numérique sur 16 bits (10 bits significatifs).

Classe Wheel

La classe Wheel est conçue pour gérer les roues du robot en utilisant le microcontrôleur AVR. Elle offre la possibilité de contrôler la vitesse des roues en utilisant un timer.

Attributs

- **_output**: un pointeur vers un registre volatile, utilisé pour configurer la valeur de comparaison du timer.
- **_wheelNumber**: Un entier non signé sur 8 bits représentant le numéro de la roue (0 ou 1).
- **MAX_COMPARE_VALUE :** Une constante qui définit la valeur maximale de comparaison pour le timer (0xFF).

Constructeur

- **Wheel (uint8_t wheelNumber)**: Construit un objet de type Wheel avec un numéro de roue spécifié. Le constructeur configure également le timer associé.

Méthodes

- **setCompareValue(uint16_t value)**: Définit la valeur de comparaison pour le timer, ce qui affecte la vitesse de la roue.

Classe Navigation

La classe Navigation est destinée à faciliter le mouvement du robot. Elle permet de contrôler la direction et la vitesse des roues de manière commune ou individuelle.

Attributs

- **_leftWheel:** Un objet de la classe **Wheel** représentant la roue gauche.
- **_rightWheel:** Un objet de la classe **Wheel** représentant la roue droite.

Constructeur

- Navigation(): Constructeur par défaut qui initialise les roues et configure les broches nécessaires.

Méthodes Publiques

- **go(uint16_t speed, bool backward):** Fait avancer ou reculer les deux roues à une vitesse donnée.
- **goLeftWheel(uint16_t speed, bool backward):** Contrôle la roue gauche pour avancer ou reculer à une vitesse donnée.
- **goRightWheel(uint16_t speed, bool backward):** Contrôle la roue droite pour avancer ou reculer à une vitesse donnée.
- **stop():** Arrête les deux roues.
- **stopLeft()**: Arrête la roue gauche.
- **stopRight():** Arrête la roue droite.

Méthodes Privées

- **forward():** Configure les registres pour faire avancer les deux roues.
- **backward():** Configure les registres pour faire reculer les deux roues.
- **leftForward():** Configure les registres pour faire avancer la roue gauche.
- **rightForward():** Configure les registres pour faire avancer la roue droite.
- **leftBackward():** Configure les registres pour faire reculer la roue gauche.
- **rightBackward()**: Configure les registres pour faire reculer la roue droite.
- **_validateSpeed(uint16_t speed):** Valide et ajuste la vitesse donnée pour s'assurer qu'elle est dans une plage acceptable.

Classe Memoire24CXXX

La classe Memoire24CXXX permet la communication avec une mémoire EEPROM 24CXXX via I2C, conservant les données même après extinction du microcontrôleur.

Attributs

- **PAGE_SIZE:** Une constante qui représente la taille d'une page de mémoire, initialisée à 128.
- **m_adresse_peripherique**: Une variable statique qui stocke l'adresse du périphérique I2C de la mémoire EEPROM.
- **_rightWheel:** Un objet de la classe **Wheel** représentant la roue droite.

Constructeur

- **Memoire24CXXX():** Le constructeur initialise l'interface I2C et configure les paramètres nécessaires pour communiquer avec la mémoire EEPROM.

Méthodes

- **init():** configure les registres pour initialiser l'interface I2C.
- **choisir_banc(uint8_t banc):** sélectionne le banc de mémoire.
- **lecture(const uint16_t adresse, uint8_t *donnee):** lit un octet de l'adresse spécifiée, et le stocke à l'adresse pointée par ***donnee**
- **lecture(const uint16_t adresse, uint8_t *donnee, uint8_t longueur):** lit 'longueur' octets depuis l'adresse spécifiée, les stocke à l'adresse pointée par *donnee.
- **ecriture(const uint16_t adresse, const uint8_t donnee):** écrit 'donnee' à l'adresse spécifiée dans la mémoire EEPROM.

Partie 2: Décrire les modifications apportées au Makefile de départ

Ce nouveau *Makefile* permet l'archivage de tous les fichiers sources de la librairie en un fichier "libstatique.a".

Modifications

- PROJECTNAME=libstatique – *Nom du projet*
- PRJSRC= LED.cpp Timer.cpp Bouton.cpp Wheel.cpp Navigation.cpp memoire_24.cpp Communication.cpp can.cpp Debug.cpp – *Fichier source à inclure dans la librairie*
- TRG=$(PROJECTNAME).a – *Fichier cible de l'archivage*
- Règle de création de la bibliothèque – *Règle de création de la bibliothèque statique*

```
$(TRG): $(OBJDEPS)
        $(AR) rcs $(TRG) $(OBJDEPS)
```

Additions

- AR=avr-ar – *Pour utiliser l'utilitaire de création d'archive, génère la bibliothèque statique*

Makefile commun

Ce Makefile sert à exécuter les deux makefiles l'un après l'autre en une seule commande. (compilation puis installation). Il permet l'exécution de *clean, debug, install* et*, all*, pour les deux dossiers à partir d'une seule commande.