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

#1618

Noms:

Thierry Champion
Nikolai Olekhnovitch
Raissa Oumarou-Petitot
Rym Touati

Date:
31 octobre 2023

Partie 1 : Description de la librairie

La librairie contient plusieurs classes qui donnent accès aux nombres de fonctionnalités d'un robot avec un microprocesseur ATmega324pa. De plus, nous avons implémenté quelques utilitaires pour nous aider à faciliter nos manipulations. Finalement, plusieurs types énumérés nous facilitent la lecture et sont listés à la fin.

Classes principales

Classe **Button**

La classe Button simplifie la gestion des boutons connectés à un microcontrôleur ATmega324pa. Elle offre des fonctionnalités pour configurer, activer, et désactiver les interruptions, ainsi que pour détecter l'état du bouton et gérer les événements liés à celui-ci.

Attributs

- ReadPin _pin: Broche à laquelle le bouton est connecté.
- GeneralInterrupt _interrupt: **GeneralInterrupt** associée au bouton.
- bool _isActiveHigh: Indique si le bouton est actif à l'état haut.

Méthodes

- Button(GeneralInterrupt interrupt, bool activeHigh): Construit un bouton. Permet de contrôler quel **GeneralInterrupt** est utilisé pour le bouton et s'il est actif haut ou bas.
- bool isButtonPressed() const: Vérifie si le bouton est actuellement appuyé ou relâché en appliquant l'antirebond. Antirebond réglé à 30 ms.
- void setSenseControl(SenseControl control): Configure le mode de déclenchement de l'interruption associé au bouton (SenseControl).
- void enable(),void disable(): Active ou désactive l'interruption associée au bouton.
- void clearButtonEvents(): Efface les événements du bouton, donc réinitialise l'état de la détection.

Classe **Comm**

La classe Comm permet la transmission de données via UART. Une chaîne de caractère est transmise, caractère par caractère, grâce à la méthode transmitData(). Cette diffusion est affichée du robot à la console grâce à serieViaUSB, un programme qui nous est fourni. Le réglage des registres nous a également été fourni lors du TP5 et il se fait lors de la construction d'un objet de la classe.

Méthodes

- Comm(): Construit un objet de communication.
- static void transmitData(const uint8_t *data, uint8_t length): Transmet des données par RS-232.

Classe **GeneralInterrupt**

La classe GeneralInterrupt est conçue pour gérer les interruptions sur un microcontrôleur ATmega324pa. Elle permet de configurer le type d'interruption, le niveau de déclenchement et de contrôler l'activation et la désactivation des interruptions.

Attributs

- GeneralInterruptType _type : Type d'interruption GeneralInterruptType.
- Flag _controlFlag1, Flag _controlFlag2, Flag _maskFlag, Flag _interruptFlag: Contiennent les drapeaux de contrôle et de masquage spécifique à chaque interruption.
- SenseControl _control : Stocke le mode de déclenchement de l'interruption via SenseControl.

Méthodes

- GeneralInterrupt(const GeneralInterruptType type): Construit et initialise les attributs associés au type d'interruption voulu (GeneralInterruptType).
- void enable(), void disable(), void clear(), void setFlags(): Active, désactive, efface le drapeau d'interruption correspondant ou initialise les drapeaux de contrôle et de masquage en fonction du type d'interruption.
- void setSenseControl(SenseControl control), void applySenseControl(): Définit et applique le mode de déclenchement spécifié aux broches d'interruption (SenseControl).
- ReadPin getRequiredPin() const: Renvoie la broche associée à l'interruption.
- void debugRequiredPin() const: Affiche des infos de débogage sur la broche associée à chaque type d'interruption.

Classe **Led**

La classe Led permet l'utilisation de la lumière DEL libre à usage général intégrée à notre carte mère. Elle permet de facilement afficher une couleur verte ou rouge grâce à une seule méthode publique qui prend la couleur en paramètre (elle sera éteinte sans paramètre). Cette dernière se sert de 3 méthodes privées pour fonctionner.

Attributs

- WritePin _greenPin: Associe une broche **WritePin** à la couleur verte (borne négative).
- WritePin _redPin: Associe une broche **WritePin** à la couleur rouge (borne positive).

Méthodes

- Led(Port port, uint8_t greenPosition, uint8_t redPosition): Construit une DEL et associe deux **WritePin** au port et aux positions voulues.
- void setColor(LedColor color): Ouvre la DEL de la couleur voulue via le type énuméré LedColor.
- void setOff(): méthode privée qui ferme la DEL.
- void setGreen(): méthode privée qui allume la DEL en vert.

- void setRed(): méthode privée qui allume la DEL en rouge.

Classe **Navigation**

La Classe Navigation est la classe qui s'occupe de la mobilité du robot en utilisant des objets Wheel (voir **Wheel**). La classe utilise les broches B3 et B5 pour la roue de gauche et B2 et B4 pour la roue de droite.

Attributs

- Wheel _leftWheel: Associe la roue de gauche.
- Wheel _rightWheel: Associe la roue de droite.
- Timer0 _timerPWM: Associe un **Timer0** qui génère le PWM.

Méthodes

- Navigation(WritePin dirLeftPin, WritePin dirRightPin): Construit la navigation et associe les deux broches aux bonnes directions via des objets **WritePin**.
- ~Navigation(): Arrête le timer.
- void controlledTurn(Side turn, Orientation orientation, float speed, uint8_t turnStrength): Effectue un virage contrôlé selon le côté (Side) et l'orientation (Orientation) spécifié. L'angle de virage et la vitesse sont également ajustables.
- void move(Orientation orientation, float speed): Déplace vers l'avant ou l'arrière (via Orientation), à la vitesse voulue.
- void stop(): Arrête le robot.

Classe **Timer0, Timer1 et Timer2**

Les classes suivantes sont utilisées pour gérer les trois minuteries du ATmega324pa. Puisque ces trois composantes ont toutes des particularités distinctes et que le polymorphisme n'est pas avantageux dans le contexte de ressources limitées, trois classes séparées ont été développées. De plus, pour faciliter la compréhension, 5 types énumérés ont été créés: TimerCompare, TimerCompareMode, TimerInterrupt, TimerPrescalar et TimerWaveMode.

Attributs

- bool _isTicking
- TimerWaveMode _waveMode,
- TimerInterrupt _interrupt,
- TimerPrescalar _prescalar: respectivement le mode, les interruptions actives et le préscalaire choisi.

Méthodes

- Timer(): Construit un Timer initialement arrêté.
- void start(): Démarrer le timer. Initialise les registres pour les interruptions et le préscalaire.
- void stop(): Arrêter le timer. Dégage les registres pour les interruptions et le préscalaire.
- bool isRunning() const: Retourne si le timer est démarré.

- void setCounterValue(uint8_t/uint16_t value),
- void setCompareValue(TimerCompare compare, uint8_t/uint16_t value): Modifient respectivement le compteur du timer et la valeur de comparaison voulue. La taille du paramètre est différente pour Timer1.

- void setWaveMode(TimerWaveMode mode),
- void setCompareMode(TimerCompare compare, TimerCompareMode mode),
- void setInterrupt(TimerInterrupt interrupt),
- void setPrescalar(TimerPrescalar prescalar): Ces méthodes modifient respectivement le mode, la méthode comparaison, les interruptions et le préscalaire. Dans le cas des interruptions et du présidescalaire, les registres sont uniquement modifiés lorsque le timer est actif.

- void applyInterrupt,
- void applyPrescalar: Deux méthodes privées simplifient l'implémentation de setInterrupt et setPrescalar.

Classe **Wheel**

La classe wheel est la classe qui gère les mouvements d'une roue via le **timer0**. On peut ajuster sa vitesse, son sens et la broche D. Comme c'est le timer0 qui est utilisé, la broche E(Enable) peut être soit PB3 ou PB4.

Attributs

- Side _side: Sélection de la Side de la roue. Pour la droite, on sera sur PB3. Pour la gauche, sur PB4.
- Timer0 *_timerPWM: Pointeur vers le **Timer0** en phase PWM de Navigation.
- WritePin _directionPin: Objet **WritePin** qui gère la broche de direction de la roue.

Méthodes

- Wheel(WritePin directionPin, Side side, Timer0 *timerPwm): Construit avec un objet **WritePin**, une Side et un pointeur vers le **Timer0** utilisé.
- setSpeed(Direction direction, float speed): Modifie la vitesse et Direction de la roue.

Classes utilitaires

Classe **Debug**

Outre les classes utilisées directement par le robot, nous avons ajouté une classe utilitaire Debug pour avoir accès à une impression sur la console via la classe **Comm**. Elle définit un macro d'impression (PRINT) lorsqu'on compile en Debug. Ce dernier crée un objet Comm pour transmettre les données à être affichées.

Méthodes

- void display(uint8_t x); affiche une donnée de type uint8_t
- void display(uint16_t x); affiche une donnée de type uint16_t
- void display(float x); affiche une donnée de type float
- void display(const char *x); affiche une chaîne de caractère

Classe **Memoire24CXXX**

Classe fournit permettant d'effectuer le protocole I2C pour l'écriture sur la EEPROM. Le protocole et la classe sont assez complexes, donc elles ont été gardées telles quelles. Memoire24CXXX permet l'utilisation d'une mémoire supplémentaire qui pourrait être utile s'il faut garder des données quelconques en mémoire.

Classe **Can**

Can est une classe fournie et nous permet d'utiliser le contrôle analogique/numérique intégré à la carte mère. Cette classe est incluse dans la librairie au cas où nous aurions à utiliser ce périphérique dans le cadre du projet final.

Classe **ReadPin et WritePin**

Ces deux classes sont un utilitaire pour nous aider à mieux construire les classes qui utilisent les différents ports, régler leurs modes (écriture/lecture) et cibler les broches voulues. Permet notamment une mauvaise utilisation des registres DDRx, PORTx et PINx.

Attributs

- Register _mode: Abstraction du registre DDRx voulu.
- Register _pin/_port: Abstraction des registres PINx (**ReadPin**) ou PORTx (**WritePin**).
- uint8_t _position: Position de la broche voulue.

Méthodes

- Constructeur(Port port, uint8_t position): Construit soit une **ReadPin** ou une **WritePin** sur Port à la position fournie.
- uint8_t ReadPin::read(): Retourne la lecture de la broche.
- void WritePin::set(): Met la valeur de la broche à 1.
- void WritePin::clear(): Met la valeur de la broche à 0.

Types Énumérés

- GeneralInterruptType: INT_0, INT_1, INT_2
- LedColor: OFF, GREEN, RED
- Orientation: FORWARD, BACKWARD
- Port: A, B, C, D
- SenseControl: LOW_LEVEL, FALLING_EDGE, RISING_EDGE, ANY_EDGE
- Side: RIGHT, LEFT
- TimerCompare: A, B
- TimerCompareMode: DISCONNECTED, TOGGLE, CLEAR
- TimerInterrupt: NONE, COMPARE_A, COMPARE_B, BOTH
- TimerPrescalar: STOPPED, NO_PRESCALAR, EIGHT, THIRTY_TWO, SIXTY_FOUR, ONE_TWENTY_EIGHT, TWO_FIFTY_SIX, THOUSAND_TWENTY_FOUR
- TimerWaveMode: NORMAL, CTC, PWM_PHASE_CORRECT

Partie 2 : Décrire les modifications apportées au Makefile de départ

Modifications du Makefile pour la librairie

- Définition de l'archiveur AR=avr-ar
- Définition globale du macro F_CPU avec le flag gcc -DF_CPU=8000000UL
- Définition du flag pour l'archiveur ARFLAGS=-crs
- Changement du target TRG=$(PROJECTNAME).a pour créer la librairie en sortie.
- Modification du .PHONY pour $(TRG) avec $(AR) $(ARFLAGS) $(TRG) $(OBJDEPS). Le makefile génèrera donc la librairie statique.
- Suppression de tout ce qui est relié au HEXROMTRG, HEXTRG et au install, ce qui est inutile pour la librairie.

Modifications du Makefile pour le code de test

- Inclusion des fichiers .h de la librairie avec INC=-I ../lib
- Liaison de la librairie avec LIBS=-l robot
- Définition du chemin vers la librairie LIBDIR=../lib
- Ajout des flags pour compiler avec la librairie -lm $(LIBS) -L $(LIBDIR)

Modifications communes entre les deux Makefiles

- Généralisation des fichiers sources $(wildcard *.cpp). Il n'est plus nécessaire de rajouter les fichiers sources à la main.
- Ajout du .PHONY pour debug qui ajoute -DDEBUG -g aux CXXFLAGS et CCFLAGS