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 1013

Noms:

Yacoubi, Ali
Falouh, Ghadi
Foroudian, Kimia
Desroches, Kim

Date:
31 octobre 2023

Partie 1 : Description de la librairie

La librairie a été formée à l'aide de multiples fichiers comportant différentes classes, soit la classe Can, LED, Memoire24CXXX, Navigation, RS232, Timer et Button. Un fichier Delay et un fichier Debug ont également été ajoutés à la librairie afin de pouvoir améliorer la structure du code et son débogage. Les différentes classes et fonctions nommées ci-haut sont expliquées davantage en profondeurs dans les sous-points si dessous.

1.1. Classe Can

La classe CAN, préalablement fournie, est un convertisseur analogique numérique qui permet à l'Atmega324PA de convertir un signal analogique en données numériques. Effectivement, la valeur numérique fournie en sortie est proportionnelle à celle de la tension d'entrée.

Constructeur : Le constructeur de CAN active/initialise le convertisseur sans démarrer la conversion. Une seule conversion s'effectuera en 832 cycles d'horloge.

Méthode : La méthode *lecture* effectue la conversion voulue et retourne la valeur numérique correspondant à la valeur analogique sur le port A. Le résultat sera affiché sur 16 bits, mais uniquement les 10 bits de poids faible sont significatifs.

1.2. Fichier Debug

Le fichier Debug permet de définir les macros associées à « #define DEBUG ». Lorsque la commande « make debug » est exécutée, DEBUG est défini, ce qui permet de satisfaire la condition « ifdef DEBUG ». Ainsi, la fonction *debugPrint* est définie via DEBUG_TRANSMIT et permet la transmission de données via RS-232.

1.3. Fichier Delay

Le fichier Delay est implémenté pour permettre un délai en ms dans le reste du code. Cette fonction permet de rendre la variable de délais constante. Ainsi, cette fonction peut être appelée dans les différentes méthodes d'une classe telles que la classe Led.

1.4. Classe Led

La classe Led est responsable de contrôler la del présente sur la carte mère du robot. Elle permet d'allumer la couleur selon l'énumération *ledColor*.

Constructeur : Le type « Register » représente le type « volatile uint8_t* ». Il permet d'associer la del avec les broches désirées sur la carte mère. Par exemple, Led(&PORTA, &DDRA, PA0, PA1) permet d'associer la led avec le port A et sur les broches PA0 et PA1.

Méthodes publiques :
controlColor permet d'allumer la del de la couleur passée en paramètre.
pwm s'occupe d'éteindre graduellement la del.
blinkingLed permettra à la del de clignoter pour une durée prédéterminée.
La méthode *off* éteint la led.

Méthodes privées :
La méthode *amber* permet d'afficher la couleur ambre.

1.5. Classe Memoire24CXXX

La classe Memoire24CXXX permet la lecture et l'écriture en mémoire grâce au protocole I^2C. Cette classe utilise les broches C0 et C1 sur un microcontroleur Atmel ATMega244pa. Le constructeur n'a pas besoin d'aucun argument et appelle la procédure *init*. L'écriture en mémoire peut se faire de deux façons. En précisant la valeur d'un octet et son adresse ou d'écrire un bloc d'octets (n'excédant pas 127). Les méthodes de lecture ont également deux variantes, soit la lecture d'une donnée à la fois ou d'un bloc de données.

Cette classe est intégrée à la librairie afin de permettre l'écriture et la lecture en mémoire.

1.6. Classe Navigation

La classe Navigation contient des méthodes pour effectuer le contrôle des moteurs (des roues) du robot d'où son nom. Le timer0 est utilisé pour générer les signaux PWM. Le mode choisi est PWM phase correct avec une valeur top de 0xFF et un prescaler de 8. Le mode de comparaison est « Clear OC0A and OC0B on compare match ».

Définitions : « RIGHT_MOTOR_DIRECTION » et « LEFT_MOTOR_DIRECTION » permettent respectivement de mettre à « 1 » PB5 et PB2. Ces broches sont associées à la direction des roues (avant/arrière). Les définitions « RIGHT_MOTOR_PIN » et « LEFT_MOTOR_PIN » permettent d'associer « un » à PB3 et PB4 pour la sortie des signaux PWM.

Constructeur : Les broches associées à la direction des roues et aux signaux PWM sont mis en sortie (DDRB0.

Méthodes publiques : La méthode *setSpeed* permet l'ajustement des signaux PWM pour les deux moteurs par l'entrée de paramètres représentant le ratio entre A et B. Donc, il faut donner des pourcentages en paramètres (i.e. 75 pour 75% de la vitesse). Elle est privée, car elle est appelée directement dans les autres méthodes de la classe.

Les méthodes *foward* et *reverse* permettent respectivement de changer la direction des roues. Elles prennent en paramètre le pourcentage de la vitesse. La méthode *stop* permet d'arrêter la rotation des roues.

Les méthodes *turnRight* et *turnLeft* permettent de tourner à droite et à gauche en mettant des directions opposées aux deux roues. Le paramètre « duration » permet de déterminer à quel point le robot tournera vers la direction désirée.

1.7. Classe RS232

Cette classe gère le protocole de communication RS232 pour le port UART0. Cela permet de transférer des données entre le microcontrôleur et l'ordinateur. Ce constructeur configure la communication à une vitesse de 2400 bauds et définit le format des trames à 8 bits de

données, 1 bit d'arrêt et sans bit de parité.

Constructeur : Comme cette classe ne contient pas d'attribut, le constructeur choisi est vide, ce qui permet de ne pas nécessiter la création d'une instance de RS232. Les méthodes sont « static », car elle n'agit pas sur des variables d'instance mais uniquement sur la classe.

Méthodes : La méthode *initialise* permet de configurer les registres pour permettre la communication RS232. *readData* devise les données qu'on veut transmettre en octets et les envois vers la méthode privée *transmitUART* responsable d'envoyer les octets reçus vers le PC.

1.8. Classe Timer

Nous avons décidé de créer une classe *Timer* qui permet de configurer les registres d'une minuterie à l'aide du Timer 1 en mode CTC (Clear Timer on Compare Match). Elle permettra de générer une interruption après une durée donnée.

Constructeur : Son rôle est de prendre la durée (en secondes) de la minuterie avant qu'une interruption soit déclenchée, le prescaler (division d'horloge) avec lequel on voudrait travailler et la fréquence du CPU (nécessaire à la conversion de la durée en cycle d'horloge)

Méthodes: La méthode *StartTimer* permet de configurer les registres de la minuterie afin de pouvoir déclencher une interruption après la durée initialisée dans le constructeur. Le prescaler et la fréquence initialisée au préalable dans le constructeur permettent de faire la conversion de la durée en secondes en cycles d'horloge. La méthode prend en paramètre la variable volatile booléenne *isTimerExpired*.

1.9. Classe Button

Cette classe est responsable de générer des interruptions. Elle est faite pour travailler avec un bouton connecté au port D2.

L'enum « interruptionType » permet de choisir la sensibilité des interruptions générée par le bouton. Ce choix doit être l'un des quatre types: low level, any edge, falling edge et rising edge.

Constructeur: Il prend le rôle d'une routine d'initialisation qui ajuste les paramètres des registres responsables des interruptions. Cela permet à notre microcontrôleur de générer des interruptions en fonction des besoins du bouton.

Partie 2 : Décrire les modifications apportées au Makefile de départ

2.1. Makefile de la librairie

Les modifications apportées au Makefile de la librairie sont les suivantes :

- **PROJECTNAME = libstatic**
  Permet de nommer la librairie.

- **PRJSRC = $(wildcard *.cpp)**
  Permet de récupérer tous les fichiers .cpp du répertoire.

- **AR = avr-ar**
  Permet d'ajouter une librairie statique.

- **TRG=$(PROJECTNAME).a**
  Permet de spécifier l'extension du fichier libstatic.

- **$(TRG): $(OBJDEPS)**
  $(AR) -crs $(TRG) $(OBJDEPS)
  Permet de faire l'implémentation de la cible AR avec les options -crs (créer l'archive, insérer les fichiers membres et produire l'index des fichiers objets).

2.2. Makefile de l'exécutable

Les modifications apportées au Makefile de l'exécutable sont les suivantes :

- **PROJECTNAME=test**
  Permet de nommer le projet.

- **PRJSRC=project.cpp**
  Permet de récupérer le fichier project.cpp du répertoire.

- **INC= -I../lib**
  Permet de spécifier le chemin des inclusions additionnels.

- **LIBS= -L../lib -lstatic**
  Permet d'indiquer le chemin vers la librairie à lier.

- **LIB_DIRECTORY = ../lib**
  Permet d'indiquer le chemin vers le répertoire de la librairie.

- **clean_all:**
  $(REMOVE) $(TRG) $(TRG).map $(OBJDEPS) $(HEXTRG) *.d *elf.map
  $(MAKE) -C $(LIB_DIRECTORY) clean
  Permet d'effacer tous les fichiers générés lors de la compilation de la librairie et de l'exécutable.

- **lib:**
  $(MAKE) -C $(LIB_DIRECTORY)

Permet de compiler la librairie à partir du répertoire de l'exécutable

- **debug: CXXFLAGS += -DDEBUG -g**
  debug: CCFLAGS += -DDEBUG -g
  debug: $(TRG) $(HEXROMTRG)

Permet de définir une macro DEBUG dans le fichier project.cpp lorsqu'on y ajoute « #define DEBUG » (option -DDEBUG) et de compiler les fichiers.