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 5565

Noms:

Anh Pham
Younes Lazzali
Oscard Arcand
Ben Jemaa Manel

Date:
13 mars 2022

Table des matières

Partie 1 : Description de la librairie

Pour notre librairie nous avons choisis d'inclure les notions suivantes : Les Boutons poussoirs, contrôle DEL, l'utilisation du PWM avec minuterie, la liaison série RS232, le convertisseur analogique numérique (CAN), les timers et enfin la manipulation des mémoires externe en écriture et en lecture.
Nous avons choisi ses notions car ce sont des fonctionnalités générales qu'on pourra les utiliser plus tard dans notre projet final dans cette partie nous allons discuter les choix des notions choisis et décrire les librairies des différentes parties.

1. PWM avec minuterie et moteur

Pour faire tourner les roues d'un moteur (avant, arrière) nous avons besoin du PWM avec timer pour alimenter le pont H.
Le choix du timer repose sur le fait qu'on laisse notre microcontrôleur s'occuper d'autre chose et on utilise les ressources interne.
Pour ce faire nous avons créé une classe dans le fichier entête qui contient 4 méthodes pour la mobilité (avancer, reculer, droite et gauche) du robot, une méthode pour l'arrêt du robot et enfin une méthode permettant l'initialisation des différents registres du notre microcontrôleur afin d'utiliser PWM.

2. La liaison série RS232

L'utilisation du RS232 nous permet d'envoyer des données sur un port série qu'on peut servir par la suite dans la partie du débogage.
RS232 est une liaison série qui nous permet d'envoyer caractère par caractère. Pour se faire, nous avons besoin de 3 modules: initialisation des registres ensuite un module pour transmettre les données du coté RS232 et un module pour recevoir les données du côté PC

3. Mémoire_24

Cette classe nous permet d'utiliser la mémoire externe sur le robot. La classe nous permet de lire et écrire sur la mémoire externe au lieu de bruler la mémoire interne de l'atmega324PA.
Cette partie nous a été servie par le professeur.

4. Le convertisseur analogique numérique (CAN)

Le convertisseur analogique numérique sert à convertir un signal analogique (donc qui est continu) en une donnée numérique (donnée discrète). La donnée est stockée originalement sur 10 bits, les deux bits les moins significatifs sont éliminés, donc la sortie est sur 8 bits.
Cette partie aussi a été donnée par le professeur.

5. Bouton

La classe bouton sert tout simplement à contrôler les interactions avec un bouton, de plus nous avons créé une fonction dans le fichier «.cpp» qui permettraient d'utiliser n'importe quel port du microcontrôleur, nous avons aussi inclus la notion de l'anti-rebond afin de mieux savoir à quel état se trouve le bouton-poussoir.
Les états du bouton sont: pressé, maintenu, relâché ou inactif, ont été définies à l'aide de «Switch/case» et de boucle «If/else».

6. Interruption

La classe interruption a une seule fonctionnalité, initialiser les interruptions. Nous avons choisi d'initialiser les 3 timers qu'il existe dans notre microcontrôleur, (INT_0, INT_1 et INT_2) indépendamment les unes des autres.

7. LED

Allumer la LED sur le robot est essentiel depuis le début du projet. On veut implémenter une fonctionnalité sur notre atmega qu'on ne peut pas voir concrètement on peut utiliser LED, par exemple si le microcontrôleur transmet correctement les données via la liaison RS232, la LED va

s'allumer vert, sinon elle va s'allumer rouge, donc il sera intéressant qu'on ait une classe LED.
Pour se faire nous avons créé une classe dans laquelle on a implémenté les méthodes des différentes couleurs, (rouge, vert, éteint, ambre) de plus nous avons ajouté une méthode qui nous permet de choisir quel port et quels pins on va utiliser sur notre atmega.

8. Timer

La classe timer nous permet de contrôler à peu près tout quand vient au timer utilisé. On peut choisir le mode du timer, son prescaler, sa valeur de départ, sa valeur finale, s'il utilise des interruptions, puis la même chose pour les autres timers.

9. Debug

On a choisi d'intégrer le module de débogage, car c'est très utile, que ce soit dans ce projet ou dans les projets qui viennent. Si on a un problème dans notre code, ça sera plus facile d'utiliser debug pour voir où se trouve l'erreur. Pour se faire, nous avons affiché les 3 types que nous avons besoin (uint8_t, const char et char), puis nous avons pris le module du RS232 qu'on a déjà fait, afin de faire l'affichage sur open serial monitor de SimulIDE.

Toutes les méthodes qu'on a implémentées se trouve dans le schéma ci-dessous

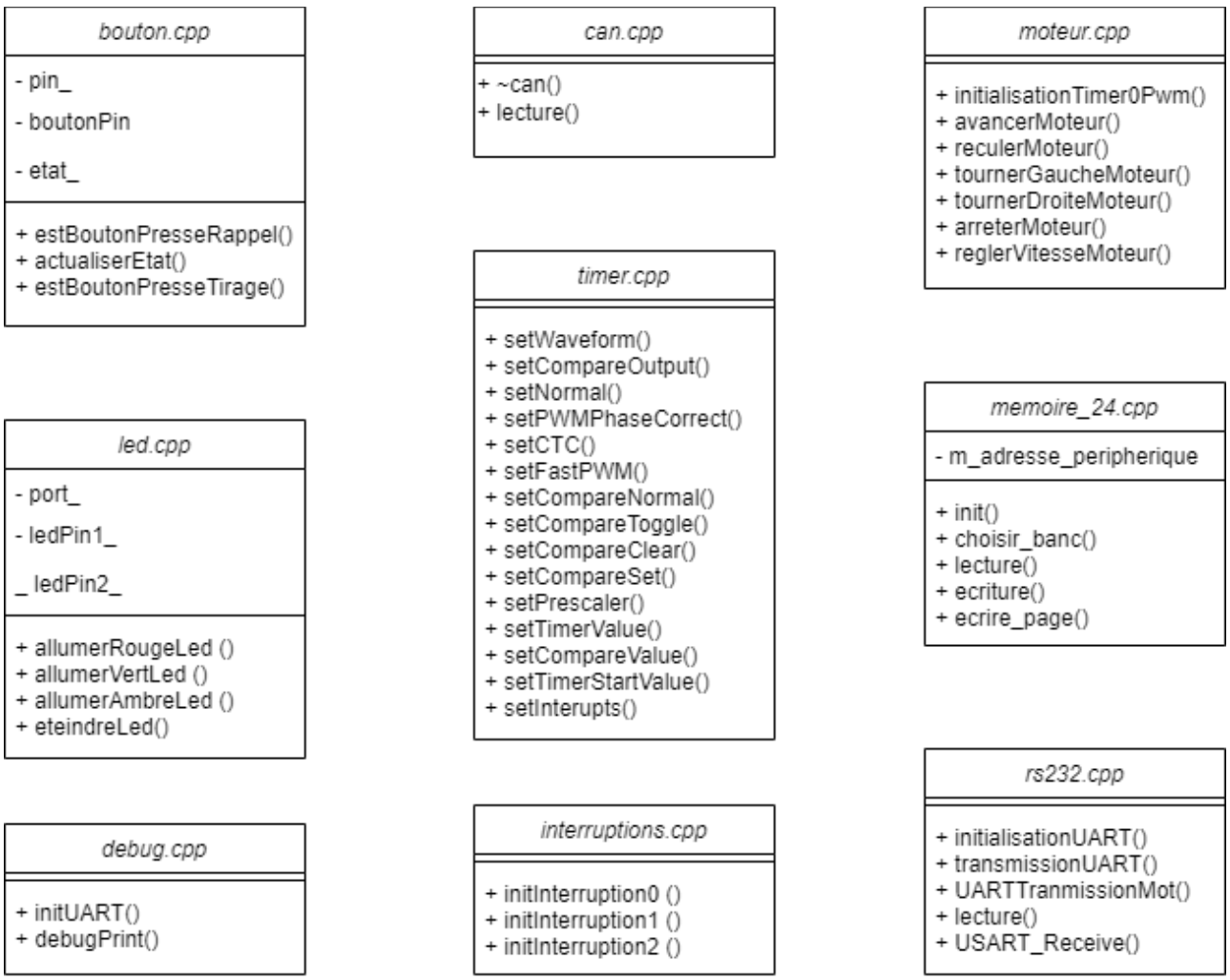

Figure 1: les méthodes dans la librairie

Dans le TP7, nous avons favorisé l'utilisation des classes, car l'approche orientée objet favorise l'encapsulation de notre code sous forme de classes. Le code est plus lisible et plus compréhensible, enfin, si on a besoin de rajouter du code qui dérive de nos classes déjà créées, il suffit d'utiliser de l'héritage ou du polymorphisme pour résoudre le problème. Donc, si on veut générer une librairie facilement extensible et lisible (bien encapsulé), il vaut mieux opter pour l'orienté objet.
De plus, certains algorithmes demandent d'utiliser des valeurs globales, par exemple pour initier le PWM ou l'accès à la mémoire. Comme les variables globales sont à éviter dans une librairie, nous avons opté pour la création de classes afin de pouvoir y stocker les valeurs initialisées une seule fois, à la création de l'objet

TP8 Partie 2 : Décrire les modifications apportées au Makefile de départ

Notre projet a été séparé en deux. Il y a un sous-répertoire « exec » qui contient un Makefile qui créé des exécutables et le code qui sera compilé, puis envoyé au robot. Il y a aussi un sous-répertoire « lib » qui contient un Makefile qui créé les librairies ainsi que les codes sources et les fichiers « include » qui seront compilés, puis combinés pour former la librairie. Pour avoir un Makefile qui puisse former des librairies, nous avons dû modifier celui fourni. Les premières modifications sont la création de nouvelles variables.
Nouvelles variables dans le Makefile de lib :

- AR = ar : Variable pour le compilateur « ar »
- exec_dir = ../exec : Chemin du dossier « lib » vers le dossier « exec »

La variable « AR » nous permet de ne pas « hard-code » le nom du compilateur « ar » dans le Makefile. Ensuite, la variable « exec_dir » permet au Makefile d'aller chercher des fichiers dans le dossier « exec ». La prochaine modification porte sur la variable « TRG ».

- De « TRG=$(PROJECTNAME).elf » à « TRG=lib$(PROJECTNAME).a »

Ce changement fait que ce qui sera compilé par le Makefile sera nommé « lib(nom du projet).a ». Nous avons fait ainsi, car une librairie doit toujours commencer par le préfix « lib ». De plus, puisqu'il s'agit d'une librairie statique, il doit avoir l'extension « .a ». Cette modification a pour effet de rendre inutile toutes les commandes qui manipulent les fichiers avec l'extension « .elf » puisque nous n'utilisons plus de fichiers « .elf » dans ce MakeFile. Ces commandes ont donc été mise en commentaires. La prochaine modification est l`ajout d'une règle.

- vars: …

L'ajout de cette règle a pour but de faciliter le débogage du Makefile puisque nous pouvons afficher les unités de compilation, les fichiers objets et la librairie sur le terminal. Ensuite, nous avons modifier la variable PRJSRC.

- PRJSRC= $(wildcard *.cpp)

Ce changement fait qu'on compile tous les fichiers .cpp dans le même répertoire que le MakeFile. Finalement, la dernière modification porte sur le transfert vers le microcontrôleur.

Puisque nous générons une librairie et qu'une librairie ne devrait pas être transmise à un microcontrôleur, nous avons mis en commentaire les commandes qui impliquent avrdude ainsi que la règle install.

Pour ce qui est du Makefile dans le répertoire exec, nous avons juste fait quelques modifications.

- lib_dir = ../lib
- LIBS = -ltest

- INC = -I $(lib_dir)
- $(TRG): $(OBJDEPS)
- $(CC) $(LDFLAGS) -o $(TRG) $(OBJDEPS) \
- -L $(lib_dir) -lm $(LIBS)

Nous avons inséré « lib_dir » dans la variable « INC ». Ces modifications permettent au Makefile de chercher des fichiers qu'il aura besoin dans le répertoire exec. Ensuite, nous avons inséré la librairie construite par le premier Makefile « -ltest » dans la variable « LIBS ». Puis, nous avons ajouté la ligne « -L $(lib_dir) dans l'éditeur des liens pour qu'il puisse utiliser la librairie lorsqu'il créé l'exécutable. En plus de ces changements, nous avons introduit une nouvelle règle.

- debug : CFLAGS += -DDEBUG
- debug: $(TRG) $(HEXROMTRG)
-

Cette nouvelle règle va définir le macro DEBUG ainsi que compiler le projet. Ceci a pour effet de de définir DEBUG_PRINT comme étant une fonction d'affichage et non du code mort.

Conclusion :

Dans ce rapport nous avons d'abord expliqué les différentes notions que nous avons choisies, ainsi que leur utilité, que ce soit dans ce tp ou dans notre projet final. Ensuite, nous avons ajouté une explication sur les méthodes implémentées, et pour finir les modifications que nous avons faites dans le makefile, afin qu'on puisse compiler et déboguer notre travail.