Polytechnique Montréal

Département de génie informatique et génie logiciel

Cours INF1995:
Projet initial en génie informatique et travail en équipe

Travail pratique 8

Makefile et production de librairie statique

Par l'équipe

No 0315

Par :
Maxime Brousseau
Christophe Kedzierski-Tanguay
Tom Avedissian
Aladin Riabi

Date:
1er novembre 2016

Description de la librairie :

Dans notre librairie, nous avons inclus différentes classes concernant plusieurs aspects principaux de la programmation de notre robot. Parmi ces derniers, on retrouve une classe sur le contrôle de la LED, une sur la conversion analogique numérique, une sur la gestion de la mémoire du robot, une sur le type de port à utiliser et finalement une sur le PWM.

La classe de la LED est très pratique à inclure dans la librairie, car elle contient des fonctions que nous utilisons très souvent pour une panoplie de tests concernant autant les LED que les moteurs. Dans cette classe, on retrouve plusieurs fonctions, comme *allumerR(), allumerV(), allumerA()* et *eteindre()*. Ces fonctions allument respectivement la LED avec la couleur rouge, verte, ambrée et aucune couleur (éteinte). On retrouve également une fonction *modifierPort()*, qui permet comme son nom le dit, de modifier le port sur lequel on fera les branchements, afin d'allumer la LED. Le port en question est déterminé dans le fichier main.cpp.

Dans la classe de conversion analogique numérique can, on retrouve une seule fonction *lecture()*, qui permet de convertir un signal analogique lu en un signal numérique et de le retourner. Cette classe peut nous être dans utile dans une situation où nous devons capter un signal qui vient de l'environnement extérieur (un signal lumineux par exemple) afin de faire réagir le robot en fonction de ce même signal.

Nous incluons également une classe mémoire à l'intérieur de notre librairie. Cette classe est très pratique lorsque nous avons à utiliser la mémoire externe sur la carte-mère. On y retrouve les fonctions suivantes : *choisir_banc(), lecture(), ecriture()* et *ecrire_page()*. Toutes ces fonctions permettent au microcontrôleur de communiquer avec la mémoire externe, à l'aide du protocole I^2C. On retrouve trois fonctions d'écriture et deux de lecture qui prennent différents paramètres, permettant ainsi de varier les façons dont le microcontrôleur communique avec la mémoire.

Dans la classe de gestion du port à utiliser, on retrouve tout simplement un enum TypePort qui comprend quatre variables A, B, C et D, qui correspondent évidemment aux différents ports utilisables sur la carte-mère.

Finalement, dans la classe du PWM, nous utilisons deux fonctions qui permettent de générer un signal à une fréquence de 60Hz ou 400Hz. Ces deux fonctions seront très utiles lorsque nous aurons besoin de faire fonctionner les moteurs à un certain pourcentage de leur capacité totale.

Description du Makefile :

Nous utilisons deux Makefile au total dans notre dossier de projet.

Commençons par les modifications apportées au Makefile chargé de compiler et d'envoyer à notre robot les différents codes que nous lui fournissons. Pour ce Makefile les quelques modifications apportées ont été ajoutées afin de permettre l'utilisation d'une bibliothèque pour nos futurs codes, permettant d'éviter le plus possible la répétition de code dans nos différents fichiers.

Tout d'abord nous avons attribué à la variable LIBS les options -lrobot et -Llibs/include. L'option -l précise le nom de notre librairie (robot dans ce cas-ci) et –L le chemin à parcourir pour la retrouver.

Notons que bien que notre librairie se nomme librobot, nous précisons uniquement le mot clé robot. En effet, le mot clé lib est considéré comme défini de base.

Nous avons également attribué à la variable INC le chemin vers les fichiers sources de notre librairie. Cela nous a permis d'éviter de préciser le chemin des fichiers .h lors des include dans notre code utilisant notre librairie. Cette variable est d'ailleurs utilisée dans la variable contenant les flags nécessaires à la compilation des fichiers .c (CFLAGS) via l'option -Ilibs/include (-I précise la location de nos fichiers d'en-têtes de notre librairie).

Nous avons également réalisé un MakeFile nous permettant de générer notre librairie. Originellement nous avions décidé de modifier le Makefile fourni afin de la générer, cependant cette méthode provoquait certains warning (notamment avec le linker) et rendait le fichier inutilement complexe à lire.

Ainsi, nous avons décidé d'enlever la majorité de l'ancien Makefile, pour nous assurer qu'il ne génère uniquement les fichiers nécessaires à la formation de notre librairie (plus de génération de .elf et de .hex). Tout d'abord, nous avons ajouté une variable CL, contenant la commande avr--ar (appel permettant de générer une libraire statique). Nous avons

également attribué à la variable PRJSRC, *.cpp afin que tous les fichiers .cpp du dossier courant soit ajouté à la libraire.

Nous avons changé la règle all également. Désormais, les commandes de cette règle permettent de compiler tous les fichiers .cpp du dossier courant et d'utiliser les .o générés pour faire notre librairie.

La compilation des fichiers .cpp se fait via la commande : $(CC) -Os -mmcu=$(MCU) -c *.cpp.

CC contient la commande avr-gcc permettant d'appeler le compilateur. L'option -Os permet de limiter l'optimisation et d'éviter plusieurs erreurs. -mmcu=$(MCU) permet de définir l'implémentation de l'avr, ainsi MCU contient atmega324pa. Enfin -c *.cpp précise que tous les fichiers .cpp du dossier sont sources.

La création de notre librairie se fait elle, avec la ligne de commande $(CL) crs lib$(LIBNAME).a *.o

Ici, nous avons déjà précisé le rôle de CL plus haut. Dans crs, c permet de créer l'archive, r d'insérer les fichiers membres et s produit l'index de notre librairie. Ensuite, vient le nom de notre librairie avec son extension (ici on peut également préciser qu'on peut préciser un chemin afin que notre librairie soit générée dans le dossier de notre choix) et enfin *.o pour préciser que tous les fichiers .o du dossier courant serviront à générer l'archive.

Enfin, nous avons ajouté la règle delete qui a le même comportement que clean, à la différence qu'il supprime notre librairie également.