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 4563

Noms :
Dominique Deschênes
Bruno Bousquet
Julio Hervé-Dandjinou
Staelle Foko

10 mars 2016

Partie 1 : Description de la librairie

La librairie est constituée de plusieurs fichiers échafaudés au fil de la session. En effet, on y retrouve notamment les fichiers tp.cpp, can.cpp, memoire_24.cpp et usart.cpp avec leurs fichiers d'entête bien sûr.

Pour commencer, le premier fichier tp.cpp et son fichier d'entête contiennent quelques fonctions qui nous ont été utiles depuis le début de la session et qui risquent d'être encore utiles dans le futur. Par exemple, la fonction powerRoue(), qui gère les moteurs pour contrôler la vitesse du robot. Cette fonction génère un PWM selon les intrants qu'elle reçoit, soit la vitesse (0, 25, 50, 75, 100), la direction, le temps de rotation des moteurs et la fréquence désirée du PWM. Aussi, les fonctions relatives aux interactions avec le bouton-poussoir sont présentes dans ce fichier. La fonction getEtatInterrupteur() retourne simplement l'état instantané du bouton. D'autres fonctions déterminent, entre autre grâce au « debouncing », si le bouton est définitivement actif ou même s'il est actif pour la première fois depuis sont inactivité. C'est aussi dans ce fichier que l'on retrouve toutes les définitions de nos constantes dont la fréquence du CPU ou les valeurs hexadécimales nécessaires à la configuration des ports.

Ensuite, le fichier usart.cpp contient les deux fonctions qui nous permettent de gérer le périphérique USART. Ce périphérique sert à communiquer avec, par exemple, un ordinateur. Il peut transmettre et recevoir des données. La première fonction contenue dans le fichier nous sert à l'initialisation du périphérique, pour l'utiliser comme voulu, et la deuxième nous permet de transmettre des données à travers le périphérique.

De plus, deux classes ont été intégrées à notre librairie. En effet, nous avons jugé essentiel d'avoir les classes Memoire24CXXX et CAN à portée de main. La première classe permet la gestion de la mémoire. On y retrouve la fonction d’initialisation, pour spécifier l'utilisation de la mémoire qui sera faite. Puis, des fonctions de lecture et d'écriture sont aussi présentes pour manipuler la mémoire.

L'autre classe permet de gérer le convertisseur analogique numérique. C'est un aspect très important du projet, car les périphériques ne traitent pas tous l'information de la même façon. Il est donc très utile de pouvoir manier le convertisseur à notre guise avec une classe permettant de manipuler les données analogiques ou numérique.

Partie 2 : Modification du Makefile

Tout d'abord, comme les deux Makefiles sont très semblables, nous avons jugé préférable de simplement garder une base commune et l'inclure dans nos deux instances de Makefile. C'est pourquoi nous avons donc utilisé le Makefile original fourni et que nous l'avons renommé en .txt. À partir de ce moment, nous avons créé nos Makefiles et nous avons inclus Makefile_Common.txt à l'intérieur de ceux-ci. Ensuite, nous avons procédé à une analyse du Makefile original pour cerner les différences entre celui-ci et les Makefiles respectifs à notre librairie et notre exécutable bidon.

La première différence notable était clairement le type de fichier résultant de l'exécution make. Ainsi, nous avons retiré les définitions du PROJECTNAME et du TRG du Makefile commun pour les redéfinir. Dans le Makefile de la librairie, comme nous voulons créer une archive, nous l'avons appelé libmylib et son extension est .a. Le Makefile de l'exécutable, quant à lui, crée un .out.

La seconde étape consistait a énumérer correctement les fichiers sources propre à chaque Makefile. Nous avons donc simplement ajouté tous nos fichiers .cpp, qui eux contiennent les fichiers .h par inclusion, au PRJSRC du Makefile de la librairie. Du côté de l'exécutable, le seul fichier source à inclure était le .cpp du code test. Il fallait cependant inclure les fichiers .h de la librairie dans la variable INC (additional includes) à l'aide de l'option -i et la librairie elle-même dans la variable LIBS. En spécifiant le chemin et le nom de la librairie le tour était joué.

Finalement, la dernière modification à faire se trouvait dans les commandes de compilation. Les règles pour compiler sont les mêmes, hormis le fait que ce sont des fichiers différents sous la variable OBJDEPS. Par contre, pour compiler la librairie, on utilise la commande avr-ar en mode rcs, pour compiler une librairie statique. Pour compiler l'exécutable, on utilise la commande classique avr-gcc, tout en prenant soin d'inclure la librairie à la commande.