Polytechnique Montréal

Département de génie informatique et génie logiciel

Cours INF1995 :

Projet initial en génie informatique et travail en équipe

Travail pratique no 8 :

Formation d’une librairie statique avec Makefile

Équipe no 96108

Travail soumis par :

Maude Laflamme

Elisabeth Fagnan

Youcef Anis Dilmi

Frédéric Brossard-Pellerin

Date :

Le lundi12 mars 2018

# Partie 1 : Description de la librairie

La librairie libcrazyrobot est formée de différentes fonctions et classes utilisées lors des travaux pratiques faits depuis le début de la session. Elle regroupe donc les classes et fonctions nécessaires à l’utilisation de certaines composantes du robot tels que la DEL, les roues et les périphériques interne et externes du microcontrôleur. Ces fonctions sont brièvement expliquées ci bas mais le fonctionnement de chaque fonction est détaillé dans les fichiers de code source.

* La fonction init\_pwm() inscrit les valeurs nécessaires dans les bons registres pour initialiser le signal PWM.
* La fonction PWM\_PORTD() permet le contrôle des roues du robot, c’est-à-dire qu’elle contrôle la direction de la roue gauche et de la roue droite indépendamment ainsi que le rapport du signal PWM envoyé à chaque roue pour contrôler sa vitesse. Ceci à travers le PORTD.
* La fonction start\_timer\_CTC(uint16\_t top) inscrit les valeurs nécessaires dans les bons registres pour démarrer le timer1 en mode CTC activer le vecteur d’interruption (TIMER1\_COMPA\_vect) pour pouvoir exécuter une routine d’interruption après le nombre de cycles désiré.
* La fonction init\_INT0() inscrit les valeurs nécessaires dans les bons registres afin d’activer le vecteur d’interruption externe (INT0\_vect) pour permettre d’exécuter une routine d’interruption suite à une interruption externe (Dans notre cas, suite à l’appui du bouton d’interruption sur la carte mère ou un autre bouton poussoir connecter sur la PIND(2)). Il est important d’y faire appel au début d’un protocole utilisant l’interruption.
* La classe can permet de faire une conversion d’un signal d’analogique (0 à 5V) à une valeur numérique sur 8 bits(0 à 255). Le fichier pose les paramètres du périphérique de conversion analogique-numérique « can » interne au ATmega324PA.
* La fonction can\_photo\_PORTC.cpp utilise la classe can afin de convertir le signal analogique (0 à 5V) provenant d’un circuit composé d’une photorésistance en une valeur numérique de 8 bits (0 à 255). La fonction dans ce fichier permet de faire varier la couleur de la DEL du robot en fonction de l’intensité lumineuse à laquelle est exposée la photorésistance. Cette fonction utilise la fonction LED\_amber() pour allumer la DEL en couleur ambrée. Cette fonction envoie les niveaux logiques aux pin du PORTC(0,1).
* La fonction LED\_amber\_PORTC.cpp permettant de faire allumer la LED du robot de couleur ambrée, en faisant rapidement changer la couleur de la LED de vert à rouge (en inversant le sens du courant qui traverse la LED. Ceci à travers les pins du PORTC(0,1).
* La classe Memoire24CXXX permet l’accès à la mémoire externe du robot. Cette classe possède une méthode pour écrire dans la mémoire, et une méthode pour lire des données venant de la mémoire. Tout ceci par le protocole de communication I2C.
* Le fichier func\_UART.cpp définit des fonctions permettant la communication par protocole RS232, la réception de données du robot via le périphérique de transmission-réception U(S)ART.
* Le fichier errors\_PORTA.cpp définit des fonctions pour faire clignoter la DEL lorsqu’on détecte une erreur dans le fonctionnement du robot. Il y a trois clignotements d’erreur possible, rouge, vert et clignotement de rouge à vert.

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

**Dans le Makefile pour créer le fichier archive «libcrazyrobot.a», les modifications apportés au Makefile original que nous utilisons depuis le début de la session sont les suivantes :**

* Pour la compilation des fichiers «.cpp» en fichiers «.o», il n’y a pas de différence.
* Pour le linking, il faut utiliser «avr-ar» plutôt que «avr-gcc».
* Pour le linking, les LDFLAGS et autres options ont été retirés et nous ajoutons un ARFLAGS pour «avr-ar» . Ce flag contient seulement l’option «-crs».
* De plus, il faut changer l’extension de la sortie pour «.a» au lieu de «.out».

**Dans le Makefile pour créer le fichier exécutable «.out», les modifications apportées au Makefile original que nous utilisons depuis le début de la session sont les suivantes :**

* Pour la compilation, il faut indiquer à «gcc»  où trouver le code source en ajoutant dans la variable d’inclusion «INC» l’emplacement du dossier contenant le code source de la librairie soit avec l’option «-I ../lib\_dir» qui sera réécrite dans les CFLAGS.
* Nous avons ajouté dans les CFLAGS l’option «Werror=all» pour que les avertissements génèrent des erreurs et empêchent la compilation.
* Nous avons ajouté dans les CFLAGS l’option «-DF\_CPU=8000000» afin de définir la fréquence d’horloge du microcontrôleur dans chaque fichier de code source.
* Pour le linking, il faut indiquer à «gcc» quelles librairies utiliser en écrivant le nom de celles-ci dans la variable «LIB»  qui sera ajoutée aux options de linking.
* Pour le linking, il faut indiquer dans les LDFLAGS où trouver les librairies à utiliser, soit avec l’option «-L ../lib\_dir »

Le Makefile commun contient toutes les lignes qui sont communes aux deux Makefiles cités précédemment.