École Polytechnique de 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 librairies statiques

Par l'équipe

No 0723

Noms:

Feriel Charfeddine
Weiwei Liang
Samuel Rondeau
Louis Racicot

Date:
1er novembre 2014

Partie 1

En comparant les codes des deux équipes, nous avons choisi de répertorier les fonctionnalités qui nous semblaient les plus utiles.

Le code des TP 1 et TP2 présentaient plusieurs fonctionnalités utiles:

- Allumer la DEL en rouge et en vert
- Éteindre la DEL
- Vérifier si un interrupteur est appuyé
- Attendre un certain nombre de millisecondes ou de microsecondes.

Les fonctions permettant d'allumer les DEL, de même que celles qui permettent de gérer plus facilement les délais semblaient très utiles. Les fonctions de DEL peuvent être utilisées par n'importe quel autre programme dans des buts de débogage et de message de confirmation. Les fonctions de délais, elles, sont très utiles étant donné qu'elles facilitent la gestion des délais. Par contre, nous ne mettrons pas les fonctions de l'interrupteur dans la librairie étant donné que nous savons maintenant gérer cette situation à l'aide de vecteurs d'interruptions.

Le TP4 portait sur l'utilisation des moteurs d'un point de vue logiciel, nous allons cependant utiliser les fonctionnalités du TP5, où nous utilisions plutôt les moteurs avec un PWM matériel. Le TP 5 portait sur ces différents aspets:

- Gestion des vecteurs d'interruption (bouton poussoir).
- Utilisation des minuteries et des interruptions de la minuterie
- Gestion d'un PWM matériel à l'aide d'une minuterie.

Dans ces fonctionnalités, nous avons choisis de ne garder que le PWM matériel. Celui-ci nous sera très utile quand viendra le temps de faire avancer notre robot. En ce qui concerne le bouton poussoir, il n'y avait pas de "fonctionnalité" proprement dit reliée à cette action. La gestion de ce genre d'interruption se fera toujours manuellement. Il en va de même pour la gestion des interruptions de la minuteries.

Note: En ce qui concerne les interruptions de la minuterie, il y aura le registre "TIMSK1" à configurer. Nous n'avons pas fait de classe spécifique pour cela, mais nous réservons la possibilité d'en faire une si le besoin se présente.

Le TP6 portait plutôt sur l'écriture et la lecture mémoire, de même que sur la communication en série par le port USB, ce qui en fait une partie très importante dans notre projet. Nous allons évidemment garder la classe «Memoire24CXXX» fournie et

allons l'intégrer à notre librairie statique. En ce qui concerne la communication en série, une classe sera construite dans la librairie pour faciliter son implémentation.

Dans le TP7, nous utilisons une classe «can» pour contrôler le convertisseur analogique/numérique. Elle sera réutilisée dans notre projet et nous allons donc la placer dans notre librairie.

Voici une présentation détaillée des classes qui seront créées pour gérer les fonctionnalités que nous désirons conserver.

1. DEL

La classe de led représente une DEL qui pourra prendre la couleur rouge ou verte et présente les méthodes suivantes:

- **led(Ports port, uint8_t pin1, uint8_t pin2)**
  Nous devons spécifié le port (Ports::A, Ports::B, Ports::C or Ports::D) de même que les pines (0..7) sur lesquelles est branchée la DEL. Les port sont un *emun* défini en haut du fichier.s

- **void lightOnGreen()**
  Allume la DEL avec la couleur verte. Si elle tourne au rouge avec cette méthode, il faudra inverser les fils.

- **void lightOnRed()**
  Allume la DEL avec la couleur rouge. Si elle tourne au vert avec cette méthode, il faudra inverser les fils.

- **void lightOff()**
  Éteint la DEL

2. Sleep

Les fonctions usleep() et msleep() permettent d'attendre quelques instants durant l'exécution selon le FCPU delay_loop2.

- **void msleep(int ms)**
  Nous devons spécifier le nombre de millisecondes d'attente. La boucle fait en sorte que le processeur reste sans rien faire pendant un certain nombre de ms.

- **void usleep(int us)**
  Nous devons spécifier le nombre de microsecondes d'attente. La boucle fait en sorte que le processeur reste sans rien faire pendant un certain nombre de ms.

3. PWM

La classe de PMW: elle contient deux fonctions, la fonction start() et stop() qui permettent de faire bouger ou arrêter le moteur, et la fonctions adjust().

- **void adjust(uint8_t ratio255)**

  Nous devons spécifier le pourcentage de fonctionnement du PWM selon le ration de 255 (valeur maximale). La fonction ajuste les pins OCR1A ou OCR1B selon notre choix du port en constructeur.

- **void start()**

  Cette fonction ajuste le registre TCCR1A. Elle met les pins OC1A ou OC1B (selon notre choix de port en constructeur) en mode “Compare Match” ce qui permet de démarrer le moteur.

- **void stop()**

  Cette fonction ajuste le registre TCCR1A. Elle met les pins OC1A ou OC1B (selon notre choix de port en constructeur) en mode “disconnected” ce qui permet de d’arrêter le moteur.

4. Mémoire

La classe Mémoire: Le code pour la mémoire nous a été fourni et son explication est assez complexe pour notre niveau, Cependant, nous devons savoir que nous avons 6 fonctions essentielles:

- **void init()**
  Initialise le "memory bank" à zéro.
- **static uint8_t choisir_banc(const uint8_t banc)**
  Change l’adresse de la mémoire.
- **uint8_t lecture(const uint16_t adresse, uint8_t *donnee)**
  Permet la lecture de la mémoire en spécifiant l’adresser et la donnée.
- **uint8_t lecture(const uint16_t adresse, uint8_t *donnee, const uint8_t longueur)**
  Permet la lecture de la mémoire en spécifiant l’adresser, la donnée et la longueur.
- **uint8_t ecriture(const uint16_t adresse, const uint8_t donnee)**
  Permet l’écriture sur la mémoire en spécifiant l’adresser et la donnée.
- **uint8_t ecriture(const uint16_t adresse, uint8_t *donnee, const uint8_t longueur)**
  Permet l’écriture sur la mémoire en spécifiant l’adresser, la donnée et la longueur.

5. Serial

La classe Serial: Implémenter la communication en série du port USB.
Nous retrouvons deux fonctions dans cette classe:

- **void init()**

Elle permet d'initialiser les registres UBRR0H UBRR0L en leur donnant des valeurs pour éviter des complications par la suite. Il faudra aussi ajuster les vecteurs UCSR0A et UCSR0B pour permettre la réception et la transmission par le UARTO. Finalement, cette fonction ajuste le registre UCSR0C pour formater les trames selon le mode: 8 bits, 1 stop bit, none parity).

- **void send(uint8_t data)**

Cette fonction envoit un octet de donnée vers le port USB.

6. Convertisseur A/N

La classe CAN (can.h et can.cpp): permet le contrôle du convertisseur analogique ou numérique. Cette classe nous a été fournie, nous devons l'inclure pour traduire les signaux numériques que nous transmettons au robot en données numériques que le processeur pourra comprendre et traiter.

Partie 2

Le Makefile utilisé pour générer la librairie *libpout.a* est basé sur le Makefile donné au début du cours. Le première étape est évidemment de renommer le projet (*PROJECTNAME*), de modifier les fichiers sources, et de remplacer *TRG=$(PROJECTNAME).a* par *TRG=$(PROJECTNAME).a*, dans le but d'obtenir comme produit final un fichier nommé *libpout.a*. Ajoutons *.a* à *.SUFFIXES*.

On pourrait également créer une nouvelle instruction de compilation, *AR=avr-ar*, mais puisque nous ne l'utiliserons qu'une seule fois et qu'elle a besoin de l'option *crs*, nous l'ometterons cette fois. On supprime toute génération de fichiers *hex* ainsi que les options *hex*, *writeflash* et *install*, puisque la librairie elle-même n'est pas à installer sur la carte mère du robot.

On modifie deux éléments de la commande de la règle *$(TRG): $(OBJDEPS)*, soient la compilation (*avr-ar crs*) et la destination de la librairie, simplement afin de satisfaire notre architecture de travail. On ajoute *-o $@* aux commandes pour *.c.o .cc.o .cpp.o .C.o* pour générer les fichiers objet compilés à partir des fichiers source, ainsi que *-c* à *CFLAGS*, *CPPFLAGS* et *ASMFLAGS*. Nous avons alors un Makefile compilant notre librairie.

Quant au Makefile de l'exécutable bidon, on inscrit la librairie *lib/libpout.a* à *LIBS=* . L'option *-lm $(LIBS)* est déjà incluse dans le Makefile de base dans le commande de la compilation de la cible exécutable. Assurons-nous encore de la présence de *-o $@* pour la génération des fichiers objets à partir de fichiers c/c++/etc. Pour le reste, il ne suffit que de mettre à jour les fichiers sources et le nom du projet.