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 4754

Noms:
Louis-Philippe Chouinard
Alexandre Vu
Gabriel Hélie
Ibrahima Séga Jr Sangaré
Olivier St-Onge

Date:
10 mars 2016

Introduction

Les travaux pratiques précédant celui-ci ont été réalisés en équipe de deux et consistaient en la conception de petits programmes simples permettant de se familiariser avec différents concepts. Dans le cadre de ce laboratoire, l'objectif est de pouvoir rassembler ces fragments de code afin de créer une librairie statique permettant l'utilisation de ceux-ci dans les travaux futures. Ces différents résultats de travaux pratiques doivent donc être adaptés en une série de classes pour que les fonctions soient compatibles entre elles et utilisable dans un même programme. De plus le makefile doit être modifié afin de pouvoir fusionner les fichiers retravaillés en une librairie statique (utilisée lors de la compilation du programme de test grâce à un second makefile). Ce rapport est divisé en deux sections: la description des différentes classes et de leurs méthodes respectives et la description des modifications apportées aux makefiles (en comparaison au makefile de base utilisé pour les anciens travaux pratiques).

Partie 1 : Description de la librairie

Notre librairie est formée des classes Del, Moteur, Can et Minuterie afin de faciliter la réutilisation du code pour les prochaines étapes du TP.

Notre classe Del n'a pas d'attributs, mais contient plusieurs méthodes. Nous avons définit un constructeur par défaut qui initialise DDRB à la valeur 0x03. La classe Del contient aussi un destructeur, ainsi que les méthodes allumerVert, allumerRouge et faireAmbree. Ces trois dernières méthodes ont pour but d'allumer la DEL connectée aux broches 1 et 2 du port B de la couleur désirée. L'argument tempsEteint spécifie le temps en millisecondes pour lequel nous voulons fermer la DEL après l'avoir allumé afin de réduire l'intensité lumineuse au besoin. Pour ce faire, il suffit de placer une de ces fonctions dans une boucle pour l'appeler à répétitions. Plus tempsEteint est grand, plus l'intensité de la DEL est basse. Finalement, nous avons définit une méthode éteindre qui sert tout simplement à fermer la DEL. L'utilité d'avoir une classe Del est de faciliter le contrôle des signaux de sortie se rendant à la diode sans avoir à se rappeler la valeur hexadécimale correspondant à chaque couleur.

La classe Moteur n’a pas d’attribut non plus. Tout comme la classe Del, celle-ci contient un constructeur par défaut et un destructeur. Nous avons définit cinq méthodes de vitesse par fréquence, c’est-à-dire cinq méthodes à 60Hz et cinq autres à 400Hz. Nous avons la méthode eteint, qui arrête tout roulement. Nous avons la méthode premiereVitesse qui fait tourner les roues du robot à 25% selon le rapport a/b du PWM. Ensuite, la méthode deuxiemeVitesse fait rouler le robot à 50%, la méthode troisiemevitesse fait rouler le robot à 75% et la méthode quatriemeVitesse fait rouler le robot à 100%. Pour l’implémentation de ces méthodes, nous avons repris le code d’un TP précédent, mais l’avons modifié afin que les deux roues soient en mouvement plutôt qu’une seule. De plus, le port C a été sélectionné comme port de sortie pour les roues afin de permettre un contrôle du moteur et contrôle de la Del de manière simultanée. L’utilité d’une classe Moteur est de faciliter le contrôle de signaux envoyés au moteur et de choisir la vitesse sans avoir à refaire les calculs du rapport a/b du PWM à chaque fois que l’on veut modifier la vitesse du robot.

La classe Minuterie quant à elle contient un constructeur, un destructeur, une méthode permettant l’acquisition de la durée, une méthode permettant la modification de la durée ainsi que les méthodes partirMinuterie, estExpirée et setExpiree. Tout d’abord, la méthode partirMinuterie permet de lancer un compteur à l’aide des registres de l’Atmega324PA. Celui-ci compte jusqu’à la durée entrée par l’utilisateur et fait appelle à l’interruption qui permet de modifier l’attribut minuterieExpirée à 1. Puis, le compteur, étant sur le mode CTC, sera remis à zéro. D’autre part, la méthode estExpirée retourne une valeur booléene. Si la minuterie est expirée, la méthode retournera la valeur true. Dans le cas contraire, elle retournera false. Finalement, la méthode setExpirée permet de mettre l’attribut minuterieExpiree_ à 1, indiquant la fin de la minuterie. Au tout début, nous avions fait en sorte que la minuterie ne soit considérée comme expirée que lorsque la valeur maximale était atteinte, mais ça ne laissait pas le temps au programme de pouvoir s’en rendre compte puisque minuterieExpiree revenait tout de suite à 0 de manière automatique quand la minuterie recommençait. Ainsi, la méthode fut modifiée pour que dès que la minuterie atteingne la valeur maximale, minuterieExpirée valle 1 tant que le programme ne la remette pas manuellement à 0, permettant ainsi au programme d’avoir le temps d’utiliser cet attribut.

Les fichiers de la classe Can sont ceux fournis au TP précédent. Puisque nous savons que cette classe est déjà fonctionnelle, aucun test pour celle-ci n’a été implémenté dans le main.

Un fichier main.cpp a été mis en place afin de tester le bon fonctionnement des différentes classes. Ce programme utilise la minuterie afin de faire alterner la couleur émise par la DEL entre vert et rouge, en plus d'utiliser la fonction deuxiemeVitesse pour faire avancer les roues. Nous avons également ajouté une fonction d'interruption avant le main qui est activée lorsque la minuterie atteint sa valeur maximale afin de vérifier le bon fonctionnement de cette dernière. Nous avons décidé d'implémenter directement cette fonction dans le fichier de test puisqu'une fonction de type ISR ne peut être appelée à l'intérieur d'une autre fonction (et ne peut donc pas être une fonction membre d'une classe). Nous avons jugé préférable de garder le code en lien aux interruptions en commentaires dans nos différents fichiers sources pour pouvoir manuellement le copier-coller plus tard lors des travaux pratiques à venir.

Partie 2 : Décrire les modifications apportées au Makefile de départ

La première modification apportée aux makefiles se trouve à la ligne de la source du projet. Pour le makefile servant à créer la librairie, cette ligne contient l'ensemble des fichiers .cpp et des fichiers .h correspondant, tandis que le makefile servant à compiler le programme de test ne vise que le fichier main.cpp avec cette ligne.

```
PRJSRC= main.cpp    # (Makefile de l'exécutable)
```

```
 PRJSRC= delControl.cpp delControl.h Moteur.h PWMvitesse.cpp can.h can.cpp minuterie.h minuterie.cpp              # (Makefile de la librairie)
```

La deuxième modification se situe au niveau de la ligne des librairies et n'est utilisée que par le makefile qui permet la compilation du programme de test, car il utilise les fonctions d'une libraire, soit celle créée à l'aide du makefile précédent.

```
LIBS= -L./ -lTest
```

La troisième modification se trouve dans le makefile qui crée la librairie statique. Le fichier d'output que l'on désire créer est un .a et non un .o (c'est pourquoi l'extension est ici modifiée).

```
TRG=$(PROJECTNAME).a
```

La quatrième modification est effectuée dans le makefile de la librairie où l'on utilise avr-ar au lieu de avr-gcc, puisque l'on cherche à générer une archive plutôt qu'à créer un exécutable.

```
$(TRG): $(OBJDEPS)
    avr-ar -csr -o $(TRG) $(OBJDEPS)
```

Note: les deux makefiles ont été remis avec un suffixe ajouté à leurs noms afin de les différencier, suffixe qu'il faudra retirer lors de l'utilisation de la commande make.

Conclusion

En conclusion, ce travail pratique a permis de rassembler l'ensemble des différents travaux effectués au cours de la session en une seule librairie qui sera particulièrement utile pour la suite du projet, notamment quand viendra le temps d'écrire le code final du robot. La librairie n'est pas encore parfaite et il faudra y faire des ajouts, comme une méthode permettant au robot de reculer, et peut-être même des méthodes permettant le contrôle des roues de manière indépendante.