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 librairie statique

Par l'équipe

No 0622

Noms:

Abdellatif Amrani

Jackie Phung

Mohamed Amine Belaid

Bilal Itani

Date:
10 mars 2015

Partie 1 : Description de la librairie

can.h et can.cpp

can.h contient la définition de la classe can et can.cpp contient l'implémentation de ses fonctions membres. Nous avons décidé de retenir cette partie du code, car nous estimons avoir besoin plus tard de fonctions permettant la conversion entre des données analogiques vers des données numériques.

memoire_24.h et memoire_24.cpp

memoire_24.h contient la définition de la classe Memoire24CXXX et memoire_24.cpp contient l'implémentation de cette classe. Nous avons choisi de retenir cette partie du code, puisque nous pensons qu'il peut être utile plus tard pour le projet. Grâce à ce code, il est possible de stocker des données dans la mémoire et de les lire.

moteur.h et moteur.cpp

moteur.h contient la définition de la fonction ajustementPWN qui prend en paramètre deux entiers non signés sur 8 bits. moteur.cpp contient l'implémentation de cette fonction. Elle va d'abord mettre à 1 les bits COM1A1, COM1A0, COM1B1 et COM1B0 du registre TCCR1A, et elle va aussi mettre à 1 le bit WGM10 de ce même registre. De plus, les paramètres de cette fonction affectent les registres de comparaisons OCR1A et OCR1B. Finalement, le bit CS11 du registre TCCR1B va être mis à 1, ce qui va permettre une division d'horloge par 8. Cette configuration décrite ci-haut permet de générer un signal PWN dans la mesure où le timer counter 1 sera en mode PWN phase-correct, lorsque le compteur compte de 0 à 255. Il va arriver à la valeur de OCR1A/OCR1B, et va mettre à 1 les pins D4/D5. Par la suite, lorsqu'il compte de 255 à 0 et qu'il arrive à ces mêmes valeurs de OCR1A/OCR1B, il va mettre ces mêmes pins à 0. Nous avons ainsi un PWN qui permettra le contrôle de la vitesse des moteurs et donc du robot. Cette fonction est très importante pour notre robot, puisque sans celle-ci, nous ne pourrons pas faire avancer le robot comme bon nous semble. C'est pour cette raison qu'elle se trouve dans notre librairie statique.

minuterie.h et minuterie.cpp

minuterie.h contient les définitions des fonctions calculerNombreCycles, qui prend les paramètres duree et prescale, deux entiers non signés sur 16 bits, et partirMinuterie, qui ne prend en paramètre que le même paramètre duree. Minuterie.cpp contient l'implémentation de ces fonctions: partirMinuterie permet de déclencher un Timer ajusté par son registre TCCR1A au mode de comparaison à la valeur du registre OCR1A, et par son registre TCCR1B à 1 prescale de 1024 (ce choix de prescale est dû au fait que l'on cherche à utiliser des Timers dont les durées dépassent la seconde, chose que les autres prescales ne sont pas capables de fournir.), ainsi qu'au mode CTC, qui permet de réinitialiser le compteur une fois la minuterie expirée, tandis que la fonction calculerNombreCycles permet de déterminer le nombre de cycles d'horloges nécessaires afin de générer la minuterie selon la durée en ms et le prescale utilisé qui lui ont été donnés en paramètres. La valeur de calculerNombreCycles sera retournée dans le registre OCR1A.

boutonPoussoir.h et boutonPoussoir.cpp

boutonPoussoir.h contient la définition de la fonction initialisationBouton qui prend en paramètre deux booléennes le risingEdge et le fallingEdge. boutonPoussoir.cpp contient l'implémentation de cette fonction. Dans un premier temps, initialiserBouton met le bit INT0 du registre EIMSK à 1 afin de sensibliser le bouton poussoir Interrupt de la carte mère aux interruptions externes. L'étape suivante consiste à modifier les bits ISC00 et/ou ISC01 du registre EICRA en fonction des paramètres risingEdge et fallingEdge afin de sensibiliser le bouton poussoir aux interruptions dûe à un front montant et/ou descendant, selon les besoins de l'utilisateur. Durant l'initialisation du bouton, les interruptions sont momentanément bloquées.

uart.h et uart.cpp

uart.h contient la définition des fonctions initialisationUART, transmissionUART et transmissionEntierUART. Le fichier uart.cpp contient l'implémentation des fonctions mentionées précédemment. Nous avons décidé d'ajouter ces fonctions dans notre librairie, car nous risquons de les utiliser plus tard dans le projet pour tester, par exemple, le bon fonctionnement d'un capteur en affichant, par le biais d'une conversion analogique/numérique et des fonctions définies dans uart.h, à l'écran de l'information relative aux capteurs qui nous permettra ou non de confirmer/infirmer le bon fonctionnement des capteurs. On pourrait aussi utiliser l'uart pour vérifier le contenu de la mémoire du robot, notamment pour vérifier si le stockage de certaines informations dans la mémoire est fait avec succès ou non. Bref, c'est pour ces raisons que l'on décide d'inclure les fonctions de l'uart dans notre librairie.

Partie 2 : Décrire les modifications apportées au Makefile de départ

Makefile de la librarie

Nous avons dans un premier temps modifié les instructions de compilation de manière à ce que le résultat final généré soit la librarie, dont les fichiers sources (d'extension .cpp, eux-mêmes dépendants de leurs en-têtes respectifs .h) sont présents dans le dossier lib_dir. Puisque la génération d'un fichier .a est notre seule attente de ce Makefile, nous avons supprimé le reste des instructions qui nous paraissaient inutiles dans ce contexte, tel que par exemple la génération de fichiers .hex, ainsi que la commande «install». Enfin, certains éléments ont été transportés au Makefile commun, et de ce fait, l'instruction «include ../Makefile.commun» est rajoutée vers le début du fichier.

Makefile du code

Les modifications apportées dans le Makefile de l'executable sont les changements pour INC et LIBS. Dans la ligne pour affecter INC, nous devons écrire «-I ../lib_dir» pour inclure les fichiers .h en indiquant le chemin à prendre, alors que dans la ligne pour affecter LIBS, nous devons écrire «-L ../lib_dir/ -lstatique» pour charger la librairie en indiquant aussi le chemin. Le «I» dans la ligne INC signifie «include», tandis que le «L» et le «l» dans la ligne LIBS représentent respectivement «check symlink-times» et «load». Nous avons aussi rajouté une inclusion du Makefile.commun. Par la suite, nous avons suprimé les commandes de compilation pour les mettre dans le makefile commun, puisque ces commandes se retrouvent aussi dans le makefile de la librarie.

Makefile commun

Le Makefile commun comprend l'ensemble des variables et instructions qui sont communes aux deux autres makefiles. C'est-à-dire qu'il contient le nom du controleur atmega utilisé (atmega324pa), les fichiers sources sous forme abrégée ($(wildcard *.cpp) $(wildcard *.c) $(wildcard *.s)), le niveau d'optimisation, les commandes de compilation, les variables «executables» et la commande de «cleanup».