Polytechnique Montréal

Département de génie informatique et génie logiciel

Cours INF1900 :
Projet initial de système embarqué

Travail pratique 7

Makefile et production de librairie statique

Par l'équipe

No 1634

Noms :

BANCE HALIMA	[1926066]
BOCAR DIALLO	[1859433]
Ramzi Belbahri	[1963293]
Abderraouf Haddouni	[1958139]

Date :
22 octobre 2019

Partie 1 : Description de la librairie

Dans le déroulement de la conception de notre librairie, notre équipe s'est orientée vers la réalisation d'une librairie en procédurale. Toutefois nous utilisons des codes qui ont été fournis aussi par exemple memoire_24.h facilitant l'accès à la mémoire externe du robot ou can.h permettant la conversion analogique numérique. Pour ce faire, notre équipe a opté pour la création de classes dans notre librairie à savoir :

- Bouton : corresponds aux fonctionnalités du bouton poussoir
- DEL : corresponds aux fonctionnalités de la DEL
- Can : corresponds aux fonctionnalités du convertisseur analogique numérique
- Mémoire_24 : fonction d'affichage du contenu de la mémoire flash
- Minuterie : contrôle de la minuterie (ICI TIMER1)
- Roue : fonction liée au fonctionnement des roues
- UART : protocole de transmission des données

Dans les lignes suivantes, nous décrirons ces classes en précisant leur utilité.

- **CLASSE BOUTON**

  Description du choix : C'est une classe qui effectue une lecture du bouton-poussoir avec un filtre anti-bond. En effet, elle indique si le bouton poussoir a été appuyé ou non. Si nous utilisons le bouton poussoir celui pour usage général sur le robot, il s'agit d'une interruption interne et il faut placer un cavalier en INT0. Dans le cas où le bouton poussoir est un capteur, il faut retirer le cavalier placer en INT0 pour empêcher toute interruption interne.

  Utilités des fonctions choisies :

  - **bool estAppuyer () :** Cette fonction indique si le bouton poussoir a été pesé. Si nous utilisons le bouton poussoir présent sur le robot il s'agit d'une interruption interne et il faut placer un cavalier en INT0. Dans le cas où le bouton poussoir est externe, il faut retirer le cavalier placer en INT0 pour empêcher toute interruption interne.

  - **void initialisation(void) :** C'est une fonction qui effectue l'initialisation de la configuration des registres pour effectuer les interruptions à l'aide du bouton-poussoir. La routine cli est appelée pour bloquer toute interruption durant cette initialisation, suivi de la mise en entrée du PORTD, puis on active les interruptions et les changements logiques en INT0.

- **CLASSE DEL**

  Description du choix : cette classe permet d'afficher la couleur de la DEL et aussi d'effectuer le clignotement.

  Utilités des fonctions choisies :

  - **void allumerDEL (uint8_t couleur, char port) :** Cette fonction permet d'allumer la DEL libre du robot selon les paramètres passés. Elle prend en paramètre la couleur (0x00 = éteint, 0x01 = vert, 0x02 = rouge) et un char (A, B, C, D) qui détermine le port mis en mode sortie.

  - **void allumerDelAmbre (char port) :** Cette fonction prend en paramètre un char qui correspond au port a utilisé et permet d'allumer la Del libre du robot en Ambre en alternant les couleurs vertes et rouges.

- **void clignoterDel (uint8_t nbRepetition, uint8_t couleur, char port) :** Cette fonction permet de faire clignoter la DEL libre du robot de la couleur voulu. Cette fonction prend en paramètre le nombre de répétitions, la couleur et le port qui est en sortie.

CLASSE CAN

Description du choix **:** C'est une classe qui permet Conversion des données analogiques numérique.

Utilités des fonctions choisies :

- **can () :** C'est un constructeur de la classe can qui initialise le convertisseur.
- **~ can () :** C'est un destructeur de la classe can qui supprime le convertisseur.
- **uint16_t lecture (uint8_t pos) :** Cette fonction prend en paramètre pos qui est un entier sur 8 bits représentant la broche du port a connecté au capteur analogique. Elle retourne la valeur numérique correspondant à la valeur analogique sur le port A. Ici, il faut faire un décalage de deux bits vers la droite pour manipuler les 8 bits significatifs, car ses deux derniers bits ne sont pas importants.

CLASSE memoire_24

Description du choix : Cette classe permet l'affichage du contenu de la mémoire flash du atmega324pa.

Utilités des fonctions choisies :

- **Memoire24CXXX () :** C'est un constructeur de Memoire24CXXX (), il ajuste la taille de la page appelle une méthode d'initialisation ; cette dernière initialise le port série et l'horloge de l'interface I2C.
- **~Memoire24CXXX ()** : C'est un destructeur de memoire24cxxx ().
- **void init () :** C'est une fonction d'initialisation du constructeur
- **uint8_t lecture (const uint16_t adresse, uint8_t *donnee) :** La lecture qui se fait caractère par caractère et prends en paramètre l'adresse et un pointeur vers les données.
- **uint8_t lecture (const uint16_t adresse, uint8_t *donnee, const uint8_t longueur) :** Lecture qui se fait par bloc qui prend en paramètre l'adresse, un pointeur vers les données, et la longueur (127 et moins) de la chaine à lire.
- **uint8_t ecriture (const uint16_t adresse, const uint8_t donnee) :** Elle prend en paramètre l'adresse, un pointeur vers les données et copie les données dans l'adresse de la mémoire externe.
- **uint8_t ecriture (const uint16_t adresse, uint8_t *donnee, const uint8_t longueur) :** Elle prend en paramètre l'adresse, un pointeur vers les données et la longueur de la chaine.

CLASSE Minuterie

Description du choix : Cette fonction permet d'initialiser et de contrôler un compteur (ici TIMER1).

- **Minuterie () :** C'est un constructeur de la classe minuterie qui initialise les variables muniterieCommencee_ a false et minuterieExpiree a 0.

- **void partirMinuterie (uint16_t duree) :** Cette fonction prend en paramètre un entier sur 16 bits représentant la durée. Durée est comparé avec TCNT1 qui incrémente jusqu'à ce que les valeurs soient les mêmes. Une fois l'égalité atteinte une routine d'interruption se lance.

- **uint8_t getminuterieExpiree () :** C'est une fonction qui retourne la valeur de la minuterieExpiree.
- **bool getminuterieComencee () :** C'est un booléen qui indique si la minuterie a commencé.

- **void setminuterieExpiree () :** C'est une fonction qui permet de mettre la minuterieExpiree a 1.

CLASSE ROUE

Description du choix :

Utilités des fonctions choisies :

- **ROUE () :** C'est un constructeur de la classe roue.

- **void ajustementPWM (uint8_t VitesseA, uint8_t VitesseB) :**

fonction qui permet de faire tourner les roues a des vitesse différente grâce au registre .

CLASSE UART

Description du choix : Nous avons choisi de créer un fichier .h spécifique à la transmission des données. En effet il contient toutes les fonctions utiles pour transmettre une information vers le robot.

Utilités des fonctions choisies :

- **Void initialisationUART ( ) :** C'est une fonction qui initialise les registres de permettant le stockage des données pour leur.
- **Void transmissionUART (uint8_t donnee)** : C'est une fonction qui permet la transmission de la donnée mise en paramètre de l'UART vers le PC.

Partie 2 : Décrire les modifications apportées au Makefile de départ

Nous avons obtenu un Makefile au début de notre projet. Ce dernier a été modifié en deux versions d'une part pour compiler et générer notre bibliothèque et d'autre part pour compiler notre fonction de test du répertoire src. Ainsi, nous avons pu vérifier le bon fonctionnement ainsi que la liaison avec la librairie.

Makefile librairie :

- Nous commencerons par le Makefile de la librairie.
- Tout d'abord nous avons commencé par ajouter nos fichiers sources **.cpp** de la manière suivante dans **PRJSRC = $(wildcard *.cpp .h)**.
- Après cela nous avons supprimé la cible Install contenue dans l'ancien fichier makefile, car nous n'avons pas besoin de cela dans notre situation actuelle de génération de librairie.
- De plus nous avons modifié le nom du projet pour lui donner le nom de la librairie donc dans **PROJECTNAME = library**.
- Puis nous avons ajouté deux nouvelles variables représentants respectivement la commande pour remplacer avr-gcc ainsi que les options à utiliser.
- Nous avons ajouté **AR** qui représentera l'appel à **avr-ar** et donc remplacera avr-gcc et **COMMANDE** qui a pris les options **c**, **r** et **s** pour créer l'archive, insérer les fichiers membres et produire l'index.
- Dans la partie gérant les noms des cibles par défaut pour TRG nous avons remplacé l'extension précédente pour la cible finale qui était .out par notre .a représentant l'extension d'une librairie. Maintenant, nous arrivons dans la partie de l'implémentation de la cible qui est ici notre librairie. Et pour la cible $(TRG), nous avons remplacé l'appel à CC par notre **AR** pour **avr-ar** et le LDFLAGS par nos options dans **COMMANDE** donc **-crs**. Nous avons par la suite enlever -lm libs car nous n'en avons pas besoin lorsqu'on génère notre librairie.
- Enfin, nous avons modifié la cible clean pour qu'elle enlève tous les fichiers intermédiaires, mais qu'elle ne supprime pas la librairie créée.

Makefile Test :

Dans notre Makefile pour le test de la librairie nous avons apporté des modifications par rapport au Makefile original.

- Dans **LIBS**, nous avons ajouté le chemin vers notre librairie selon le dossier actuel dans lequel se trouve notre fichier de test.
- Dans la partie des inclusions additionnelles : INC nous avons ajouté le chemin vers la librairie -I../librairie
- Nous avons ajouté -I library.a -L ../librairie dans la partie **CFLAGS** pour faire l'édition des liens.
- Ensuite dans les dépendances de $(TRG) nous avons ajouté la librairie.