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 113114

Noms:

Jefferson Lam

Nathan Ramsay-Vejlens

David Saikali

Agnès Sam Yue Chi

Date:

9 mars 2020

Partie 1 : Description de la librairie

> *Décrire la librairie construite et formée (définitions, fonctions ou classes, utilité, etc...) pour que cette partie du travail soient bien documentées pour la suite du projet pour le bénéfice de tous les membres de l'équipe.*

La présente librairie *lib_c* sert d'interface de communication pour la carte mère ATmega324PA, et implémente plusieurs fonctions permettant certaines actions utiles et pertinentes lors de son fonctionnement. Cette librairie crée une couche d'abstraction entre les fonctionnalités basiques du robot telle que le contrôle des moteurs, ou bien même l'utilisation des capteurs. Ainsi, à travers cette librairie, le programmeur en question sera en mesure d'utiliser ses fonctions afin de faciliter le contrôle du robot.

Constantes

Le fichier *constantes.h* de la librairie commune définit plusieurs constantes utiles pour le fonctionnement du robot. Dans le but d'éviter la duplication et la redondance de code, ainsi que l'utilisation des chiffres magiques, des constantes reliant aux fonctionnalitées suivantes ont été définies :

- Configuration d'un port,
- Couleur que peut prendre la DEL
- Décalage de bit
- Fonctionnement des capteurs

Ce fichier fait l'inclusion des librairies nécessaire pour la programmation sur avr. En conséquent, l'inclusion du fichier de constantes inclut intrinsèquement les librairies avr.

Initialisation

Pour éviter de manuellement définir la direction des ports A, B, C et D, quatre fonctions ont été définies pour initialiser si ces ports individuels sont en entrée (0x00) ou en sortie (0xFF). Une cinquième fonction permet de définir la direction des quatre ports simultanément. De plus, ce fichier d'en-tête contient aussi les fonctions d'initialisation des autres composantes du systèmes. Par exemple, on inclut aussi l'initialisation de l'UART, et l'initialisation des interruptions.

Contrôle de la DEL (lumière)

La carte mère ATmega324PA possède une DEL libre qui s'avère utile afin de signaler un changement quelconque, ou bien simplement de transmettre un feedback du bon ou mauvais fonctionnement de notre système. Généralement, dans les systèmes réalisés à date, la DEL est de facto connecté au PORTB libre, configuré en tant que sortie. Bien qu'il soit possible de

connecter la DEL à des ports variables, la librairie suppose l'utilisation du port B pour ce qui a trait à la DEL.

Afin de simplifier et d'éviter la duplication de code, les fonctions suivantes contrôle la couleur que prend la DEL. Celle-ci prend trois couleurs possible : rouge, vert, ou ambre. Nous avons aussi défini plusieurs fonctions qui permettent d'accomplir des actions plus complexes tel que le clignotement, ou bien une atténuation de la lumière.

Contrôle des moteurs

Pour contrôler les moteurs, on utilise le timer1 du ATmega324PA pour générer deux signaux PWM sur les broches du port D. De cette façon, il ne sera pas nécessaire d'utiliser le microcontrôleur, puisque les moteurs pourront fonctionner de manière autonome. Ainsi, une première fonction *ajusterPWM()* permet de partir les moteurs avec deux valeurs de comparaison des registres OCR1A et OCR1B pour définir l'intensité du signal. C'est à partir de cette fonction que sont construites les autres fonctions:

- arreterMoteur(): Cette fonction ajuste l'intensité du signal à 0.
- dirigerRoues(uint8_t direction, uint8_t intensite): Cette fonction indique la direction dans laquelle les roues tournent et une intensité pour les deux roues.
- tournerRoues(uint8_t direction, uint8_t intensiteGauche, uint8_t intensiteDroite): Comme dirigerRoues, mais il est possible d'indiquer deux intensités différentes pour chaque roue.

Antirebond

Afin d'utiliser le bouton-poussoir sans utiliser une routine d'interruption, il est nécessaire d'avoir une fonction qui détecte le rebond lors de l'appui du bouton pour éviter que le microcontrôleur ne détecte plusieurs changements de signals. Il suffit de détecter si la tension sur la deuxième broche du PORT D est encore constante après un délai.

Transmission de données à la mémoire et protocoles de communication

Afin de permettre la communication entre l'ordinateur et le robot, nous avons dû intégrer la classe déjà fournie Memoire24CXXX qui implémente plusieurs fonctions qui permet à la fois de faire la lecture et l'écriture de données. Pour le projet final, il se peut très bien que notre robot ait besoin de communiquer avec d'autres systèmes pour bien fonctionner, et donc ce serait utile d'inclure ses fonctionnalités. En plus de cette classe, nous avons aussi implémenté quelques fonctions qui permet d'envoyer des octets à l'ordinateur, ainsi que d'écrire des messages en mémoire vers le PC. Ceci facilite l'utilisation de la classe Memoire24CXXX et étend les fonctionnalités possibles.

Capteurs et conversion analogique/numérique

Certains de nos détecteurs, comme la photorésistance, retournent des valeurs analogiques, alors que le programme n'est capable que de traiter des valeurs numériques. C'est pourquoi un convertisseur analogique-numérique est nécessaire pour traduire ces valeurs. Les fonctions de conversion sont fournies pour le cours.

Délai

Cette fonction permet de faire des délais sur lesquelles on a plus de contrôle. Considérant les limites que la fonction _delay_loop_2 possède par rapport à la taille du paramètre d'entrée, il est plus avantageux de créer une fonction qui prend en paramètre un nombre "n" afin de faire "n" fois un délai de 1 ms.

Partie 2 : Décrire les modifications apportées au Makefile de départ

> *Décrire les quelques modifications apportées au Makefile de la librairie pour démontrer votre compréhension de la formation des fichiers. Faire de même pour les modifications apportées au Makefile du code (bidon) de test qui utilise cette librairie.*

La librairie conçue dans le cadre de ce projet est constitué de plusieurs fonctions accomplissant différentes tâches en lien avec le fonctionnement du robot. Ces fonctions ont d'abord été déclarés dans des fichiers .h et organisé dans des fichiers .cpp selon les fonctionnalités que ces dernières accomplissent. De ce fait, un utilisateur de cette librairie doit simplement inclure le fichier d'en-tête pour avoir accès aux fonctions.

Afin de compiler tous ces différents fichiers .cpp en une librairie statique, il a fallu faire quelques changements au fichier Makefile. D'abord, puisque notre librairie contient plusieurs fichiers .cpp et que nous devons compiler chacune d'entre elle avant de faire l'édition des liens, il a fallu modifier la variable PRJSRC afin d'inclure tous les fichiers .cpp. En utilisant le mot clé wildcard, et en précisant *.cpp, le Makefile sera capable de reconnaître tous les fichiers présents dans le répertoire.

Ensuite, puisque nous voulons produire une librairie statique, il a fallu modifier le type de la cible d'un exécutable à une librairie, soit de *.out* à *.a.* Le type *.a,* à l'inverse du type *.s* qui produit une librairie dynamique, précise au Makefile que c'est une librairie statique qui est produit. De plus, il faut modifier la configuration de l'implémentation de la cible. Au lieu de l'utilisation du compilateur avr-gcc pour produire la librairie, c'est un appel à la commande *ar*, avec les options *r, c, et s* afin d'archiver, créer les fichiers membre, et produire les fichiers index respectivement.

Finalement, nous avons ajouté plusieurs appels à la fonction UNIX *echo* pour vérifier l'état des variables présents dans le Makefile. Il devient maintenant possible de construire la librairie en utilisant la commande make all.

Dans le Makefile du code exécutable, afin d'accéder aux fonctions définis, il a fallu indiquer au Makefile les librairies à lier dans la variable LIBS, et les inclusions additionnelles dans la variable INC, en précisant le chemin au directoire de la librairie. Afin de créer l'objet exécutable, nous retournons à l'utilisation du compilateur avr-gcc.

Le rapport total ne doit par dépasser 7 pages incluant la page couverture.

Barème: vous serez jugé sur:

- *La qualités et le choix de vos portions de code choisies (5 points sur 20)*
- *La qualités de vos modifications aux Makefiles (5 points sur 20)*
- *Le rapport (7 points sur 20)*
  - *Explications cohérentes par rapport au code retenu pour former la librairie (2 points)*
  - *Explications cohérentes par rapport aux Makefiles modifiés (2 points)*
  - *Explications claires avec un bon niveau de détails (2 points)*
  - *Bon français (1 point)*
- *Bonne soumission de l'ensemble du code (compilation sans erreurs, …) et du rapport selon le format demandé (3 points sur 20)*