Département de génie informatique et génie logiciel

Cours INF1900 :

Projet initial de système embarqué

Travaux pratiques 7 et 8

Production de librairie statique et stratégie de débogage

Par l'équipe

No 5471

Noms :

Faneva Rakotoarivony

Lina Belloui

Stefan Cotargasanu

Justine Sauquet

Date :

13 mars 2022

Partie 1 : Description de la librairie

Notre librairie comporte plusieurs éléments définis ci-après :

- **Can** :

La classe can (can.h et can.cpp) nous permettra de convertir des signaux analogiques captés par le robot en signaux numériques. Elle nous permet d'avoir un contrôle sur la traduction de différentes valeurs. Cette classe nous ont directement été fournis et contient une seule méthode : **uint16_t lecture(uint8_t pos)**.

- **La mémoire :**

Les fichiers memoire_24.h et memoire_24.cpp nous ont été fournis au tp5 et nous permettrons l'accès à la mémoire externe I^2C de la carte-mère du robot. La classe Memoire24CXXX permet toute la gestion des opérations reliées à la mémoire.

- **Les moteurs :**

Notre classe **MotorsController** nous permettra de contrôler les moteurs du robot à l'aide de signaux PWM. Les méthodes implémentées sont les suivantes :

setLeftPercentage(uint8_t percentage) et **setRightPercentage(uint8_t percentage)** qui prennent un pourcentage PWM en paramètre afin de le convertir en timer à l'aide de la méthode privée **convertPercentageToTimerValue(percentage).** Cette dernière méthode retournera une valeur correspondant à la vitesse des moteurs (**leftSpeed**_ et **rightSpeed**_). Ensuite, les méthodes privées **adjustLeftMotorSpeed()** et **adjustRightMotorSpeed()** s'occupent de l'ajustement PWM dans le timer en affectant les variables de vitesse droite et gauche aux sorties PWM OCR0A et OCR0B respectivement.

Les méthodes qui nous permettront de changer les directions des moteurs sont **changeLeftDirection()** et **changeRightDirection()**. Elles modifient simplement les entrées au port B qui sera relié aux moteurs.

getLeftPercentage() et **getRightPercentage()** retournent simplement le pourcentage PWM calculé à partir du timer avec la méthode **convertTimerValueToPercentage(uint8_t timerValue).**

- **La transmission :**

Cette librairie permettra la transmission usart afin d'afficher, si nécessaire, du texte sur l'écran. La documentation de ATmega324PA nous a permis d'implémenter les méthodes suivantes :

transmit(uint8_t data) s'occupe de prendre un octet (unsigned char) venant de la carte-mère et l'envoyer au PC par transfert USB. Cet octet représente un caractère qui est lu par le programme serieviaUSB. **receive(void)** reçoit la donnée transmisse et **transmitTextMessage(uint8_t message[], uint8_t messageLength)** prend en paramètre un message ainsi que sa longueur afin de savoir le nombre de fois que la transmission UART (transmit(message[i])) doit être appelée. Enfin, la méthode privée **initialization(void)** s'occupe d'initialiser tous éléments essentiels à la transmission USART (initialisation des premiers registres, format des trames, fonctions...) et est appelé par le constructeur de la classe.

- **Le contrôle des/de la DEL :**

Cette librairie permettra un meilleur contrôle des couleurs des LED de la carte-mère. La classe led contient un constructeur **led(volatile uint8_t* port, uint8_t pin1, uint8_t pin2)** qui prend en paramètre un port (**port**_) ainsi que 2 broches (**pin1**_ **et pin2**_) correspondant aux 2 broches du port de notre choix qui seront connectées à la DEL. Les méthodes utilisées sont **setGreen(), setRed(), setOff()** ainsi que **setAmber()** qui change simplement la valeur du port relié à la DEL dépendamment de la couleur que nous désirons. Des attributs privés constants **GREEN_AMBER_DELAY** et **RED_AMBER_DELAY** ont également été initialisés dans la classe afin de créer des délais entre les couleurs rouge et vert et ainsi permettre la couleur jaunâtre de la DEL.

- **Une fonction interruption :**

Cette librairie permet le contrôle de la configuration des interruptions générées par des boutons, elle n'implémente pas la routine d'interruption ni les variables globales. La classe ButtonInterrupt contient un constructeur **ButtonInterrupt(uint8_t interruptNumber, InterruptMode interruptMode)**. L'interruptNumber est un nombre allant de 0 à 2, correspondant au numéro de l'interruption que l'on souhaite déclencher. Le second paramètre, l'InterruptMode est une enum class, composé des modes suivants :

1- **LowLevelInterrupt** : une interruption est générée lorsque INTn est à 0,
2- **AnyEdgeInterrup** : une interruption est générée lorsque la valeur de INTn change,
3- **FallingEdgeInterrupt**: une interruption est générée lorsque INTn passe à 0,
4- **RisingEdgeInterrupt**: une interruption est générée lorsque INTn passe à 1.

- **Une fonction compteur :**

Cette librairie permet le contrôle de la configuration d'un timer sur le timer1 avec un prescaler prédéfini à 1024, elle n'implémente pas la routine d'interruption du compteur qui est effectuée sur **INT1_COMPA_vect**, ni la variable globale. Le constructeur de cette classe est le suivant **CounterInterrupt()**. Les méthodes implémentées sont les

suivantes, **setDuration(uint16_t duration)**, qui prend en paramètre l'intervalle en secondes entre chaque incrémentation du compteur, valeur qui peut être comprise entre 0 et 8.38 secondes.

La méthode **setGenerationMode(GenerationMode generationMode)** qui prend en paramètre une enum class **GenerationMode** composée des éléments suivants :

1. **Normal** : le timer augmente de 0 à 65535 et génère une interruption lorsqu'il atteint la valeur assignée,
2. **ClearTimerCompare** (CTC) : le timer augmente de 0 à la valeur assignée, génère une interruption et recommence directement à 0.

La méthode **setCompareMode(CompareMode compareMode)** qui prend en paramètre une enum class **CompareMode** composé des éléments suivants :

1. **Toggle**, un mode qui change la valeur de INTn lorsque le compteur atteint la valeur assignée.
2. **LowLevel**, un mode qui déclenche une interruption dont la sortie est 0.
3. **HighLevel**, un mode qui déclenche une interruption dont la sortie est 1.
4. Off, un mode qui désactive la comparaison et donc ne générera aucune interruption.

- **Mécanisme de débogage :**

Ce mécanisme permet de vérifier le bon fonctionnement d'un programme. Il est actif lorsque l'on fait un make debug. Il utilise la communication en RS232 pour afficher la valeur de ce qui est demandé.

Afin d'initialiser la classe debug, il faut utiliser la macro **DEBUG_INIT**. Il est aussi possible de transmettre la valeur de différentes choses, notamment la transmission d'un message avec la macro **DEBUG_PRINT_MESSAGE(uint8_t message[], uint8_t lenght)** qui prend en paramètre le message que l'on souhaite afficher, ainsi que la longueur de ce dernier.

Il est aussi possible de transmettre la comparaison entre une valeur et un certain pin avec la macro **DEBUG_COMPARE_SIGNAL(uint8_t port, uint8_t pin, uint8_t value)**, "value" étant la valeur entre 0 et 1 à laquelle on veut comparer le port en question et transmets un booléen (0 ou 1) à l'ordinateur.

Le dernier objet transmissible est la valeur entière d'une variable au travers de la macro **DEBUG_PRINT_VARIABLE(uint8_t variable)** avec la variable en paramètre.

Partie 2 : Décrire les modifications apportées au Makefile de départ

- **Librairie statique :**

Dans nos laboratoires précédents, nous avons pris l'habitude d'associer un makefile à un fichier .cpp pour la compilation. Dans ce tp, il a fallu ajouter tous nos fichiers .cpp correspondant à toutes nos fonctions et tous nos codes nécessaires à la formation de la librairie (dans la partie **PRJSRC**).

Il a fallu également créer la librairie statique. Pour ce faire, nous avons ajouté la variable **LIBRARY = ar.** L'appel à ar (pour archive) s'occupe plutôt de l'archivage qui créer un fichier de type « .a » qui sera la librairie (**TRG=$(PROJECTNAME).a** sera le nom de la cible de la librairie par default).

Nous avons ajouté un flag (**LIBFLAGS=-crs**). Comme indiqué sur l'énoncé du tp7, l'option « c » servira à créer l'archive, « r » à insérer les fichiers membres et « s » à produire l'index des fichiers objets. La création de la librairie statique se fait en regroupant les affectations faites ci-haut de la manière suivante :

$(TRG): $(OBJDEPS)

$(LIBRARY) $(LIBFLAGS) $(TRG) $(OBJDEPS)

où **TRG** représente notre cible, **OBJDEPS** la liste de tous les fichiers objets (.o), **LIBRARY** la commande archive (ar) et **LIBFLAGS** nos flags (les options –crs).

Nous avons aussi enlevé les commandes superflues, comme celles pour créer les fichiers .elf, .hex et le .o cible, car on a seulement besoin de créer la libraire, pas de fichiers pour l'ATmega.

- **Code:**

Le fichier lib_test.cpp est une manière de tester notre librairie en important les classes à tester. Afin de pouvoir compiler ce petit programme, les modifications aux makefile ont été les suivantes. Dans la partie « Inclusions additionnels », nous avons ajouté le chemin d'accès à notre librairie (**INC=-I ../lib/** et **LIBINC= -L ../lib/**). Comme indiqué dans l'énoncé du tp, l'option « -I » précise l'endroit des fichiers *includes* et l'option -L correspond à l'endroit où se trouve la librairie. **LIBS=-l rene** indique le nom de la librairie (option –l).