Cours INF1900:
Projet initial de système embarqué

Travaux pratiques 7 et 8

Production de librairie statique et stratégie de débogage

Par l'équipe

03-14

Noms:

Massoud Ibrahim
Steven Ho
Nazim Bertouche
Lawali Mahamane Yacine

Date:
31 octobre 2022

Partie 1 : Description de la librairie

Par définition, une librairie est un regroupement de fonctions utilitaires et réutilisables par des programmes[1]. Dans le cadre des TP7 et TP8, nous avons construit une librairie constituée de plusieurs classes et macros jugés comme étant essentiels à la réalisation du projet final. Comme ce dernier n'a pas encore été amorcé, il ne s'agit que d'un regroupement initial : cette librairie pourrait donc être étendue dans les semaines qui suivent et continuer d'évoluer dans le but de répondre aux besoins futurs rencontrés. Ci-dessous sont présentés les classes qui constituent notre librairie ainsi que le fichier debug.h.

Classes

1. Bouton

La classe Bouton nous permet d'initialiser nos boutons et préparer nos interruptions.
Constructeur:
Bouton(uint8_t interup = 0) : permet d'initialiser le masque pour isoler notre interruption. Par défaut nous prenons INT0.

Registres utilisés: (EIMSK, EICRA)

Méthodes publiques:

- Bool verifieEstPresse() : est une méthode qui renvoie qui vérifie si le bouton est poussé ou pas en appliquant un anti-rebonds.
- Uint8_t recupBoutonPoussoir() : permet de savoir quel bouton poussoir nous utilisons utile plus tard pour l'ISR.
- Bool estPresse() : permet de savoir si le bouton est pressé ou pas.

Méthodes privées:

- Void initialisationIntExtrene(volatile uint8_t interup , TYPE_PRESSION pression) : permet les interruptions au niveau du bon bouton poussoir et de la bonne manière front montant, front descendant ou les deux.

2. Can

Cette classe nous permet de contrôler le convertisseur analogique/numérique.

Constructeur:
D'une part, le registre ADMUX (*Analog Digital Converter Multiplexer Selection Register*) est initié avec des valeurs initiales à zéro pour les 8 bits. REFS1 et REFS0 sont initié à 0 (correspondent au voltage de référence pour le convertisseur analogue/numérique). Les bits MUX4:0 servent à sélectionner les combinaisons d'entrées analogique qui sont connectées au convertisseur « ADC ». Ils sont aussi initialisés à 0. Le bit ADLAR affecte la présentation du résultat de la conversion « ADC ». Il est initialisé de façon qu'il ajuste la conversion à droite[2].

D'autre part, les 8 bits du registre ADCSRA (*ADC Status and Control Register A*) sont initiés

[1] Techno-Science.net. (s.d.) Bibliothèque logicielle - Définition et Explications. [En ligne]. Disponible : https://www.techno-science.net/definition/1470.html#:~:text=En%20informatique%2C%20une%20bibliothèque%20ou,sans%20avoir%20%C3%A0%20les%20r%C3%A9%C3%A9crire.

[2] Description technique du ATmega324PA, p.249

de façon que le convertisseur s'active mais sans démarrer une conversion.

Destructeur : La valeur de ADEN (*ADC Enable*) du registre ADCSRA est mise à 0 pour désactiver le convertisseur.

Méthodes:

- lecture : cette méthode prend en paramètre la position (valeur allant de 0 à 7 de type uint8_t) du port où la photorésistance sera relié en entrée. Elle permet de faire une conversion et va retourner un résultat sur 16 bits. Il est important de noter que la valeur retournée par le convertisseur est de 10 bits étant donné que seulement les 10bits de poids faibles sont significatifs. Les 2 derniers bits peuvent être négligés pour réduire le format des données à 8 bits.

3. Del

La classe Del est responsable de contrôler les diodes électroluminescentes (DEL) connectées à des broches de la carte mère en sortie. Elle permet d'allumer la DEL selon une couleur représentée dans l'énumération Couleur : ETEINT, VERT et ROUGE.

Registres utilisés : (PORTx, DDRx)

Constructeur

Del(Couleur couleur = ETEINT, volatile uint8_t* port = &PORTA, uint8_t positionBroche1 = 0, uint8_t positionBroche2 = 1) : permet d'initialiser chacun des attributs du constructeur en plus d'associer via la méthode determinerDdr le port à son registre de direction de données (DDR) respectif et de mettre les broches sélectionnées en sorties.

Par défaut, une instance de la classe Del sera de couleur ETEINT, connectée au PORTA, sur les bits 0 et 1.

Méthodes publiques

- **void activer()** : méthode qui rend la DEL actif. La diode s'allumera alors à la couleur définie dans l'attribut couleur_. Tant que cette méthode n'a pas été appelée, aucune lumière ne peut être émise.

- **void fixerCouleur(Couleur couleur)** : méthode qui permet d'ajuster la couleur de la DEL avant son activation et après son activation en changeant l'attribut couleur_. Si la DEL a été activé précédemment, on pourra observer un changement de la couleur émise par la DEL. Dans l'autre cas, aucun effet sera visible.

- **void desactiver()** : méthode qui rend la DEL inactif. La couleur sera alors ETEINT et l'attribut estActif_, false.

Méthodes privées

- **void changerCouleur()** : méthode responsable d'envoyer ou non du courant dans les broches concernées et donc, d'allumer la DEL à la couleur stockée dans l'attribut couleur_.

- **void determinerDdr()** : méthode utilisée pour détecter le DDR associé au port utilisé. La méthode met ensuite en mode de sortie les broches concernées.

4. Memoire24

Objectif : il s'agit d'une classe qui permet à l'utilisateur de faire la manipulation de la mémoire externe de notre robot grâce aux ports C0 et C1.

Constructeur : Son constructeur Memoire24CXXX() va faire appelle à une méthode d'initialisation ; cette dernière initialise le port série et l'horloge de l'interface I2C.

Méthodes implémentées : les méthodes implémentées sont celles de lecture et d'écriture ;il existe deux types pour chacune des méthodes :

• Lecture :
-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.

• Écriture :
-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.

5. Moteur

Cette classe est responsable de la motricité du robot, c'est-à-dire le contrôle des moteurs par l'usage du Timer 0. Nous utilisons les broches B3 (pour le moteur gauche) et B4 (pour le moteur droit) pour ajuster le PWM (Pulse Width Modulation). L'utilisation de cette classe ne nécessite pas la déclaration d'un ISR.

Constructeur: Pour le constructeur, nous avons mis en entrée les broches B3 et B4 puisqu'elles vont nous permettre de générer un signal PWM sur les broches de comparaison (OC0A et OC0B). Le PWM a été ajusté pour chaque roue à 0 à l'aide de la fonction « ajusterPwm » puisque le robot ne devrait pas bouger par défaut.

Méthodes privées

- **ajusterPwm(rapportCycliqueGauche, rapportCycliqueDroite):** Elle permet de générer un signal PWM sur les broches de comparaison (OC0A et OC0B). Les rapports cycliques peuvent varier entre 0 et 100. Ils nous permettent d'ajuster le pourcentage voulu pour régler la période active de l'onde PWM. La valeur des registres OCR0A et OCR0B peuvent varier entre 0 et 255. Ensuite, nous utilisons le

mode PWM 8 bits, phase correcte avec une valeur de TOP fixe étant 0xFF (étant le Mode 1 du "*Waveform Generation Mode*"[3]). C'est pourquoi la section WGM du registre TCCR0A est affecté. La section COM0A1 et COM0B1 de ce registre sont affecté aussi. La section CS11 du registre TCCR0B sera affecté pour faire la division de l'horloge par 8.

Méthodes publiques

Toutes les méthodes dépendent de la fonction "ajusterPwm".

- **void accelerer(DirectionVerticale direction, uint8_t rapportCyclique):** Cette méthode permet de faire tourner les 2 roues du robot vers l'avant ou l'arrière avec la même vitesse. Le paramètre direction peut être soit AVANT ou ARRIERE.

- **void virer(uint8_t rapportCycliqueGauche, uint8_t rapportCycliqueDroit)**: le but est de pouvoir ajuster le PWM de chacune des roues séparemment afin de faire en sorte qu'une roue tourne plus rapidement que l'autre pour effectuer un virage dans la direction désirée. Il faut utiliser cette méthode avec la méthode reculer ou avancer.

- **void virer90(DirectionHorizontable direction):** cette méthode diffère de la précédente étant donné qu'elle permettra au robot de faire une rotation sur place de 90 degrés dans la direction choisie (GAUCHE ou DROITE). En ajustant le PWM, nous avons déterminé qu'un pourcentage de 75% pour le niveau haut de l'onde PWM ainsi que d'ajouter un délai de 800 ms (DELAI_VIRAGE_90) nous a permis de faire une rotation de 90°. C'est donc une fonction bloquante qui peut prendre au minimum 800 ms pour s'exécuter.

- **void arreter():** la méthode "arreter" va permettre d'ajuster le PWM à 0% pour chacune des roues afin de mettre le robot en arrêt.

6. Timer1

La classe Timer1 est une classe qui permet de partir un compteur et provoquer une interruption lorsque le registre TCNT est égal à OCR1A. Il doit être utilisée avec une fonction ISR.

Méthodes publiques

void start() : démarre le compteur.

void setMode(Mode mode) : permet de changer le mode du Timer1, soit le mode Normal ou CTC.

void setPrescaler(uint16_t prescaler) : permet de changer l'attribut prescaler_, c'est-à-dire le diviseur appliqué sur l'horloge du Timer1, qui sera utilisé par la méthode configuratePrescaler.

void setOcrValue() : permet de changer l'attribut OcrValue, la valeur du registre OCR1A.

Méthodes privées

[3] Mode 1 de la table 16-5 de la description technique du ATmega324PA, p.130

void configurateMode() : configure le Timer1 pour qu'il soit en mode normal ou CTC.

void configuratePrescaler() : configure le prescaler du Timer1 selon l'attribut prescaler_.

void enableInterrupt() : active les interruptions sur la chaîne A du Timer1.

7. Uart

Méthodes implémentées :

initialiserUart() : cette méthode va permettre d'initialiser le registre UCSR0B en affectant la section TXEN0 pour permettre la transmission de données par le UART0. Les registres UBRR0H/L sont aussi affectés afin que la communication se fait par 2400 bauds et sans parité. Les sections UCSZ02 combiné avec UCSZ01:0 du registre UCSR0C sont affecté pour que les trames de 8 bits sont séparées par un bit d'arrêt.

transmettreViaUart(char caractere): cette méthode va permettre de transmettre un caractère (un octet de la carte) vers le PC. C'est une fonction bloquante puisque le programme bloquera jusqu'à ce que les octets lui parviennent.

transmettreViaUart(const char* chaine): cette méthode va permettre de transmettre une chaine de caractères vers le PC.

8. Fichier debug.h

Le fichier debug.h contient un ensemble de directives de préprocesseur qui permettent d'envoyer des données au PC à des fins de débogage, notamment la valeur d'une variable. Il dépend de la règle «debug» du Makefile de l'exécutable.

Voici la liste des macros définis :

```
#ifdef DEBUG
        #define INITIALISER_UART() Uart uartDebug;\
                                   uartDebug.initialiserUart();
        #define DEBUG_TRANSMETTRE(x) uartDebug.transmettreViaUart(x);
#else
        #define INITIALISER_UART() do {} while (0);
        #define DEBUG_TRANSMETTRE(x) do {} while (0);
#endif
```

Lorsqu'on exécute la commande «make debug» sur la ligne de commande,
le macro DEBUG est défini (voir la partie 2), de sorte que la condition de la première directive de préprocesseur #ifdef DEBUG est satisfaite. Les macros INITIALISER_UART et DEBUG_TRANSMETTRE(x) sont alors définis et sont remplacés par du code qui permet l'initialisation de l'UART et la transmission de données par celle-ci à l'ordinateur.

Dans le cas où l'on exécute la commande «make» simplement, aucun macro DEBUG n'est défini. Par conséquent, les macros autres macros n'exécuteront que du code mort.

Partie 2 : Décrire les modifications apportées au Makefile de départ

- ***Makefile de la librairie :***
  - ***PROJECTNAME= libstatique***
    Commande qui permet de nommer la librairie
  - ***PRGSRC= $(wildcard *.cpp)***
    Cette commande permet de récuperer tous les fichiers .cpp du repertoire
  - **AR=ar**
    Cette commande permet d'ajouter une librairie statique
  - ***TRG=$(PROJECTNAME).a***
    Commande qui spécifie l'extension du fichier libstatique, dont libstatique.a est la cible par defaut
  - ***$(TRG) : $(OBJDEPS)***
    $(AR) -crs $(TRG) $(OBJDEPS)
    Commande qui fait l'implémentation de la cible ar avec son flag -crs

- ***Makefile de l'executable:***
  - **PROJECTNAME= MakeFileExec**

  Commande permettant de nommer la librairie
  - ***PRJSRC= main.cpp***
    Commande pour récuperer le main.cpp dans le repertoire
  - **INC= -I .. /lib_dir**

  Commande qui spécifie le chemin des inclusions additionnelles
  - ***LIBS= -L ../lib_dir/-llibstatique***

  Commande qui indique le chemin vers la librairie à lier
  - ***debug:***
    $(CC) $(INC) -DDEBUG -c main.cpp

  Commande qui permet de définir par ligne de commande un macro DEBUG dans le fichier main.cpp en y ajoutant la ligne #define DEBUG (option -D) et de compiler ce fichier avec toutes ses inclusions en fichier .o (option -c).