POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

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

0607

Noms :

Baba Farid
Bouchama Zekaria
Boukhezar Sid-Ali
Moualdi Sofiane

Date :
31-10-2022

Partie 1 : Description de la librairie

Notre création de bibliothèque statique comporte plusieurs classes. Grâce aux méthodes, constructeurs et propriétés, les classes créées implémentent efficacement les morceaux de code ; ces mêmes codes qui seront utiles pour le projet final. Dans ce sens on retrouve dans la librairie créée les classes suivantes :

Classe Del :

Dans cette classe, on retrouve un constructeur de classe Del() et un destructeur ~Del(). En effet, on initialise dans le fichier Del.h 4 fonctions : « rouge() », « vert() », « eteint() » et « ambre() » qu'on va définir dans le ficher Del.cpp :

-void Del :: rouge(volatile uinit8_t* port) : dont on va forcer la valeur du premier bit à 0(moins significatif) et le deuxième bit à 1 ;
-void Del :: vert(volatile uinit8_t* port) : dont on va forcer la valeur du premier bit à 1(moins significatif) et le deuxième bit à 0 ;
-void Del :: eteint(volatile uinit8_t* port) : dont on va forcer les 2 bits à 0 ;
-void Del :: ambre(volatile uinit8_t* port) : dont on va alterner durant un lapse de temps très court la couleur rouge et vert car en effet la couleur ambre est un mélange des 2 couleurs mais dont l'intensité des 2 couleurs est différente.

Dans ce sens, chaque fonction réfère à un état de couleur que prend la Del. Pour spécifier les ports auxquels on va lier la Del, on va faire un passage par un pointeur dans le paramètre de la fonction couleur, qui va ensuite référer vers le port choisit.

Classe Minuterie :

Dans cette classe, on retrouve un constructeur de classe Minuterie() et un destructeur ~Minuterie(). En effet, on initialise dans le fichier Minuterie.h 3 fonctions « initMinuterie() », « finMinuterie() et « set_MinuterieExpiree » qu'on va définir dans le ficher Minuterie.cpp et un attribut privé dit « MinuterieExpiree_ » avec le mot clé volatile pour éviter qu'elle soit optimisée par le compilateur.

On configure dans le fichier cpp notre classe Minuterie avec sa méthode initMinuterie qui prend durée (valeur) comme paramètre qui sera assigné au registre OCR1A pour indiquer la durée à laquelle on déclenchera l'interruption. De plus on a configuré le registre TCCR1B de façon a ce que la minuterie travaille avec le mode CTC et une division d'horloge à 1024. Quant au registre TIMSK1, nous l'avons activé pour qu'il puisse comparer le ORC1A avec le TCNT1 afin de déclencher une interruption une fois que les valeurs sont égales On a initialisé la variable MinuteireExpriree_ qu'on peut modifier dans la fonction set_MinuteireExpriree à 0. On a aussi initialisé les registres TCNT1, TCCR1A, TCCR1C à 0.

Classe Interrupteur :

Dans cette classe, on retrouve un constructeur de classe Interrupteur() dans lequel on initialise un port et un destructeur ~Interrputeur(). En effet, on initialise dans le fichier Interrupteur.h 2 fonctions : « intialisationInterrupteur() » et « antiRebound() » qu'on va définir dans le ficher Interrputeur.cpp :

Concernant la fonction « antiRebound », il s'agit d'un booléen. En effet, on va vérifier en premier lieu si un PINx est activé, on laisse ensuite un bref délai et refaire la même vérification et selon le return de cette vérification, on vérifie si un bouton est appuyé ou non.

Concernant la fonction « intialisationInterrupteur » il s'agit de redéfinir la fonction d'un bouton poussoir avec la méthode d'interruption. Dans ce sens, on initialise les registres « EIMSK », « EICRA » qu'on ajuste selon nos besoins. On devra bien entendu bloquer les autres interruptions qui peuvent survenir au début avec la fonction « cli() » et enlever cette condition à la fin avec la fonction « sei() ».

Classe Moteur :

Cette classe s'occupe de générer un signal PWM. On y trouve dans le fichier Moteur.h un constructeur Moteur() et un destructeur ~Moteur(). En effet, on initialise 3 fonctions : « ajustementPwm() », « avancerMoteur() » et « reculerMoteur() »qu'on va définir dans le ficher Moteur.cpp :

-void Moteur :: ajustementPwm(uinit16_t duree1, uinit16_t duree2) : Dans lequel on va fixer dans les registres OCR1A pour indiquer la durée à laquelle on déclenchera l'interruption. De plus on a configuré le registre TCCR1A et TCCR1B.
-void Moteur :: avancerMoteur(uinit16_t duree1, uinit16_t duree2) : qui prendra en paramètre la fonction « ajustementPwm » dont on va mettre en paramètre les durées correspondant aux valeurs entre 0 et 255.

-void Moteur :: reculerMoteur(uinit16_t duree1, uinit16_t duree2) :

Classe Photoresistance :

Dans cette classe, on retrouve un constructeur de classe Photoresistance() et un destructeur ~Photoresistance(). En effet, on initialise dans le fichier photoresistance.h 3 fonctions « initialisation() », « chargerEntree() et « affichage() » qu'on va définir dans le ficher Minuterie.cpp et un attribut privé dit « lumiereNum_ » avec le mot clé volatile pour éviter qu'elle soit optimisée par le compilateur. Pour la fonction initialisation(), nous avons seulement dirigé le port A en entrée et le port B en sortie. En ce qui s'agit de la fonction chargerEntree(), elle permet de convertir la lumière captée en une valeur binaire sur 10 bits, ensuite elle fait un décalage de 2

à droite pour avoir une valeur sur 8 bits stockée dans la variable lumiereNum. En dernier, la fonction affichage() permet d'assigner la valeur de la variable lumiereNum à un type d'allumage particulier de la Del avec la structure de contrôle Switch-case.

Classe Can :

Ce code nous a été fourni par le professeur dans lequel il a initié dans le fichier can.h le constructeur, le destructeur ainsi qu'une fonction lecture de type uint16_t. tandis que dans le fichier can.cpp, il a défini le constructeur, dans lequel il a initialisé les paramètres nécessaires afin de permettre l'accès au convertisseur du microcontrôleur, le destructeur ainsi que la méthode lecture() qui prend en paramètre un port qui représente la position numérique de la valeur analogique du port correspondant.

Classe Uart :

Cette classe permet la transmission des données par le protocole Uart.

Ce code nous a été fourni par le professeur dans lequel il a initialisé la classe Uart dans le fichier Uart.h. ensuite il déclaré le constructeur, le destructeur ainsi que les deux fonctions qui ne retournent rien, initialisationUART() et transmissionUART() qui prend uint8_t en paramètre. Tandis que dans le fichier Uart, il a défini la méthode initialisationUART() qui configure les registres nécessaires pour permettre la réception et la transmission de données. Il a aussi défini la méthode transmission UART() qui prend en paramètre la donnée à transmettre. cette méthode permet le transfert de données un caractère à la fois au RS232.

Classe mémoire_24 :

Le fichier « mémoire_24.h » possède quatre méthodes permettant d'interagir avec le Usart. Le constructeur de cette classe initialise le porte série et l'horloge de l'interface I2C et ajuste la taille.

On a :

1. uint8_t lecture(const uint16_t adresse, uint8_t donnee) : Lecture de donnée de chaque caractère. En paramètre, on a l'adresse et un pointeur vers les données

2. uint8_t lecture(const uint16_t adresse, uint8_tdonnee, const uint8_t longueur) : Lecture de donnée par bloc. En paramètre, on a l'adresse , un pointeur vers les données et la longueur de la chaine.

3. uint8_t ecriture(const uint16_t adresse, const uint8_t donnee) : Écriture de donnée un caractère à la fois. En paramètre, on a l'adresse et un pointeur vers les données.

4. uint8_t ecriture(const uint16_t adresse, uint8_t *donnee, const uint8_t longueur) : Écriture de donnée par bloc. En paramètre, on a l'adresse, un pointeur vers les données et la longueur de la chaine.

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.

Pour le Makefile de la librairie (celui qui génère le .a), nous avons repris le Makefile des anciens travaux pratiques. À partir de ce Makefile, nous avons commencé par modifier la source PRJSRC et nous l'avons mis à $(wildcard *.cpp) à **la ligne 32** pour que lors de l'exécution, le compilateur compile tous les fichiers .cpp contenus dans la librairie.

Ligne 92 : Ensuite, nous avons modifié l'extension du target TRG = $(PROJECTNAME).a. En effet .a permet de créer la librairie puisque la librairie n'utilise pas de .elf.

Ligne 130 : Pour l'implémentation de la cible, le target TRG, nous avons modifié son implémentation afin qu'il génère un .a en utilisant la commande avr-ar crs $(TRG) $ (OBJDEPS) qui va créer le .a en utilisant comme dépendance les .o donnés en paramètres.

Ligne 127 : Nous avons aussi enlevé $(HEXROMTRG) du all, pour que lorsqu'on fait make all, aucun .hex n'est généré puisque l'on veut uniquement que .a et .o et .d pour les fichiers CPP soient générés. On a donc uniquement laissé le target définit plus haut.

Concernant le Makefile du code, nous avons aussi démarrer avec le Makefile des anciens travaux pratiques.

Dans les inclusions additionnels INC de la **ligne 35**, nous avons ajouté le chemin d'accès vers notre répertoire où se trouve la librairie -I .. /lib.

Ensuite, à **la ligne 38** nous avons mentionné le nom de notre libraire dans LIBS afin que le Makefile du code puisse accéder au .a de librairie et qu'ils soient liés pour que le Makefile

du code puisse générer l'exécutable : -L .. /lib -lmylib. Enfin dans les **lignes 129-131** nous avons ajusté 1 fonction qui servira au débogage. Celle-ci est composée de trois commandes : CXXFLAGS += -DDEBUGS -g ; CCFLAGS += DDEBUGS -g ; all.

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

Barème: vous serez jugé sur:

- *La qualité et le choix de vos portions de code choisies (5 points sur 20)*
- *La qualité 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)*