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 211212

Noms:

Hubert Boucher

Alexandre Turcotte

David Amoussa

Hamza Karoui

Date:

17 mars 2021

Le rapport total ne doit pas 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)

Partie 1 : Description de la librairie

Pour notre librairie, nous avons choisi d'inclure ces notions: les boutons-poussoirs, le contrôle de del, l'utilisation d'un PMW avec une minuterie, l'accès à la mémoire d'une EEPROM et le convertisseur analogique/numérique.

Nous les avons choisies, car ces notions importantes représentent la majorité de ce que nous avons appris jusqu'à maintenant dans le cours. De plus, ce sont des fonctionnalités générales qui peuvent être utilisées antérieurement. Certaines notions demandent des recherches et de la lecture de librairies et de documentations. On peut penser à la minuterie et à l'accès de la mémoire EEPROM. Ainsi, le fait de mettre une fonction pour le PWM et une autre pour la lecture et l'écriture d' une mémoire EEPROM dans notre librairie raccourcit notre temps de recherche dans les documentations. Pour les boutons-poussoirs et les del, nous avons décidé de les mettre au cas où on aurait besoin de différents boutons pour contrôler des lumières différentes. Cela facilite encore une fois notre écriture du code.

Pour continuer, au niveau de la structure des fichiers, nous avons décidé d'utiliser des classes et des fonctions pour chaque concept. Ainsi, notre librairie est composée de fichiers «.cpp» qui correspondent aux fonctions principales et de fichiers «.hpp» qui nous servent à la définition des classes de nos éléments.

Nous avons créé un fichier global.hpp qui va être inclus dans tous les fichiers «.hpp», afin de centraliser tous les «#include» à des librairies extérieures dans un seul fichier «.hpp». Ce fichier nous permet aussi de déclarer des «enum» qui seront utiles pour les fichiers «.hpp». C'est aussi dans ce fichier que nous avons défini la fonction _delay_ms() et la fréquence du microcontrôleur.

Pour les boutons-poussoirs, nous avons créé des fonctions dans le fichier «.cpp» qui permettraient d'utiliser n'importe quel port du microcontrôleur. De plus, nous avons inclus la notion de l'anti-rebond afin de mieux savoir à quel état se trouve le bouton-poussoir. Les fonctions ont été définies à l'aide de «Switch/case» et de boucle «If/else». Dans le fichier «.hpp», nous avons défini les états de la classe du bouton (unpushed, pulse et pending) à l'aide de «Enum».

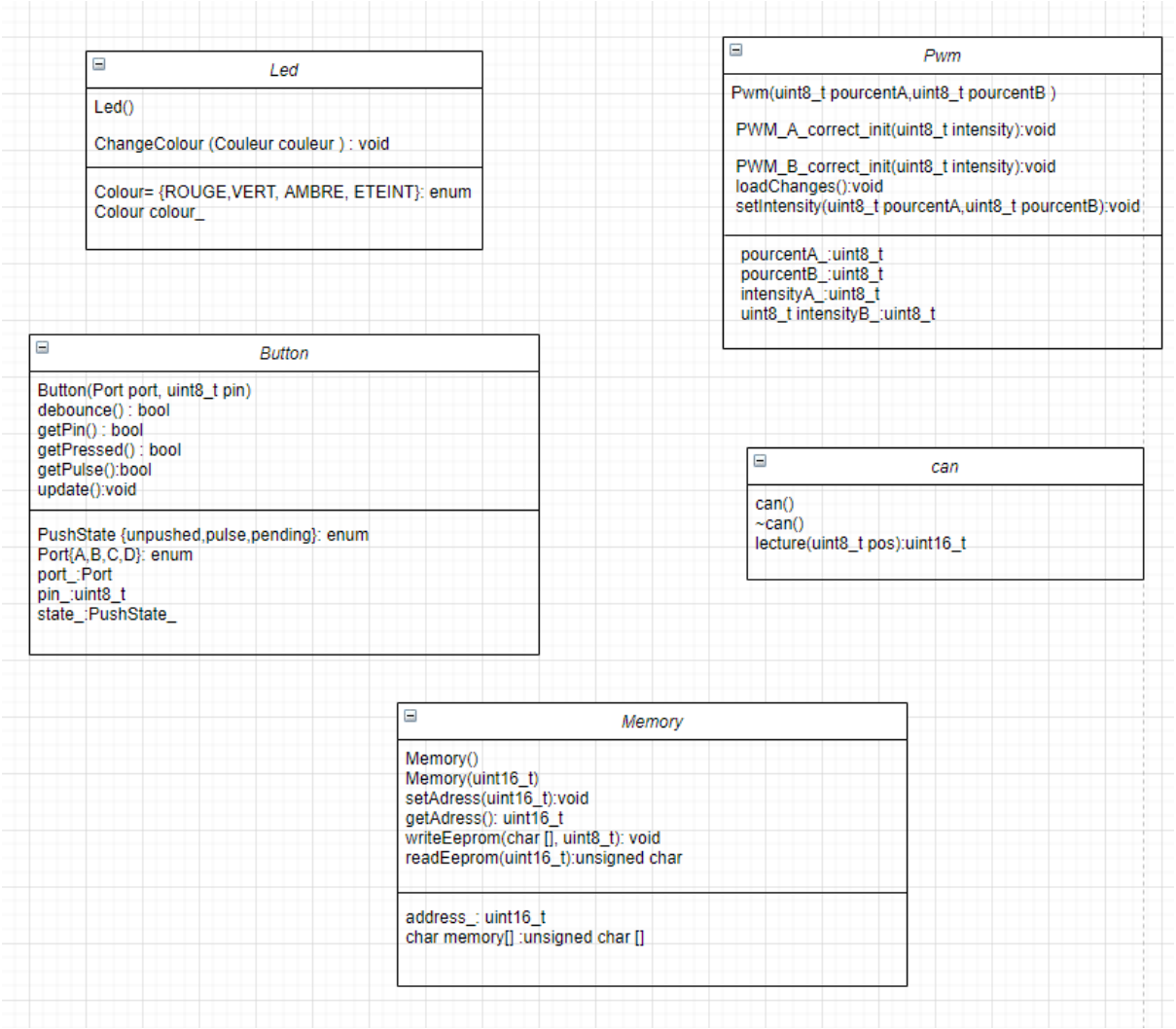
Pour le contrôle de del, dans le fichier d'en-tête, nous avons définis les états de la del avec un «Enum» et nous avons créé aussi dans la classe de la del, une méthode de changement de couleur. Ainsi, dans la fonction pour l'implémentation, le but de la del est de changer de couleur.

Pour continuer, pour l'utilisation du PMW, nous avons créé une classe dans le fichier d'en-tête qui pourrait contrôler 2 moteurs simultanément avec des intensités différentes. Dans le fichier contenant les fonctions principales, nous avons ajusté les registres pour que la minuterie puisse être fonctionnelle. Ainsi, les moteurs peuvent fonctionner sans assistance.

Au niveau de l'accès à la mémoire d'une EEPROM, dans notre fichier «.hpp», nous avons déclaré des méthodes qui permettraient de se positionner à une adresse, de

retrouver l'adresse à laquelle on se situe, d'écrire et de lire dans la mémoire ainsi qu'une méthode pour comparer si deux chaînes de caractère sont pareilles. Dans le fichier principal des fonctions, nous avons défini les méthodes grâce à la librairie AVRLibC.

Pour l'utilisation du convertisseur analogique/numérique, nous avons une classe composé de can.hpp et can.cpp qui permet une lecture des données et retourne le résultat sur 16 bits. Ces fichiers contenant du code ont été fournis par l'enseignant.



Nous avons aussi décidé de laisser la responsabilité des initialisations des timers, tes entrées/sorties des port et des autres initialisations à l'exécutable plutôt qu'à la librairie. Ainsi, les concepteurs qui utiliseront notre librairies auront plus facilement accès aux paramètres du microcontrôleur et pourrons définir des interruptions et autres fonctionnalités de manière plus libre.

Pourquoi choisir de faire des classes?

Nous avons préféré l'approche orientée objet parce que ça favorise l'encapsulation de notre code sous forme de classes. Ainsi, cela rend plus lisible et plus compréhensible ce dernier, en plus du fait que cette approche est plus flexible. C'est-à-dire, si on a besoin de rajouter du code qui dérive de nos classes déjà créées, il suffit d'utiliser de l'héritage ou du polymorphisme pour résoudre le problème. Donc, si on veut générer une librairie facilement extensible et lisible (bien encapsulé), il vaut mieux opter pour l'orienté objet.

De plus, certains algorithmes demandent d'utiliser des valeurs globales, par exemple pour initier le PWM ou l'accès à la mémoire. Comme les variables globales sont à éviter dans une librairie, nous avons opté pour la création de classes afin de pouvoir y stocker les valeurs initialisées une seule fois, sur création de l'objet.

Partie 2 : Décrire les modifications apportées au Makefile de départ

Tout d'abord, nous avons décidé d'enlever l'option de «install» puisque nous n'aurons pas à installer l'exécutable sur un microcontrôleur physique. Ensuite, nous avons compris que la commande «all» n'était pas utile pour l'utilisation du Makefile de la librairie que nous faisions.

Pour ce qui est du Makefile dans le dossier lib, nous avons enlevé tout ce qui était en lien avec la création de fichier ayant pour extension "elf" et "hex", car nous n'avons ni à communiquer avec le microcontrôleur, ni à créer d'exécutable. De ce fait, nous avons aussi pu supprimer les variables «AVRDUDE» et «HEXFORMAT». Pour la variable «PRJSRC», nous lui avons attribué la valeur $(wildcard *.cpp) pour qu'elle inclut tous nos fichier .cpp. Nous avons aussi attribué le nom de «lib211212» à la variable «PROJECTNAME». Suite à ces retraits, nous avons jugé utile d'écrire une variable «AR», qui prend la valeur de la commande «avr-ar -crs». Notre équipe a aussi décidé qu'une seule cible, nommée «LIB», serait utilisée. Cette cible prend comme valeur «$(PROJECTNAME).a». Finalement, notre équipe a modifié l'implémentation de la cible pour avoir une libraire fonctionnelle.

Par contre, pour le Makefile du dossier de projet (exec), la commande «all» était essentielle pour la création du fichier .hex, afin de communiquer notre exécutable à la micropuce. Les seules modifications que nous avons apportées à la version d'origine a été de donner les valeurs aux variables «INC» et «LIBS».