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 121135

Noms:

Justin Boulet
Félix Perreault
Ralph Alaile
Ireina Hedad

Date:
13 mars 2023

Partie 1 : Description de la librairie

Notre première classe est pour la LED. Nous avons 4 méthodes qui permettent d'allumer chacune une couleur différente parmi rouge, vert, ambre et l'éteindre. Ces trois fonctions ne prennent donc rien en paramètre et ne font qu'activer les ports adéquats sauf pour la couleur ambre qui fait alterner dans une boucle infinie les couleurs verte et rouge de la LED avec un délai après chaque couleur. Pour mettre une certaine durée, nous avons deux autres fonctions seulement présente pour la couleur verte et rouge, prenant en paramêtre une duré et permettant d'allumer les LED un certain temps. LEs fonctions pour diminiuer "dimRedLight" et "dimGrennLight" commencent par initialiser un delay à 1 , un compteur timeOn intialisé à 250 qui débute au maximum et décrémente de 1 à chaque 12 boucles dans la fonctionpuis un compteur timeOff à 0 qui incrémente de 1 à chaque 12 itérations. Les 12 boucles sont comptés dans la constante DelayRatio et tant que cette variable n'est pas 12, on rentre dans une boucle qui dépendement de si l'on veut l'allumer ou l'éteindre progressivement, va faire ceci:

```
this->turnRed();
 for(int j = 0; j < timeOn; j++)
  { _delay_us(4);}
 this->turnOff();
```

Lorsque 12 itérations sont faites, on décrémente timeOn ou incrémente timeOff puis remet le compteur delayRatio à 1. Il y a deux autres fonctions qui permettent de faire clignoter les LED rouge et verte, prenant en paramètre une durée ainsi qu'une fréquence en Hertz (1/s). Ces deux méthodes alterne entre la couleur et l'éteindre dans une boucle qui s'arrête lorsque la durée multipliée au taux par seconde est atteinte. Ensuite, nous avons deux autres méthodes qui permettent de diminuer l'intensité de la LED pour les couleurs rouges et vertes.La couleur ambre ne peut diminuer en intensité ni clignoter, nous trouvions l'implémentation de ces méthodes non nécessaires et rendant la classe plus simple. On trouvait plus simple de garder qu'en sortie les mêmes pins, nous avons choisis les pins A0 et A1 et les avons initialisés avec le constructeur de la classe. Une dernière fonction utile à été ajouté puisque nous ne pouvons jamais appellé delay_ms avec une variable pour un argument alors cette fonction se charge d'appeller nos délais dans nos fonctions avec une l'option de choisir la durée puisqu'elle est prise en paramètre.

Notre deuxième classe est la routine d'interruption nommé Interrupt, elle prend dans son constructeur ceci:

```
{cli();
 DDRD &= ~(1 << DDD2);
 EIMSK |= (1 << INT0);
   sei();}
```

Cli() est une routine qui bloque toutes les interruptions tandis que sei () permet de recevoir à nouveau des interruptions. DDRD est notre sortie qui sera toujours la meme lorsque l'on utilise cette classe pour qu'on ne permet pas de paramètre au constructeur. Finalement, EIMSK permet les interruptions externes sur le INT0 qui correspond au port D2. Cette classe sera utilisée avec la routine ISR (int0_vect) seulement et donc seulement avec l'interrupteur sur le port D2. Pour l'utiliser on devra déclarer la routine d'interruption

puis dans le main, utiliser une instance de la classe et y joindre la méthode choisissant le mode voulu.
Elle a 3 méthodes qui permettront de choisir parmi les différents modes d'interruptions parmi le front montant, le front descendant ou peu importe. Ceci sera implémenté en modifiant simplement les registres EICRA selon ce que l'on souhaite et en ayant toujours les routine cli() et sei() pour chaque fonction.
Nous avons 3 autres méthodes qui vont nous permettre de vérifier si un bouton est pressé ainsi que deux fonctions qui permettent un délai pendant que le bouton est pressé et lorsqu'il est relâché. Ces trois méthodes nous seront utile lorsque l'on voudra faire différents programmes ou les changements à l'interrupteurs devront être détectés. Celle-ci comprennent un antirebond (delai_ms) et pour les deux fonctions waitButtonRelease() et WaitButtonPressed(), il y a une boucle (while) infini puisque l'on en sort lorsque le bouton est relaché ou actionné.

Notre troisième classe nous a déjà été offerte, nous l'avons simplement mis dans notre fichier de Librairie. La classe Memoire24 permet l'accès à la mémoire EEPROM, donc l'accès à l'écriture et la lecture. "L'écriture peut se faire de deux façons. Un octet seul peut être écrit en précisant sa valeur et son adresse. Une autre possibilité est d'écrire un bloc d'octets (n'excédant pas 127) d'un seul coup en commençant à l'adresse spécifiée. La lecture se fait d'une manière similaire. On peut lire un octet à la fois ou en passant un vecteur de longueur précise pour lire un bloc continu."[1] La classe contient deux donnés membres : const uint8_t PAGE_SIZE et static uint8_t m_adresse_peripherique qui sont utilisés dans certaines méthodes puis une autre méthode privée ecrire_page. Il y a bien sûr les 4 méthodes de lecture et écriture publiques et même une méthode choisir_banc, prenant en paramètre un const uint8_t permettant de choisir un ban de mémoire.

Nous avons créé une nouvelle classe RS232 pour pouvoir relier ce qui est lu et écrit au PC (cavalier en DbgEn) . La méthode USART_transmit envoie les valeurs à notre carte jusqu'au PC et la méthode USART_received permet le contraire, soit la transmission de données du PC vers la carte mère ce qui est permit par nos registres UCSR0A et UCSR0B. Le constructeur s'implémente ainsi:

```
RS232::RS232(){
UBRR0H = 0;
 UBRR0L = 0xCF;
 UCSR0A = 0 ;
 UCSR0B = (1<< RXEN0)| (1<< TXEN0) ;
UCSR0C = (1<< UCSZ00)|(1<<UCSZ01) ;
}
```

Les registres UBRR0H et UBRR0L sont mit aux valeurs correspondantes à 2400 bauds. RXENn0 et TXEn0 sevrvent à activer ou désactiver la transmission et la reception des bits du registre USCR0B. Le format voulue est décider grâce aux bits UCSZ00 et UCSZ01 du registre UCSR0C (8 bits, 1 stop bit, sans parité). Le constructeur ne prenant rien en paramètre, il n'est pas vraiment possible de modifier l'initialisation, ce qui nous ne cause pas vraiment de problème.

[1] INF1900, Projet initial de système embarqué Travail pratique No. 5 Mémoires et protocoles de communication, https://cours.polymtl.ca/inf1900/tp/tp5/

Nous avons seulement deux méthodes qui permettent la réception et l'envoie des messages. Pour la transmission des donnés, notre méthode USART_Transmit() prend en parametre un caractère (unsigned char data). Il y a donc transmission d'un seul bit à la fois et la boucle permet d'attendre jusqu'à ce que le tampon de transmission soit vide en regardant que le bit UDRE0 du registre USCR0A. Lorsque vide, on ecrit les donnés à transmettre dans le registre UDR0. La deuxieme méthode fait sensiblement le contraire puisque'elle sert à la reception des donnés (encore une fois bit par bit). Elle attend que les donnés soient présente dans le tampon de reception en vérifiant RXC0 puis list les donnés et les retournent.

La classe CAN nous a également été donnée, celle-ci nous permettait d'utiliser un convertisseur analogique numérique. Elle n'avait qu'une seule méthode permettant la lecture puis un constructeur qui initialise les bits de ADMUX et ADCSRA. Des explications sur la façon de l'utiliser avec le matériel de laboratoire se retrouve sur le site du cours : https://cours.polymtl.ca/inf1900/tp/tp6/.

Une autre classe que nous avons ajouté à notre librairie est la classe Motor afin de contrôler la motricité du robot. Celle-ci permet de créer un signal PWM à l'aide de la minuterie interne timer0 de la carte ATMega324PA. À l'initialisation d'un objet de la classe Motor, les ports PB3 et PB4 sont placés en sortie. Pour l'instant, cette classe comporte une seule méthode nommée motorForward() qui prend deux entiers non signés de 8 bits (uint8_t) afin de choisir la vitesse des roues. Cette méthode configure le timer0 en mode PWM de 8 bits en phase correct. Voici les registres modifiés:

```
TCCR0A |= ((1 << COM0A1) | (1 << COM0B1) | (1 << WGM00));
TCCR0B |= (CS01);
```

De plus, nous avons ajouté la classe Timer qui permet aux utilisateurs d'activer une minuterie en plusieurs modes différents. Cette minuterie utilise le timer1 interne de 16 bits à la carte ATMega324PA. Les modes que la classe fournissent sont le mode CTC et le mode PWM phase correct à 8 bits dans deux méthodes différentes.
Pour le mode CTC, voici les registres modifiés:

```
TCCR1A = 0;
TCCR1B |= ((1 << ICES1) | (1 << WGM12) | (1 << CS12) | (1 << CS10));
TCCR1C = 0;
TIMSK1 = ((1 << ICIE1) | (1 << OCIE1A));
```

Les registres TCCR1A, B et C sont les registres des timers. ICES1 utilisé dans TCCR1B est un bit activant la capture d'interruption sur un front montant. WGM12 indique l'utilisation du mode de comptage CTC. CS12 et CS10, encore des bits de TCCR1B, sont les bits configurant le prescaler pour le timer 1 à une fréquence de 1024. TIMSK1 active grace aux bits choisis les interruptions de capture (ICIE1) et de comparaison (OCIE1A).
Pour le mode PWM phase correct 8 bits:

```
TCCR1A |= ((1 << COM1A1) | (1 << COM1B1) | (1 << WGM10));
TCCR1B |= (1 << CS11);
TCCR1C = 0;
```

Le constructeur de cette classe est vide, l'initialisation n'a pas a être fait, les bits des registres adéquats sont choisis dans les deux méthodes précédentes.

Finalement, la librairie comporte un système de débogage qui permette d'installer des opérations DEBUG_PRINT((string)) dans le code. Dans le cas contraire, ils sont ignorés lors de la compilation. La commande DEBUG_PRINT utilise la transmission par RS232 du robot.
En créant un debug (make Debug), on cree une instance de DEBUG, si elle existe, elle sera afficher grâce à la transmission par RS232. On pourra lire ce qui est envoyé par la fonction en faisant la commande serieViaUSB -l sur le terminal. Si par exemple l'instance nm'existe pas, notre code présent dans debug.cpp fait en sorte que rien ne sorte et donc rien ne sera afficher.

Partie 2 : Décrire les modifications apportées au Makefile de départ

Pour le Makefile de la librairie, nous sommes parties d'un makefile quelquonque d'un ancien tp puis avons modifier plusieurs lignes et même enlever quelques autres.
Nous avons tout d'abord modifié le nom de PRJSOURCE pour $(wildcard *.cpp) pour prendre tous nos fichiers .cpp.

Nous avons modifié notre target : TRG= INF1900-121135.a. Notre target représente maintenant le nom de notre librairie, ce que l'on veut créer. Nous avons aussi ajouté la liste des objets que nous voulons créer: OBJDEPS=$(src:.cpp=.o). Ce sont donc tous les documents de notre fichier librairie finissant par .o.
Nous avons mis ar -crs $(TRG) $(OBJDEPS) sous l'implémentation de la cible puisque cette commande crée une archive de librairie statique nommée &(TRG), dans notre cas libINF1900-121135, en combinant les fichier objets (.o) définis précedement dans le makefile. Pour crs, c: crée l'archive s'il n'existe pas, r: ajoute les fichiers à l'archive ou les remplace si ceux ci existent déjà et ne sont pas à jours, s: crée un index pour l'archive qui permet un "Linking "plus.

Nous avons enlevé les fichier.hex ainsi que la commande install puisque nous n'avons plus besoin. Nous avons enlevé la définition de HEXROMTRG et HEXTRG ainsi que tout les règles et dependances qui sont liées à la création des fichiers .hex
Nous avons finalement changé la commande clean pour convenir au retrait de hex, elle est maintenant: $(REMOVE) $(TRG) $(OBJDEPS) *.d ,ceci enlève du répertoire tout les fichiers .o et .d ainsi que la librairie . Nous avons enlever le install, puisque nous n'avons aucun programme à installer. Nous avons ajouté une règle vars, pour que si on fait make vars nous pouvons voir l'état de nos variables pour des fins de débogage et ceci en utilisant la commande echo pour afficher les variable demandées.

Pour le makefile de l'exécutable, nous sommes également parties du makefile de base et avons modifier plusieurs lignes. La première et simplement le nom du fichier que l'on veut compiler dans le project source.
Nous avons ajoutés ces lignes:
LIBS= INF1900-121135

lib_dir=../lib
“LIBS” correspond au nom de la librairie à lier et lib_dir correspond au path de l’executable vers la librairie. Changer l’executable nous obligerait donc à également modifier cette ligne pour aller le rechercher. Nous avons du également changer “lm $(LIBS)” en “-l ($LIBS)” parceque la librairie ne semblait pas être reconnu autrement. “-L $(lib_dir)” a àtà ajouté dans LDflags pour indiquer le répertoire ou se trouve la librairie et puis aussi “-I $(lib_dir)” dans CFLAGS.

Pour la partie débogage, nous avons tout d’abord créer le fichier Debug.cpp et Debug.h à insérer dans notre librairie. Par la suite, le makefile de l’éxécutable à été modifié dans l’étiquette Phony, où nous avons ajouté dans debug dans <<Phony: all install clean debug >> ce qui permet de debugger tout nos fichiers sources. Nous avons ensuite ajouté ces 3 lignes en dessous de all:
debug: CXXFLAGS += -DDEBUG -g
debug: CFLAGS += -DDEBUG -g
debug: install
Celles-ci permettent d’ajouter l’option déboguage pour les fichiers sources CPP et C ainsi que l’ibnstallation en même temps, permettant ainsi de débugger notre main.