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 1112

Noms:

Maxime Laroche
Marie-Claire Taché
Eduardo Falluh
Ahmed Gafsi
Nemro Yapmi Nji Monluh

Date:
17 Mars 2021

Partie 1 : Description de la librairie

Décrire la librairie construite et formée (définitions, fonctions ou classes, utilité, etc.) pour que cette partie du travail soit bien documentée pour la suite du projet au bénéfice de tous les membres de l'équipe.

Cette librairie contient 11 documents incluant le Makefile.

HBRIDGE :

Hbridge.h Hbridge.cpp
La classe Hbridge permet de contrôler la position du robot: le faire avancer, reculer, tourner à droite, droite à gauche.

void foward(uint8_t speed): cette fonction permet d'avancer, elle prend en paramètre speed de type uint8_t qui permet de contrôler la vitesse.

void backward(uint8_t speed) : cette fonction permet de reculer, elle prend en paramètre speed de type uint8_t qui permet de contrôler la vitesse de recul.

Hbridge::Hbridge(volatile uint8_t& port, const uint8_t right_direction_pin, const uint8_t left_direction_pin) : C'est un constructeur qui permet d'initialiser la position de départ.

Nous avons inclus dans ce fichier uniquement des fonctions “plug and play” qui s’utilisent pour des cas clés, mais tout de même spécifiques dû au manque de temps. L'implémentation de méthodes permettant de choisir d’autres modes et de modifier des registres précis sera alors faite plus tard.
Il sera utile dans le futur d’inclure une fonction pour pouvoir affecter des vitesses différentes aux deux moteurs du Hbridge pour permettre à celui-ci de tourner.

IO: Le fichier nommée IO.cpp continent les trois fonctions suivantes :

- void setPin(volatile uint8_t &port, const uint8_t pin, PinState state)
  - Cette fonction place l’état state à la pin et le port donné en paramètre.
- void setLED(volatile uint8_t &port, const uint8_t pins[], Color col)
  - Cette fonction s’ajoute aux fonctionalités de setPin.
  - La fonction requiert qu’une LED bicolore rouge et verte dont l’anode est passé à `pin[0]` et la cathode, à `pin[1]` (anode et cathode pour partie verte de la DEL.

    - Les state possibles sont `OFF, RED` et `GREEN`
- void setInputInterrupts()
    - Cette fonction active les input interrupts `INT0` et `INT1` situé sur les pins `PA0` et `PA1`.
    - Dans une fonction ISR, passer en paramètre `INT0_vect` ou `INT1_vect` pour gérer l'interruption au rising edge du signal.
    - Cette fonction est très spécifique comme application mais peu être facilement adaptée pour être plus généralisable quant aux port désiré pour les interruptions et le type d'interruption.
        - Nous n'avons pas fait ces modifications dû au temps limité pour produire la librairie

TIMER: Le fichier contient une classe contenant les fonctionalités clés qui font usage des timer du AVRT

- `enum TimerNum`
    - Ce enum sert à spécifier le timer à utiliser dans d'autres méthodes.
    - Contient: `TIMER0`, `TIMER1, TIMER2`
- `void startTimerCTC(TimerNum timer, uint16_t prescaler, uint16_t cycles)`
    - Cette fonction commence un timer à 0, qui compte à une vitesse égale à la vitesse de l'horloge interne divisé par le prescaler.
    - La valeur du timer est comparé à cycle et génère une interruption lorsque les deux valeurs sont égales dans `TIMERx_COMPA_vect` ou x est le numéro du timer (0, 1 ou 2)
- `void setPrescaler(volatile uint8_t &reg, uint16_t value);`
    - cette fonction applique le prescaler donné au registre reg
    - Registre: TCCRnB
    - value: 1, 8, 64, 256 ou 1024
- `uint16_t getSlowFreqCyclesKHZ(uint16_t prescaler)`
    - Cette méthode donne le nombre de cycles nécessaires pour écouler `time_step_ms` millisecondes avec le `prescaler` (fréquence ralentie) dans un timer choisi. Cette valeur est initialisé à 100.
    - La raison pour laquelle on ne donne pas l'option d'entrer le temps désiré est parce qu'il n'y aurait pas d'application:
        - le timer1 est celui qui prend le plus de temps à se rendre à sa valeur maximale (il est de 16 bits). Le temps maximal représentable sur 16 bits est 1024/(8 *10^6) = 0.128ms. Il n'a donc pas de cas général où l'on voudra attendre si peu de temps. C'est à l'utilisateur d'avoir un compteur qui incrémente ou décrémente à chaque fois que l'interruption est appelée " pour garder la notion du temps".

- void Timer::setPWMMode(volatile uint8_t & TCCRnA, uint8_t COMnx1)
    - Cette fonction met le timer TCCRnA en mode PWM n compare A ou B.

- `void Timer::setPWM(uint8_t value,volatile uint16_t & OCRnx)`
    - Cette fonction affecte la valeur value au registre comparateur OCRnx pour générer le signal PWM.
    - Il est requis d'avoir appeler la fonction setPWMMode() avant d'utiliser cette méthode pour être dans le bon mode.

UART : Le fichier UART.cpp contient les fonctions suivantes:

- void UARTInitialisation() :

    - Cette fonction permet d'initialiser le UART transmission et réception des données qui devront être affichées dans notre monitor seriel qui se situe dans le SimulIDE.

- void UARTTransmission(uint8_t data):
    - Dans le cas de cette fonction, le UDR0 est en effet le registre qui représente "data" qui est nommé dans le paramètre. Le bit UDRE0 du registre UCSR0A permet de connaître si le registre est prêt pour recevoir le data qui sera transmis. Juste qu'à ce que celui-ci ne soit prêt, le code restera dans la boucle while. D'autre part, lorsque le registre sera prêt à recevoir des données, celui-ci sortira de la boucle et assignera le data au registre UDR0.

- void readUntil(uint8_t *startAddress, uint8_t endValue):

    - Permet d'indiquer au programme ou commencer à lire et transmettre au UART les données trouvées à partir d'une adresse. Comme expliqué ci-dessus, ont lit le programme a partir du start address et cette while loop va sortir lorsque nous tombons sur une valeure équivalente a endValue.

CAN:

Can.h et Can.cpp :
Cette classe permet le contrôle de la conversion analogique/numérique du microcontrôleur. Elle utilise le PORTA. Celle-ci sera très utile à notre robot quand on va utiliser les capteurs analogiques simples.

can() : Ce constructeur permet l'initialisation du convertisseur
~can(): Ce destructeur arrête le convertisseur pour sauver sur la consommation.
uint16_t lecture(uint8_t pos) : Cette fonction prend en paramètre un uint8_t qui représente la connexion du port A au capteur analogique. Elle fait ces traitements et retourne la valeur numérique correspondant à l'entrée analogique.

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.

Makefile du projet :

Ce makefile comporte deux changements par rapport à l'original.

LIBS=librairie/librairie.a

Cette commande spécifie le chemin vers notre librairie.

$(REMOVE) $(TRG) $(TRG).map $(OBJDEPS) $(HEXTRG) *.d *.a

Dans cette commande nous avons ajoutées le “*.a” qui spécifie le type de fichier

Makefile de la librairie

Ce makefile export vers un point a (.a) et non un point hex (.hex), comment la librairie du projet. Elle contient aussi moins de Flags.

target = librairie.a

Cette commande spécifie l’extension vers la librairie.a qui est la cible

all: $(target)

Cette commande permet de compiler le fichier cible, soit target = librairie.a

librairie.a:$(OBJ)

Commande qui implémente la cible avr-ar et le flag -crs comme dans la commande suivante :

avr-ar -crs $(target) $(OBJ)