POLYTECHNIQUE DE MONTRÉAL

INF1995
PROJET INITIAL EN TRAVAIL ET ÉQUIPE

TP 8 : Mise en commun de code et formation de librairies

William Balea (1904906)
Simon Ritchot (1637043)
Gabriel-Andrew Pollo-Guilbert (1837776)
Dalyna Pak (1865507)

Remis à
Jérome COLLIN

12 mars 2018

Table des matières

1 Description de la librairie **1**
1.1 Contrôle des moteurs . . . 1
1.1.1 `motor_init()` . . . 1
1.1.2 Fonction supplémentaires . . . 1
1.2 Communication U(S)ART . . . 1
1.2.1 `uart_init` . . . 1
1.2.2 `uart_putchar` . . . 1
1.2.3 `uart_printf` . . . 2
1.3 Utilisation de la mémoire flash . . . 2
1.3.1 `memory_init` . . . 2
1.3.2 `memory_read_byte` et `memory_read_block` . . . 2
1.3.3 `memory_write_byte` et `memory_write_block` . . . 2
1.4 Utilisation des pins analogiques . . . 2
1.4.1 `adc_init` . . . 3
1.4.2 `adc_read` . . . 3
1.5 Configuration d'une minuterie . . . 3
1.5.1 `timer_init` . . . 3
1.5.2 `timer_start` . . . 3

2 Modifications au Makefile **4**
2.1 `Makefile.common` . . . 4
2.2 `Makefile` de la librairie . . . 4
2.3 `Makefile` du robot . . . 4
2.4 `Makefile` à la racine . . . 4

Références **4**

1 Description de la librairie

1.1 Contrôle des moteurs

1.1.1 `motor_init()`

La première routine contenue dans notre librairie s'occupe du contrôle des moteurs. La fonction `motor_init()` effectue essentiellement trois opérations. La première opération s'occupe d'effectuer la configuration du registre de contrôle A du compteur 0. En activant les bits de `COM0A1` et de `COM0A0` cela nous permet de définir un mode de comparaison et de réinitialisé la valeur du compteur une fois celle-ci atteinte à 0 (au bas).

Ensuite, la routine définit un mode d'opération des signaux. Dans notre cas, ceux-ci sont définit au mode «fast PWM». Étant donné que le microcontrôleur fonctionne à plus de 8 millions de cycles par secondes, il est possible de modifier le *prescaler* du compteur en ajustant la macro `MOTEUR_PRESCALER`. Elle définit une valeur qui divise les cycles d'horloge du microcontrôleur. Les valeurs possibles sont les suivantes : 1, 8, 64, 256 et 1024. Finalement, la dernière étape de la fonction `motor_init()` est d'initialiser les pins de sorties du compteur.

1.1.2 Fonction supplémentaires

Jusqu'à maintenant, il n'y a aucune fonction pour contrôler les roues directement d'un point de vue à haut niveau. Idéalement, on aurait des fonctions comme `avancer` et `tourner` qui faciliterait le mouvement du robot. On désire implémenter ces fonctions plus tard, car il faut effectuer plus de tests avec les deux roues.

1.2 Communication U(S)ART

1.2.1 `uart_init`

Les prochaines routines comprises dans notre librairie permettent d'effectuer une communication en utilisant le protocole RS232. Pour initialiser une communication, il faut avoir définit au moins les trois aspects suivants :

— le baud de la transmission (*baud rate*)
— le format de transmission
— l'émetteur ou le récepteur

Pour se faire, les registres `UBRR0H` et `UBRR0L` sont utilisé pour définir la valeur de la vitesse de transmission. La macro `BAUD` se doit d'être définie dans le code. Ensuite, les émetteurs et récepteurs sont activés à l'aide du registre `UCSR0B`. Puis, le format de transmission de 8 bits est définit à l'aide du registre `UCSR0C`.

1.2.2 `uart_putchar`

Pour envoyer un octet en transmission, la fonction `uart_puchar()` est utilisée. Pour se faire, il suffit d'écrire un octet dans le registre `UDR0` et le microcontrôleur se charge d'envoyer l'octet selon les bons paramètres. Par contre, il faut s'assurer que le dernier octet écrit a bel été envoyé avant d'en écrire un autre. Pour se faire, on attendre qu'un certain flag (`UDRE0`) du registre `UCSR0A` soit activé afin capté le moment où la transmission du dernier octet est terminée.

1.2.3 `uart_printf`

Puisque seulement des données en format textuel seront envoyées, on a écrit un *wrapper* afin de simulaer la fonction `printf` de la librairie C afin de pouvoir facilement formater du texte et l'envoyer par RS232.

1.3 Utilisation de la mémoire flash

Ensuite, nous avons introduit des routines pour utiliser la mémoire flash du robot dans notre librairie. Le robot utilise une puce 24LC512 [1] du fabricant Microchip. Cette mémoire externe de 512 KB est utilisable par le ATmega324PA via une communication I^2C/TWI.

Dans ce protocole, un seul bus est utilisé par plusieurs composantes qui discutent une à la fois. Lorsqu'une composante veut envoyer un message à une autre, elle commence par envoyer un octet de contrôle afin de s'adresser à l'autre composante. Il s'avère que la puce 24LC512 permet jusqu'à 8 puces sur le même bus. Nous avons apporter des modifications au code original pour supporter ce fait.

1.3.1 `memory_init`

Pour pouvoir lire ou écrire dans la mémoire, on doit d'abord l'initialiser avec cette fonction. Compte tenu que l'objet original `Memoire24CXXX` n'avait pas vraiment d'état, le modèle orienté objet de la mémoire fut supprimé, ce qui simplifie son utilisation.

1.3.2 `memory_read_byte` **et** `memory_read_block`

Cette première fonction permet de lire la mémoire un octet à la fois. Il suffit de fournir une adresse `addr` dans la mémoire où la lecture s'effectura. Pour la suivante, elle permet de lire une série d'octets de taille `len` dans un tampon `data`. La valeur maximal de `len` ne doit pas dépasser 127 octets.

1.3.3 `memory_write_byte` **et** `memory_write_block`

Cette fonction permet de écrire la mémoire un octet à la fois à l'adresse `addr`. Il suffit de fournir un tampon de données `data` à envoyer à la mémoire. Pour pouvoir écrire plusieurs octets à la fois, il faut spécifier la longueur `len` du tampon en paramètres. La valeur de celle-ci ne doit pas dépasser 127 octets.

1.4 Utilisation des pins analogiques

Par la suite, il y a l'introduction des routines concernant la lecture des pins analogiques. Ces dernières permettent le contrôle du convertisseur analogique/numérique (*Analog to Digital Converter – ADC*).

En effet, le microcontrôleur du robot possède des ports qui sont aptes de convertir un signal analogique en valeur numérique. En d'autres mots, la conversion analogique/numérique permet d'associer une valeur de voltage entre 0 et 5 volts à une donnée numérique allant jusqu'à 16-bits de précision.

Ces pins concernés, c'est-à-dire celles du port A, peuvent lire jusqu'à huit signaux. Celles-ci sont essentielles lorsque le robot a besoin de capteurs qui sont privés d'une transmission numérique.

1.4.1 `adc_init`

Pour initialiser l'ADC de ce système, on utilise cette fonction. Il est possible de modifier le facteur de division de la lecture en ajustant la macro `ADC_FACTEUR_DIVISION`. Compte tenu du bruit et la qualité de l'exactitude des ADCs du ATmega324PA, on applique un facteur de division de 64 par défauts. Alors les valeurs lues ont une précision de 10-bits.

1.4.2 `adc_read`

Cette méthode débute la conversion analogique/numérique. Le résultat retourné par cette fonction est sur 16 bits. Cependant, ce sont seulement les 10 bits les moins significatifs qui sont importants, mais un décalage de 2 bits vers la droite peut être effectué pour obtenir les 8 bits de poids faible. En d'autres mots, il y a les derniers 2 bits du 10 bits peuvent être ignorés. Donc, il est possible d'obtenir un résultat sur 8 bits. On envoie en paramètres la pin que l'on désire effectuer la lecture.

1.5 Configuration d'une minuterie

Finalement, on intègre à la librairie quelques routines pour partir une minuterie. On utilise le timer 1, car il offre un compteur de 16-bits pouvant aller jusqu'à environ 8 secondes avec le prescaler maximal.

1.5.1 `timer_init`

Cette fonction initialise le timer 1 du ATmega324PA. Il est possible de configurer le prescaler avec la macro `TIMER_PRESCALER`.

1.5.2 `timer_start`

On utilise un modèle semblant à un *callback*. C'est-à-dire on démarre une minuterie avec un temps et une fonction d'appel. Lorsque la minuterie expire, la fonction est appelée. Par conséquent, l'utilisateur de la librairie n'a pas toucher aux interruptions directement, ceux-ci sont géré dans la librairie et font appels aux fonctions.

Puisque le timer 1 a deux canaux, le canal A et le canal B, il est possible d'avoir deux minuteries en même temps. Pour configurer une minuterie, on utilise la structure `callback` définit comme suit

```
struct callback {
    uint16_t time;
    void (*func)(void);
};
```

On écrit le temps en millisecondes dans `time` et la fonction d'appel dans `func`. La fonction `timer_start` prend en paramètre deux de ces structures, une pour chaque canal.

2 Modifications au Makefile

2.1 `Makefile.common`

La première étape de la réfactorisation fut de prendre toute la configuration et les règles générales dans un fichier commun, appelé `Makefile.common`, afin de pouvoir diminuer le plus possibles la duplication. Dans ce fichier, il y a les configuration du compilateur, de l'éditeur de lien, de l'archiveur et du programmeur. Il y a aussi les règles générales pouvant générer les objets à partir d'un fichier C ou C++.

De plus, on y met des paramètres tels que l'horloge du microcontrôleur, la vitesse de transmission RS232 ou encore la vitesse de transmission I^2C/TWI. Normalement, ces derniers se doivent d'être définis comme macro dans les fichiers sources, mais il est possible d'ajouter ces définitions directement à partir du compilateur (*gcc*) sans avoir à les définir directement dans les fichiers.

Finalement, les règles générales pour compiler des objets à partir d'un fichier C ou C++ sont dans ce fichier. Ces règles fonctionnent bien, mais ne prennent pas en compte de la dépendance des fichiers. Les dépendances doivent être spécifiées directement dans le `Makefile` incluant `Makefile.common` afin d'optimiser les construction subséquentes.

2.2 `Makefile` de la librairie

Ce `Makefile` est assez simple puisque la majorité des règles proviennent de `Makefile.common`. Il spécifie tout simplement ses objets avec optionellement leurs dépendances. Finalement, il crée une archive des objets qui va être utilisée comme librairie.

2.3 `Makefile` du robot

Celui-ci est similaire à cel de la librairie. Il spécifie ces objets et leurs dépendances. Par contre, le fichier final généré est un fichier programmable sur le microcontrôleur. Il appel aussi le `Makefile` de la librairie afin de construire l'archive à priori. De plus, il doit s'assurer d'éditer les liens afin d'ajouter la librairie au code. Finalement, il contient aussi la règle pour installer le programme sur le microcontrôleur.

2.4 `Makefile` à la racine

Un dernier `Makefile` présent dans le répertoire se trouve dans le dossier parent de la librairie et du code du robot. Celui-ci facilite l'appel des autres `makefile`s. Par exemple, il offre les règles `clean` et `mrproper` qui nettoient les deux sous-projets. Sa règle par défaut consiste à construire le projet au complet en appelant le `Makefile` du robot.

Références

[1] *256K* I^2C ^{TM} *CMOS Serial EEPROM*. 24LC256. Microchip Technology Inc. Juin 2004. URL : `http://ww1.microchip.com/downloads/en/DeviceDoc/21203M.pdf`.

[2] *8-bit Atmel Microcontroller with 16/32/64/128K Bytes In-System Programmable Flash*. ATmega324PA. Rev. 8272G. Atmel Corporation. Jan. 2015. URL : http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-8272-8-bit-AVR-microcontroller-ATmega164A_PA-324A_PA-644A_PA-1284_P_datasheet.pdf.