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 5365

Noms:

Alexandre Gélinas
Ioana Daria Danciu
Maxence Sigouin
Vincent Grenier

Date:
17 mars 2021

Partie 1 : Description de la librairie

Bouton:

Bouton contient deux parties. Une fonction pour l'anti-rebond et une autre pour le relâchement. Antirebond permet de vérifier si le bouton est actionné lorsqu'on appuie et s'il le demeure 10 ms plus tard, afin d'annuler l'effet d'antirebond. Elle prend deux paramètres, pin qui prend la valeur du port et position qui est la broche à modifier en hexadécimal. Relâchement a comme but de retourner True lorsque le bouton est relâché. Ses paramètres sont les mêmes que ceux d'antirebond.

Can:

Can est une classe déjà fournie par le professeur. Elle permet d'initialiser le convertisseur et de convertir une valeur. Cette valeur convertie est retournée sur 16 bits. C'est donc une valeur numérique qui est convertie pour correspondre à la valeur analogique du port A. Le paramètre pos doit être de 0 à 7.

Del:

Del a comme but de changer certains bits de pin. Elle possède trois paramètres. Pin qui est la valeur que le port prend, position qui est une valeur qui permet d'indiquer la position des bits à modifier dans pin et finalement, valeur qui est un bool qui permet de décider si les bits devraient être remplacés par 1 ou 0, donc true ou false.

Atmel:

Atmel permet de modifier DDRX, donc de décider si les ports sont en entrée ou sortie. Les paramètres sont les ports A, B, C et D. Les ports peuvent avoir plusieurs modes selon ce qu'ils doivent prendre. Les modes possibles sont mode 0 qui ne fait rien, mode 1 qui met le port en entrée et mode 2 qui met le port en sortie.

Interruption:

Interruption sert à s'adapter plus facilement aux modes, en plus d'activer et de désactiver les interruptions. Le fichier inclut la classe Interruption et comporte six méthodes qui n'ont aucune valeur de retour. La première est Interruption() et a comme paramètres mode et numero. Ce dernier sera assigné à l'attribut du même nom, puis la méthode appellera deux méthodes, soit inititialiserMasqueInterruption() et initialiserDetectionInterruption(). La première n'a pas de paramètres et va déterminer la valeur de EIMSK selon la valeur du paramètre numero_. Ce dernier registre a comme fonction de générer une demande d'interruption aussitôt qu'un signal est reçu sur les broches de EICRA. La deuxième, quant à elle, a comme but de contrôler la détection d'interruptions à l'aide du registre EICRA, en déterminant ses ISCn0 et ISCn1. Elle prend le mode en paramètre et il y a quatre modes possibles :

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request
0	1	Any edge of INTn generates asynchronously an interrupt request
1	0	The falling edge of INTn generates asynchronously an interrupt request
1	1	The rising edge of INTn generates asynchronously an interrupt request

Note: 1. n = 2, 1 or 0.

Figure 1 : modes d'interruption[1]

La méthode modifierFlagInterruption() consiste à contrôler les Flags d'interruptions externes, en assignant le bon INTFn, qui dépend de numero_ au registre EIFR.

Les méthodes activer et désactiver, qui n'ont rien en paramètre, permettent d'activer les interruptions et de les désactiver, grâce aux fonctions sei() et cli().

Eeprom:

Eeprom inclut la classe eeprom et contient quatre méthodes, deux surcharges d'opérateurs, ainsi que le constructeur de la classe. Ce dernier a comme argument le pointer adresse, qui sera assigné à l'attribut privé du même nom. Les trois méthodes lire, écrire et mettre à jour sont respectivement les équivalents des fonctions eeprom_read_block(), eeprom_write_block() et eeprom_update_block().

La méthode changerAdresse prend un pointeur vers un entier et permet de changer la valeur de l'adresse (c'est un setter).

Les surcharges d'opérateur permettent d'additionner et soustraire l'adresse d'un objet EEPROM en lui additionnant/soustrayant un entier.

Timer:

Une des fonctionnalités de Timer est de déterminer un délai de temps personnalisé. Pour ce faire, nous avons énuméré deux unités de temps, ms et us, que nous avons ensuite introduites comme paramètre unite dans la fonction delaiVariable qui établit le délai personnalisé. Un autre paramètre, delai, est inclus dans cette méthode.

Il y a aussi une classe Timer qui a les principales fonctions des 3 Timers, Timer0, Timer1 et Timer2. Ces Timers possèdent des points communs comme les méthodes qu'ils possèdent ainsi que les paramètres. Les paramètres sont modeMinuterie, qui est le fonctionnement de la minuterie, modeComparaison, qui est le mode de comparaison de la sortie, et prescaler qui est le mode de prescaler de la minuterie.

La méthode initialiser prend en paramètre les différents réglages des timers et appelle les fonctions décrites plus haut pour initialiser un timer plus facilement selon nos besoins.

Les méthodes modifier registre de chaque timer permettent de choisir le registre que nous voulons modifier dépendamment du timer utilisé (donc si on l'appelle avec timer2, OCRnB devient OCR2B et ainsi de suite).

[1] **Atmel Corporation** (2015). « 8-bit Atmel Microcontroller with 16/32/64/128K Bytes In-System Programmable Flash. Datasheet », sur le site *microchip*. Consulté le 15 mars 2021. (fichier pdf, 659 p.) http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-8272-8-bit-AVR-microcontroller-ATmega164A_PA-324A_PA-644A_PA-1284_P_datasheet.pdf

La méthode obtenirPWM permet d'obtenir le PWN du Timer et retourne un unsigned int en prenant en paramètre un bool. 8-bit Atmel Microcontroller with 16/32/64/128K Bytes In-System Programmable Flash

ObtenirTop est une autre méthode qui prend un bool en paramètre, mais elle retourne un uint16_t. Cette fonction permet d'obtenir la valeur maximale du Timer.

Les fonctions de modifications (modifierModeMinuterie, modifierModeComparaisonA, modifierModeComparaisonB, modifierPrescaler, modifierInterruption et modifiercompteur) permettent de modifier les différents réglages du Timer selon nos besoins. Les réglages à modifier et leur paramètre dépendent du nom de la méthode. Ces méthodes ne retournent rien.

Timer0 (8 bits): voir Figure 2 dans l'annexe

Timer1 (16 bits): voir Figure 3 dans l'annexe

Timer2 (8 bits): voir Figure 4 dans l'annexe

Usart:

La classe Usart nous permet d'activer la communication avec le robot, par exemple pour écrire en mémoire ou la lire, via les protocoles RS232. Elle contient un constructeur qui prend en paramètre un entier, qui permet de régler la durée de modulation du signal (donc les bauds), un booléen qui permet de choisir le nombre de bits d'arrêt ainsi qu'un entier qui permet de choisir le nombre de bits envoyés entre chaque bit d'arrêt.

Les classes USART0 et USART1 dérivent de USART et ont deux méthodes. Les constructeurs permettent d'ajuster la durée de modulation du signal (les bauds) et appellent modifierRegistre de sa classe respective (USART0 l'appelle pour la classe USART0 et USART1 pour USART1). Les méthodes modifierRegistre ne prennent aucun paramètre et ne retournent rien, elles permettent de choisir les registres que nous voulons modifier dépendamment du USART qui est utilisé.

Les méthodes activerReception et activerTransmission permettent d'ajuster le registre UCSR0B afin de pouvoir communiquer en utilisant le USART du microcontrôleur. Ces deux méthodes ne retournent rien et ne prennent rien en paramètre.

La méthode modifierStopBit prend en paramètre un booléen sans rien retourner, et permet de choisir entre 1 et 2 bits d'arrêt pour le USART choisi.

La méthode modifierNombreBits ne retourne rien, prend un entier en paramètre et permet de choisir le nombre de bits transmis entre les bits d'arrêt.

La fonction transmettre ne retourne rien et prend un entier en entrée. Elle permet d'envoyer des données vers le PC.

Partie 2 : Décrire les modifications apportées au Makefile de départ

Makefile pour la librairie:

La première modification survient à la ligne pour changer le nom du projet : nous y avons inscrit le nom de notre librairie. Ensuite nous avons indiqué les différents fichiers source qui forment la librairie. Pour les include, nos fichiers .hpp se trouvent dans le répertoire ./include, nous avons donc écrit -I ./include pour que le Makefile soit capable de trouver les fichiers lors de la compilation.

En ce qui concerne le compilateur, nous avons ajouté une variable CLIB, qui n'est que l'outil utilisé pour créer la librairie, soit ar. Nous pouvons donc utiliser cette variable lorsque vient le temps de transformer les fichiers .o en un fichier .a et avons conservé le compilateur avr-gcc, qui est nécessaire pour transformer les .cpp en .o.

Par après, nous avons remplacé tout ce qui était dans LDFLAGS par -crs, car cette variable n'est utilisée que par make all pour faire la cible (dans notre cas la librairie) à partir des fichiers objets. Nous n'avions donc pas besoin de ce qui s'y trouvait précédemment.

Nous avons ensuite remplacé le .elf de TRG par .a, car nous voulions construire un fichier .a.

Pour la commande make all, nous avons ajouté un echo qui vient confirmer que la commande s'est bien exécutée. Nous avons aussi retiré la variable HEXROMTRG car nous n'avons pas besoin de fichiers .hex pour la librairie. Nous avons aussi retiré la règle pour produire ces fichiers pour la même raison. Nous avons aussi retiré le -lm $(LIBS) car nous n'avons pas besoin de lier une librairie à notre librairie. Nous avons ajouté l'option -o et la variable automatique $@ à la règle pour construire les fichiers objets pour que ces fichiers soient placés dans le même dossier que leur fichier source.

Nous avons ajouté des echo pour les commandes clean et install, qui confirment que ces commandes se sont exécutées correctement. Finalement, nous avons ajouté *.a et *.o à la commande make clean, afin d'effacer correctement ces fichiers lorsque nécessaire. Il pouvait arriver que certains fichiers n'étaient pas effacés même par le $(OBJDEPS) car ils étaient créés dans un différent répertoire, ce qui nous a poussé à ajouter le *.o.

Makefile de test:

Ce Makefile est utilisé pour construire une application très simple qui lie notre librairie.
Premièrement, nous avons créé les variables EXECDIR et LIBDIR, qui sont les répertoires ou se trouvent les fichiers pour l'application et les fichiers pour la librairie. Nous avons modifié le nom du projet et les fichiers source comme à la normale, en utilisant les variables créées dans ce but.

Pour lier la librairie, nous avons indiqué le répertoire dans lequel elle se trouve à l'aide de l'option -L et de la variable LIBDIR, et nous avons indiqué son nom avec l'option -l. Il y a un deux-points (:) juste après l'option puisque le nom de notre librairie ne commence pas par lib et avr-gcc n'arrive pas à lier notre librairie autrement.

Comme le but était de créer des fichiers comme dans les autres travaux pratiques, nous n'avons pas changé autre chose à par la commande make clean pour qu'elle puisse effacer les fichiers .d dans le répertoire ou se trouve les fichiers de l'application.

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)*

Annexe 1

```
*                Mode
*
*            Mode d'opération     |   Valeur Plafond      |   Mise à jour de OCRnX    |   TOVn Flag
*
* Mode  0 :        Normal          |        0xFF           |          Immédiat         |      MAX
* Mode  1 :   PWM, Phase Correct   |        0xFF           |            TOP            |     BOTTOM
* Mode  2 :         CTC            |        0CRA           |          Immédiat         |      MAX
* Mode  3 :    Fast PWM, 8-bit     |        0xFF           |           BOTTOM          |      TOP
* Mode  4 :      (Réservé)         |        ---            |            ---            |      ---
* Mode  5 :   PWM, Phase Correct   |        0CRA           |            TOP            |     BOTTOM
* Mode  6 :      (Réservé)         |        ---            |            ---            |      ---
* Mode  7 :       Fast PWM         |        0CRA           |           BOTTOM          |      TOP
*
* MAX    = 0xFF
* BOTTOM = 0x00
*
*             Prescaler
*
* Mode  0  : Pas de source de la minuterie
* Mode  1  : clk /    1
* Mode  2  : clk /    8
* Mode  3  : clk /   64
* Mode  4  : clk /  256
* Mode  5  : clk / 1024
* Mode  6  : Source de la minuterie sur Tn, minuterie sur front descendant
* Mode  7  : Source de la minuterie sur Tn, minuterie sur front montant
*
*             Mode de comparaison de la sortie
*
* Mode  0  : Opération normal, OCnA/OCnB déconnecté
* Mode  1  : Toggle on Match compare
* Mode  2  : Clear on Match compare (met à 0)
* Mode  3  : Set on Match compare (met à 1)
*
*              Gestion des interruptions
*
* Mode 0   : Rien
* Mode 1   : Compare Match A
* Mode 2   : Compare Match B
* Mode 3   : Compare Match Overflow
```

Figure 2 : Timer 0

```
*                Mode
*
*            Mode d'opération     |   Valeur Plafond      |   Mise à jour de OCRnX    |   TOVn Flag
*
* Mode  0 :        Normal          |       0xFFFF          |          Immédiat         |      MAX
* Mode  1 :   PWM, Phase Correct   |       0x00FF          |            TOP            |     BOTTOM
* Mode  2 :   PWM, Phase Correct   |       0x01FF          |            TOP            |     BOTTOM
* Mode  3 :   PWM, Phase Correct   |       0x03FF          |            TOP            |     BOTTOM
* Mode  4 :         CTC            |       0CRnA           |          Immédiat         |      MAX
* Mode  5 :    Fast PWM, 8-bit     |       0x00FF          |           BOTTOM          |      TOP
* Mode  6 :    Fast PWM, 9-bit     |       0x01FF          |           BOTTOM          |      TOP
* Mode  7 :    Fast PWM, 10-bit    |       0x03FF          |           BOTTOM          |      TOP
* Mode  8 : PWM-Phase-Frequency C. |        ICRn           |           BOTTOM          |     BOTTOM
* Mode  9 : PWM-Phase-Frequency C. |       0CRnA           |           BOTTOM          |     BOTTOM
* Mode 10 :   PWM, Phase Correct   |        ICRn           |            TOP            |     BOTTOM
* Mode 11 :   PWM, Phase Correct   |       0CRnA           |            TOP            |     BOTTOM
* Mode 12 :         CTC            |        ICRn           |          Immédiat         |      MAX
* Mode 13 :      (Réservé)         |        ---            |            ---            |      ---
* Mode 14 :       Fast PWM         |        ICRn           |           BOTTOM          |      TOP
* Mode 15 :       Fast PWM         |       0CRnA           |           BOTTOM          |      TOP
*
*             Prescaler
*
* Mode  0  : Pas de source de la minuterie
* Mode  1  : clk /    1
* Mode  2  : clk /    8
* Mode  3  : clk /   64
* Mode  4  : clk /  256
* Mode  5  : clk / 1024
* Mode  6  : Source de la minuterie sur Tn, minuterie sur front descendant
* Mode  7  : Source de la minuterie sur Tn, minuterie sur front montant
*
*             Mode de comparaison de la sortie
*
* Mode  0  : Opération normal, OCnA/OCnB déconnecté
* Mode  1  : Toggle on Match compare | 0CnB déconnecté en PWM
* Mode  2  : Clear on Match compare (met à 0)
* Mode  3  : Set on Match compare (met à 1)
*
```

Figure 3 : Timer 1

```
*                Mode
*
*            Mode d'opération     |   Valeur Plafond      |   Mise à jour de OCRnX    |   TOVn Flag
*
* Mode  0 :        Normal          |        0xFF           |          Immédiat         |      MAX
* Mode  1 :   PWM, Phase Correct   |        0xFF           |            TOP            |     BOTTOM
* Mode  2 :         CTC            |        0CRA           |          Immédiat         |      MAX
* Mode  3 :    Fast PWM, 8-bit     |        0xFF           |           BOTTOM          |      TOP
* Mode  4 :      (Réservé)         |        ---            |            ---            |      ---
* Mode  5 :   PWM, Phase Correct   |        0CRA           |            TOP            |     BOTTOM
* Mode  6 :      (Réservé)         |        ---            |            ---            |      ---
* Mode  7 :       Fast PWM         |        0CRA           |           BOTTOM          |      TOP
*
* MAX    = 0xFF
* BOTTOM = 0x00
*
*             Prescaler
*
* Mode  0  : Pas de source de la minuterie
* Mode  1  : clk /    1
* Mode  2  : clk /    8
* Mode  3  : clk /   32
* Mode  4  : clk /   64
* Mode  5  : clk /  128
* Mode  6  : clk /  256
* Mode  7  : clk / 1024
*
*             Mode de comparaison de la sortie
*
* Mode  0  : Opération normal, OCnA/OCnB déconnecté
* Mode  1  : Toggle on Match compare
* Mode  2  : Clear on Match compare (met à 0)
* Mode  3  : Set on Match compare (met à 1)
*
*              Gestion des interruptions
*
* Mode 0   : Rien
* Mode 1   : Compare Match A
* Mode 2   : Compare Match B
* Mode 3   : Compare Match Overflow
```

Figure 4 : Timer 2