Examen de Systèmes Embarqués
partie Microcontrôleur

EILCO, ING2
Semestre 1, session 2
P.-A. Hébert
07/05/2024

Consignes

- Sans document. Pas de calculatrice.
- Durée : 1h30.
- Énoncé : 9 questions sur 22 points, sur 5 pages + 4 pages d'annexe (programmation timer).
- Codez l'ATmega328p, en langage Arduino (C/C++).
- Justifiez tous les calculs.
- En cas de souci, supposez la question résolue, et passez à la suivante.

Exercice 1 : programmation d'un afficheur 7-segments

Fonctionnement d'un afficheur 7-segments

Un afficheur 7-segments est un afficheur rudimentaire, constitué de 7 segments (des « bâtons-leds ») + 1 point lumineux, qui sert en particulier à la représentation des chiffres, par exemple sur les « radio-réveils » (cf. musée des objets du quotidien du siècle dernier).

Le schéma de l'afficheur (cf. figure 1) montre ses 7 segments indépendants, nommés - dans le sens horaire - de **A** (segment horizontal du haut) à **G** (segment du milieu). Chacun de ces segments est piloté individuellement par une ligne d'entrée (de même nom). Un segment s'allume si son entrée est au niveau haut (5V), et s'éteint dans le cas contraire. Pour allumer le segment du haut, il faut donc appliquer un niveau de tension haut sur l'entrée **A**.

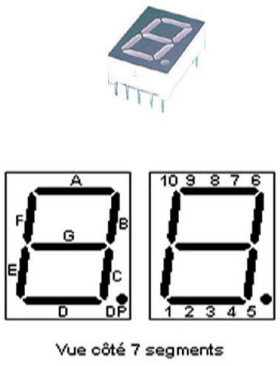


FIGURE 1 – Afficheur 7-segments

L'afficheur est **connecté aux broches 8 à 14** de l'arduino (**respectivement**, aux entrées **A** à **G**). (Pour simplifier, on considère en effet que la broche 14 est bien utilisable). Enfin, remarquez que l'entrée du point (DP) n'est pas connectée (inutile ici).

Par exemple, l'affichage du chiffre « 7 » s'obtient en mettant sous tension les broches 8, 9 et 10 (car elles sont connectées respectivement aux segments A, B et C), et en mettant à 0V les 4 autres entrées de l'afficheur.

Description et rôle du bouton poussoir

Un bouton poussoir relié à la masse (GND, 0V) est connecté à la broche 7 de l'arduino (donc sur le **port D**). Son appui sert à provoquer l'affichage d'un chiffre sur l'afficheur (sinon il est éteint).

Questions

Q 1. Écrivez la fonction `init_ports` qui configure les broches de l'Arduino Uno utiles au contrôle de l'afficheur et du bouton. *(4 points)*

Vous utilisez exclusivement les registres de l'ATmega328p, en veillant à ne pas modifier la configuration des lignes d'entrée-sortie non exploitées.

Q 2. Pour chaque chiffre de 0 à 4, déterminez la valeur à écrire dans le registre de données du port D pour l'afficher. *(2 points)*

- Le 8ième bit (celui correspondant au point) sera systématiquement mis à 0 (car inutilisé).
- Placez ces valeurs dans un tableau `chiffres` déclaré **globalement** (utile dans la question suivante). Autrement dit, déclarez et initialisez ce tableau.
- Précisez son type C/C++, en optant pour le format le plus économe en mémoire.

Q 3. Écrire la fonction `loop` de manière à afficher les chiffres de 0 à 4, en boucle. *(4 points)*

- Après l'affichage du 4, **revenez au 0**.
- Le changement de chiffre se fait toutes les **demi-secondes** (utilisez la fonction `void delay(int ms)`)
- Un chiffre ne s'allume lors du changement, qui si le bouton est **pressé**. Dans le cas contraire, l'afficheur n'affiche rien pendant la demi-seconde.
- Manipulez les **registres** des ports, en veillant à **préserver** (=maintenir) les parties de registres qui concernent des broches non utilisées.
- Évitez - de préférence - les instructions « blocantes ».
- Définissez des variables globales si nécessaire.
- Ne programmez pas d'interruptions.

Exercice 2 : programmation d'une régulation

L'exercice est **indépendant** du précédent.

Fonctionnement d'une régulation de température proportionnelle

On souhaite mettre en place une régulation de température d'une pièce, avec une *loi de commande proportionnelle*. L'idée est de chauffer la pièce dès que la température mesurée est inférieure à la *température de consigne* (= température désirée), et ce **d'autant plus fortement** que l'écart entre les températures est important. Le radiateur dispose en effet d'une entrée analogique qui permet de régler sa puissance de chauffe : celle-ci est directement proportionnelle à son niveau d'entrée (100 % pour 5V, 0 % pour 0V).

Voici en outre la *loi de commande*, qui sert à déterminer la puissance de chauffe requise en fonction de la *température mesurée*, pour une *température de consigne* donnée (cf. figure 2).

Pour produire ce signal analogique (de tension variable, entre 0 et 5V), il suffit de générer un signal numérique de type PWM (ou MLI en français : « Modulation de Largeur d'Impulsion »). En filtrant ce signal numérique, il est en effet possible de moyenner/lisser sa tension, et d'obtenir une tension

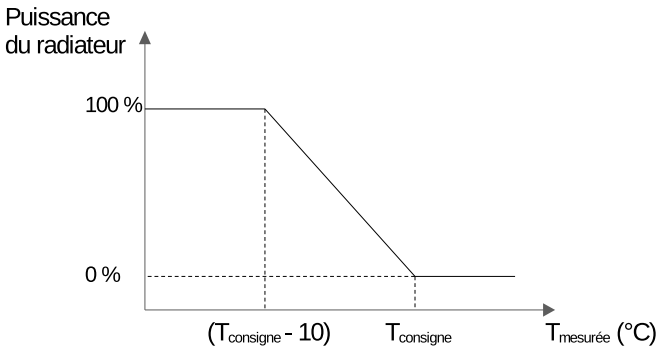


FIGURE 2 – Loi de commande du chauffage.

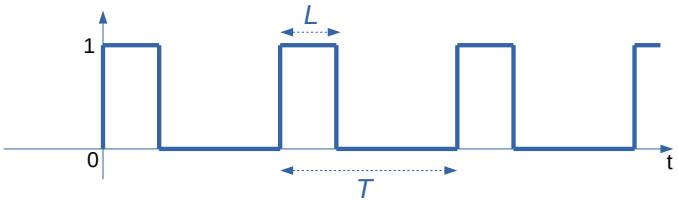
continue, proportionnelle au **rapport cyclique** α (c'est le ratio entre la durée de l'impulsion et la durée de la période).

En résumé : pour obtenir une puissance de chauffage p (exprimée en pourcentage), il suffit de générer un signal MLI de rapport cyclique p.

Génération d'un signal *MLI* (ou *PWM* en anglais)

Vous allez mettre en place la génération d'un signal à Modulation de Largeur d'Impulsion (*MLI* ou *PWM* en anglais).

FIGURE 3 – Description d'un signal MLI.



Un signal `MLI` a donc les caractéristiques suivantes :

- C'est un signal logique de **période** T. En sortie de l'Arduino, les états logiques 0 et 1 correspondent respectivement aux tensions 0V et 5V.
- Outre sa période, il est caractérisé par une **largeur d'impulsion** L ($L \leq T$). Mais on lui préfère souvent son **rapport cyclique** : $\alpha = \frac{L}{T} \in [0, 100\%]$.

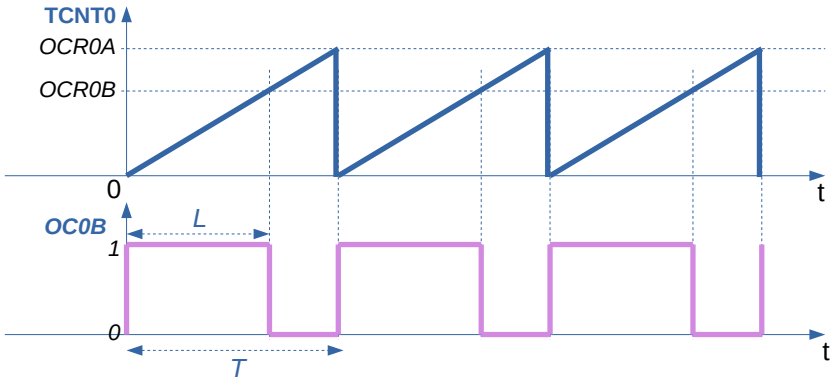
Dans la suite, vous allez travailler avec le **timer 0** (on suppose qu'il n'est pas monopolisé par les fonctions de temps Arduino).

Vous le programmerez dans un mode « *Fast PWM* » (cf. figure 4) :

- Attention, il y a 2 modes possibles ! Il faudra sélectionner celui qui définit le plafond du domaine de comptage (valeur TOP) avec la valeur du registre de comparaison A (`OCR0A`).
- Dans ce mode, le registre de comptage évolue de 0 jusque `OCR0A` (comme dans le mode CTC).

— Si activées (cf. registre `TCCR0A`), les broches de sortie-comparaison du timer 0 ($OC0A$ et $OC0B$) sont remises à 1 en début de cycle de comptage, et passent à 0 sur une comparaison A (pour $OC0A$), ou une comparaison B (pour $OC0B$).
— Pour notre objectif, seule la sortie $OC0B$ est utile : elle repasse à 1 en début de période, et s'annule sur la comparaison B, pour abréger l'impulsion.
— Attention, le signal généré diffère donc de celui du mode CTC. En particulier, sa période égale la durée du cycle de comptage. Ne vous trompez pas de formule.
— C'est donc bien un mode MLI : sa période T se règle grâce au registre de comparaison A, et la largeur d'impulsion L grâce au registre de comparaison B.

FIGURE 4 – Description du mode Fast PWM.



Questions

Q 4. Déterminez et justifiez la valeur du couple *(prédiviseur, valeur de comparaison A)* qui génère un signal de fréquence $F \simeq 1600Hz$, avec une *résolution temporelle* **la plus faible possible**.*(2 points)*

Pour rappel, la *résolution temporelle* est la *période d'incrémentation*, soit la durée qui sépare 2 incrémentations successives d'un timer. Et l'Arduino Uno est cadencé à **16 Mhz**.

Aide au calcul : $\frac{10000}{128} = 78.125$. Mais attention, le prédiviseur 128 n'est pas disponible sur le timer 0, il reste encore un peu de calcul.

Q 5. Écrivez la fonction `init_timer0`. *(3 points)*

Elle doit :
— Produire un signal MLI de fréquence $F \simeq 1600Hz$ sur la broche $OC0B$.
— Initialiser la largeur d'impulsion à l'aide du `registre de comparaison B`. Au départ, on ne veut pas de chauffage, donc une tension moyenne de 0V, donc un rapport cyclique proche de 0%, le plus petit possible (sa mise à jour fait l'objet de la prochaine question).

Vous produirez le signal MLI en manipulant les bits $COM0A1$, $COM0A0$ (pour la comparaison A), et $COM0B1$, $COM0B0$ (pour la comparaison B).

Note : ces 2 derniers bits ont exactement le même rôle que les 2 autres, mais ils contrôlent la broche de sortie $OC0B$ au lieu de $OC0A$.

Q 6. Écrivez la fonction `setup` pour mettre en place la génération du signal MLI. *(1 point)*
Elle doit configurer le timer 0, et orienter la broche $OC0B$.

Q 7. Écrivez la fonction `void rapport_cyclique(float p)`. *(2 points)*

Elle règle le rapport cyclique du signal MLI au pourcentage p transmis en paramètre (une marge d'erreur est tolérée).

Q 8. Écrivez la fonction `float commande()`. *(3 points)*

Elle retourne le pourcentage de puissance p, calculé - grâce à la loi de commande (cf. figure 2) - en fonction de la température mesurée et de la température de consigne.

— La température mesurée (en °C) est donnée par la fonction `float get_temperature()` (supposée définie).
— La température de consigne (en °C) est contenue dans la variable globale `Tc` (supposée définie).

Q 9. Écrivez la fonction `loop`. *(1 point)*

Elle doit mettre à jour le rapport cyclique du signal émis par le timer, en fonction de la température mesurée, aussi souvent que possible.